SDC

STORAGE DEVELOPER CONFERENCE

SNIA SANTA CLARA, 2017

A New Key-Value Data Store For Heterogeneous Storage Architecture

brien.porter@intel.com

wanyuan.yang@intel.com

yuan.zhou@intel.com

jian.zhang@intel.com

Intel APAC R&D Ltd.

Agenda

- Introduction
- Background and Motivation
- Hybrid Layout Key-Value Data Store (HLKVDS)
  - Design and Implementation
  - Performance Overview
- Summary & Next Step

Introduction

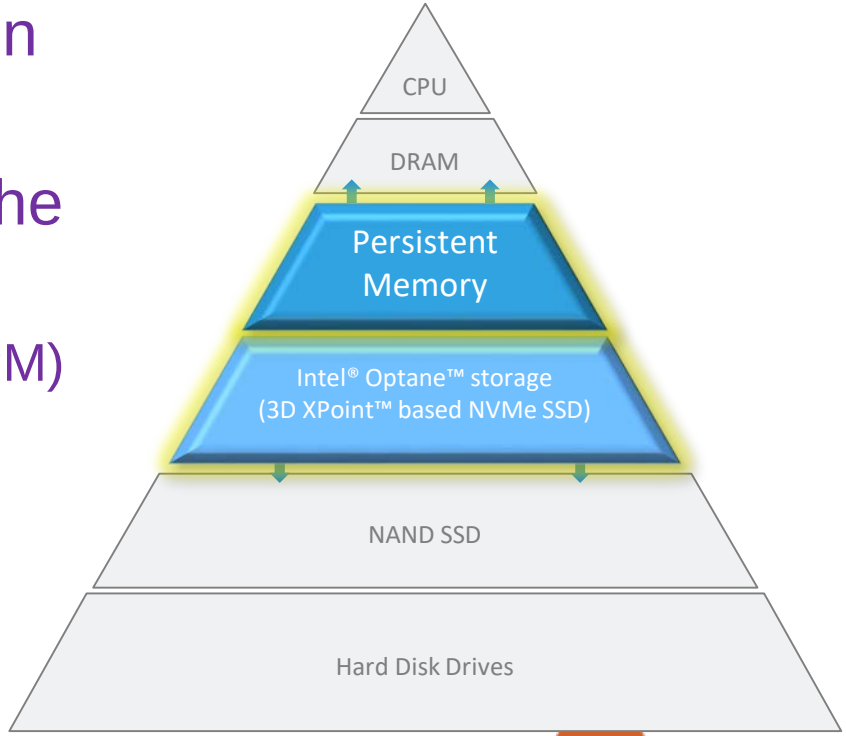
Introduction

- Intel Big Data Technologies Team is part of Intel's Software Service Group
- Tasked to deliver optimized BigData and open source solutions on Intel® platforms
  - Open-source @Spark*, Hadoop*, OpenStack*, Ceph*, etc.
- Working closely with community and customers
- Technology and Innovation oriented
  - Real-time, in-memory, complex analytics
  - Structure and unstructured data
  - Agility, Multi-tenancy, Scalability and elasticity
  - Bridging advanced research and real-world applications

Background and Motivation

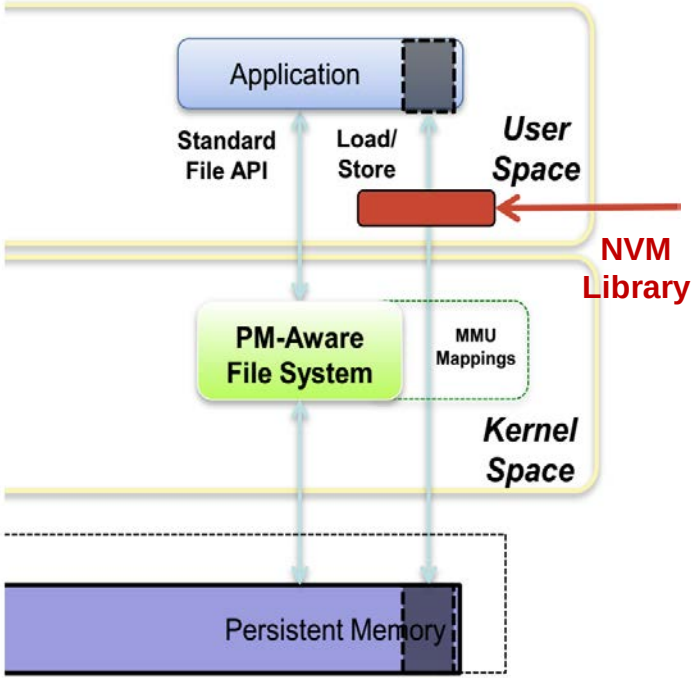
The Storage Hierarchy of Tomorrow

- There is a Memory-Storage gap in traditional storage hierarchy
- 3D XPoint™ technology breaks the memory/storage barrier
  - Memory-like non-volatile memory (PM)
    - Appears as memory to application
    - Application load/store data directly
  - Disk-like non-volatile memory
    - Appears as disk drives to application
    - Accessed with write/read



NVM Library

- NVM Library help developer build application on PM directly
- Open Sourced
  - http://pmem.io
- Providing transactional access method
  - libpmemobj
  - libpmemblk
  - libpmemlog



[Picture Source] https://www.snia.org/sites/default/orig/NVM2014_ppt/Rudoff-Overview-of-the-NVM-Programming-Model.pdf

Persistent memory programming will be easier with NVM Library

Motivation

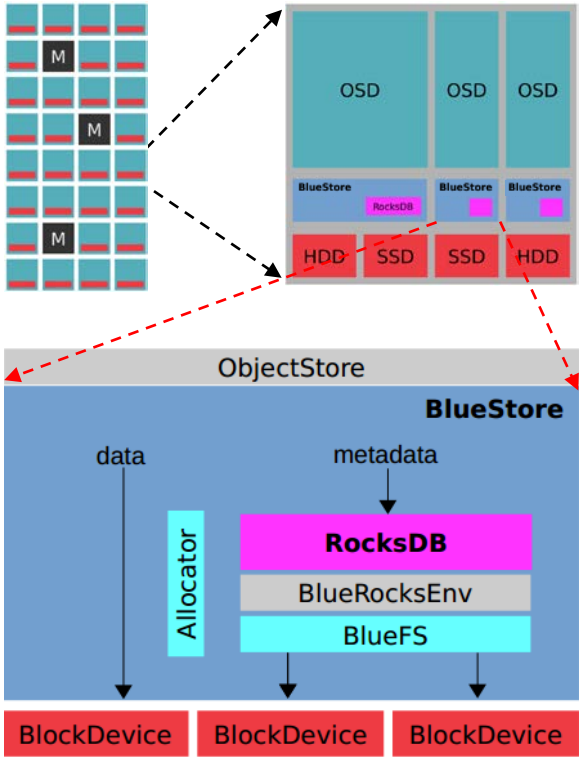
- New storage technology is emerging
  - HW - 3D XPoint™ technology
  - SW - new program model
- Persistent Memory will be a popular and powerful storage media in future storage solutions
- There are challenges in today's Key-Value Store solutions for Cloud Storage that need to be resolved
- There is need to re-design the storage engine and re-think how best to leverage Persistent Memory in cloud storage

Key-Value Store Engine usage model in Cloud Storage

- Mission critical metadata store
  - As a Key-Value metastore for any distributed storage system
  - Started with Ceph, A distributed object, block, and file storage platform
    - http://ceph.com
- Hybrid storage engine
  - As a cache or storage engine for distributed storage system
  - Started with a storage engine for HDCS (hyper-converged cache solutions for the cloud)
    - https://github.com/Intel-bigdata/HDCS
- As a high performance Key-Value database

Key-Value Store engine as Meta Store in Ceph BlueStore

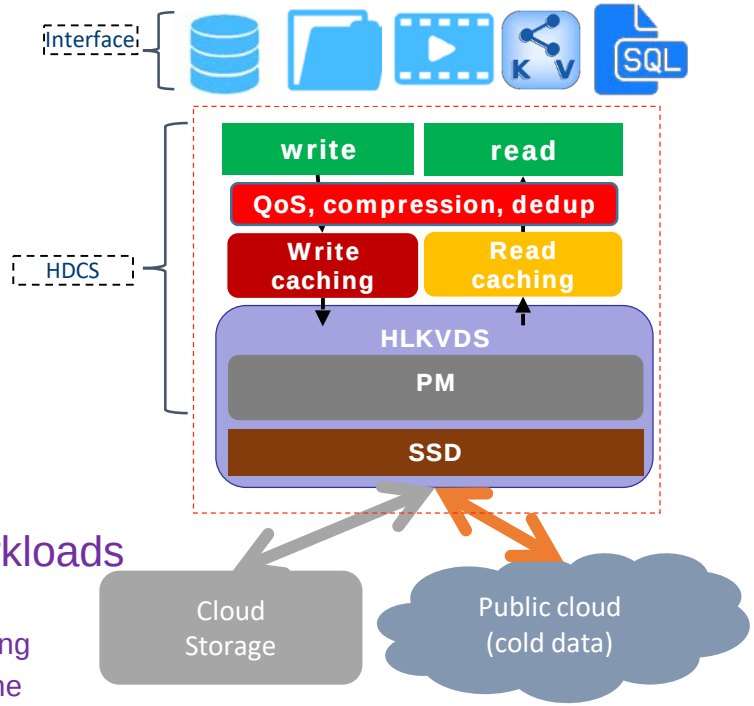
- **Ceph** is one of the most popular open source storage solutions
- Ceph contains two types of daemons
  - **Monitor** provides extremely reliable and durable storage of cluster membership, configuration, and state.
  - **OSD** provides specific data storage
- **BlueStore** is the newest OSD store backend
  - consumes raw block device(s)
  - key/value database (RocksDB) for metadata
  - data written directly to block device
- Challenges in Ceph Key-Value Store
  - RocksDB Compaction brings unstable performance
  - Need a Persistent Memory aware Key-Value Store



[Picture Source] https://www.slideshare.net/sageweil1/bluestore-a-new-faster-storage-backend-for-ceph

Key-Value Store Engine in Hyper-converged clouds

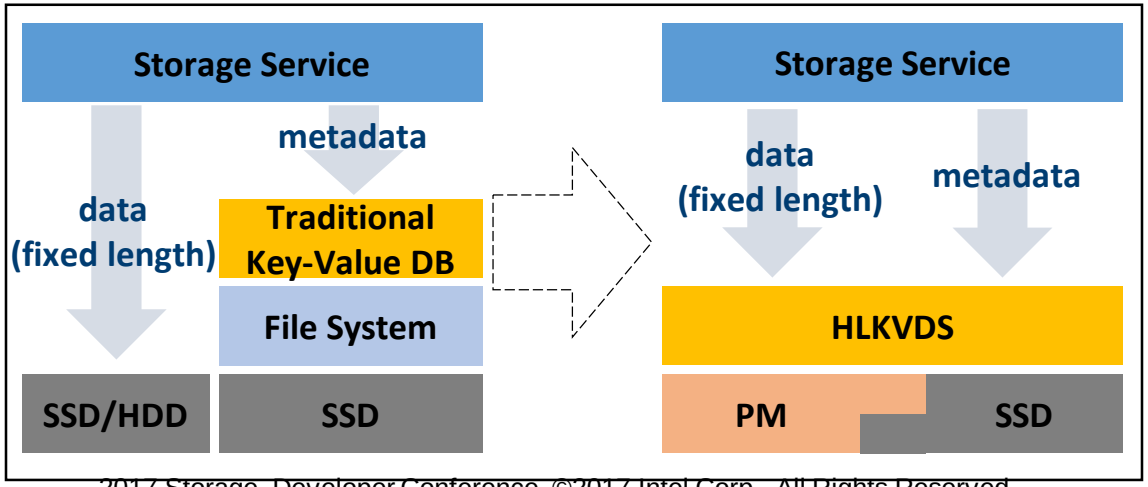
- Hyper-converged Distributed Cache Store
  - Providing cache for different APIs
    - Block, Object, File cache
    - Supporting various cloud storage backbends
      - Ceph, GlusterFS, Swift
  - Advanced service
    - de-duplication, compression, QoS, snapshot
  - Connecting with different backend
    - Private cloud as a warm tier
    - Public cloud as cold tier
  - https://github.com/Intel-bigdata/HDCS
- Unique I/O characteristics in Hyper-converged workloads
  - High concurrency, typically fixed length request (e.g., 4K).
  - Advanced data services like de-duplication, compression, data tiering
  - Extremely high performance, low latency – which might eliminate the dependency on local file system



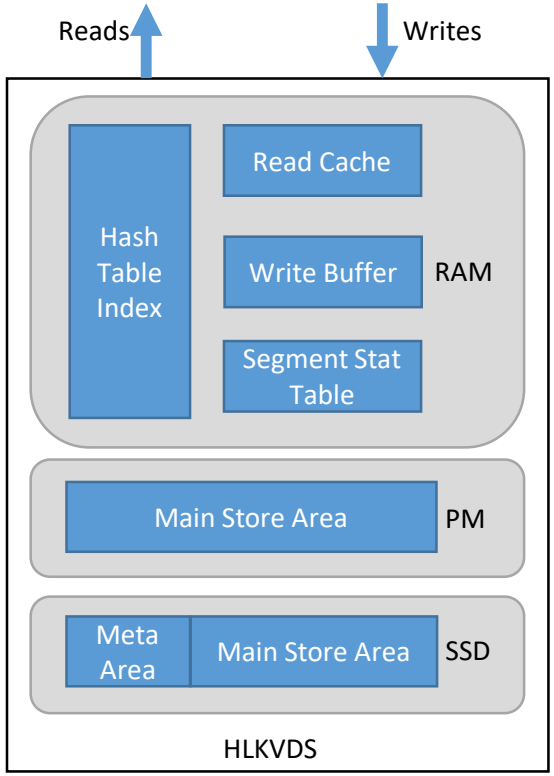
Hybrid Layout Key-Value Data Store (HLKVDS) Design and Implementation

Design Goal

- A simple and general solution
- Designed for the cloud - With a usage scenario of large, concurrent fixed-length random write/read requests
- A focus on high performance - Hybrid storage engine that supports both Persistent Memory and SSD



HLKVDS Architecture Overview



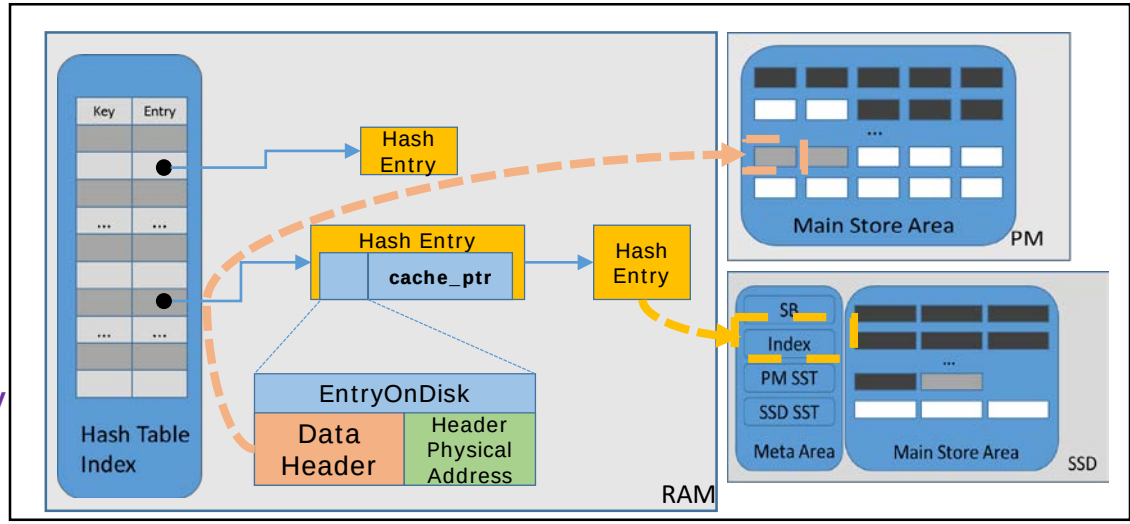
- A Key-Value store engine
  - With Put, Delete, Get, Iterator APIs
- The hybrid storage engine supports various hardware
  - DRAM, Persistent Memory, SSD
- Provides for rich features, optimized for Persistent Memory & SSD devices
  - Space provisioning & Recycling
  - Data tiering
  - Cache semantic
  - Optimized data layout based on media

HLKVDS Design Details

In-Memory Hash Index Table

- HKey: RMD160 (Key) -> Digest
  - Fixed-length entry in memory

  Notes: use RIPEMD-160 to generate key digest. The actual probability of collision is infinitesimally small or non-existent in practice for billions of keys.
- HValue: Hash(Digest) -> HashEntry
  - Separate-chaining with linked lists
- Write amplification
  - Write hash entry twice on device
  - Sacrifice space to increase random read performance



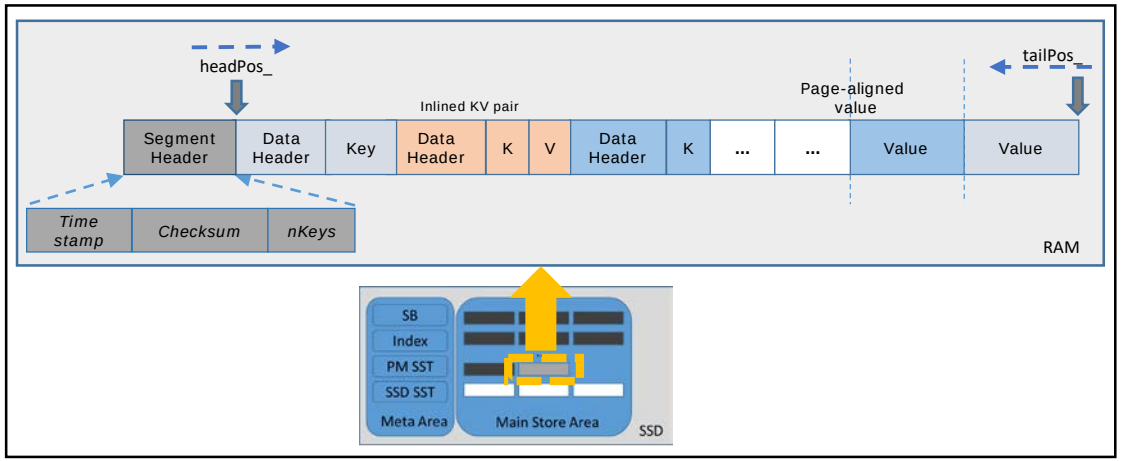
Data Header	
Digest	20Bytes
Key Size	4Bytes
Data Size	4Bytes
Data Offset	4Bytes
Next Header Offset	4Bytes

Header Physical Address	
Location (PM or SSD)	1Byte
Physical Offset	8Bytes

HLKVDS Design Details

Data layout

- Segment: Semantic abstraction of data placement
  - Optimized data layout adapt to SSD friendly IO pattern
    - Store fixed-length value with page align



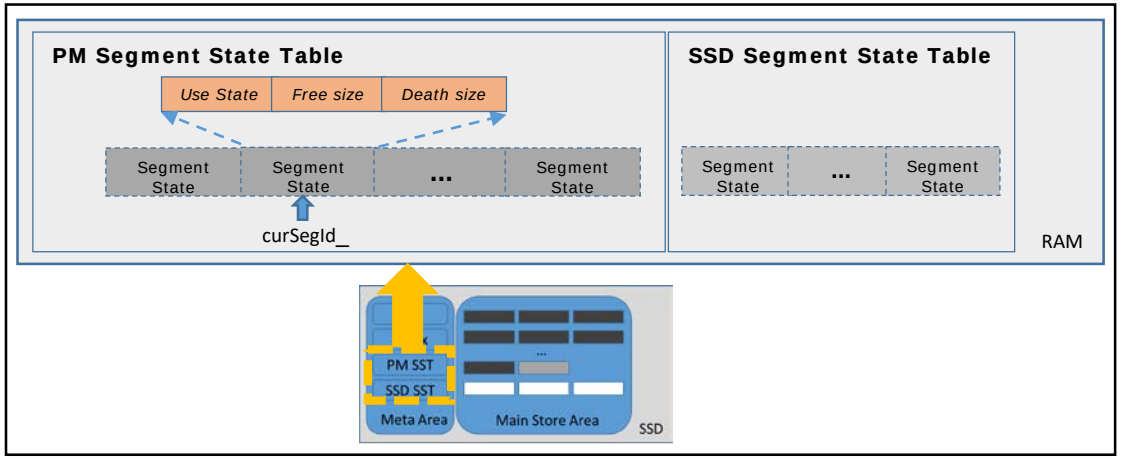
Data Header	
Digest	20Bytes
Key Size	4Bytes
Data Size	4Bytes
Data Offset	4Bytes
Next Header Offset	4Bytes

Data Header
Store the relative position of Key , Value , Next Data Header in the segment

HLKVDS Design Details

Space Provisioning

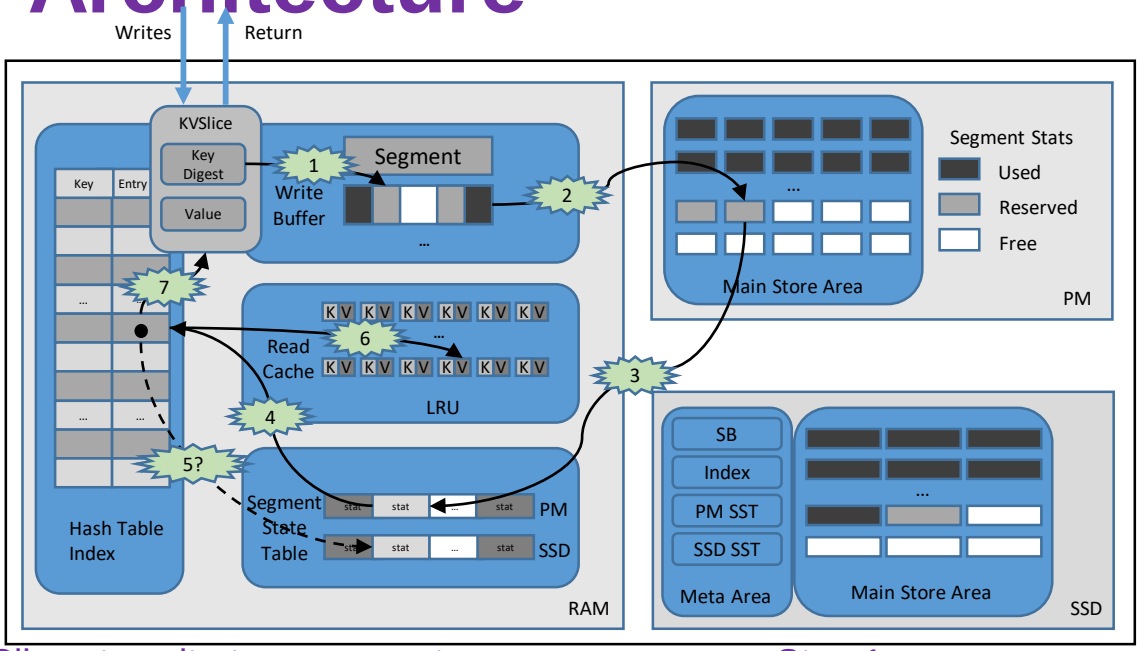
- Segment State Table maintains the whole segment state in RAM
- Segment State Tables persist to SSD



- **UseState**: State of the segment (free, used and reserved).
- **FreeSize**: record how much free space remained in the segment when the segment write to device
- **DeathSize**: record how much invalid key-value size in the segment. (when a key-value is deleted or updated, the origin segment store KV pairs are invalid)

HLKVDS Architecture

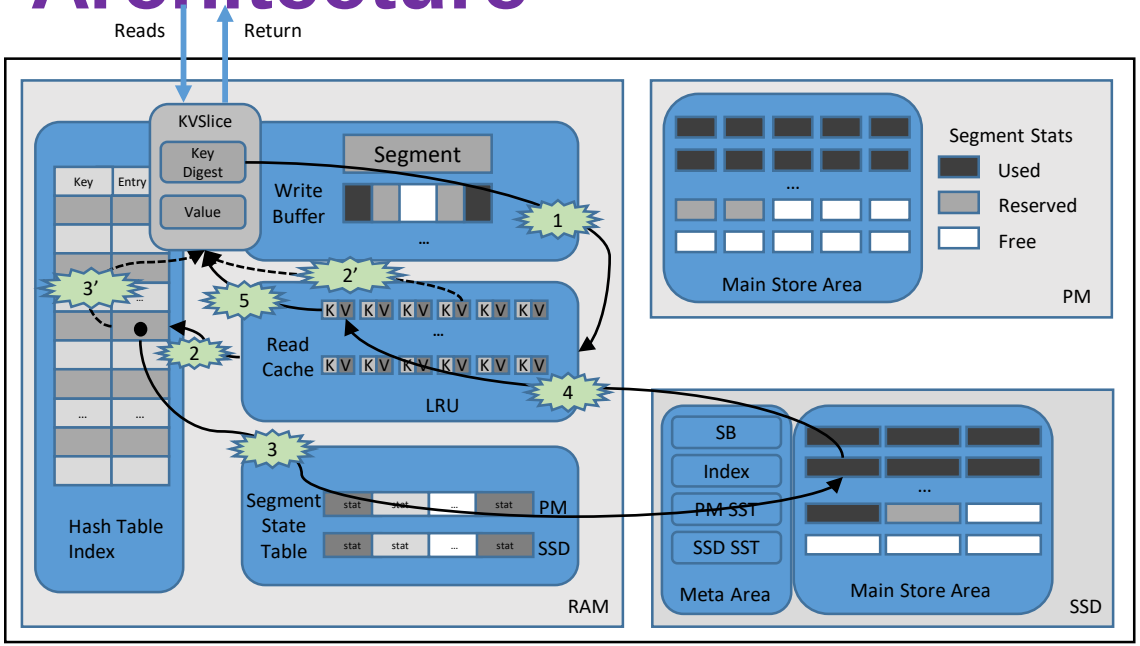
Write workflow



- Aggregate KVSlices to write to a segment (Timeout & Immediacy) – Step 1
- Write segment to Persistent Memory with log appending – Step 2
  - Data will be tiered to SSD by space recycling
- Update in-memory metadata - Step 3, 4, 5, 6
  - Delete operation is equivalent to inserting a KVSlice with a value of null (step 5)

HLKVDS Architecture

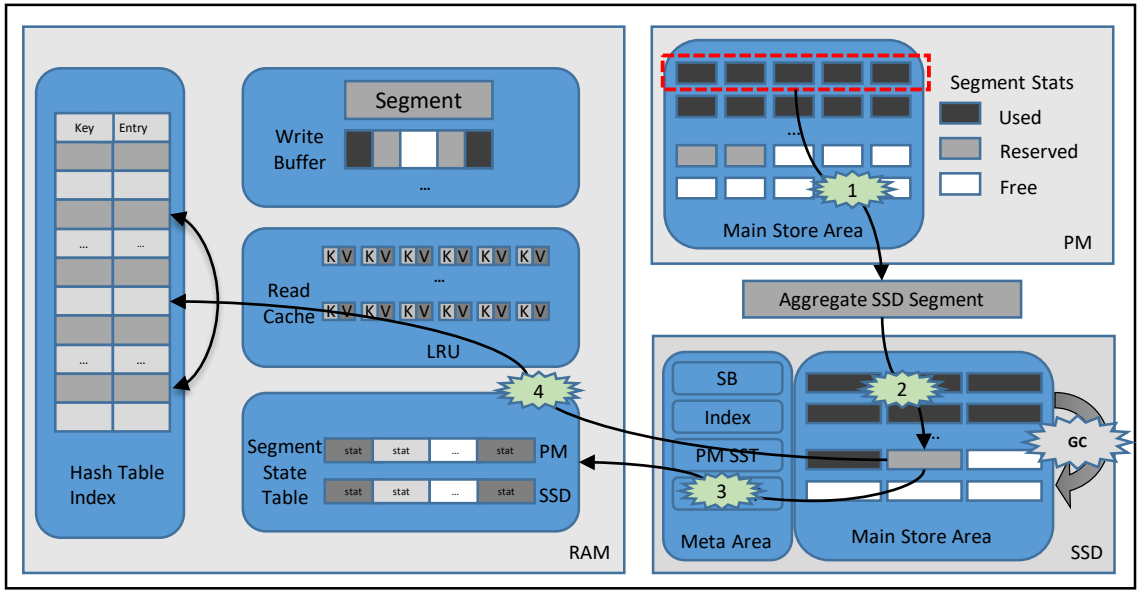
Read workflow



- Accelerated read with key-value LRU cache
- The value is obtained by a hash calculation and a read from the disk.

HLKVDS Architecture

Space recycling

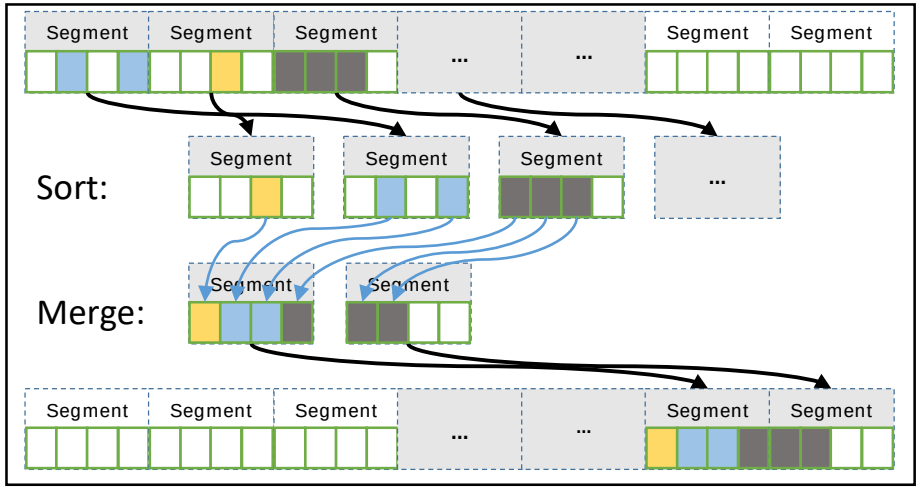


- Data on Persistent Memory will be tiered to SSD
  - Based on Threshold
- Data on SSD will do Garbage Collection

HLKVDS Architecture

Space recycling: SSD Garbage Collection

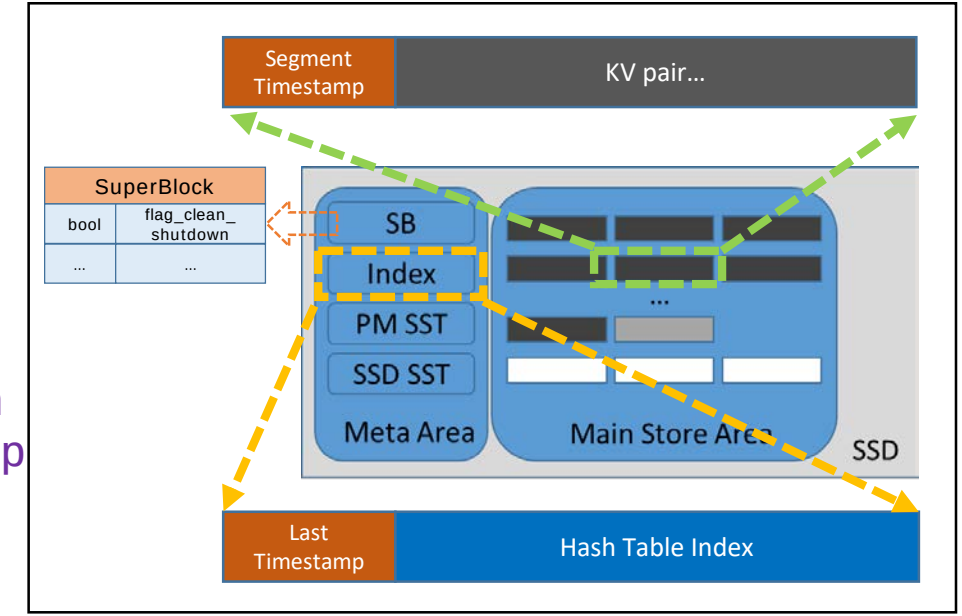
- Semantic abstraction of space recycling
  - Pick strategy base on timestamp for Persistent Memory
  - Pick strategy base on valid space capacity for SSD
    - Always pick segments which are most worthy of recycling



HLKVDS Design Details

Fast Recovery

- The Super Block sets flag when it is cleanly shutdown
- Writes the Hash Table index to the device periodically
- In a recovery, the system loads the last written Hash Table index checkpoint, then scan segments with timestamp after last saved timestamp and insert key-value pairs into the restored Hash Table index



Hybrid Layout Key-Value Data Store (HLKVDS) Performance

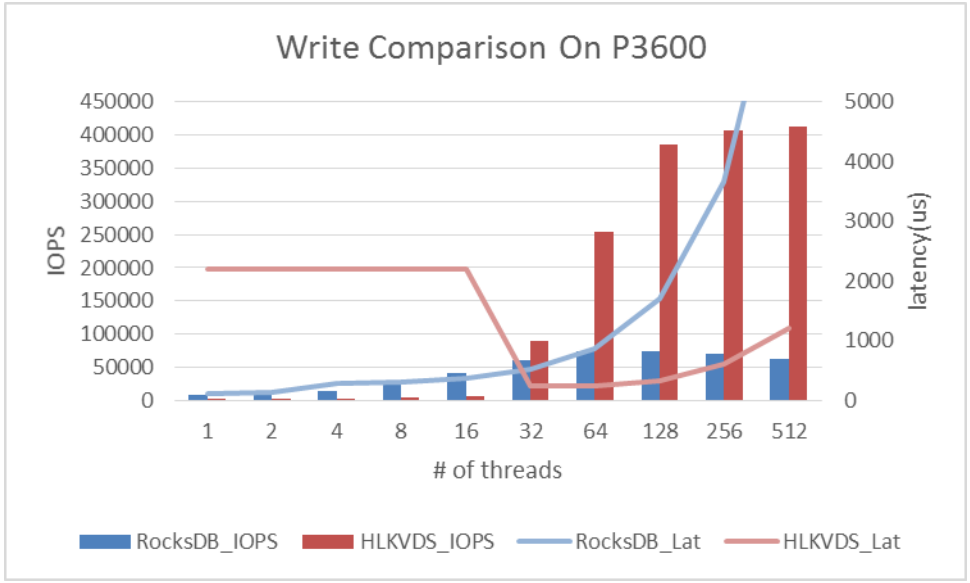
Test Configurations & Methodology

- **Test Configuration**
  - Each key size is 10 Bytes, value size is 4096 Bytes
  - Different concurrency(1-512) to simulate cloud storage scenarios
- **Benchmarks**
  - A simulation of db_bench*
- **Write Test**
  - Inserted 1 million key-value pairs with different concurrency
- **Read Test**
  - Inserted 10 million key-value pairs to warmup first
  - Get 1 million key-value pairs with different concurrency

Test Environment	
CPU	2 CPUs, HT enable 40 vCPUs, 10 Cores with Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GH
Memory	64GB
OS	Ubuntu 14.04
Device	Intel® SSD DC P3600 (2.0TB)
Misc	RocksDB(v.5.2) with ext4 filesystem; HLKVDS segment_size = 128KB

* Refer to backup for details

Write Performance



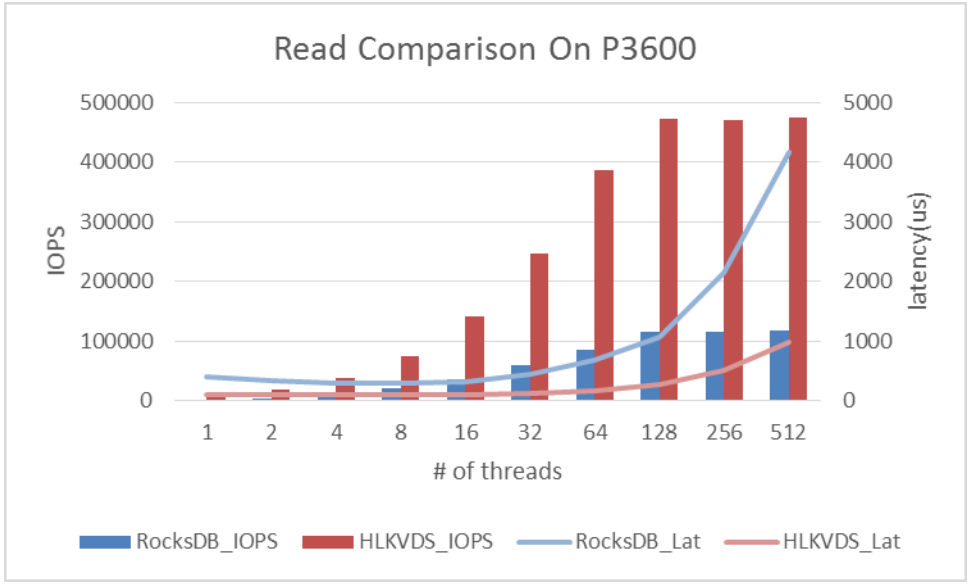
Intel® SSD DC P3600 Specification	
Sequential Read	2600MB/s
Sequential Write	1700MB/s
Random Read	450000 IOPS
Random Write	56000 IOPS

4K Write Performance	Throughput	Average Latency
Best Latency	90K IOPS	0.25ms
Peak IOPS	413K IOPS	1.2ms

Best latency when concurrency=32, ~1.5X performance advantage over RocksDB and latency < 0.5ms

Peak IOPS when concurrency=512, ~6.5X performance advantage over RocksDB and latency < 1.5ms

Read Performance



Intel® SSD DC P3600 Specification	
Sequential Read	2600MB/s
Sequential Write	1700MB/s
Random Read	450000 IOPS
Random Write	56000 IOPS

4K Read Performance	Throughput	Average Latency
Best Latency	8.5K IOPS	0.11ms
Peak IOPS	450K IOPS	0.25ms

- Simulated cloud storage scenarios with a large number of random reads

When concurrency=128, ~4X read performance advantage over RocksDB and latency < 0.25ms.

Summary & Next

Summary & Next Step

- Summary
  - HLKVDS is a new key-value data store designed for hybrid hardware environment like DRAM, PM and SSD.
  - It drives good performance in cloud storage scenarios of large concurrent fixed-length random write/read requests.
- Next Step
  - Performance evaluation as Ceph BlueStore metadata store
  - Performance evaluation on PM
- Call to action
  - Go to https://github.com/Intel-bigdata/HLKVDS to download
  - Share with us your feedback to wanyuan.yang@intel.com , brien.porter@intel.com

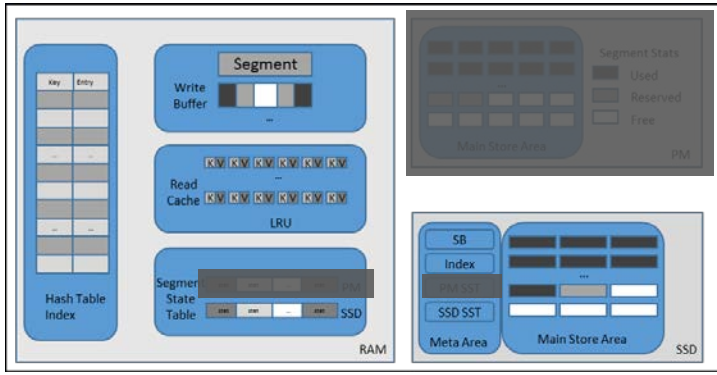
Legal notices

- Copyright © 2017 Intel Corporation.
- All rights reserved. Intel, the Intel logo, Xeon, Intel Inside, and 3D XPoint are trademarks of Intel Corporation in the U.S. and/or other countries.
- *Other names and brands may be claimed as the property of others.
- Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Backup

Performance Test

- Implemented the prototype with 1-tier
  - Aggregate write with timeout strategy
- Implemented benchmark tool
  - Reference to db_bench [1]
- Simulated cloud storage scenarios
  - Access data with different concurrent number
  - 4K Fixed-length value
- Performance comparison with RocksDB



[1] https://github.com/facebook/rocksdb/wiki/performance-benchmarks