SSD & FTL 从底向上（P3完）

Original Accela Zhao Accela推箱子 2018-01-06

（续前文：SSD & FTL从底向上 (P2)）

设计变迁

SSD具有很多新特性，被存储系统广泛使用后，也带来了许多设计上的变化（Design Shifts）。SSD带来的高性能让CPU有些跟不上。其中，许多改进是优化了原本存储软件对并行和锁的处理，更加高效地利用多核。Linux内核以及上下文切换（Context switch）带来的延迟也拖慢了SSD的速度，于是有Intel DPDK/SPDK等在用户空间处理IO，绕过内核的做法。

Seastar是一套高并发框架。Scylla目标是改进Cassandra数据库，使用C++语言，提高并发、消除瓶颈，提高10x速度；Seastar作为其中的关键框架被开源出来。它大量使用FCP异步编程（Future，Continuation，Promise），将应用程序针对CPU核分片（sharding）；其动机在于更好地利用多核和并行。

另一特点是，Seastar将TCP/IP栈放在用户空间中，绕过Linux内核（Kernel bypassing），使用Intel DPDK；类似地，存储IO栈也使用Intel SPDK绕过内核。这么做的动机是，SSD速度更快，而Linux内核及上下文切换反而成为增大延迟的瓶颈，于是绕过它。

[Traditional (Left) vs ScyllaDB Design (Right)] (https://adtmag.com/articles/2015/09/23/scylladb-cassandra.aspx)

[Seastar/ScyllaDB How to Make Cassandra 10x Faster] (http://www.slideshare.net/TzachLivyatan/seastar-scylladb-or-how-we-implemented-a-10times-faster-cassandra)

另外，有一个Linux内核IO栈的分解图，可以看到应用程序的写请求是如何经过文件系统、页缓存（Page Cache）、块IO（Block IO）、驱动层，最后被写入物理磁盘或SSD的。

[Linux Kernel IO Stack](https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram)

Ceph BlueStore是继Ceph的FileStore和KVStore后新推出的底层存储组件。Ceph的架构解耦了分布式管理层和单节点的存储组件。OSD节点通过接口配置，可以自由选择使用

FileStore、KVStore还是BlueStore。FileStore使用文件系统存储对象，可以使用XFS、Ext4等文件系统。KVStore使用LevelDB作为存储对象。

BlueStore的动机在于，首先，文件系统是通用，其提供的功能远超Ceph自身所需。但是，超出的功能带来的性能开销，却是Ceph需要额外承担的。因而，BlueStore自己实现了BlueFS，这一轻量级的、专门为Ceph定制的文件系统。不过也有反面意见，通用文件系统的周边工具、稳定和调试等更为成熟；而BlueFS作为全新的、不通用的文件系统，则给用户的理解和运维带来负担。

另一方面，随着SSD的引入，Ceph的底层存储组件需要为SSD优化。实际上，BlueStore使用RocksDB作为后端；RocksDB社区成熟，其为SSD做过许多优化。BlueStore使用RocksDB来存储对象元数据，并且借用RocksDB的事务管理省去了自己做WAL（Write-ahead Log）带来的双写（Double-write，即一个用户写需要先写Journal，再写实际数据）问题。

另外，在BlueStore的软件层做了很多优化，例如提高并发、优化队列带来的阻塞开销等等。这点也类似Seastar；为SSD更高的读写性能而更加充分地利用CPU。过去，Hadoop所最初提出的IO很慢而CPU太快，用Java实现来牺牲CPU换开发效率和社区面的想法，如今已经不很适用了。

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

RocksDB由Facekbook开源，在社区广泛使用，被作为分布式存储或者数据库的单节点存储引擎。它基于LevelDB开发，增加了众多功能和性能改进，也为SSD做了许多优化。

[RocksDB: Key-Value Store Optimized for Flash-Based SSD](https://www.percona.com/live/data-performance-conference-2016/sessions/rocksdb-key-value-store-optimized-flash-based-ssd)

LevelDB是基于LSM-Tree的数据库，这种数据结构对写优化，但是落盘读被牺牲；而后台需要时不时进行合并（Compact）操作，顺序地读取和写入大量硬盘数据，占用资源也带来写放大的问题。RocksDB对合并操作改进不少，优化其空间占用、读写的放大问题。另一

方面，RocksDB的一个经典功能Universal Compaction允许在用户自由配置写放大、读放大、占用空间的优化与牺牲（Trade-off）。

[Discussions about Universal compaction](https://github.com/facebook/rocksdb/issues/1014)

此外，RocksDB的优化也减少了锁开销，提高读和写的并发能力；以及能够支持并行的合并操作。

LSM-Tree写放大，可以说是其独具优势的数据结构所带来的代价；另一方面，LSM-Tree又在各种存储和数据库中广泛使用。在下面的论文中有详细的描述。为了优化写操作，它们被首先缓存起来，写入硬盘；之后通过合并操作，才被放到最终的存储位置。相当于写被滞后了，从而由随机变成了顺序写入。

[WiscKey: Separating Keys from Values in SSD-consciousStorage](https://www.usenix.org/conference/fast16/technical-sessions/presentation/lu)

在数据的生命周期中，一份数据被多次读取和写入硬盘。论文中的测试支持，写放大甚至可以达到50x，读放大达到300x。对于磁盘而言，LSM-Tree减少随机写获益更大。而对于SSD，虽然减少随机写有益，但写放大带来的磨损（Wear-out）问题不可忽视。

WiscKey为解决这个问题，将存储键（Key）和值（Value）分离；只把键存储于LSM-Tree中。对于值的体积远长于键的情况，这个优化对减少写放大效果显著。

基于内容寻址（Content-based Addressing）是一种在SSD场景下适用的数据分布（Data placement）方法。传统上，一般是用过数据的键（Key）、对象的名字等，取哈希（Hash），来决定这个数据应该放置于哪个节点；或者，通过分配器为新数据分配一个位置，然后元数据表中记录下来。而基于内容寻址，则是给数据本身取哈希，得出其应存储于何处的地址。

这种方法会导致写完全是随机化的，对磁盘无法适用；而SSD能够高性能地支持随机读写。另一方面，基于内容寻址自动实现了数据去重（Data deduplication），因为相同的数据会得到相同的哈希值，被存储在同一处。相比以往，数据去重是存储系统中实现起来难度很

大的一块功能，需要全局的指纹（例如SHA1）缓存和比对、以及复杂的元数据管理；在大部分开源存储系统中甚至没有去重功能。

在全闪存阵列中，EMC的XtremIO和SolidFire采用了基于内存寻址。下表是SolidFire制作，出自链接的其视频；而XtremIO也有（极其）精彩的架构解析。Pure全闪存阵列也是领域内的出色产品。

[SolidFire: Comparing Modern All-Flash Architectures](https://www.youtube.com/watch?v=AeaGCeJfNBg)

[EMC: XtremIO Architecture](https://www.youtube.com/watch?v=lIIwbd5J7bE)

机柜级SSD（Rack-scale SSD），代表产品是EMC收购的DSSD。分布式的全闪存阵列，通常思路是由软件实现分布式管理，硬盘仍然使用SSD，SSD盘内硬件FTL负责管理本地SSD的块分配、垃圾回收等功能。

[EMC Introduces The DSSD D5]
(http://www.storagereview.com/emc_introduces_the_dssd_d5)

而DSSD则类似采用一个分布式FTL层，接入的是Flash；由分布式FTL全局管理分配、垃圾回收等功能；相比SSD各自为政，全局协调能够有更高的效率。DSSD采用专门设计的硬件，与用户节点连接时也为了降低延迟而采用专门设计的总线和协议。软件上，DSSD也采用了类似绕过Linux内核的做法。

使用方式上，可以把DSSD作为一个可以由多节点共享的、分布式的、持久的存储。PCIe SSD虽然能够做到比DSSD更低的延迟，因其没有分布式软件和硬件连线的开销；但其无法提供多节点共享，没有DSSD巨大的存储容量（DSSD是Rack-scale）。DSSD因其极低的延迟，甚至可以被当作大共享内存使用。一些金融用户可能会更加青睐DSSD。

[EMC Community: 极致性能存储DSSD的故事]
(https://community.emc.com/thread/225826?tstart=0)

FlashRAID是传统RAID为高性能地支持SSD而作的改进。可以看到传统RAID在许多设计和细节上，对SSD的支持并不理想。例如，SSD盘的失效曲线、写放大、随机写、RISL（Random Input Stream Layout，识别用户不同IO流的模式），零填充（Zero-fill）、文件系统Trim、EC（Erasure-coding）部分写（Partial-stripe Write）等问题都需要处理。Alan Wu的博客中有更多更全面的解析，例如

[Alan Wu的博客：RAID2.0核心思想：数据保护与物理资源管理域分离]
(http://blog.51cto.com/alanwu/1876942)

（**全文完**。注：本文为个人观点总结，作者工作于微软）