Linux Kernel Extensions for Open-Channel SSDs

Matias Bjørling
Member of Technical Staff
Dealing with flash chip constrains is a necessity

No way around the Flash Translation Layer (FTL)

Embedded FTLs enabled wide SSD adoption - esp. for Client computing:

Client: single host, single SSD, low I/O efficiency, wide variety of applications

Server systems have a much different profile:

Server: multi-host, multi-SSD, high I/O efficiency, limited # of applications
Embedded FTL’s introduce **significant limitations for Server compute**:  

- Hardwire design decisions about data placement, over-provisioning, scheduling, garbage collection, and wear leveling.  
- Designed on more or less explicit assumptions about the application workload.  
- Introduces redundancies, missed optimizations, and underutilization of resources.
Limited number of SSDs in the market with embedded FTLs for specific:

- Workloads (e.g., 90% reads)
- Applications (e.g., SQL Server, Key-value stores)

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?
Open-Channel SSDs share control responsibilities with the Host in order to implement and maintain features that typical SSDs implement strictly in the device firmware.

**Device information:**
- SSD offload engines & responsibilities
- SSD geometry
  - NAND media
  - Channels, timings, etc.
  - Bad blocks list
  - ECC

**Host gains:**
- Data placement
- I/O scheduling
- Over-provisioning
- Garbage collection
- Wear-leveling

* CNEX WestLake is a commercial class Open-Channel SSD controller ASIC

Enables Quality of Service
Open-Channel SSD: Architecture

**Software:**
- Block Target
- Vendor-Specific Target
- Direct Flash Target
- Managed Geometry
- Block Manager (Generic, Vendor-specific, ...)
- Raw NAND Geometry

**Kernel**
- File-System
- Key-Value/Object/FS/Block/etc.

**User-space**

**Hardware:**
- Open-Channel SSDs (NVMe, PCI-e, RapidIO, ...)
  - Block Copy Engine
  - XOR Engine
  - Metadata State Mgmt.
  - ECC Engine
  - Bad Block State Mgmt.
  - GC Engine
  - Error Handling
  - Etc.

**LightNVM Framework**
Open-Channel SSD: Configurability

1. Target across SSDs (with different vendor SSDs)
2. Global Garbage Collection
3. Single Address Space

Target(s)
Manage multiple SSDs

Generic Block Manager
Vendor Block Manager

Vendor A SSD
Vendor B SSD
Vendor C SSD

BMs expose a generic interface
Same target (FTL) across SSDs. Same behavior over many SSDs
Open-Channel SSD: More Benefits

- Over-provisioning can be greatly reduced,
  - E.g., 20% lower cost for the same performance
- SSD steady state can be considerably improved
- Predictable latency
  - Reduce I/O outliers significantly
## Open-Channel SSD: Host Overhead

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Native Latency(us)</th>
<th>LightNVM Latency(us)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read</td>
<td>Write</td>
<td>Read</td>
</tr>
<tr>
<td>Kernel and fio overhead</td>
<td>Submission and completion (4K)</td>
<td>1.18</td>
<td>1.21</td>
</tr>
<tr>
<td>Completion time for devices</td>
<td>High-performance SSD</td>
<td>10us (2%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Null NVMe hardware device</td>
<td>35us (0.07%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Common SSD</td>
<td>100us (0.002%)</td>
<td></td>
</tr>
</tbody>
</table>

SSD: ECC, Translation & Bad block table metadata offloaded to device.

Low overhead negligible to hardware overhead
0.16us on reads and 0.23us on writes
Software-defined storage solutions:

- Storage is managed centrally across multiple Open-Channel SSDs
  - Petabytes of flash
- Open-Channel SSDs are “software programmable”
  - Versus “Hardware/Firmware configurable”
  - Applications that have specific and/or evolving needs
- Applications can define their own FTLs based on their workload
- FTL optimizations that change over time
- New target (e.g., Customer-specific) implementations
- Different Application personalities:
  - Transactions, archiving, video processing, backup, vm-aware storage

Open-Channel SSDs -> Application-driven Storage
1. How do we support applications that benefit from custom FTLs?

2. What is the role of the OS in this architecture?

3. How can we hide NAND media complexity from the application (and the OS)?

- Application-Driven Storage
  - Generic interface for programmable SSDs to abstract the hardware
  - Avoid multiple layers of translation
  - Leverage optimization opportunities
  - Minimize overhead when manipulating persistent data
  - Make better decisions regarding latency, resource utilization, and data movement (compared to the best-effort techniques seen today)
Open Channel SSDs: RocksDB Use-case

Flash Memory Summit 2015
Santa Clara, CA

Prototype in progress

Kernel
- Direct Flash Target
- Managed Geometry
- Block Manager
- Raw NAND Geometry

Open-Channel SSDs
- Bad Block State Mgmt.
- Metadata State Mgmt.
- XOR Engine
- ECC Engine

User-space

RocksDB
- Writes to flash aligned log
- Streams is separate into flash blocks
- Vectored reads is optimized using flash lun busyness
- Rocks implements device FTL (GC & Placement)

liblightnvm

DirectFlash

Flash Memory Summit 2015
Santa Clara, CA
LightNVM: Linux kernel support for Open-Channel SSDs
- Open, flexible, extensible, and scalable layer for Open-Channel SSDs for the Linux kernel
- Development: https://github.com/OpenChannelSSD
- Supports multiple block managers and targets
Pluggable Architecture
  • Block Managers – Generic, Vendor specific, etc
  • Targets – Block, Direct Flash

Supported drivers:
  • NVMe, Null driver (FTL performance testing and debugging)

Push into the Linux kernel. v7 posted to LKML (7/7-15).
Users may extend, contribute, and develop new targets for their own use-cases.
Direct integration with RocksDB under development.
Thank you

Development: https://github.com/OpenChannelSSD/

Interface Specification: http://goo.gl/BYTjLI

Contact: Matias Bjørling
matias@cnexlabs.com