**BlueStore源码分析-分配器**

**磁盘空间管理**

一般磁盘空间管理可以通过位图或者基于段的管理方式来记录一个范围的磁盘的存储使用状态.既然状态只有使用和未使用两种情况,那理论上就可以做到用一个bit标识一个块是否被使用,在bluestore中使用的就是该方法,而段的管理方式主要可以解决在大容量的存储空间时候位图占用内存过大的问题,具体做法是用一个(offset, length)来代表一个存储区域的使用情况,一共就两个字段,每个都是64位的,这种的可以用在大容量存储.而bluestore单机存储的容量不会特别大,例如单机1TB的容量,块大小按照4KB计算,差不多是256MB,在现在的服务器上使用这个大小的内存是在很多存储场景下是能接受的.当然也可以使用段.

bluestore中,FreeListManager负责管理空闲空间列表,Allocator组件负责管理已分配空间列表,两种组件又各有段和位图两种实现方式.因为bluestore默认使用位图形式,这里我们先介绍两种组件的位图实现,分别对应数据结构BitmapFreelistManager和BitmapAllocator。

**BitmapFreelistManager**

1. class BitmapFreelistManager : public FreelistManager {
2. std::string meta\_prefix, bitmap\_prefix;
3. KeyValueDB \*kvdb;
4. ceph::shared\_ptr<KeyValueDB::MergeOperator> merge\_op;
5. std::mutex lock;
7. uint64\_t size; ///< size of device (bytes)
8. uint64\_t bytes\_per\_block; ///< bytes per block (bdev\_block\_size)
9. uint64\_t blocks\_per\_key; ///< blocks (bits) per key/value pair
10. uint64\_t bytes\_per\_key; ///< bytes per key/value pair
11. uint64\_t blocks; ///< size of device (blocks, size rounded up)
13. uint64\_t block\_mask; ///< mask to convert byte offset to block offset
14. uint64\_t key\_mask; ///< mask to convert offset to key offset
16. bufferlist all\_set\_bl;
18. KeyValueDB::Iterator enumerate\_p;
19. uint64\_t enumerate\_offset; ///< logical offset; position
20. bufferlist enumerate\_bl; ///< current key at enumerate\_offset
21. int enumerate\_bl\_pos; ///< bit position in enumerate\_bl
22. ...
23. };

以块为粒度，将连续，数量固定的多个块进一步组成一个段，从而将整个磁盘空间划分为若干连续的段进行管理，每个段以其在磁盘中的对应起始地址进行编号，可以得到一个Bluestore实例内唯一的索引，从而可以使用kvDB固化BitmapFreelistManager中的所有段信息。如前所述，单个块的状态可以使用一个比特进行标记，因此每个段的值部分是一个长度固定的比特流，比特流中的某个比特置位，表明对应的块已经被分配，反之则表明对应的块空闲。另外，因为使用kvDB存储段信息，所以需要合理调整BitmapFreelistManager中的段大小设置，过大，value过大，过小，kv数量过多。

系统运行过程中，BitmapFreelistManager中的块需要频繁的在空闲和占用两种状态之间切换，固定使用"1"作为掩码，基于异或运算可以实现每个块在"空闲"在和"占用"两种状态之间快快速切换，主要有六种核心接口。

create()

通过bluestore的mkfs()接口创建一个BitmapFreelistManager.因为BitmapFreelistManager中的一些关键参数例如块大小，每个段包含的块数目等等是可以配置的，随意需要通过create()固化到kvDB.后续上电这些参数会直接从kvDB中读取，防止因为配置变化而导致BitmapFreelistManager无法工作。

init()

初始化BitmapFreelistManager,上电时调用，用于从kvDB中加载块大小，每个段包含的块数目等可配置参数。

allocate()

从BitmapFreelistManager分配指定范围[offset, offset+length]空间。

release()

从BitmapFreelistManager中释放指定范围[offset, offset+length]空间。

enumerate\_reset()，enumerate\_next()

通过这两个接口在Bluestore上电时遍历BitmapFreelistManager中所有空闲段,并将其从Allocator中同步移除，从而还原上一次下电时Allocator。

**BitmapAllocator**

这个结构是不需要持久化的，是一个内存版本的空间分配器。组织结构是树状形态。

BitmapAllocator{BitmapAreaInternal{BitmapAreaLeaf{BitmapAreaZone{BitmapEntry.....}....}....}.....}

BitmapEntry是一个64位无符号的整型，代表64个连续的块。

BitmapAreaZone是单次最大可分配单位，大小可调。它拥有独立的锁逻辑，所有API都是原子的。所以不同的BitmapAreaZone可并发执行。

中间的BitmapAreaInternal是为了在数据量比较大的时候提高索引效率的，它就是一个多叉树，叶子节点是BitmapAreaLeaf,非叶子节点是BitmapAreaInternal,根据BitmapAreaInternal中单个中间或者叶子节点的跨度决定BitmapArea的高度。最终BitmapAreaInternal会到达根结点BitmapAllocator。

因为Bluestore一般把不同类型的数据放在不同的设备上，而且这些设备可能还是不同的存储介质，所以，一般一个Bluestore实例上会有多个bitmapAllocator实例同时运行。