Open-Channel Solid State Drives

Matias Bjørling

2015/03/12

Vault
Solid State Drives

• Thousand of IOPS and low latency (<1ms)
• Hardware continues to improve
  – Parallel architecture
  – Larger flash chips

• Replaceable for traditional harddrives
• Embedded software maintains complexity
Embedded FTLs: No Future

• Dealing with flash chip constraints is a necessity
  – No way around some form of FTL
• Embedded FTLs were great to guarantee adoption, but have critical limitations:
  – Hardwire design decisions about data placement, overprovisioning, scheduling, garbage collection and wear leveling
  – Based on more or less explicit assumptions about the application workload
  – Resulting in redundancies, missed optimizations and underutilization of resources
Market Specific FTLs

• SSDs on the market with embedded FTLs targeted at specific workloads (90% reads) or applications (SQL Server, KV store)
• FTL is no longer in the way of a given application
• What if the workload or application changes?
• What about the other workloads or applications?
Open-Channel SSDs

Physical flash exposed to the host (Read, write, erase)

Host
• Data placement
• IO Scheduling
• Over-provisioning
• Garbage collection
• Wear levelling
Where are Open-Channel SSDs useful?

• Data centers with multi-tenancy environments
• Software-defined SSDs
  – Managed storage centrally across open-channel SSDs.
  – NAND flash shared at fine-granularity
• Applications that have specific needs can be serviced by a FTL tailored to their needs (Application-driven FTLs).
New Logical Abstractions

• How is flash exposed to the host?
  – Traditional Flash Translation Layer
    • Both metadata and data are managed by the host
  – New interfaces
    • LUNs (The parallel unit of SSDs)
    • Key-value database (e.g. LevelDB and RocksDB)
    • Object-store (OSSD)
    • Application-driven (New research area)
    • File-system (NVMFS)
    • Hybrid FTL (Traditional FTL is expensive, offload metadata consistency to device)
  – Manage multiple devices under a single address space
    • Including garbage collection (Global FTL)
What should the host know?

• SSD Geometry
  – NAND idiosyncrasies
  – Die geometry (Blocks & Pages)
  – Channels, Timings, Etc.
  – Bad blocks
  – Error-Correcting Codes (ECC)

• Features and Responsibilities
Kernel Integration

• Generic core features for flash-based SSD management such as:
  – List of free and in-use blocks, handling of flash characteristics, and global state.
• Targets that expose a logical address space, possibly tailored for the needs of a class of applications (e.g., key-value stores or file systems)
Architecture
Hybrid Target

• Host-side Translation table and reverse mapping table (for GC) in host
• Device maintains metadata consistency
  – Offloads metadata overhead at the cost of disk also maintaining translation table
• Sequential mapping of pages within a block
• Cost-based garbage collection
• Inflight tracking
  – Guarantee atomicity of writes
# Hybrid Target per Request

<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</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>

System: i7-3960K, 32GB 1600Mhz – 4K IOs

Low overhead compared to hardware overhead
0.16us on reads and 0.23us on writes
### Key-value Target

#### 1 MB Writes

<table>
<thead>
<tr>
<th>Metric</th>
<th>Native</th>
<th>LightNVM-Page</th>
<th>LightNVM Key-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>29GB/s</td>
<td>28.1GB/s</td>
<td>44.7GB/s</td>
</tr>
<tr>
<td>Latency</td>
<td>32.04μs</td>
<td>33.02μs</td>
<td>21.20μs</td>
</tr>
<tr>
<td>Kernel Time</td>
<td>66.03%</td>
<td>67.20%</td>
<td>50.01%</td>
</tr>
</tbody>
</table>

**Kernel time overhead 30% serving 1MB writes.**

Opportunities for application-driven FTLs
Industry Vendors

• MemBlaze
  – Available hardware

• PMC Sierra
  – Builds support in user-space

• IIT Madras
  – Builds HW using RapidIO SRIO

• Stealth startups and others
  – Storage Arrays
  – Applications
Source Layout

• Open-channel SSD initialization
  – /block/blk-nvm.c – Initialization/Registration
  – /include/linux/blkdev.h – Common NVM structures

• Targets
  – /drivers/nvm
    • Round-robin page-based with cost-based GC FTL (rrpc)
Open-channel SSD initialization

• Device drivers register the block device as an Open-channel SSD device
  – Device is queried for geometry and configured
• blk_nvm_register(struct request_request *,
  struct blk_nvm_ops *)
• struct blk_nvm_ops
  – identify
  – get_features
  – set responsibility
  – get l2p
  – erase_block
Block Layer Structures
/includes/linux/blkldev.h

- struct nvm_dev (1 per device)
  - Describes the characteristics of the device
- struct nvm_lun (N per device, 8-16)
  - Information about luns and status of flash blocks
- struct nvm_block (M per device, 1024+)
  - Information about each flash block state
Target Interface

• Uses the interface provided by the block layer
  – blk_nvm_get_blk(struct nvm_lun *)
  – blk_nvm_put_blk(struct nvm_block *)

• Target reserves flash blocks and writes data

• Reads can either be resolved by device or physical LBAs

• Implements target_type interface
  – make_request, prep_rq/unprep_rq, init/exit
RRPC Flow

1. Read IO from /dev/nvm0
   - submit_bio

2. Direct to /dev/nvme0n1
   - blk_mq/sq_make_request
   - prep_rq
   - queue_rq

3. Setup target context
   - submit_bio
   - blk_mq_end_request

4. Setup physical addresses in request
   - unprep_rq

5. Unlock logical address
   - bio complete
Future

• Integrate with
  – Ceph
  – RocksDB
  – Percona
  – Openstack
  – And many others

• Kernel upstream

• Finalize interface specification together with vendors

https://github.com/OpenChannelSSD
Thanks for Listening

https://github.com/OpenChannelSSD