Build a Key Value Flash Disk Based Storage System

Outline

- Introduction，What's Key Value Disk
- A Evolution to Key Value Flash Disk Based Storage System
- Three Technologies to Setup a Flash Disk Based Stroage System

What's Key Value Drive

- ❑Drive Key Value is a drive level interface which can be adopted by multiple high-level Key Value storage applications.

- ❑Drive Key Value is a transportation agnostic interface and can run over Fabric(FC, RoCE, iWarp, TCP, et al) and PCIe architecture.

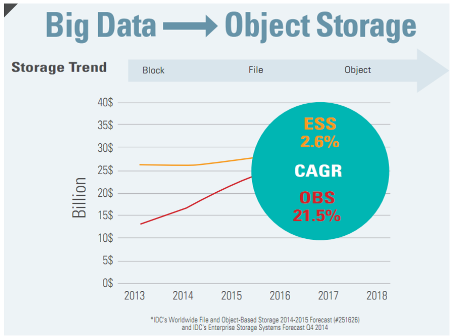
- ❑Drive Key Value supersedes and replaces the traditional block interface in some areas.

Board

Drive

JBOD

A Typical Key Value Scenario - Object Storage



❑Server SAN based solutions, like

- ➢Scale out object storage (like Amazon Cloud Storage S3).
- ➢Key Value Databases (like Redis, RocksDB)
- ➢In Memory DBs (IMDB, like Gemfire)
- ➢Distributed caching (memcached)
- ➢Object Storage Device（Ceph、Dsware）

All use key-value as a basic interface.

Medical

Financial

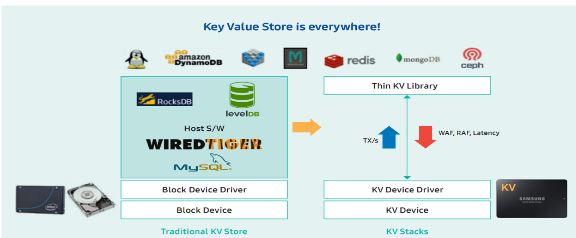
Industry

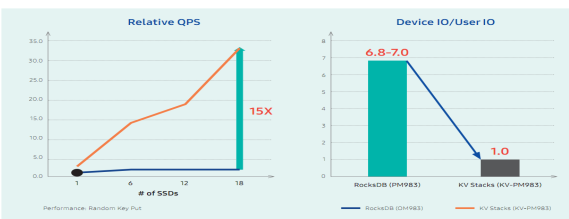
DC (Huawei Cloud)

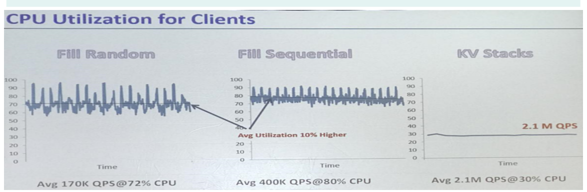
Flash Memory SUMMIT

Ecology – Some Vendors Have Launched NVMe Key Value

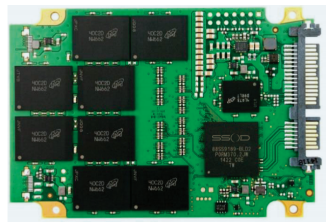
Samsung Key Value SSD

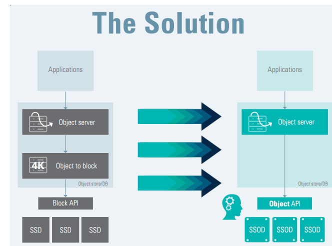


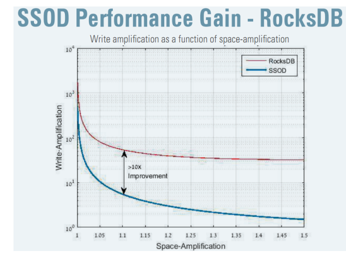




SSOD Key Value SSD







Sponsors

SAMSUNG

TOSHIBA
Leading Innovation

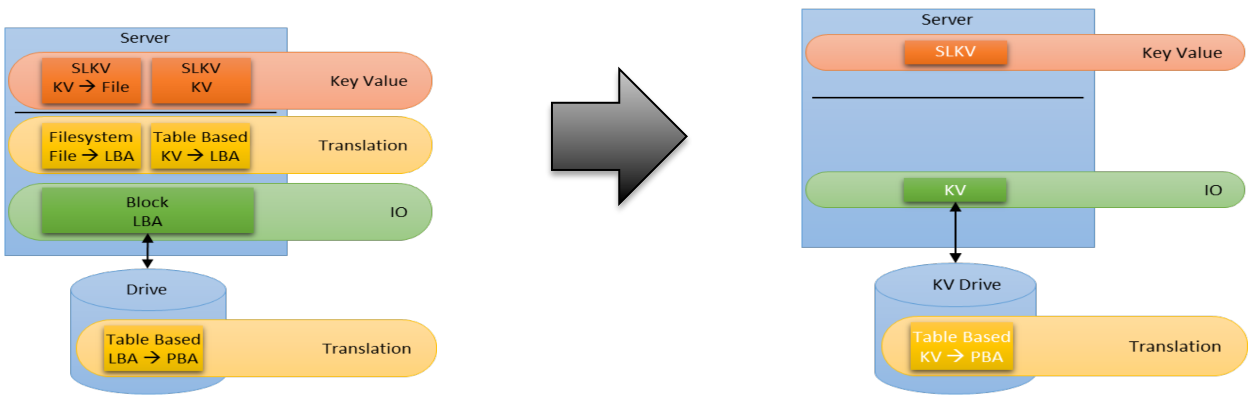
HUAWEI

NetApp

SEAGATE

SSOD

Storage Architecture Evolution I：Server side KV



Advantage：

❑**Less CPU consumption**

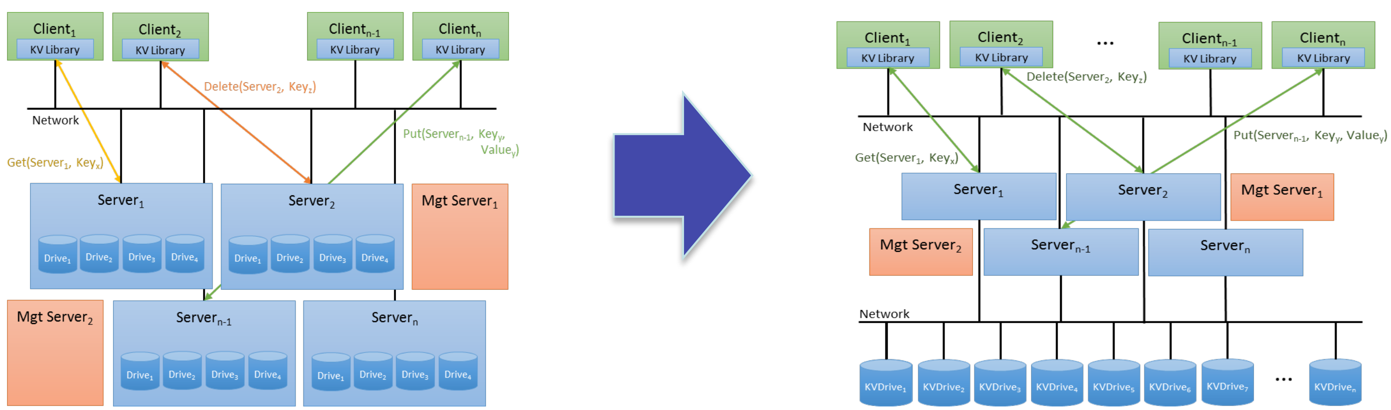
- ✧ Get rid of complexity translation between KV and Block.
- ✧ Get rid of redundant useless block protocol.
- ✧ SLKV and Disk Drive translation is more efficient.

❑**Less high-speed memory**

- ✧ Get rid of most of metadata.

❑**Low Latency/High Performance**

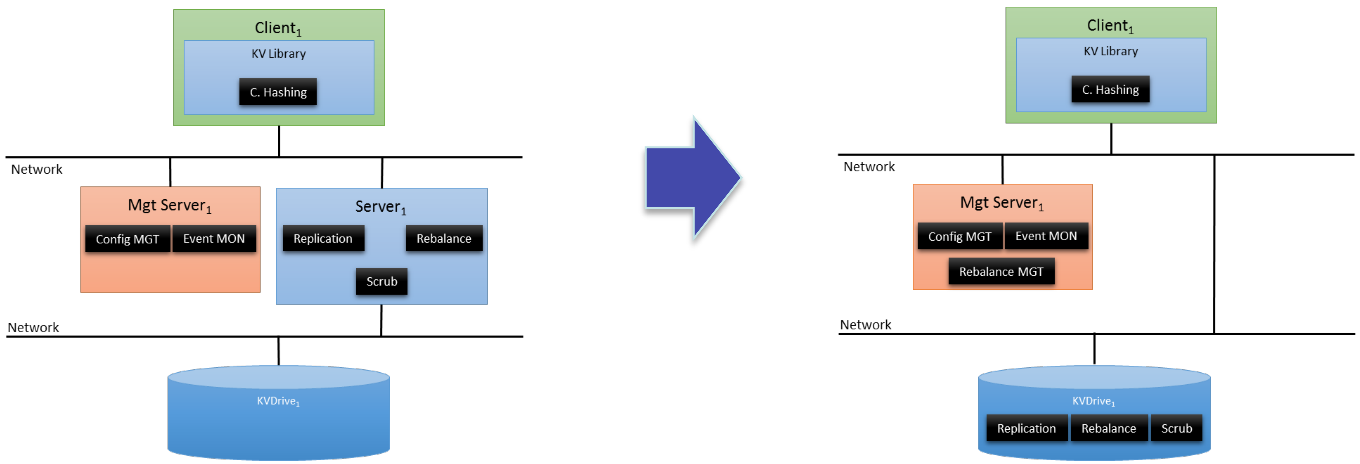
Storage Architecture Evolution II: Disaggregated Scale-out Object Storage



Advantage:

- Decouple KV Disk Drive and Server, Reduce the servers, Cost down.
- Flexible architecture, dynamic configuration to address the changing storage requirement.
- Small Fault Recovery Range.

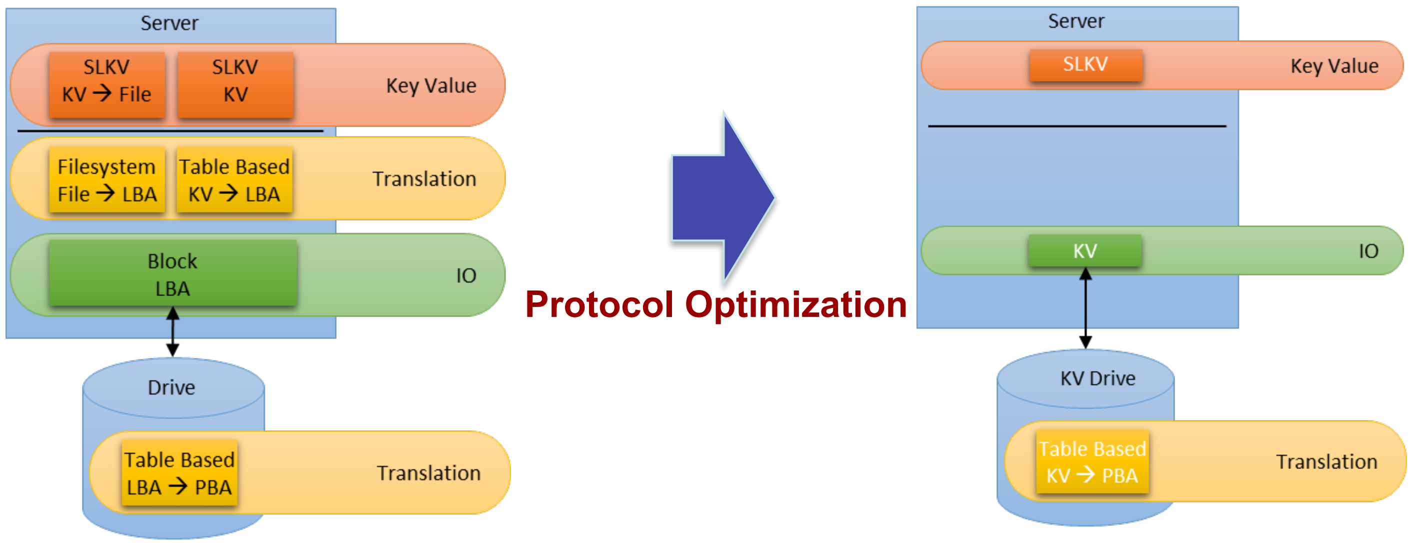
Storage Architecture Evolution III：Server-less Scale-out Object Storage



More Advantages besides "**Disaggregated** " architecture:

- ❑No Storage Sever required. Cost, space and power savings. Stronger Scale-out feature.
- ❑Workload offload. Fine-granularity(Disk Drive Level) parallel operation.
- ❑Small/Flexible Fault Recovery Range.

Key Technology innovation-I： Protocol Optimization

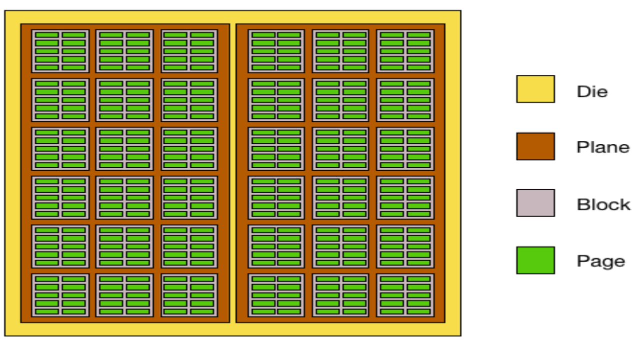


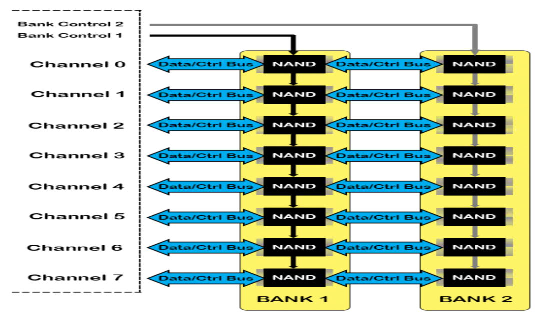
Previous Drive Protocol Stack

New Drive Protocol Stack

Key Technology innovation-II: FTL Optimization

Beyond protocol stack/transfer mechanism optimization, a new FTL is required if NVMe Key Value is adopted. A more complex FTL layer is the downside for NVMe Key Value. The new FTL is a key component to achieving high performance.





NAND Banks and Channels Illustration

Key Technology innovation-III: Standardized Key Value Interface

☐ New KV NVM Command/New data replacement.

✓New bit should be added to indicate that key-value or block data is transferred.

✓New parameter should be added to indicate the formation of key-value if key-value is supported. For instance,

- ◆Need a parameter to indicate whether one key value pair or multiple key value pairs are transferred in a command.

✓If multiple key value pairs are transferred in one command, need a parameter to indicate Key and Value stored interleaved with each other, or Key and Value stored separated in the command.

✓Should we support arbitrary length keys and values? If yes, parameters should be added in the command to indicate the length of Key and Value.

☐ New Transfer Mechanism

✓One method would be to transfer one or multiple key value pairs via one command if the key value pair do not larger than the max size of the command.

✓Another alternative to have the key transferred in the NVMe command and the value transferred thru DMA/RDMA.

✓However with variable length keys the previous method may not work. Another approach would be to create a metadata structure that fully describes the key-value operation including references to the key and value. This structure could then transfer both the key and value via DMA/RDMA.

The foregoing modification should be addressed by NVMe Key Value Standard.

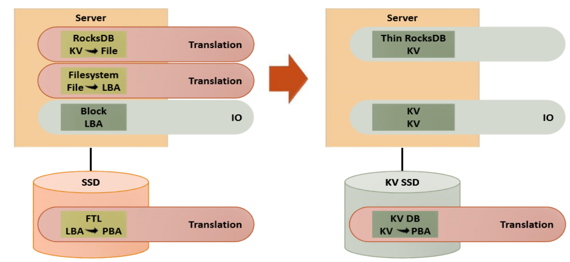
NVMe Technical Proposal Authorization Request

Technical Proposal Name:	NVMe Key Value enhancements
Date:	May 23, 2017
Impacted Specification:	NVMe 1.4
Sponsor(Samsung, Huawei)	Samsung: Bill Martin, Judy Brock, Yang Seok Ki Huawei: Philip Kufeldt, Robert Quixin NetApp: Fred Knight, David Slik

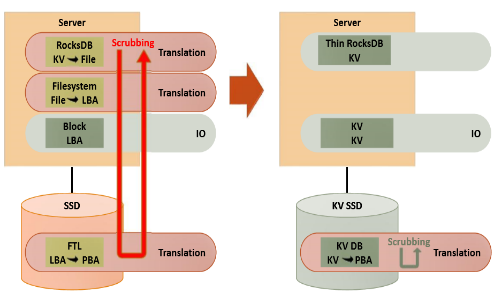
Solution Verification Step1 – RocksDB Optimization

应用优化	• 优化RocksDB的复杂的KV存储流程，去掉文件系统层，直接对接KV Lib
KV 库	• 提供KV API给应用使用，透传KV的访问通道，对接KV Driver
NVMe 驱动适配	• 开源NVMe驱动适配KV接口的盘，提供KV设备的初始化能力
控制器适配	• 控制器实现的KV->PBA转换，提供KV接口能力 • 后续可以提供更加高级的KV特性接口：scrubbing、GC、SCM cache、压缩等特性

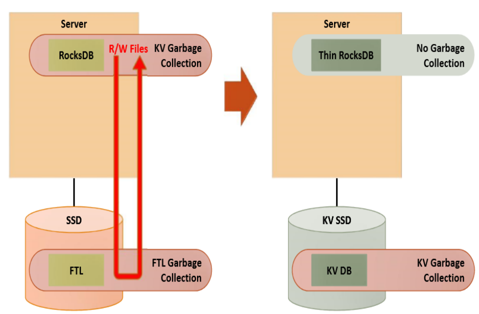
Translation Layers



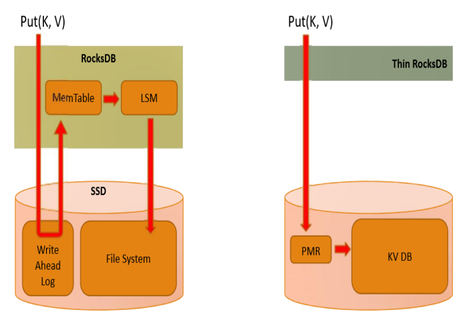
Scrubbing Too Far Away



Duplicated Work

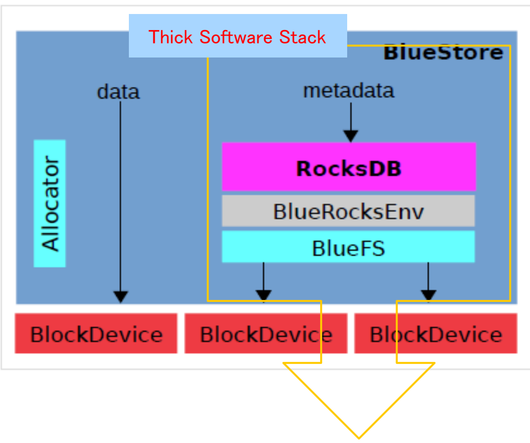


Memtable to PMR

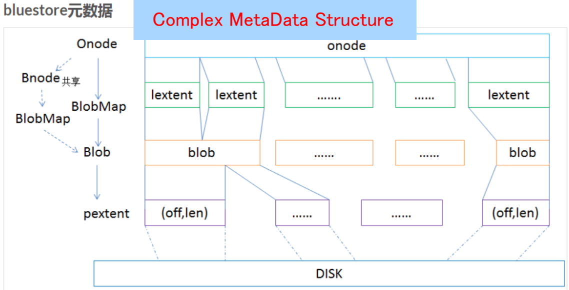


Solution Verification Step1 - Ceph Bluestore Optimization

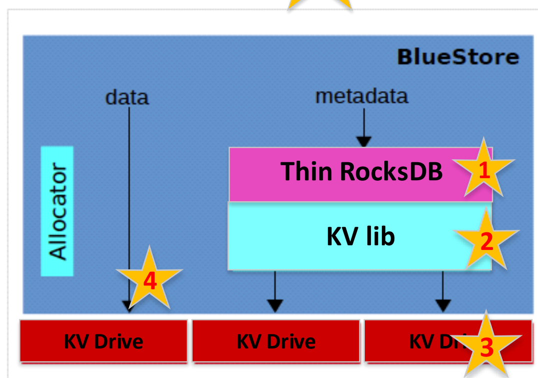
bluestore整体架构



bluestore元数据



bluestore整体架构 NEW



Bluestore背景：
Ceph后端因为filestore在写数据前需要先写journal，会有一倍的写放大，并且filestore一开始只是对于机械盘进行设计的，没有专门针对SSD做优化考虑，Bluestore初衷就是为了减少写放大，并针对SSD做优化

Bluestore存在的缺陷：
1、Bluestore本身沿用了RocksDB，还需要存在文件系统层，未从根本解决复杂软件堆栈
2、RocksDB后端对接的还是Block接口，需要KV→LBA→PBA的转化流程，增加了元数据的管理的复杂度；
3、RocksDB在Bluestore的主要线程之一_kv_sync_thread线程中的CPU占比超过90%

Bluestore New 架构的方案：
1、Bluestore采用Thin RocksDB，去掉文件系统层，从根本解决复杂软件堆栈，减少内存copy消耗，降低内存成本
2、RocksDB后端直接对接KV 接口，简化元数据的管理及软件处理开销，降低CPU消耗，大幅提升CEPH性能

CPU OverHead is High

```
+ 100.00% clone
  + 100.00% start_thread
    + 100.00% Thread::entry_wrapper()
      + 100.00% BlueStore::KVSyncThread::entry()
        + 100.00% BlueStore::_kv_sync_thread()
          + 90.50% RocksDBStore::submit_transaction(std::shared_ptr<KeyValueDB::TransactionImpl>)
          | + 89.50% rocksdb::DBImpl::Write(rocksdb::WriteOptions const&, rocksdb::WriteBatch*)
          | | + 89.50% rocksdb::DBImpl::WriteImpl(rocksdb::WriteOptions const&, rocksdb::WriteBatc
          | |   + 76.00% rocksdb::WriteBatchInternal::InsertInto(rocksdb::autovector<rocksdb::Writ
          | |   | + 75.50% rocksdb::WriteBatch::Iterate(rocksdb::WriteBatch::Handler*) const
```

Thank you

Questions?