1. Blobstore
   1. blobstore介绍

Blobstore被定义为一种块分配的存储池，从层次上讲，blobstore是构建在底层块设备上的一种逻辑块分配的存储系统，类似于传统意义上的文件系统，但并非文件系统。Blobstore也不支持posix语义，同时为了和传统posix语义上的文件系统区分开，blobstore使用了blob对象来表示传统文件系统中的files和objects对象。

从应用上，blobstore为上层的应用提供了基础的块存储切片服务，从而可以基于blobstore构建很多现有的上层存储服务。例如：逻辑卷、数据库、文件系统（BlobFS和RocksDB等）、key/value存储等等，甚至包括SAN、NAS和各类分布式存储系统。

* 1. blobstore概念

Blobstore建立了抽象的层次式存储结构，其中包含的对象有：blobstore、blob、cluster、page以及logical block（逻辑块）。

逻辑块：底层块设备的最小访问单元，一般512B或4KB

Page页：包含了固定的逻辑块数目，page中的逻辑块一定是物理连续的。Page大小一般4KB，SSD设备必须能够原子的读写一个Page。

Cluster簇：包含了固定数目的Page，这个数目是创建blobstore时，指定的。一般一个cluster大小1MB或者4MB。Cluster内的Page也是物理上里西安许的。

Blob：blob是一个有序簇组成的簇链表，应用程序会直接操作blob。Blob是物化到块设备的，因此断电和重启数据不丢失。应用程序通过blobstore中保存的blob标识符来访问特定的blob，blob的读写访问以page为单位，通过指定page在blob中的起始偏移即可。应用程序可以往blob中存储key/value键值对，这些键值对实际形成了blob的属性（xattrs）。

Blobstore: 一个blobstore管理了整个底层块设备，基于blobstore的应用程序，被称作一个blobstore。Blobstore由元数据区域和多个blobs集合组成。

下图描述了blobstore中的逻辑层次结构：



1. blobstore的关键数据结构

从物理布局上，blobstore中有两类元数据，一类是blobstore的元数据，主要包括占用一个page大小的super\_blob和3张位图数组。3张位图数组用于管理blobstore资源的分配，主要包括blob的元数据页的分配、簇的分配、blobid的分配。另一类是blob的元数据，blob的元数据记录在blob元数据页上，主要包括blob的flag标志位、blob的属性（xattrs）、blob的extents（一个extent由一段物理上连续的多个簇组成）。

* 1. spdk\_bs\_super\_block

该结构存储了blobstore的管理数据，其中使用了3张位图数组管理了整个blobstore的资源分配，主要包括簇分配位数组、blob元数据页（元数据Page）分配位数组和blobid分配位数组。每一个blobid的分配都对应一个blob的生成，blobstore的所有blob中，存在一个特殊的blob，即super\_blob。在创建blobstore时，会同时创建super\_blob，这个blob的主要作用是记录了用户程序对该blobstore的一些定制内容，例如逻辑卷应用中，会使用super\_blob来记录逻辑卷组的名称以及生成的uuid，这些信息以key/value对形式记录到blob的属性中（xattrs）。另外，super\_blob区别于普通blob，它不分配数据簇。super\_blob主要字段解释如下：

struct spdk\_bs\_super\_block {

         uint8\_t              signature[8];

         uint32\_t        version;

         uint32\_t        length;

         uint32\_t   clean;

         spdk\_blob\_id   super\_blob;

         uint32\_t   cluster\_size;

         uint32\_t   used\_page\_mask\_start;

         uint32\_t   used\_page\_mask\_len;

         uint32\_t   used\_cluster\_mask\_start;

         uint32\_t   used\_cluster\_mask\_len;

         uint32\_t   md\_start;

         uint32\_t   md\_len;

         struct spdk\_bs\_type        bstype;

         uint32\_t   used\_blobid\_mask\_start;

         uint32\_t   used\_blobid\_mask\_len;

         uint8\_t              reserved[4012];

         uint32\_t   crc;

};

* 1. spdk\_blob\_store

该结构缓存了blobstore运行时关键的数据，其中一部分是从super\_block中拷贝过来，一部分是通过计算得到的，另一部分是运行时可变的，例如三张位图数组。可以知道，该结构中的数据需要及时刷下去，否则会造成断电丢失。主要字段解释如下：

struct spdk\_blob\_store {

         uint64\_t                     md\_start;

         uint32\_t                     md\_len;

         struct spdk\_io\_channel            \*md\_channel;

         uint32\_t                     max\_channel\_ops;

         struct spdk\_thread           \*md\_thread;

         struct spdk\_bs\_dev          \*dev;

         struct spdk\_bit\_array               \*used\_md\_pages;

         struct spdk\_bit\_array               \*used\_clusters;

         struct spdk\_bit\_array               \*used\_blobids;

         pthread\_mutex\_t                       used\_clusters\_mutex;

         uint32\_t                     cluster\_sz;

         uint64\_t                     total\_clusters;

         uint64\_t                     total\_data\_clusters;

         uint64\_t                     num\_free\_clusters;

         uint32\_t                     pages\_per\_cluster;

         spdk\_blob\_id                      super\_blob;

         struct spdk\_bs\_type                  bstype;

         struct spdk\_bs\_cpl           unload\_cpl;

         int                                unload\_err;

         TAILQ\_HEAD(,spdk\_blob)                 blobs;

         TAILQ\_HEAD(,spdk\_blob\_list)         snapshots;

};

* 1. blob的元数据结构体

（1）blob的flags标志位

struct spdk\_blob\_md\_descriptor\_flags {

         uint8\_t              type;

         uint32\_t   length;

         uint64\_t   invalid\_flags;

         uint64\_t   data\_ro\_flags;

         uint64\_t   md\_ro\_flags;

};

（2）blob的属性xattrs

struct spdk\_blob\_md\_descriptor\_xattr {

         uint8\_t              type;

         uint32\_t   length;

         uint16\_t   name\_length;

         uint16\_t   value\_length;

         char          name[0];

};

（3）blob的簇extents

struct spdk\_blob\_md\_descriptor\_extent {

         uint8\_t              type;

         uint32\_t   length;

         struct {

                   uint32\_t        cluster\_idx;

                   uint32\_t        length;

         } extents[0];

};

1. Blobstore的物理存储

第2章节描述了blobstore和blob相关的数据结构，这些数据结构中有三类，会分别存储到底层物理块设备上。第一，super\_block；第二，资源分配数组；第三，blob的元数据页。具体物理存储结构如下图：



3.1. 元数据页的分配计算

（1）super\_block

super\_block固定占据块设备的第0个page（4KB）。

（2）位图数组

Blobstore默认为每一个簇分配一个Page的元数据页，这是为了支持最小的blob（仅一个簇）时，blob至少需要一个Page来记录blob的元数据。

假设blobstore的总的簇数量为N，那么md\_len为N。

由于第0个page是super\_block，那么used\_page\_mask\_start恒为1，used\_page\_mask\_len的计算如下：

divide\_round\_up(sizeof(struct spdk\_bs\_md\_mask)+

                                                divide\_round\_up(bs->md\_len, 8),SPDK\_BS\_PAGE\_SIZE)

后面的used\_cluster和used\_blobid位图数组依次往后排，并按照上述方式计算。

如下的一个super\_block，可以看到used\_page\_mask\_start为1，used\_page\_mask\_len为12；used\_cluster\_mask\_start为13，used\_cluster\_mask\_len为12；used\_blobid\_mask\_start为25，used\_blobid\_mask\_len为12。可以知道，默认情况下三张位图的大小是一样的，这里都占据12个page页。

（3）Blob的元数据页

位图数组一旦分配完，就得到了blob元数据页的起始page，如上md\_start为37，是由1 + 12 + 12 + 12 得到的。而md\_len固定由底层块设备的簇数量决定，这里为381546个Page。由此可知元数据共占据381546 + 37 = 381583个Page。

（4）数据簇

将上述元数据按照占用的page数量进行簇整数边界对齐后，去掉这些簇，剩余的簇就是数据簇，数据簇由blobstore在创建blob时分配给blob（瘦分配在写入时才进行分配）。分配数据簇时要修改位图数组，因此位图数组需要及时刷到物理块设备上。

如上，元数据使用的簇数量divide\_round\_up(num\_md\_pages, bs->pages\_per\_cluster)

= divide\_round\_up(381583, 4MB / 4KB)

=373。

因此数据簇数量381546 – 373 = 381173。

至此也可以计算blobstore的空间利用率：100% \* 381173 / 381546.0 = 99.90223%，元数据管理浪费的空间不到千分之一。