Ceph BlueStore 和双写问题

Original Accela Zhao Accela推箱子 2018-01-28

论开源分布式存储，Ceph大名鼎鼎。用同一个存储池融合提供块存储、对象存储、集群文件系统。在国内有近年使用量迅速攀升，Ceph Day峰会也搬到北京来开了。

大型公司内部研发云虚拟化平台，常使用开源方案Openstack或者Kubernetes，配套的为虚机或容器提供块存储的开源方案，几乎为Ceph莫属。对象存储几年发展迅速，图像、视频、网站资源等皆可适用，有初创公司基于Ceph搭建存储服务方案。企业存储方面，国外有Redhat收购了Inktank，后者由Ceph初创作者Sage Weil创建；国内有XSky星辰天合，聚集了大量从早期就开始专注Ceph的专家。（P.S. 关于国内谁在大规模使用Ceph，上Ceph Day看Slides可以知道。）

可以将Ceph理解为分布式管理层，加上每个存储节点（OSD）的存储后端。社区成熟的存储后端使用FileStore，用户数据被映射成对象，以文件的形式存储在文件系统上。文件系统可以是EXT4、BtrFS、XFS等。最近两年，因为FileStore的种种问题，由Sage Wei推动，Ceph社区合力推出了新的存储后端，BlueStore。

BlueStore有独特的架构，解决了Ceph社区一直烦恼的FileStore的日志双写问题，测试性能比FileStore提高了一倍。这让人非常想深入剖析BlueStore。另一方面，公有云内部开发的存储系统也如同Ceph，历久年月不断翻新；像Ceph社区这样，能够提出全新架构，把性能提升一倍，是非常值得借鉴的。

关于Ceph BlueStore的资料

CDM（Ceph Developer Monthly）是Ceph开发者之间的分享会议，技术细节原汁原味。下面的视频非常全面地覆盖了Ceph BlueStore的动机、设计、工作流程、未来发展等等。Slides链接如下，但许多内容只在视频中（虽然视频2016但Slides是2017，但内容大体一致）。本文使用了其中的不少插图。（P.S. 经过两年的开发，如今BlueStore成果显著；相比应用开发，也可看出底层存储开发周期之慢…）

[2016-JUN-21 -- Ceph Tech Talks: Bluestore – YouTube](https://www.youtube.com/watch?v=kuacS4jw5pM)

[BlueStore, A New Storage Backend for Ceph, One Year In](https://www.slideshare.net/sageweil1/bluestore-a-new-storage-backend-for-ceph-one-year-in)

System Notes博客发布了多篇非常深入的Ceph BlueStore解析，甚至也是国内最早的。本文也直接使用了其中的插图。下面的链接是其中一篇

[System Notes: ceph 存储引擎 bluestore 解析](http://www.sysnote.org/2016/08/19/ceph-bluestore/)

笔记社区WuXiangWei的文章有多篇非常深入的Ceph BlueStore剖析。例如BlueFS设计、对象到磁盘的映射等。本文也使用其中的插图。下面的链接是其中的一篇

[WuXiangWei：Ceph BlueFS分析](http://www.bijishequ.com/detail/271710)

关于Ceph BlueStore以及其它几种存储后端的写行为和写放大的深入解析，有一篇论文。论文中有对各种后端的写路径和相关特性的详细描述。

[Understanding Write Behaviors of Storage Backends in CephObject Store](http://storageconference.us/2017/Papers/CephObjectStore.pdf)

关于Ceph的写路径，最全面的资料在它的开发文档里。其中列举了BlueStore应对不同类型写入所采取的策略，结合前述Slides看更加容清楚。

[BlueStore Internals](https://github.com/ceph/ceph/blob/master/doc/dev/bluestore.rst)

为什么需要BlueStore

在上文的CDM的BlueStore介绍中详细解释，归结起来，主要有这些方面：

首先，Ceph原本的FileStore需要兼容Linux下的各种文件系统，如EXT4、BtrFS、XFS。理论上每种文件系统都实现了POSIX协议，但事实上，每个文件系统都有一点“不那么标准”的

地方。Ceph的实现非常注重可靠性，因而需要为每种文件系统引入不同的Walkaround或者Hack；例如Rename不幂等性，等等。这些工作为Ceph的不断开发带来了很大负担。

其次，FileStore构建与Linux文件系统之上。POSIX提供了非常强大的功能，但大部分并不是Ceph真正需要的；这些功能成了性能的累赘。另一方面，文件系统的某些功能实现对Ceph并不友好，例如对目录遍历顺序的要求，等等。

另一方面，是Ceph日志的双写问题。为了保证覆写中途断电能够恢复，以及为了实现单OSD内的事物支持，在FileStore的写路径中，Ceph首先把数据和元数据修改写入日志，日志完后后，再把数据写入实际落盘位置。这种日志方法（WAL）是数据库和文件系统标准的保证ACID的方法，但用在Ceph这里，带来了问题：

1）数据被写了两遍，即日志双写问题，这意味着Ceph牺牲了一半的磁盘吞吐量。

2）Journaling of Journal问题，这个在上述Write Behaviors论文中有讲。Ceph的FileStore做了一遍日志，而Linux文件系统自身也有日志机制，实际上日志被多做了一遍。

3）对于新型的LSM-Tree类存储，如RocksDB、LevelDB，由于数据本身就按照日志形式组织，实际上没有再另加一个单独的WAL的必要。

4）更好地发挥SSD/NVM存储介质的性能。与磁盘不同，基于Flash的存储有更高的并行能力，需要加以利用。CPU处理速度逐渐更不上存储，因而需要更好地利用多核并行。存储中大量使用的队列等，容易引发并发竞争耗时，也需要优化。另一方面，RocksDB对SSD等有良好支持，它为BlueStore所采用。

另外，社区曾经为了FileStore的问题，提出用LevelDB作存储后端；对象存储转换为KeyValue存储，而不是转换问文件。后来，LevelDB存储没有被推广开，主流还是使用FileStore。但KeyValue的思路被沿用下来，BlueStore就是使用RocksDB来存储元数据的。

展望未来，ScanDisk开源的ZetaScale存储能够更加出色地发挥SSD/NVM/PersistentMemory的性能。它有智能内存缓存、最大化并发和减小响应时间、支持原子操作/快照/事务，等等特色。Ceph可能将它作为新的存储后端，或者替换掉

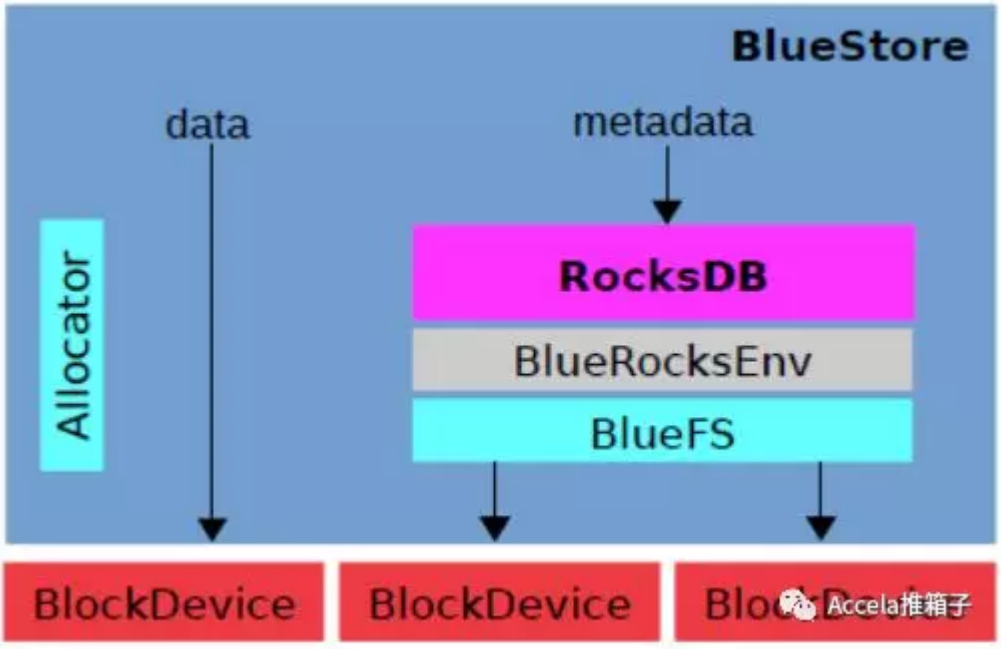
BlueStore中的RocksDB；当然也可以等RocksDB发展得更好。（P.S.其实大半年前已经开始做了。）

BlueStore的架构

BlueStore的出发点其实应验了这样的哲学，存储的最常用写路径应该尽量地短、尽量地简单，这样才能有最好的性能，尽管另外的异常处理路径可能是非常复杂的。BlueStore的设计有如下特色

1）Ceph并不需要POSIX文件系统。抛弃它，实现一个尽量简单的文件系统，专门给RocksDB使用。这个文件系统叫作BlueFS。

2）元数据存储在RocksDB中，用KeyValue的方式正合适。而数据不需要文件系统，直接存储在裸块设备上即可。我们在块设备上需要的，其实是一个空间分配器（Allocator）。



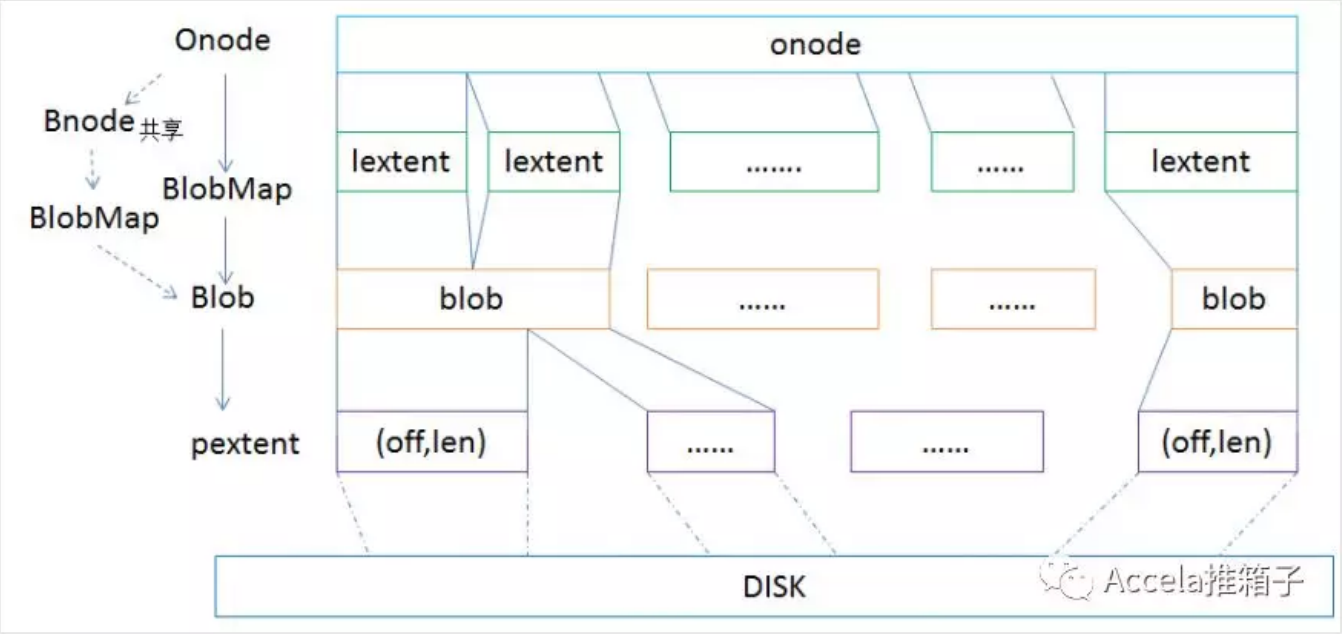
查看BlueStore的代码，相比FileStore小了很多。Allocator是可插拔更换策略的，大约3000行代码。BlueFS本就是极简的文件系统，约3000行代码。事务的实现借用RocksDB对事务的支持，简化很多。而且没有自己实现日志的需要了，剩下了FileStore中的Journal一块。

还有一点，如上图所示，BlueStore中不同组件可以使用不同的块设备。例如给RocksDB的WAL文件配备NVRAM，给SST文件配备SSD，给数据文件配备磁盘；方案是灵活的。

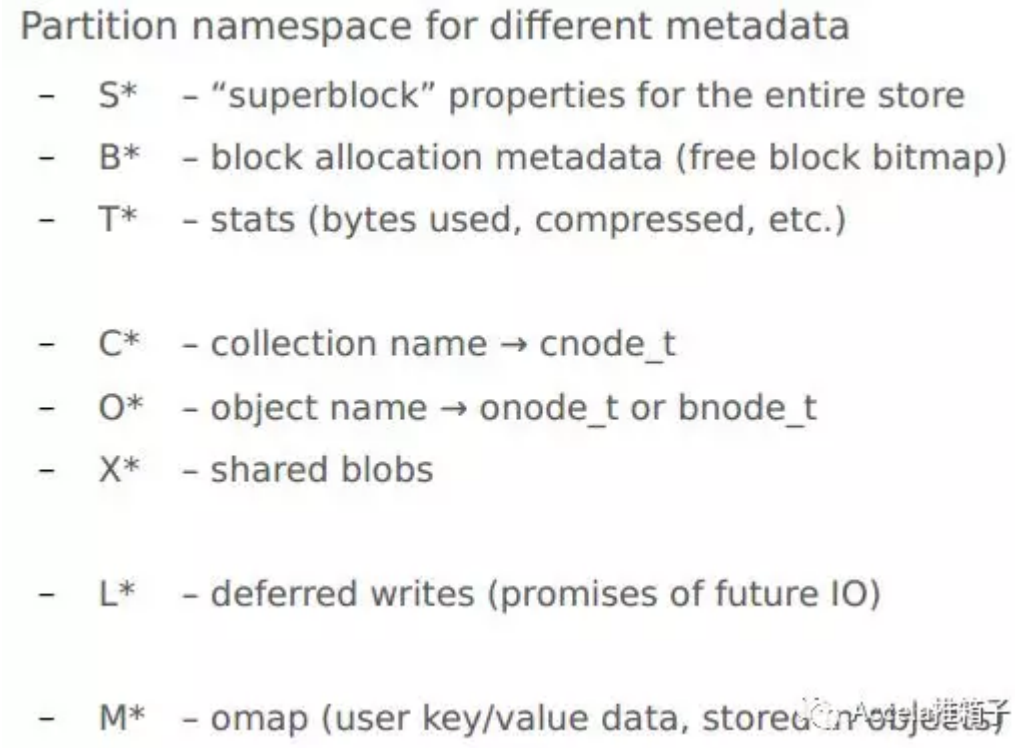
BlueStore的元数据管理

在涉及写路径之前，先看看Ceph BlueStore如何管理元数据。首先的问题是，对象如何映射成磁盘数据结构（Ceph的底层是对象存储，向上封装出块存储、文件系统）？

Onode代表对象，名字大概是从Linux VFS的Inode沿袭过来的。Onode常驻内存，在RocksDB中以KeyValue形式持久化；关于内存Cache的结构，在CDM的Slides中有讲。Onode包含多个lextent，即逻辑extent。Blob通过映射pextent、即物理extent，映射到磁盘上的物理区域。Blob通常包括来自同一个对象的多段数据，但是也可能被其它对象引用。Bnode是对象快照后，被用于多个对象共享数据的。



上面仅是关于对象映射的。更进一步，RocksDB中存储有许多类型的元数据，包括块分配、对象集合、快照、延迟写（Deferred Writes）、对象属性（Omap，即一个对象上可以附加一些KeyValue对作为属性，例如给图片加上地点、日期等），等等。在CDM的Slides中有详述。



BlueStore的写路径

写路径包含了对事务的处理，也回答了BlueStore如何解决日志双写问题。

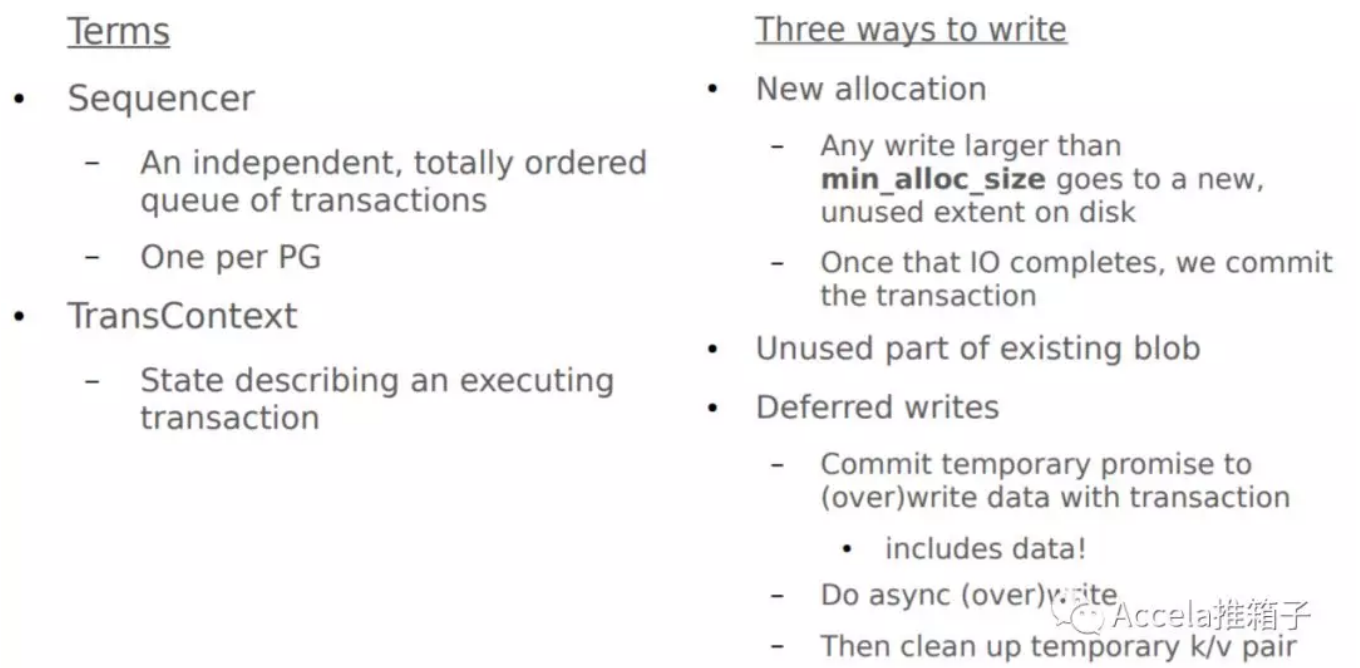
首先，Ceph的事务只工作于单个OSD内，能够保证多个对象操作被ACID地执行，主要是用于实现自身的高级功能。每个PG（Placement Group，类似Dynamo的vnode，将hash映射到同一个组内的对象组到一起）内有一个OpSequencer，通过它保证PG内的操作按序执行。事务需要处理的写分三种：

1）写到新分配的区域。考虑ACID，因为此写不覆盖已有数据，即使中途断电，因为RocksDB中的元数据没有更新，不用担心ACID语义被破坏。后文可见RocksDB的元数据更新是在数据写之后做的。因而，日志是不需要的。在数据写完之后，元数据更新写入RocksDB；RocksDB本身支持事务，元数据更新作为RocksDB的事务提交即可。

2）写到Blob中的新位置。同理，日志是不需要的。

3）Deferred Writes（延迟写），只用于覆写（Overwrite）情况。从上面也可以看到，只有覆写需要考虑日志问题。如果新写比块大小（min_alloc_size）更小，那么会将其数据与元数据合并写入到RocksDB中，之后异步地把数据搬到实际落盘位置；这就是日志

了。如果新写比块大小更大，那么分割它，整块的部分写入新分配块中，即按（1）处理，；不足的部分按（3）中上种情况处理。



上述基本概述了BlueStore的写处理。可以看到其是如何解决FileStore的日志双写问题的。首先，没有Linux文件系统了，也就没有了多余的Journaling of Journal问题。然后，大部分写是写到新位置的，而不是覆写，因此不需要对它们使用日志；写仍然发生了两次，第一次是数据落盘，然后是RocksDB事务提交，但不再需要在日志中包含数据了。最后，小的覆写合并到日志中提交，一次写完即可返回用户，之后异步地把数据搬到实际位置（小数据合并到日志是个常用技巧）；大的覆写被分割，整块部分用Append-only方式处理，也绕开了日志的需要。至此，成为一个自然而正常的处理方式。（P.S.总之，个人感觉日志双写不是一个该存在的问题，不知为何成了一个问题，好在今天终于不是问题了。）

更深入地，Ceph的开发文档中列出了所有的写策略处理方式。可以看到Inline Compression也是BlueStore的功能点之一；其中也有对Partial-write问题的处理。

Small write strategies

- *U*: Uncompressed write of a complete, new blob.
  - write to new blob
  - kv commit
- *P*: Uncompressed partial write to unused region of an existing blob.
  - write to unused chunk(s) of existing blob
  - kv commit
- *W*: WAL overwrite: commit intent to overwrite, then overwrite async. Must be chunk_size = MAX(block_size, csum_block_size) aligned.
  - kv commit
  - wal overwrite (chunk-aligned) of existing blob
- *N*: Uncompressed partial write to a new blob. Initially sparsely utilized. Future writes will either be *P* or *W*.
  - write into a new (sparse) blob
  - kv commit
- *R+W*: Read partial chunk, then to WAL overwrite.
  - read (out to chunk boundaries)
  - kv commit
  - wal overwrite (chunk-aligned) of existing blob
- *C*: Compress data, write to new blob.
  - compress and write to new blob
  - kv commit

Possible future modes

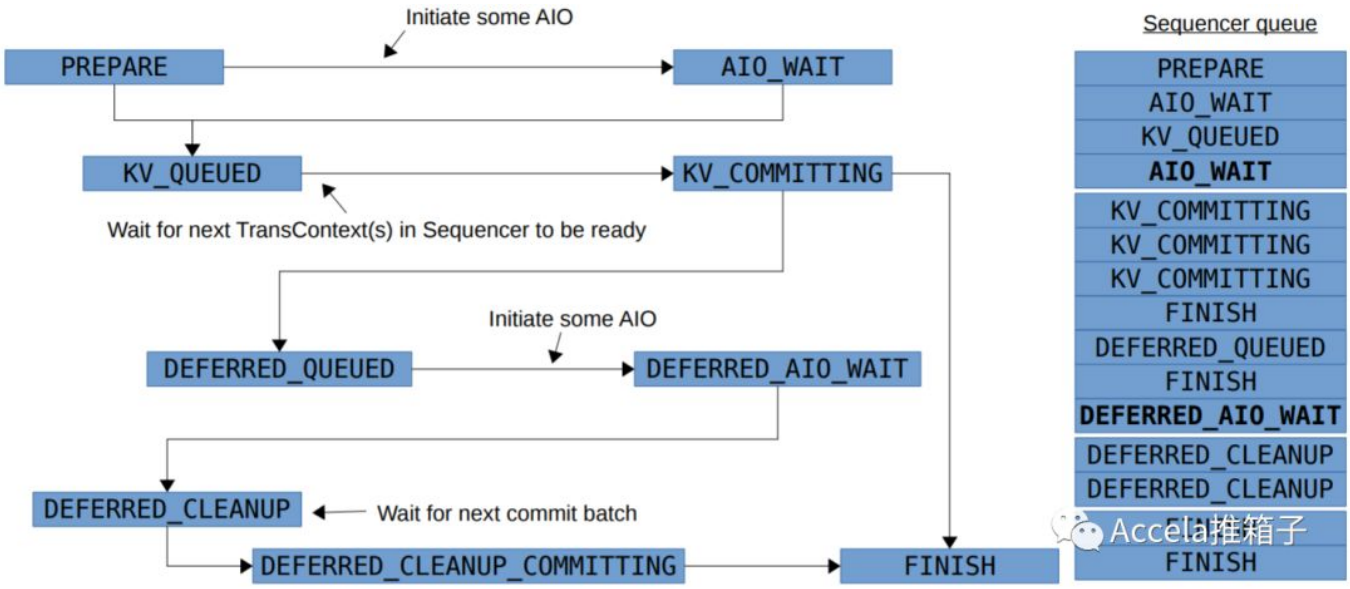
- *F*: Fragment lextent space by writing small piece of data into a piecemeal blob (that collects random, noncontiguous bits of data we need to write).
  - write to a piecemeal blob (min_alloc_size or larger, but we use just one block of it)
  - kv commit
- *X*: WAL read/modify/write on a single block (like legacy bluestore). No checksum.
  - kv commit
  - wal read/modify/write

Mapping

This very roughly maps the type of write onto what we do when we encounter a given blob. In practice it's a bit more complicated since there might be several blobs to consider (e.g., we might be able to *W* into one or *P* into another), but it should communicate a rough idea of strategy.

	raw	raw (cached)	csum (4 KB)	csum (16 KB)	comp (128 KB)
128+ KB (over)write	U	U	U	U	C
64 KB (over)write	U	U	U	U	U or C
4 KB overwrite	W	P | W	P | W	P | R+W	P | N (F?)
100 byte overwrite	R+W	P | W	P | R+W	P | R+W	P | N (F?)
100 byte append	R+W	P | W	P | R+W	P | R+W	P | N (F?)

CDM的Slides中有BlueStore写的状态机图。状态机是存储中常用的处理方式，处理写路径，Ceph的PG Peering过程也有相应的状态机。数据落盘，对应的是PREPARE->AIO_WAIT间的“Initiate some AIO”一步。之后经过多个队列，向RocksDB提交事务，以及完成Deferred Write和Cleanup。直到最终完成。



另外，BlueStore使用Direct IO提交数据，这样数据会立即落盘，而不是在内核中缓存；从而，存储系统可以完全自主地控制写的持久化。这是一个如今常见的做法。但代价是，不能利用内核缓存，需要自己处理缓存问题；也必须处理好数据对齐，以及写小于一扇区时的Partial-write问题。

BlueFS的架构

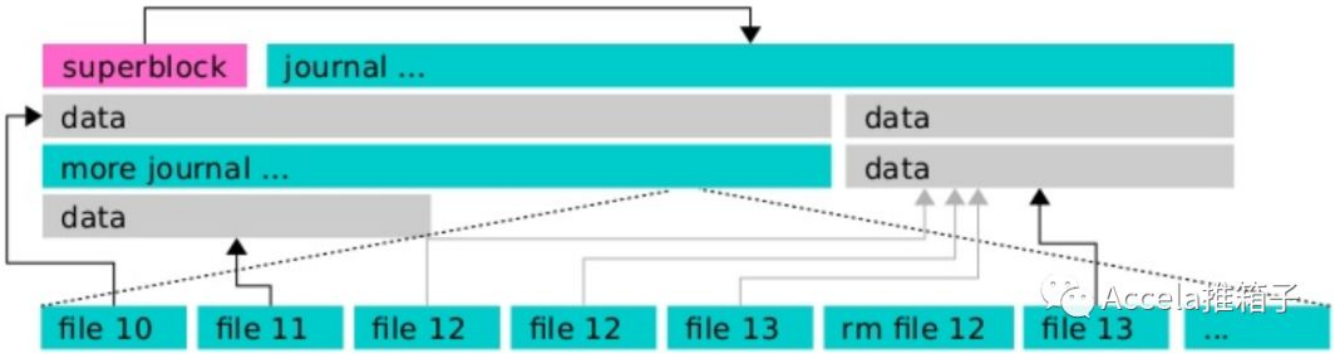
BlueFS以尽量简单为目的设计，专门用于支持RocksDB；RocksDB总之还是需要一个文件系统来工作的。BlueFS不支持POSIX接口。总的来说，它有这些特点：

1）目录结构方面，BlueFS只有扁平的目录结构，没有树形层次关系；用于放置RocksDB的db.wal/，db/，db.slow/文件。这些文件可以被挂载到不同的硬盘上，例如db.wal/放在NVMRAM上；db/包含热SST数据，放在SSD上；db.slow/放在磁盘上。

2）数据写入方面，BlueFS不支持覆写，只支持追加（Append-only）。块分配粒度较粗，越1MB。有垃圾回收机制定期处理被浪费掉的空间。

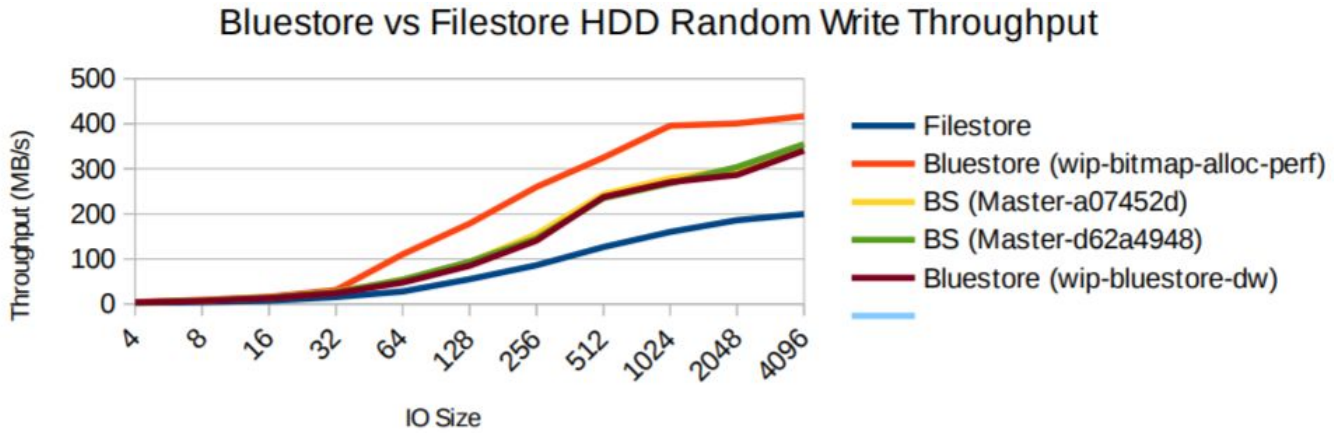
3）对元数据的操作记录到日志，每次挂载时重放日志，来获得当前的元数据。元数据生存在内存中，并没有持久化在磁盘上，不需要存储诸如空闲块链表之类的。当日志过大时，会进行重写Compact。

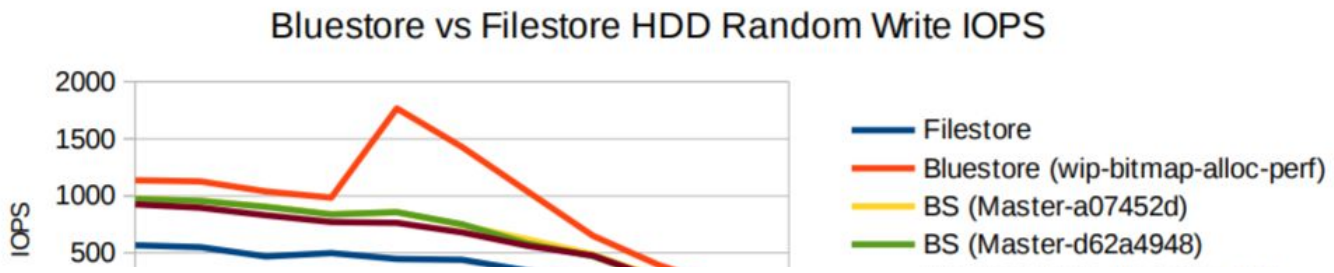
下图见于CDM的Slides，显示了BlueFS在磁盘上的数据结构。Superblock用于存储整个文件系统级别的元数据，日志和数据本着尽量简单的设计，按照追加的方式不断写入。关于写放大的问题，这是Append-only式通有的，在Write Behaviors论文中有详述。

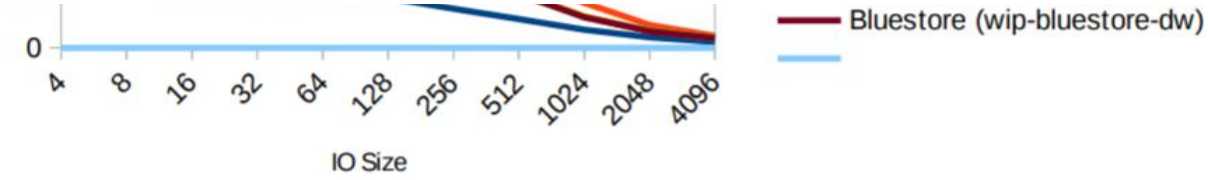


BlueStore的性能

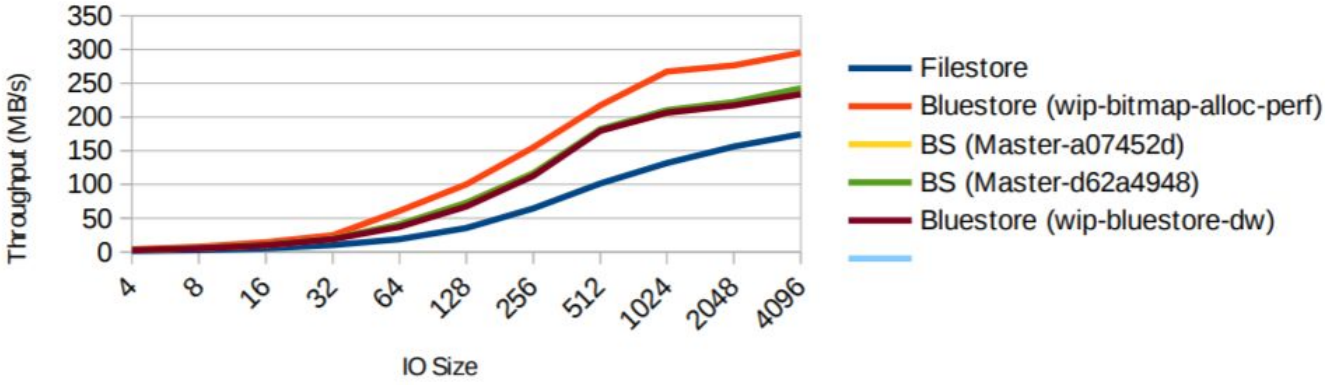
如果问为什么BlueStore相比FileStore能够提高越一倍的吞吐量，可能在于其更加简单、更加短的写路径；解决了双写问题，大部分数据不再需要在日志中多写一遍；借用RocksDB处理元数据，后者实现成熟，对SSD优化良好。



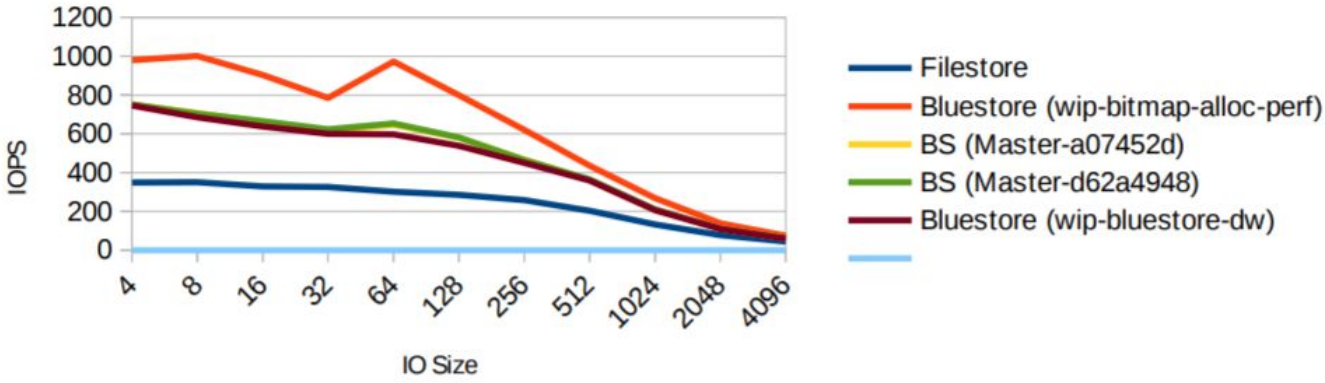




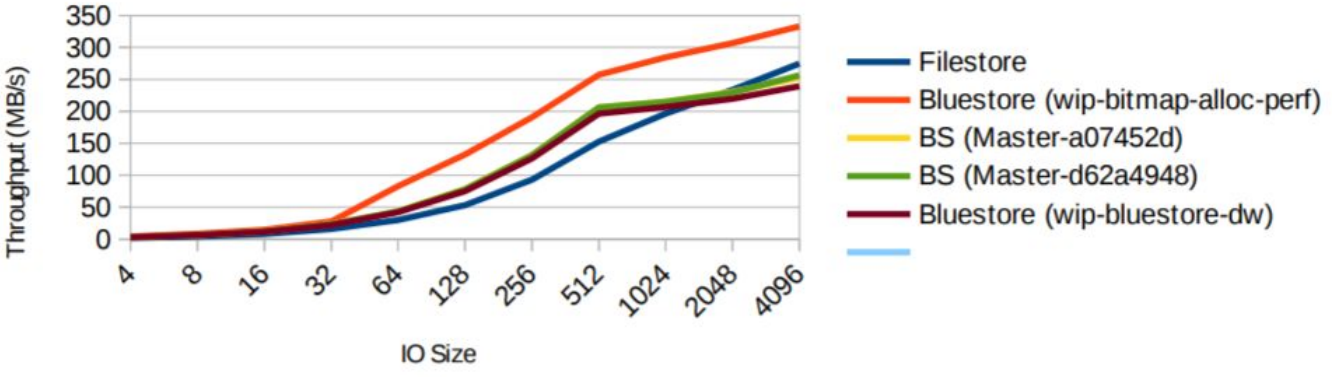
Bluestore vs Filestore HDD Random RW Throughput



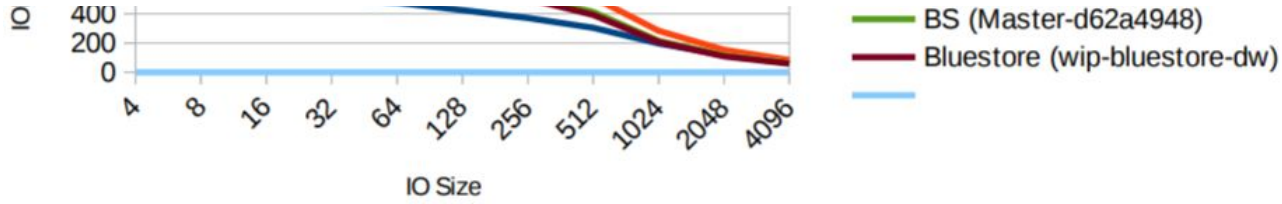
Bluestore vs Filestore HDD Random RW IOPS

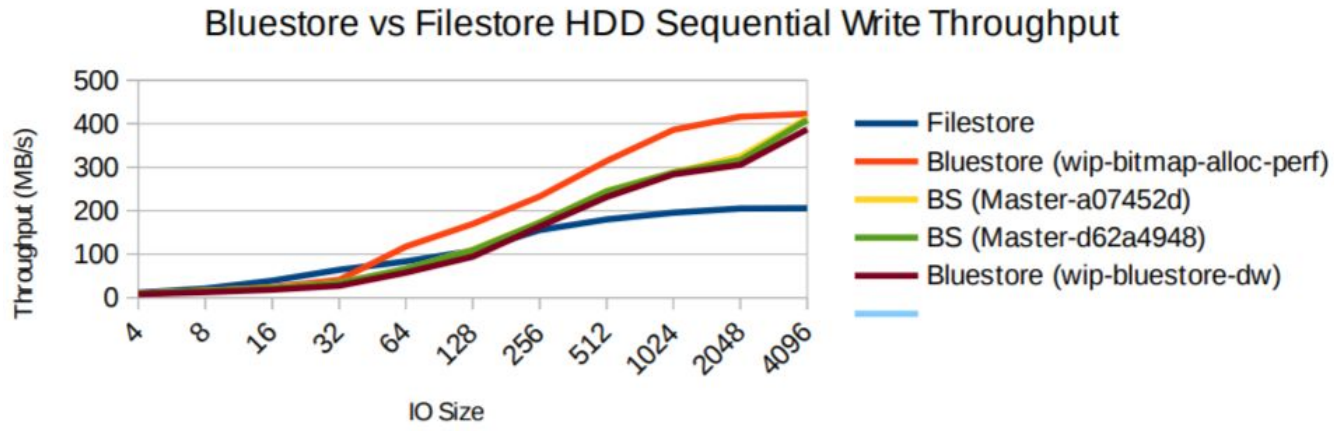


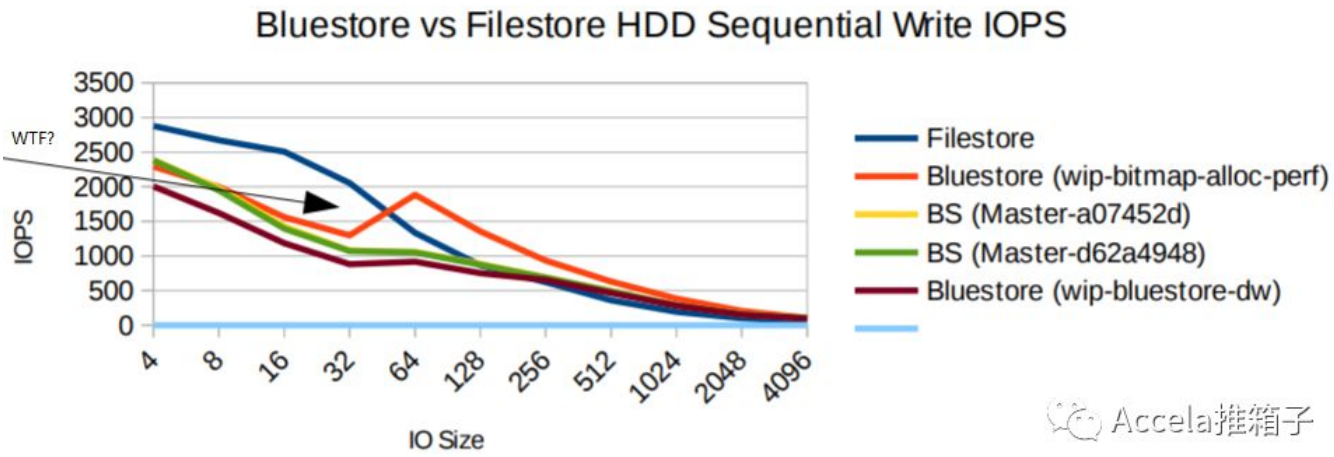
Bluestore vs Filestore HDD/NVMe Random RW Throughput



Bluestore vs Filestore HDD/NVMe Random RW IOPS







更重要的，和Ceph类似，公有云内部开发的存储系统也历久年月不断翻新；像Ceph社区这样，能够提出全新架构，把性能提升一倍，是非常值得借鉴的。