Open-Channel Solid State Drives
Matias Bjørling

Linux Software Foundation Vault 2016
Solid State Drives

- High Throughput
- Low Latency
- High Parallelism
Solid State Drives and Flash Translation Layer

- **Flash Translation Layer (FTL)**
  - Read/Write/Erase -> Read/Write
  - Translation map
    - Data placement decisions
  - Garbage Collection
    - Device performs garbage collection in background
  - Wear-leveling
    - Flash has a limited amount of writes available

- **Applications**
  - Applications work with (and optimize for) LBAs
  - Use trim to hint FTL
  - Makes assumption about good data placement strategies
Embedded FTL Solid State Drives

- Hardwires design decisions about data placement, scheduling, garbage collection, and wear leveling.
- Designed with explicit assumptions about application workload.
- Non-optimal write workloads increase write-amplification
- Performance is tied to the cost of heavy over-provisioning
- Hints from applications are best effort
- Predictable latencies cannot be guaranteed – 99 percentiles
Solid State Drive Workloads

• SSDs are targeted typical workloads:
  • Accesses (e.g., 90%, 75%, 50% reads)
  • I/O Applications (e.g., MySQL and RocksDB)
  • Latency requirements (Quality of Service)

• Cost and lack of flexibility for these “hard-wired” solutions is prohibitive:
  • What if the workload changes (at run-time)?
  • What about new workloads?
  • And new applications?
Common Solid State Drive Design

- **Write Line**
  - Also known as virtual block, raid line, erase unit, ...
  - Optimizes for Throughput – Across all channels and LUNs
  - Considerably reduces metadata requirements
  - Single digit GB write lines are not uncommon

- **Critical Sections**
  - Write/Erase before Read -> Read must wait 1-2ms

- **Common Strategies for efficient reads**
  - Erase/Program Suspend -> Delays erases and writes
  - Reconstruction from parity -> Extra reads

LUN -> One unit of parallelism
- Either read (~50-80us), write (~1-2ms), or erase (~1-2ms)
Open-Channel Solid State Drives

• Host in Control
  - Data placement
    • I/O scheduling, Wear-leveling, Over-provisioning
  - Garbage Collection
  - Physical Page Addresses (PPAs) vs LBA + Range
  - Mapping Table <> Block-based vs Page-based

• Device maintains responsibilities
  - Bad block management
  - ECC Calculations
  - XOR Calculations

• Media optimized I/O Commands
  - Programming Mode, Multiple PPAs, Read retry, ...
Open-Channel SSD: Usages

• Predictability
  - Predictable I/O with consistent and low latency QoS requirements
  - Decrease write amplification
  - Reduce ‘noisy neighbor’ problem

• Management
  - Manage all flash media globally
  - Petabytes of storage with flash
  - Single level address space

• Application-Driven Storage
  - Programmable SSDs versus Hardware/Firmware configurable
  - Integrate with I/O application

Adopters

<table>
<thead>
<tr>
<th>Web-scale Data Centers</th>
<th>Hyper-converged Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Array Vendors</td>
<td>High-Performance Computing</td>
</tr>
</tbody>
</table>
Open-Channel SSD: Architecture

Common Data Structures
- Append-only
- Key-Value

Provisioning Interface
- Get block
- Put block

LightNVM Spec. Interface

LightNVM
- Block Device Target (pblk)
- Other Targets
- Full Stack FTL

General Media Manager

NVMe Device Driver

User-space
- I/O Apps
- liblightnvm
- Apps

Linux Kernel

File-System(s)

Open-Channel SSD
- Metadata State Mgmt.
- ECC Engine
- XOR Engine

Block Storage

Hardware Offload
Open-Channel SSD: Specification

• LightNVM Specification
  - Works with NVMe
  - Vendor-based command set
  - Identification through PCI IDs
  - Path to standardilization
    • ”Physical Page Addressing” Command Set

• Vendors implement their own
  - ECC algorithms
  - XOR engines
  - Offload engines
To support Physical Page Addressing, a new identify structure is implemented.

**Identify**
- Flash Media Type (SLC, MLC, TLC, QLC)
- Number of channels
- Number of LUNs, planes, block, pages, sector size, etc.
- Multi-plane Operation Support (Single, Dual, Quad, ...)
- Media and Controller Capabilities (SLC, Scramble, Encryption, etc.)
- Page Index Table – Let host know the order of writes to MLC/TLC flash media
- PPA Format (Device-side PPA format)

**Bad block Management**
- Get and Set Bad Block State – Easy Recovery
LightNVM Specification: I/O Commands

• Write PPA – Enable host to write to a PPA in a unit of a sector
• Read PPA – Enable host to read PPAs in a unit of a sector
• Erase PPA – Enable host to erase a one or more NV memory blocks
• Read/Write
  - Limited Retry (Only try to read once, else apply all means to recover data)
  - FUA – Force data to media
  - SLC Mode – Write only to lower page of MLC/TLC media
  - Suspend Enable – Apply Erase/Program Suspend to the command
  - Plane operation – Access with either single, dual or quad plane commands
Open-Channel SSD: General Media Manager

- One media manager instance per SSD
- Block media information
  - Manages physical address space of a device
  - Initialization/recovery of block states
  - Can manage multiple blocks as a single block
  - Manages addressing mode for device
    - Global PPA <-> Device PPA
  - Provides Wear-leveling
  - API – Manage address ranges of physical address space
    - Get/Put Block of the Physical Address Space
    - Block size is typically one or more physical flash blocks.
pblk – Physical Block Target (FTL)
- Exposes media as a LBA block device
- Implements write buffering
- Manages logical to physical address table (L2P)
- Orchestrates data placement
- Garbage collection
- Multiple targets per device
Open-Channel SSD: Steady State Example

- Manage your over-provisioning
  - SSDs employ 10-30% over-provisioning for performance
  - Improve the steady state of the drive
  - Predictable latency
  - Reduce I/O outliers significantly
Westlake ASIC Platform
- NVMe Compliant
- PCIe G3x8 or dual G3X4
- 4x10GbE NVMe
- 40 bit DDR3
- 16 CH SLC/eMLC/MLC/TLC NAND

Performance Measurements:
- 16 Channels
- 2TB MLC Flash
- 400MT/s
- Host Interface PCIe G3x8
- User space performance
- Preconditioning & garbage collection enabled

ASIC Performance (NAND @400MT)
- Seq. READ: 4.0 GB/s
- Seq. WRITE: 4.1 GB/s
- Ran. 4K READ: 983K IOPS
- Ran. 4K WRITE: 892K IOPS
Westlake ASIC: Read/Write Latency

*FPGA Controller - Not final performance numbers
Westlake ASIC: Read/Write Performance

Read/Write Performance (MB/s)

- Write KB/s
- Read KB/s

*1T - Not final performance numbers
Application-Driven Storage with Open-Channel SSDs
Application-Driven Storage: Motivation

- I/O applications with log-structured data structures
  - Writes are sequential
  - Reads are random
  - Great for Open-Channel SSDs

- Handle physical data placement and garbage collection and avoid expensive translation layers
  - Multiple layers of translation
    - Application -> Logical data (e.g. record, key-value pair, blob) to physical file locations
    - File-systems -> File offset and range to LBAs
    - SSD -> LBAs to PPAs
  - Convenience of the traditional storage stack becomes a bottleneck

- Mismatch between embedded FTLs capabilities and I/O applications expectations
Application-Driven Storage: Applications

- Single level of translation – Applications map data to physical storage
- Fast path for both reads and writes
- Use application data structures to make informed decisions regarding latency, resource utilization and data movement
- Reduce SSD write amplification by accurate flash block alignment
- Translation table overheads are stored within application data structures
Application-Driven Storage: liblightnvm

- User-space library to access Open-Channel SSDs
  - Generic interface for programmable SSDs
  - Open-Channel SSD configuration exposed through library and sysfs
  - ioctl for read/write/erase
- Applications integrates with Open-Channel SSDs by using the exposed configuration and ioctl.
Sysfs Integration

• Each device exposes its configuration
  - Number of block, channels, lun, pages, planes, page size, sector size, ...
  - Registered media manager
  - Device PPA format

• A registered target exposes its
  - Configuration
  - Devices that it is attached to
  - And any other variables

• Applications may use liblightnvm or query sysfs for device geometry

Available from 4.7 of the kernel
liblightnvm Data Commands

• **Physical Page Address Commands**
  - Three buffers
    • Data buffer (sector aligned data buffer)
    • Metadata buffer (Out-of-band buffer)
    • Physical Page Address (PPA) List
      • If > 1 PPAs, buffer with list of PPAs to access, else PPA passed directly
  - # of pages (i.e. sectors defined in the PPA list)
  - Type of command (read, write, erase)
  - Flags
    • Program/Erase Suspend
    • Single, dual, quad plane operation
    • Polling
    • Vendor-specific – Encryption, compression, etc.

```c
struct ppa_cmd {
    void *databuf;
    void *metadatabuf;
    union {
        void *ppa_list;
        u64 ppa_addr;
    }
    int nr_pages;
    int rw; /* r/w/e */
    int flags;
};
```

*Preliminary design*
RocksDB and Open-Channel SSDs
RocksDB: Overview

• Embeddable Key-Value persistent store where keys and values are arbitrary byte streams.
• Fork from LevelDB
• Based on Log-Structured Merge Tree
• Optimized for fast storage, many CPU cores and low latency
• Used by Facebook, LinkedIn, Yahoo, AirBnb, ...
RocksDB: Log-Structured Merge (LSM) Tree

- Persisted In memory: Memtable

- e.g. 64MB
  - Append-only ->
  - L0
    - Most Updated
    - A Z
  - Sstable
    - sync

- e.g. 128MB
  - L1
    - A Z
    - compaction

- e.g. 1G
  - Ln
    - Least Updated
    - A
    - ... Z
    - compaction

- e.g. 100G

- Sstable Size
  - User data age

- Grows 10X
RocksDB: Log-Structured Merge (LSM) Tree

- Storage unit
- Approximate size - configurable (top-down)
- Data Block, Filter Block, Index Block, Metadata

• Data locality per sstable (> L0)
• All data in sstable is invalidated simultaneously
  - sstables are immutable
  - Merges involve the whole sstable

• But after FTL...
  - sstable data not guaranteed to be sequential on physical media
  - Append-only inserts translate into: 1 invalidation + 1 page write + GC
    → Device-added write amplification
    → GC introduces long unpredictable latency tails
RocksDB on Open-Channel SSDs

- **Objective: Optimize RocksDB for Open-Channel SSDs**
  - Control data placement:
    - Align data in sstables and WAL to physical media (same block, adjacent blocks)
    - Other metadata continues to be stored on posix compatible block storage
  - Exploit Parallelism:
    - Define virtual blocks based on I/O patterns issued by the storage backend
    - What is the sustained write requirement?
  - Reduce GC and minimize over-provisioning
    - Use LSM merging strategies to remove device-side GC and over-provisioning
    - Drive SSD in a stable state
  - Control I/O scheduling
    - Prioritize I/Os based on the LSM persistent needs (e.g., WAL, L0 and lower levels move from hot to cold)
RocksDB + liblightnvm: Challenges

• sstables and write ahead logging
  - Fit sstables and WAL to block sizes in L0 and further level (merges + compactions)
  - No need for GC on SSD side - RocksDB merging acts as GC
  - Keep block metadata to reconstruct sstable in case of host crash

• Other Metadata
  - Keep superblock metadata and other data structures on block storage to easily recover the database
Rocksdb: Match flash block size

- SSTables and WAL are matched to the size of the physical blocks.
- Writes are issued in parallel across multiple blocks (and preferably across multiple LUNs).
- In the case a write fails (very rare), a new pblk is added and the data, for the broken pblk, is rewritten to the new block.
- Physical blocks associated is managed by MANIFEST file.
RocksDB: Block Metadata

Flash block (vblk)

First Valid Page
- Block Metadata
- RocksDB Data
- OOB Area

Intermediate Page
- RocksDB Data
- OOB Area

Last Page
- RocksDB Data
- Block Metadata
- OOB Area

First Page
- Filename (RocksDB GID)

OOB
- Page State (Fully written, partially written)
- # Valid Bytes

Last Page
- CRC
RocksDB: LightNVM Architecture

- LSM is the FTL
  - Tree nodes (files) control data placement on physical flash
  - sstables and WAL are on Open-Channel SSD - rest is in posix file-system
  - Garbage collection takes place during compaction

- LightNVM manages flash
  - liblightnvm takes care of flash block provisioning; RocksDB works directly with PPAs
  - Media manager handles wear-levelling (get/put block)
LightNVM and Open-Channel SSDs Status
LightNVM Subsystem: Status

• LightNVM available from Linux kernel 4.4
• Pluggable Architecture
  - Media Managers
    - General Media Manager
    - CNEX FTL Media Manager (Full FTL exposed as a block device)
  - General Media Manager targets
    - pblk – Expose PPA SSDs as a block device (mapping table on host)
    - rrpc – Expose Hybrid PPA SSDs as a block device (mapping table both on host and device)
    - direct flash (to be folded into liblightnvm)
• Supported device drivers:
  - NVMe support for devices that implements the LightNVM specification
  - null_blk for performance testing
• Hardware available
  • CNEX Labs Westlake SDK
LightNVM Subsystem: Ongoing Work

• Linux kernel 4.7-4.8
  - Upstream pblk target
  - Sysfs integration
  - Persistent block state management

• Add support for LightNVM in nvme-cli

• Application-driven Storage
  - liblightnvm
    • Integrate with sysfs integration
    • Standardize data transfer API
  - RocksDB
    • Move RocksDB DFlash storage backend logic to liblightnvm
    • Implement efficient page caching
    • Improve I/O submission (ioctl + libaio)
    • Exploit device parallelism within RockDB’s LSM
LightNVM
Open-Channel SSDs

• LightNVM: http://lightnvm.io
• Interface Specification: http://goo.gl/BYTjLI
• Documentation: http://openchannelssd.readthedocs.org
Tools and Libraries

LightNVM Subsystem Development
https://github.com/OpenChannelSSD/linux

lnvm – Administrate your Open-Channel SSDs
https://github.com/OpenChannelSSD/lnvm

liblightnvm – Integrate lightnvm into your application
https://github.com/OpenChannelSSD/liblightnvm

Test tools
https://github.com/OpenChannelSSD/lightnvm-hw

QEMU NVMe with Open-Channel SSD Support
https://github.com/OpenChannelSSD/qemu-nvme