Open-Channel SSDs

Matias Bjørling

LinuxCon North America 2015
Solid State Drives

High Throughput + Low Latency

Parallelism + Controller
Why Open-Channel SSDs

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
Why Open-Channel SSDs

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.
Why Open-Channel SSDs

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 SSD: Overview

• 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 manages:
• Data placement
• I/O Scheduling
• Over-Provisioning
• Garbage Collection
• Wear-leveling

This architecture enables Quality of Service for SSDs.
Open-Channel SSD: Architecture

Targets
- Exposes physical media to user-space

Block Managers
- Manages physical SSD characteristics

Open-Channel SSD
- Responsibility
- Offload engines
Open-Channel SSD: Configurability

1. Target across SSDs
2. Global Garbage Collection
3. Single Address Space

BM devices expose a generic interface.

SSD Vendor-agnostic
Open-Channel SSD: Example

- 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></td>
<td>Read</td>
<td>Write</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></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Null NVMe hardware device</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Common SSD</td>
<td></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
Open-Channel SSD: Where are they useful?

Software-defined storage solutions:
- Manage storage centrally across multiple SSDs
  • Petabytes of flash
- Open-Channel SSDs are “software programmable”
  • Versus “Hardware/Firmware configurable”
- Applications can define their own FTLs based on their workload
- FTL optimizations that change over time
- Multi-tenancy environments

Open-Channel SSDs -> Application-driven Storage
Open Channel SSDs: Application-Driven Storage

- Generic interface for programmable SSDs to abstract the hardware
- Avoid multiple layers of translation
- Minimize overhead when manipulating persistent data
- Make informed decisions regarding latency, resource utilization, and data movement (compared to the best-effort techniques today)

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)?
Open Channel SSDs: RocksDB Use-case

Prototype in progress

- 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)

Talk to Javier Gonzalez if you want to know more
Kernel Support

• **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](https://github.com/OpenChannelSSD)

• Supports multiple block managers and targets
LightNVM Status

• 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@cnexlabs.com