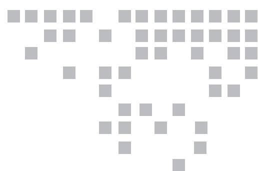
互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



第 1 章 Chapter 1

计算为王

—— 基于可扩展哈希的受控副本分布策略 CRUSH

HZ BOOKS 华章图书

大部分存储系统将数据写入到后端存储设备之后，数据很少会在设备之间再次移动。这就存在一个潜在问题：即使一个数据分布已经趋于完美均衡的系统，随着时间的推移，新的空闲设备不断加入，老的故障设备不断退出，数据也会重新变得不均衡，这种情况在大型分布式存储系统中尤为常见。

一种可行的解决方案是将数据以足够小的粒度打散，然后完全随机地分布在所有存储设备之间，这样，从概率上而言，如果系统运行的时间足够长，所有设备的空间利用率将会趋于均衡。当新设备加入后，数据会随机地从不同的老设备迁移过来；同样，当老设备因为故障退出后，其原有数据会随机迁出至其他正常设备。这样整个系统将一直处于动态平衡过程之中，从而能够适应任何类型的负载和拓扑结构变化。进一步的，因为任何数据（可以是文件、块设备等）都被打散成为多个碎片然后写入不同的底层存储设备，从而使得在大型分布式存储系统中获得尽可能高的 I/O 并发和汇聚带宽成为可能。

一般而言，使用哈希函数可以达到上述目的，但是实际应用中还需要解决两个问题：一是如果系统中存储设备数量发生变化，如何最小化数据迁移量从而使得系统尽快恢复平衡；二是在大型（PB 级及以上）分布式存储系统中，数据一般包含多个备份，如何合理分布这些备份从而尽可能地使得数据具有较高的可靠性。因此，需要对普通哈希函数加以扩展，使之能够解决上述问题，Ceph 称为 CRUSH（Controlled Replication Under Scalable Hashing）。

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

顾名思义，CRUSH 是一种基于哈希的数据分布算法。以数据唯一标识符、当前存储集群的拓扑结构以及数据备份策略作为 CRUSH 输入，可以随时随地通过计算获取数据所在的底层存储设备（例如磁盘）位置并直接与其通信，从而避免查表操作，实现去中心化和高度并发。CRUSH 同时支持多种数据备份策略，典型如镜像、RAID 及其衍生的纠删码等，并受控地将数据的多个备份映射到集群不同物理区域中的底层存储设备之上，从而保证数据可靠性。上述这些特性使得 CRUSH 特别适用于对可扩展性、性能以及可靠性都有极高要求的大型分布式存储系统。

本章按照如下形式组织：首先，我们介绍 CRUSH 需要解决的问题，由此引出 CRUSH 最重要的基本选择算法——straw 及其改进版本 straw2；其次，完成基本算法分析后，我们介绍如何将其进一步拓展至具有复杂层级结构的真实集群，为了实现上述目的，我们首先介绍集群拓扑结构和数据分布规则的具体描述形式，然后以此为基础介绍 CRUSH 的完整实现；最后，由于实际应用中集群拓扑结构千变万化，CRUSH 配置相对复杂，我们结合一些实际案例，分析如何针对 CRUSH 进行深度定制，以满足生产环境中形式各异的数据分布需求，同时介绍如何通过人工调整的方式来解决生产环境中常见的、因为各种各样的因素所导致的数据分布不均衡问题。

1.1 straw 及 straw2 算法简介

Ceph 在设计之初被定位于管理大型分级存储网络，网络中的不同层级具有不同程度的灾难容忍程度，因此也称为容灾域（或者安全域）。图 1-1 是一个典型 Ceph 集群的层级结构：

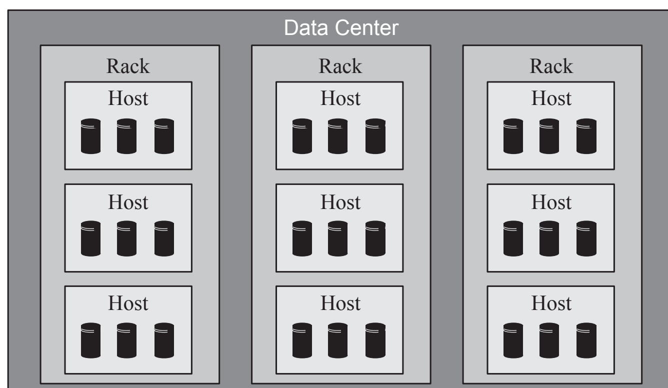


图 1-1 一个典型的 Ceph 集群

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

图 1-1 中，单个主机包含多个磁盘，每个机架包含多个主机并采用独立的供电和网络交换系统，从而可以将整个集群以机架为单位划分为若干容灾域。为了实现高可靠性，实际上要求数据的多个副本分布在不同机架的主机磁盘之上。因此，CRUSH 首先应该是一种基于层级的深度优先遍历算法。此外，上述层级结构中，每个层级的结构特征也存在差异，一般而言越处于顶层其结构变化的可能性越小，反之越处于底层则其结构变化越频繁，例如大多数情况下一个 Ceph 集群自始至终只对应一个数据中心，但是主机或者磁盘数量随时间流逝则可能一直处于变化之中。因此，从这个角度而言，CRUSH 还应该允许针对不同的层级按照其特点设置不同的选择算法，从而实现全局和动态最优。

在 CRUSH 的最初实现中，Sage Weil 一共设计了 4 种不同的基本选择算法，这些算法是实现其他更复杂组合算法的基础，表 1-1 罗列了它们各自的优缺点。

表 1-1 CRUSH 基本选择算法对比

通过对比每种算法“添加元素”和“删除元素”产生的数据迁移量来评判其好坏

算法 / 对比项	unique	list	tree	straw
时间复杂度	$O(1)$	$O(N)$	$O(log(N))$	$O(N)$
添加元素	差	最好	好	最好
删除元素	差	差	好	最好

由表 1-1 可见，unique 算法执行效率最高但是抵御结构变化能力最差；straw 算法执行效率较低但是抵御结构变化能力最好；list 和 tree 算法执行效率和抵御结构变化能力介于上述两者之间。如果综合考虑呈爆炸式增长的存储空间需求（导致需要添加元素）、在大型分布式存储系统中某些部件故障是常态（导致需要删除元素）以及愈发严苛的数据可靠性需求（导致需要将数据副本存储在更高级别的容灾域中，例如不同的数据中心），那么针对任何层级采用 straw 算法都是一个不错的选择。事实上，这也是 CRUSH 算法的现状，在大多数将 Ceph 用于生产环境的案例中，除了 straw 算法之外，其他 3 种算法基本上形同虚设，因此我们将重点放在 straw 算法的分析上。

顾名思义，straw 算法将所有元素比作吸管，针对指定输入，为每个元素随机计算一个长度，最后从中选择长度最长的那个元素（吸管）作为结果输出，这个过程也被形象地称为抽签（draw），对应元素的长度称为签长。

显然 straw 算法的关键在于如何计算签长。理论上，如果所有元素构成完全一致，那

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

么只需要将指定输入和元素自身唯一编号作为哈希输入即可计算出对应元素的签长。因此，如果样本容量足够大，那么最终所有元素被选择的概率是相等的，从而保证数据在不同元素之间均匀分布。然而实际中前期规划的再好的集群，其存储设备随着时间推移也会逐渐趋于异构化，典型如因为批次不同而导致的磁盘容量差异（参照摩尔定律，磁盘容量每 18 个月翻一番），显然，在此情况下，我们不应该也无法对所有设备一视同仁，因此需要在 CRUSH 算法中引入一个额外的参数——权重来体现这种差异，让权重大（对应容量大）的设备分担更多的数据，权重小（对应容量小）的设备分担更少的数据，从而使得数据在异构存储网络中也能合理的分布。

上述过程应用于 straw 算法，则可以通过使用权重对签长的计算过程进行调整来实现，即我们总是倾向于让权重大的元素获得更大的签长，让权重小的元素获得更小的签长。因此，此时 straw 算法执行结果取决于三个因素：固定输入、元素编号和元素权重，这其中元素编号起的是随机种子的作用，所以针对固定输入，straw 算法实际上只受元素权重的影响。进一步的，如果每个元素的签长只和自身权重相关，则可以证明此时 straw 算法对于添加元素和删除元素的处理都是最优的，我们以添加元素为例进行论证：

1）假定当前集合中一共包含 n 个元素：

$(e_1, e_2, \cdots, e_n)$

2）向集合中添加新元素 e_{n+1}：

$(e_1, e_2, \cdots, e_n, e_{n+1})$

3）针对任意输入 x, 加入 e_{n+1} 之前，分别计算每个元素签长并假定其中最大值为 d_{max}：

$(d_1, d_2, \cdots, d_n)$

4）因为新元素 e_{n+1} 的签长计算只和自身编号及自身权重相关，所以可以使用 x 独立计算其签长（同时其他元素的签长不受 e_{n+1} 加入的影响），假定为 d_{n+1}；

5）又因为 straw 算法总是选择最大的签长作为最终结果，所以：

如果 $d_{n+1} > d_{max}$，那么 x 将被重新映射至新元素 e_{n+1}；反之对 x 的已有映射结果无任何影响。

可见，添加一个元素，straw 算法会随机地将一些原有元素中的数据重新映射至新加

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

入的元素之中；同理，删除一个元素，straw 算法会将该元素中全部数据随机地重新映射至其他元素之中。因此无论添加或者删除元素，都不会导致数据在除被添加或者删除之外的两个元素（即不相关的元素）之间进行迁移。

理论上 straw 算法是非常完美的，然而在 straw 算法实现整整 8 年之后，得益于 Ceph 应用日益广泛，不断有用户向社区反馈每次集群有新的 OSD 加入或者旧的 OSD 删除时总会引起不相关的数据迁移，Sage 被迫开始针对 straw 算法已有实现重新进行审视。原 straw 算法伪代码如下：

```
max_x = -1
max_item = -1
for each item:
    x = hash(input, r)
x = x * item_straw
if x > max_x:
max_x = x
max_item = item
return item
```

可见，算法选择的结果取决于每个元素根据输入（input）、随机因子（r）和 item_straw 计算得到的签长，而 item_straw 通过权重计算得到：

```
reverse = rearrange all weights in reverse order
straw = -1
weight_diff_prev_total = 0
for each item:
    item_straw = straw * 0x10000
    weight_diff_prev = (reverse[current_item] - reverse[prev_item]) * items_remain
    weight_diff_prev_total += weight_diff_prev
    weight_diff_next = (reverse[next_item] - reverse[current_item]) * items_remain
    scale = weight_diff_prev_total / (weight_diff_prev_total + weight_diff_next)
    straw *= pow(1 / scale, 1 / items_remain)
```

原 straw 算法实现中，将所有元素按其权重进行逆序排列后逐个计算每个元素的 item_straw，计算过程中不断累积当前元素与前后元素的权重差值，以此作为计算下一个元素 item_straw 的基准，因此原有实现中 straw 算法的最终选择结果不但取决于每个元素的自身权重，而且也和集合当中所有其他元素的权重强相关，从而导致每次有元素加入当前集合或者从当前集合中删除时，都会引起不相关的数据迁移。

出于兼容性考虑，Sage 引入了一种新的算法对原有的 straw 算法进行修正，称

HZ Books 华章图书

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

为 straw2。修正后的 straw2 算法在计算签长时仅使用元素自身权重，因此可以完美反映 Sage 的初衷（也因此得以避免不相干的数据迁移），同时计算也更加简单，其伪代码如下：

```
max_x = -1
max_item = -1
for each item:
  x = hash(input, r)
x = ln(x / 65536) / weight
if x > max_x:
    max_x = x
    max_item = item
return max_item
```

上述逻辑中，针对输入和随机因子执行哈希后，结果落在 [0,65535] 之间，因此 x / 65536 必然小于 1，对其取自然对数（ln(x / 65536)）后结果为负值。进一步地，将其除以自身权重（weight）后，则权重越大，结果越大（因为负得越少），从而体现我们所期望的每个元素权重对于抽签结果的正反馈作用。

1.2 CRUSH 算法详解

HZ BOOKS 华章图书

CRUSH 算法基于权重将数据映射至所有存储设备之间，这个过程是受控的并且高度依赖于集群的拓扑描述——cluster map，不同的数据分布策略通过制定不同的 placement rule 实现，后者实际上是一组包括最大副本数、容灾级别等在内的自定义约束条件，例如针对图 1-1 所示的集群，我们可以通过一条 placement rule 将互为镜像的 3 个数据副本（这也是 Ceph 的默认数据备份策略）分别写入位于不同机架的主机磁盘之上，以避免所有副本同时掉电从而导致业务中断。

针对指定输入 x，CRUSH 将输出一个包含 n 个不同目标存储对象（例如磁盘）的集合。CRUSH 的计算过程中仅仅使用 x、cluster map 和 placement rule 作为哈希函数的输入，因此如果 cluster map 不发生变化（一般而言 placement rule 不会轻易变化），那么结果就是确定的；同时因为使用的哈希函数是伪随机的，所以 CRUSH 选择每个目标存储对象概率相对独立（然而我们在后面将会看到——受控的副本策略改变了这种独立性），从而可以保证数据在整个集群之间均匀分布。

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

1.2.1　集群的层级化描述——Cluster Map

cluster map 是 Ceph 集群拓扑结构的逻辑描述形式。实际应用中 Ceph 集群通常具有形如“数据中心→机架→主机→磁盘”（参考图 1-1）这样的树状层级关系，所以 cluster map 可以使用树这种数据结构来实现——每个叶子节点都是真实的最小物理存储设备（例如磁盘），称为 device；所有中间节点统称为 bucket，每个 bucket 可以是一些 devices 的集合，也可以是低一级的 buckets 集合；根节点称为 root，是整个集群的入口。每个节点都拥有唯一的数字 ID 和类型，以标识其在集群中所处的位置和层级，但是只有叶子节点，也就是 device 才拥有非负 ID，表明其是承载数据的最终设备。节点的权重属性用于对 CRUSH 的选择过程进行调整，使得数据分布更加合理，上一级节点权重是其所有孩子节点的权重之和。

表 1-2 列举了 cluster map 中一些常见的节点（层级）类型。

表 1-2　cluster map 常见的节点类型

类型 ID	类型名称	类型 ID	类型名称
0	osd	6	pod
1	host	7	room
2	chassis	8	datacenter
3	rack	9	region
4	row	10	root
5	pdu		

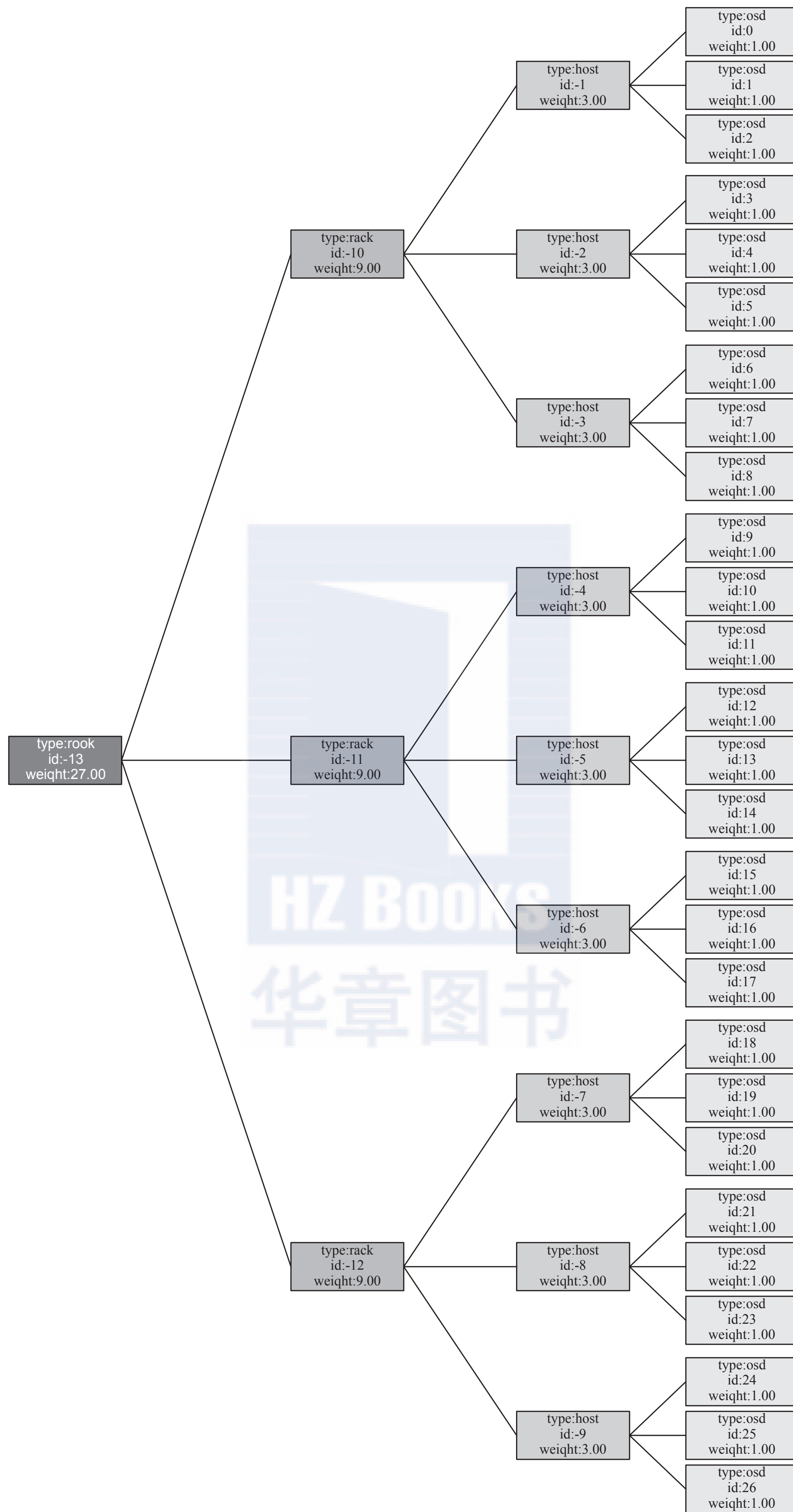
需要注意的是，这里并非强调每个 Ceph 集群都一定需要划分为 11 个层级，表 1-2 中每种层级类型的名称也不固定，而是都可以根据自己的喜好进行修改和裁剪。假定所有磁盘规格一致（这样每个磁盘的权重一致），我们可以给出图 1-1 所示集群的 cluster map 描述，如图 1-2 所示。

实现上，类似图 1-2 中这种树状的层级关系在 cluster map 中是通过一张二维映射表建立起来的：

```
<bucket, items>
```

树中的每个节点都是一个 bucket（device 也被抽象为一种 bucket 类型），每个 bucket 都只保存自身所有直接孩子的编号。当 bucket 类型为 device（对应图 1-2 中的 osd）时，容易知道此时其对应的 items 列表为空，即 bucket 为叶子节点。

互动出版网
www.china-pub.com
专业图书网上第一专营店
www.china-pub.com
独家提供样章



HZ BOOKS
华章图书

图 1-2 集群的 cluster map 描述（对应集群参考图 1-1）

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

1.2.2 数据分布策略——Placement Rule

使用 cluster map 建立对应集群的拓扑结构描述之后，可以定义 placement rule 来完成数据映射。

每条 placement rule 可以包含多个操作，这些操作共有 3 种类型：

（1）take

take 从 cluster map 选择指定编号的 bucket（即某个特定的 bucket），并以此作为后续步骤的输入。例如系统默认的 placement rule 总是以 cluster map 中的 root 节点作为输入开始执行的。

（2）select

select 从输入的 bucket 当中随机选择指定类型和数量的条目（items）。Ceph 当前支持两种备份策略——多副本和纠删码，相应的有两种 select 算法——firstn 和 indep。实现上两种算法都是深度优先，并无显著不同，主要区别在于纠删码要求结果是有序的，因此，如果无法得到满足指定数量（例如 4）的输出，那么 firstn 会返回形如 [1,2,4] 这样的结果，而 indep 会返回形如 [1,2,CRUSH_ITEM_NONE,4] 这样的结果，即 indep 总是返回要求数量的条目，如果对应的条目不存在（即选不出来），则使用空穴进行填充。select 执行过程中，如果选中的条目故障、过载或者与其他之前已经被选中的条目冲突，都会触发 select 重新执行，因此需要指定最大尝试次数，防止 select 陷入死循环。

（3）emit

emit 输出最终选择结果给上级调用者并返回。

可见，一条 placement rule 中真正起决定性作用的是 select 操作。

为了简化 placement rule 配置，select 操作也支持容灾域模式。以 firstn 为例，如果为容灾域模式，那么 firstn 将返回指定数量的叶子设备，并保证这些叶子设备位于不同的、指定类型的容灾域之下。因此，在容灾域模式下，一条最简单的 placement rule 可以只包含如下 3 个操作：

```
take(root)
select(replicas, type)
emit(void)
```

上述 select 操作中的 type 为想要设置的容灾域类型，例如设置为 rack，则 select 将

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

保证选出的所有副本都位于不同机架的主机磁盘之上；也可以设置为 host，那么 select 只保证选出的所有副本都位于不同主机的磁盘之上。

图 1-3 以 firstn 为例展示了 select 从指定的 bucket 当中查找指定数量条目的过程：

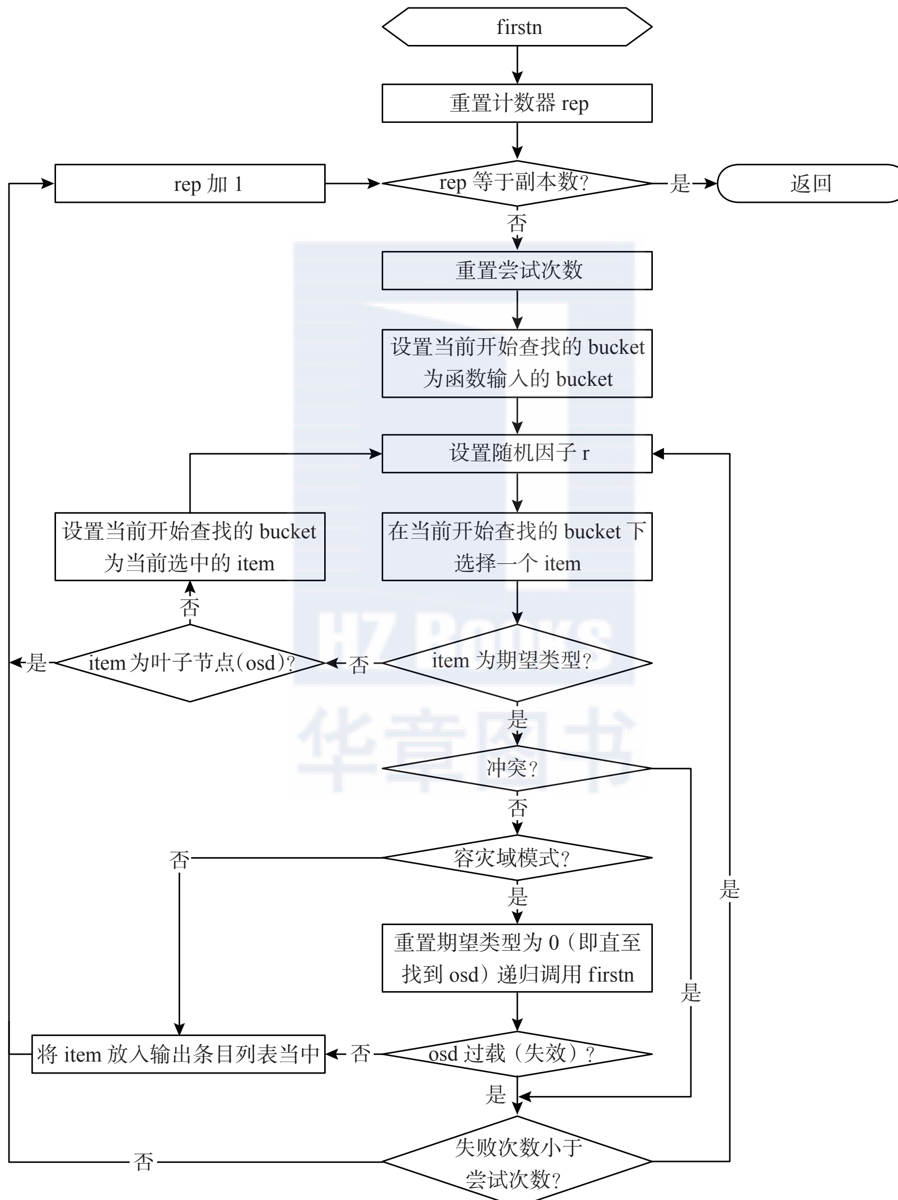


图 1-3　基于 firstn 的 select 执行过程

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

图 1-3 中几个关键处理步骤的补充说明如下：

（1）如何从 bucket 下选择一个条目（item）？

构建集群的 cluster map 时，通过分析每种类型的 bucket 特点可以为其指定一种合适的选择算法（例如 straw2），用于从对应的 bucket 中选择一个条目。因此从 bucket 选择条目的过程实际上就是执行设定的选择算法。这个选择算法的执行结果取决于两个因素：一是输入对象的特征标识符 x，二是随机因子 r（r 实际上是作为哈希函数的种子）。因为 x 固定不变，所以如果选择失败，那么在后续重试的过程中需要对 r 进行调整，以尽可能输出各种不同结果。目前 r 由待选择的副本编号和当前的尝试次数共同决定。

为了防止陷入死循环，需要对选择每个副本过程中的尝试次数进行限制，这个限制称为全局尝试次数（choose_total_tries）；同时因为在容灾域模式下会产生递归调用，所以还需要限制产生递归调用时作为下一级输入的全局尝试次数，因为这个限制会导致递归调用时的全局尝试次数成倍增长（按照递归的概念，多次递归后这个全局尝试次数应该成指数增长，但是实际上至多会递归调用一次，所以这里是将原始输入的全局尝试次数放大 N 倍后作为下一级输入的全局尝试次数），所以实现上采用一个布尔变量（chooseleaf_descend_once）进行控制，如果为真，则在产生递归调用时下一级被调用者至多重试一次；反之则下一级被调用者不进行重试，由调用者自身重试。为了降低冲突概率（如前，每次尽量使用不同的随机因子 r 可以降低冲突概率），也可以使用当前的重试次数（或者其 2^{N-1} 倍，这里的 N 由 chooseleaf_vary_r 参数决定）对递归调用时的随机因子 r 再次进行调整，这样产生递归调用时，其初始随机因子 r 将取决于待选择的副本编号和调用者传入的随机因子（称为 parent_r）。

值得一提的是，Jewel 版本之前，容灾域模式下作为递归调用时所使用的副本编号是固定的，例如调用者当前正在选择第 2 个副本，那么执行递归调用时的起始副本编号也将是 2。按照上面的分析，副本编号会作为输入参数之一对递归调用时的初始随机因子 r 产生影响，有用户反馈这在 OSD 失效时会触发不必要的数据（PG）迁移，因此在 Jewel 版本之后，容灾域模式下会对递归调用的起始副本独立编号（这个操作受 chooseleaf_stable 控制），以进一步降低两次调用之间的相干性。

在老的 CRUSH 实现中，因为选择的过程是执行深度优先遍历，所以如果对应集群的层次较多，并且在中间某个层次的 bucket 下因为冲突而选择条目失败，那么可以在当前的 bucket 下直接进行重试，而不用每次回归到初始输入的 bucket 之下重新开始重试，这样可以稍微提升算法的执行效率，此时同样需要对这个局部重试过程的次数进行

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

限制，称为局部重试次数（choose_local_retries）。此外，因为进入这种模式的直接原因是 bucket 自带选择算法冲突概率较高（即使用不同的 r 作为输入也反复选中同一个条目），所以针对这种模式还设计了一种备用的选择算法。这种后备选择算法的基本原理是将对应的 bucket 下的所有条目进行随机重排，只要输入 x 不变，那么随着 r 的变化，算法会不停记录前面已经被选择过的条目，并将其从本次候选条目中排除，从而能够有效降低冲突概率，保证最终能够成功选中一个不再冲突的条目。切换至后备选择算法需要冲突次数达到一定限制，这个限制主要由当前 bucket 的规模决定（原实现中要求冲突次数至少大于当前 bucket 下条目数的一半），当然切换至后备选择算法时，也可以再次限制启用后备选择算法进行重试的次数（choose_local_fallback_retries）。上述过程因为对整个 CRUSH 的执行过程进行了大量人工干预从而严重损伤了 CRUSH 的伪随机性（即公平性），所以会导致严重的数据均衡问题，因此在 Ceph 的第一个正式发行版 Argonaut 之后即被废弃，不再建议启用。

表 1-3 汇总了如上分析的、所有影响 CRUSH 执行的可调参数：

表 1-3 CRUSH 可调参数

表中的默认值和最优值针对 Jewel 版本而言

参数名称	默认值 / 最优值	说明
choose_local_tries	0/0	已被废弃，不建议进行调整
choose_local_fallback_tries	0/0	
choose_total_tries	50/50	如果集群比较大，层次比较多，或者每个主机下的磁盘数量比较少，可能会导致 CRUSH 无法选出足够的 OSD 完成所有副本映射，此时可以通过设置更大的 choose_total_tries 加以解决
chooseleaf_descend_once	1/1	控制容灾域模式下产生递归调用时的重试次数。 至多产生一次递归调用，递归时如果此标志位置位，至多重试一次。 不建议进行调整
chooseleaf_vary_r	1/1	不建议进行调整
chooseleaf_stable	1/1	不建议进行调整
straw_calc_version	1/1	用于对 straw 类型 bucket 下的条目权重计算过程进行校正。因为 straw 算法已经被废弃，所以本参数对于新建集群无影响

（2）冲突

冲突指选中的条目已经存在于输出条目列表之中。

（3）OSD 过载（或失效）

虽然哈希以及由哈希派生出来的 CRUSH 算法从理论上能够保证数据在所有磁盘之间均匀分布，但是实际上：

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- 集群规模较小，集群整体容量有限，导致集群 PG 总数有限，亦即 CRUSH 输入的样本容量不够。
- CRUSH 本身的缺陷——CRUSH 的基本选择算法中，以 straw2 为例，每次选择都是计算单个条目被选中的独立概率，但是 CRUSH 所要求的副本策略使得针对同一个输入、多个副本之间的选择变成了计算条件概率（我们需要保证副本位于不同容灾域中的 OSD 之上），所以从原理上 CRUSH 就无法处理好多副本模式下的副本均匀分布问题。

这些因素导致在真实的 Ceph 集群，特别是异构集群中，出现大量磁盘数据分布悬殊（这里指每个磁盘已用空间所占的百分比）的情况，因此需要对 CRUSH 计算结果进行人工调整。这个调整同样是基于权重进行的，即针对每个叶子设备（即磁盘，亦即 OSD），除了由其基于容量计算得来的真实权重（weight）之外，Ceph 还为其设置了一个额外的权重，称为 reweight。算法正常选中一个 OSD 后，最后还将基于此 reweight 对该 OSD 进行一次过载测试，如果测试失败，则仍将拒绝选择该条目，这个过程如图 1-4 所示。

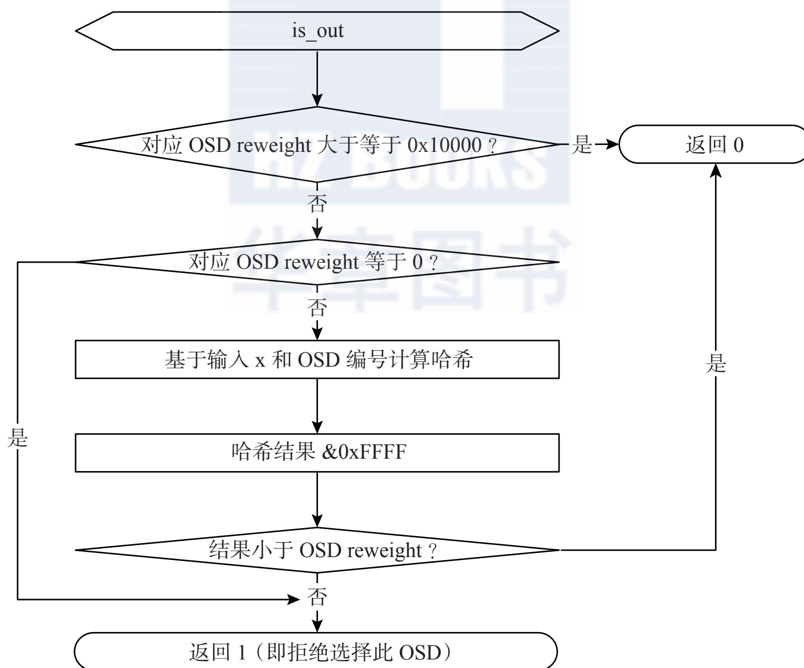


图 1-4　过载测试

由测试过程可见：对应 OSD 的 reweight 调整得越高，那么通过测试的概率越高（例

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

如手动设置某个 OSD 的 reweight 为 0x10000，那么通过测试的概率是 100%），反之则通过测试的概率越低。因此在实际应用中，通过降低过载 OSD 或者（和）增加空闲 OSD 的 reweight 都可以触发数据在 OSD 之间重新分布，从而使得数据分布更加合理。

引入过载测试的另一个好处在于可以对 OSD 暂时失效和 OSD 被永久删除的场景进行区分，区分这两者的意义在于：如果 OSD 暂时失效（例如对应的磁盘被拔出超过一定时间，Ceph 会将其设置为 out），可以通过将其 reweight 调整为 0 从而利用过载测试将其从候选条目中淘汰，进而将其中的数据迁移至其他 OSD，这样该 OSD 正常回归时，将其 reweight 重新调整为 0x10000 即可将原来归属于该 OSD 的数据再次迁回，而迁回过程中只需要同步该 OSD 不在线期间产生的新数据即可，即只需要进行增量同步；相反，如果是删除 OSD，此时会同步将其从对应的 bucket 条目中删除，这样即使该 OSD 后续被重新添加回集群，因为其在 cluster map 中的唯一编号可能已经发生了变化（参考前面的分析：条目编号会作为 straw2 算法输入参数之一，所以 OSD 编号改变会导致 CRUSH 选择结果产生变化），所以也可能承载与之前完全不同的数据。

初始时 Ceph 将每个 OSD 的 reweight 都设置为 0x10000，因此上述过载测试对 CRUSH 的最终选择结果不会产生任何影响。

HZ BOOKS 华章图书

1.3 调制 CRUSH

按照 Ceph 的设计，任何涉及客户端进行数据寻址的场景都需要基于 CRUSH 进行计算，所以提升 CRUSH 的计算效率具有重要意义。调制 CRUSH 即针对表 1-3 中的参数进行调整，从而使 CRUSH 正常工作的同时能够花费尽可能小的计算代价，以提升性能。需要注意的是，为了使调整后的参数正常生效，需要保证客户端和服务器对应的 Ceph 版本严格一致。

调制 CRUSH 最简单的办法是使用现成的、预先经过验证的模板（profile），命令如下（以下命令均以 Jewel 版本为准）：

```
ceph osd crush tunables {profile}
```

一些系统已经预先定义好的模板如表 1-4 所示。

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

表 1-4 系统自定义 CRUSH 可调参数模板（截至 Jewel 版本）

模板名称	说明
argonaut	最初的 CRUSH 版本，支持后备选择算法（通过设置 choose_local_tries 和 choose_local_fallback_tries 启用），该功能已被废弃。 此外这个模板中 choose_total_tries 值为 19，已经被证明在大部分用于生产环境的集群中都无法正常工作（无法选出足够的副本数）
bobtail	将 choose_total_tries 设置为经过大量生产环境验证、更合理的 50； 引入 chooseleaf_descend_once，对容灾域模式下的重试次数进行控制
firefly	引入 chooseleaf_vary_r，对容灾域模式下，产生递归调用时作为下一级输入的随机因子 r 进行调整，降低冲突（失败）概率
hammer	增加 straw2 算法支持
jewel	引入 chooseleaf_stable，减少不相关数据迁移
legacy	同 argonaut
optimal	同 jewel
default	同 firefly，这也是一个新集群创建时默认所启用的 CRUSH 可调参数

当然，在一些特定的场景（例如集群比较大同时主机中的磁盘数量比较少）可能使用上述模板仍然无法使得 CRUSH 正常工作，那么此时需要手动进行参数调整。

1.3.1 编辑 CRUSH Map

为了方便将 CRUSH 的计算过程作为一个相对独立的整体进行管理（因为内核客户端也需要用到），Ceph 将集群的 cluster map 和所有的 placement rule 合并成一张 CRUSH map，因此基于 CRUSH map 可以独立实施数据备份及分布策略。一般而言，通过 CLI 即可方便地在线修改 CRUSH 的各项配置，当然也可以通过直接编辑 CRUSH map 实现，步骤如下：

（1）获取 CRUSH map

大部分情况下创建集群成功后，对应的 CRUSH map 已经由系统自动生成，可以通过如下命令获取：

```
ceph osd getcrushmap -o {compiled-crushmap-filename}
```

上述命令将输出集群的 CRUSH map 至指定文件。当然出于测试或者其他目的，也可以手动创建 CRUSH map，命令如下：

```
crushtool -o {compiled-crushmap-filename} --build --num_osds Nlayer1 …
```

其中，`--num_osds Nlayer1` …将 N 个 OSD 从 0 开始编号，然后在指定的层级之间

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

平均分布，每个层级（layer）需要采用形如 <name, algorithm, size>（其中 size 指每种类型的 bucket 下包含条目的个数）的三元组进行描述，并按照从低（靠近叶子节点）到高（靠近根节点）的顺序进行排列，例如可以用如下命令生成图 1-2 所示集群的 CRUSH map（osd->host->rack->root）：

```
crushtool -o mycrushmap --build --num_osds 27 host straw2 3 rack straw2 3\
    root uniform 0
```

需要注意的是，上述两种方式输出的 CRUSH map 都是经过编译的，需要经过反编译之后才能被正常编辑。

（2）反编译 CRUSH map

执行命令：

```
crushtool -d {compiled-crushmap-filename} -o {decompiled-crushmap-filename}
```

即可将步骤（1）中输出的 CRUSH map 转化为可编辑版本，例如：

```
crushtool -d mycrushmap -o mycrushmap.txt
```

（3）编辑 CRUSH map

得到反编译后的 CRUSH map 之后，可以直接以文本形式打开和编辑，例如可以直接修改表 1-3 中的所有可调参数：

```
vi mycrushmap.txt

# 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 chooseleaf_vary_r 1
tunable straw_calc_version 1
```

也可以修改 placement rule：

```
vi mycrushmap.txt
# rules
rule replicated_ruleset {
    ruleset 0
    type replicated
    min_size 1
    max_size 10
    step take root
```

HZ BOOKS 华章图书

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    step chooseleaf firstn 0 type host
    step emit
}
```

上述 placement rule 中各个参数含义如表 1-5 所示。

表 1-5　CRUSH map 中 ruleset 相关选项及其具体含义

选项名称		含义
ruleset		对应规则（集）的唯一编号。 不同的 pool 可以使用不同的 ruleset
type		副本策略，包含如下选项： ● replicated ● erasure
min_size		用于对选择副本数的范围进行约束
max_size		
step	take	这三个操作具体含义参考 1.2.2 节，这里仅对“step chooseleaf firstn 0 type host”补充说明如下： ● chooseleaf，容灾域模式，可以替换为 choose，后者对应非容灾域模式。 ● firstn，两种选择算法之一，可以替换为 indep。 ● 0，表示由具体的调用者指定输出的副本数，例如不同的 pool 可以使用同一套 ruleset（拥有相同的备份策略），但是可以拥有不同的副本数。 ● type，对应 chooseleaf 操作，指示输出必须是分布在由本选项指定类型的、不同的 bucket 之下的叶子节点；对应 choose 操作，指示输出类型
	chooseleaf	
	emit	

因此上述规则指示“采用多副本数据备份策略，副本必须位于不同主机的磁盘之上”。

（4）编译 CRUSH map

以文本形式修改后的 CRUSH map，还需要经过编译，才能被 Ceph 识别，执行命令：

```
crushtool -c {decompiled-crush-map-filename} -o {compiled-crush-map-filename}
```

（5）模拟测试

在新的 CRUSH map 生效之前，可以先进行模拟测试，以验证对应的修改是否符合预期，例如可以使用如下命令打印输入范围为 [0,9]、副本数为 3、采用编号为 0 的 ruleset 的映射结果：

```
crushtool -i mycrushmap --test --min-x 0 --max-x 9 --num-rep 3 --ruleset 0 \
    --show_mappings
CRUSH rule 0 x 0 [19,11,3]
CRUSH rule 0 x 1 [15,7,21]
CRUSH rule 0 x 2 [26,5,14]
CRUSH rule 0 x 3 [8,25,13]
```

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
CRUSH rule 0 x 4 [5,13,21]
CRUSH rule 0 x 5 [7,25,16]
CRUSH rule 0 x 6 [17,25,8]
CRUSH rule 0 x 7 [13,4,25]
CRUSH rule 0 x 8 [18,5,15]
CRUSH rule 0 x 9 [26,3,16]
```

也可以仅统计结果分布概况（这里输入变为 [0,100000]）：

```
crushtool -i mycrushmap --test --min-x 0 --max-x 100000 --num-rep 3 \
    --ruleset 0 --show_utilization
rule 0 (replicated_ruleset), x = 0..100000, numrep = 3..3
rule 0 (replicated_ruleset) num_rep 3 result size == 3: 100001/100001
    device 0:          stored : 11243   expected : 11111.2
    device 1:          stored : 11064   expected : 11111.2
    device 2:          stored : 11270   expected : 11111.2
    device 3:          stored : 11154   expected : 11111.2
    device 4:          stored : 11050   expected : 11111.2
    device 5:          stored : 11211   expected : 11111.2
    device 6:          stored : 10848   expected : 11111.2
    device 7:          stored : 10958   expected : 11111.2
    device 8:          stored : 11203   expected : 11111.2
    device 9:          stored : 11031   expected : 11111.2
    device 10:         stored : 10997   expected : 11111.2
    device 11:         stored : 11165   expected : 11111.2
    device 12:         stored : 10993   expected : 11111.2
    device 13:         stored : 11188   expected : 11111.2
    device 14:         stored : 11150   expected : 11111.2
    device 15:         stored : 11222   expected : 11111.2
    device 16:         stored : 11152   expected : 11111.2
    device 17:         stored : 11103   expected : 11111.2
    device 18:         stored : 11044   expected : 11111.2
    device 19:         stored : 11056   expected : 11111.2
    device 20:         stored : 11023   expected : 11111.2
    device 21:         stored : 11514   expected : 11111.2
    device 22:         stored : 11026   expected : 11111.2
    device 23:         stored : 10888   expected : 11111.2
    device 24:         stored : 11025   expected : 11111.2
    device 25:         stored : 11069   expected : 11111.2
    device 26:         stored : 11356   expected : 11111.2
```

（6）注入集群

新的 CRUSH map 验证充分后，可以重新注入集群，使之生效，执行命令：

```
ceph osd setcrushmap -i  {compiled-crushmap-filename}
```

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

1.3.2 定制 CRUSH 规则

在上一节中，我们介绍了编辑 CRUSH map 的一般方法，本节我们介绍如何对 CRUSH 规则进行灵活定制，以满足特定需求，我们仍将以图 1-1 所示的集群为例进行说明。

最常见的场景是需要提升容灾域，例如，默认的容灾域一般为 host 级别，可以提升为 rack，修改对应的 ruleset（当然也可以新建一条 ruleset）如下：

```
vi mycrushmap.txt
# rules
rule replicated_ruleset {
    ruleset 0
    type replicated
    min_size 1
    max_size 10
    step take root
    step chooseleaf firstn 0 type rack
    step emit
}
```

测试结果如下：

```
crushtool -i mycrushmap --test --min-x 0 --max-x 9 --num-rep 3 \
    --ruleset 0 --show_mappings
CRUSH rule 0 x 0 [19,15,3]
CRUSH rule 0 x 1 [15,2,18]
CRUSH rule 0 x 2 [26,5,14]
CRUSH rule 0 x 3 [8,20,13]
CRUSH rule 0 x 4 [5,13,19]
CRUSH rule 0 x 5 [7,25,10]
CRUSH rule 0 x 6 [17,25,5]
CRUSH rule 0 x 7 [13,4,18]
CRUSH rule 0 x 8 [18,8,11]
CRUSH rule 0 x 9 [26,1,16]
```

可见此时所有副本都位于不同 rack 的 OSD 之上。

也可以限制只选择某个特定 rack（例如 rack2）下的 OSD，例如：

```
vi mycrushmap.txt
# rules
rule replicated_ruleset {
    ruleset 0
    type replicated
    min_size 1
```

HZ BOOKS 华章图书

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    max_size 10
    step take rack2
    step chooseleaf firstn 0 type host
    step emit
}
```

测试结果如下：

```
crushtool -i mycrushmap --test --min-x 0 --max-x 9 --num-rep 3 \
    --ruleset 0 --show_mappings
CRUSH rule 0 x 0 [19,21,26]
CRUSH rule 0 x 1 [20,23,26]
CRUSH rule 0 x 2 [26,20,22]
CRUSH rule 0 x 3 [22,25,18]
CRUSH rule 0 x 4 [21,26,18]
CRUSH rule 0 x 5 [21,25,19]
CRUSH rule 0 x 6 [19,25,23]
CRUSH rule 0 x 7 [21,18,25]
CRUSH rule 0 x 8 [18,24,21]
```

可见此时所有副本都被限制在 rack2（包含 OSD 编号范围 [18,26]）之下的 OSD 上。

另外需要注意的是，在 CRUSH map 中，除了 OSD（叶子节点）之外，其他层级关系都是虚拟的（不管其有无实际的物理实体对应），这为灵活定制 CRUSH 提供了更大的便利，例如在下面这个极端的例子中，我们通过新建一个虚拟的 host，以此为基础限制所有副本都必须分布在编号为 0、9、18 这三个特定的 OSD 上：

```
vi mycrushmap.txt
host virtualhost {
    id -14          # do not change unnecessarily
    # weight 3.000
    alg straw2
    hash 0  # rjenkins1
    item osd.0  weight 1.000
    item osd.9  weight 1.000
    item osd.18 weight 1.000
}
# rules
rule customized_ruleset {
    ruleset 1
    type replicated
    min_size 1
    max_size 10
    step take virtualhost
```

HZ BOOKS 华章图书

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    step chooseleaf firstn 0 type osd
    step emit
}
```

实际测试结果如下：

```
crushtool -i mycrushmap --test --min-x 0 --max-x 9 --num-rep 1 \
    --ruleset 1 --show_mappings
CRUSH rule 0 x 0 [0]
CRUSH rule 0 x 1 [9]
CRUSH rule 0 x 2 [9]
CRUSH rule 0 x 3 [0]
CRUSH rule 0 x 4 [18]
CRUSH rule 0 x 5 [18]
CRUSH rule 0 x 6 [18]
CRUSH rule 0 x 7 [18]
CRUSH rule 0 x 8 [18]
CRUSH rule 0 x 9 [9]
```

更复杂的例子可以通过将上面这些例子进行适当的组合得到，这里不再赘述。

1.3.3　数据重平衡

在 1.2 节，我们曾经提及如果集群数据分布不均衡，那么通过手动调整每个 OSD 的 reweight 可以触发 PG 在 OSD 之间进行迁移，以恢复数据平衡。上述数据重平衡操作可以逐个 OSD 或者批量进行。

首先查看整个集群的空间利用率统计：

```
ceph osd df tree
```

找到空间利用率较高的 OSD，然后逐个执行：

```
ceph osd reweight {osd_numeric_id} {reweight}
```

上述命令中各个参数含义如表 1-6 所示。

表 1-6　reweight 命令中的参数及其含义

参数	含义
osd_numeric_id	必选，整型。 OSD 对应的数字 ID
reweight	必选，浮点类型，[0,1]。 待设置的 OSD 的 reweight。reweight 取值越小，将使得更多的数据从对应的 OSD 迁出

HZ BOOKS 华章图书

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

也可以批量调整，目前有两种模式：一种是按照 OSD 当前空间利用率（reweight-by-utilization）；另一种是按照 PG 在 OSD 之间的分布（reweight-by-pg）。为了防止影响前端业务，可以先测试执行上述命令后，将会触发 PG 迁移数量的相关统计（以下都以 reweight-by-utilization 相关命令为例进行说明），以方便规划进行调整的时机：

```
ceph osd test-reweight-by-utilization {overload}{max_change}\
    {max_osds}{--no-increasing}
```

上述命令中各个参数含义如表 1-7 所示。

表 1-7 [test-]reweight-by-utilization 命令中的参数及其含义

参数	含义
overload	可选，整型，≥ 100；默认值为 120。 当且仅当某个 OSD 的空间利用率大于等于集群平均空间利用率的 overload/100 时，调整其 reweight
max_change	可选，浮点类型，[0,1]；默认值受 mon_reweight_max_change 控制，目前为 0.05。 每次调整 reweight 的最大幅度，即调整上限。实际每个 OSD 调整幅度取决于自身空间利用率与集群平均空间利用率的偏离程度——偏离越多，则调整幅度越大，反之则调整幅度越小
max_osds	可选，整型；默认值受 mon_reweight_max_osds 控制，目前为 4。 每次至多调整的 OSD 数目
--no-increasing	可选，字符类型。 如果携带，则从不将 reweight 进行上调（上调指将当前 underload 的 OSD 权重调大，让其分担更多的 PG）；如果不携带，至多将 OSD 的 reweight 调整至 1.0

例如：

```
ceph osd test-reweight-by-utilization 105 .2 4 --no-increasing
no change
moved 197 / 11016 (1.78831%)
avg 344.25
stddev 90.4592 -> 92.0923 (expected baseline 18.2618)
min osd.11 with 61 -> 60 pgs (0.177197 -> 0.174292 * mean)
max osd.31 with 424 -> 379 pgs (1.23166 -> 1.10094 * mean)

oload 105
max_change 0.2
max_change_osds 4
average 0.156612
overload 0.164443
osd.20 weight 1.000000 -> 0.844574
osd.27 weight 1.000000 -> 0.869125
osd.31 weight 1.000000 -> 0.890121
osd.13 weight 1.000000 -> 0.895248
```

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

由输出可见，本次如果真正执行 rewegith-by-utilization 命令将导致：

- 197 个 PG 发生迁移。
- 每个 OSD 承载的平均 PG 数目为 344.25，执行本次调整后，标准方差将由 90.4592 变为 92.0923。
- 当前负载最轻的 OSD 为 osd.11，只承载了 61 个 PG，执行本次调整后，将承载 60 个 PG。
- 当前负载最重的 OSD 为 osd.31，承载了 424 个 PG，执行本次调整后，将减少到 379 个 PG。
- 执行本次调整后（命令可以重复执行），共计对 osd.20、osd.27、osd.31、osd.13 在内的 4 个 OSD 的 reweight 进行了调整。

输入以下命令将确认执行调整：

```
ceph osd reweight-by-utilization 105 .2 4 --no-increasing
```

1.4 总结与展望

作为 Ceph 设计基础之一的 CRUSH 算法已经走过了 10 年的时间。CRUSH 良好的设计理念使其具有计算寻址、高并发和动态数据均衡、可定制的副本策略等基本特性，进而能够非常方便地实现诸如去中心化、有效抵御物理结构变化并保证性能随集群规模呈线性扩展、高可靠等高级特性，因而非常适合类似于 Ceph 这类对可扩展性、性能和可靠性都具有严苛要求的大型分布式存储系统。

然而回到具体实现上，无论是 list 和 tree 算法被先后废弃，还是原创的 straw 算法被完全重写，无不证明 CRUSH 从来就不是完美的。兴许 CRUSH 最为人诟病之处在其引入扩展的副本策略支持之后所导致的数据不均衡问题[⊖]，虽然从设计者的角度无比希望将这个问题交由 CRUSH 自身加以圆满解决，然而实现上由于 Ceph 所支持的集群可能具有复杂的层级拓扑结构使得解决这个问题困难重重而变得遥遥无期，在现阶段以及可预见的将来依然只能依托于用户进行人工干预。此外，在生产环境中，用户向社区抱怨在一些异构集群中 CRUSH 选不出足够副本数的声音从未停止，虽然可以通过针对 CRUSH 的一些参数进行调整加以解决（然而这从来就不是一件轻松的事情），但是相应的会以牺

⊖ http://tracker.ceph.com/issues/15653

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

牲 CRUSH 的计算性能作为代价，因此也远远无法作为一个商业级成熟软件的解决方案。最后，基于计算寻址的设计使得无论何时针对 CRUSH 升级都要求客户端和服务端同时进行，这通常会为内核客户端（krbd、kcephfs）类型的用户带来极大的困扰，同时因为 Ceph 流控机制相对薄弱，升级过程中潜在的数据迁移问题有极大可能会影响到正常业务，进而成为平滑在线升级方案中的隐患。

当然，伴随着 Ceph 社区的不断发展壮大，相信上述这一切最终都能够得到妥善解决。