Linux Kernel Abstractions for Open-Channel SSDs

Matias Bjørling

Javier González, Jesper Madsen, and Philippe Bonnet

2015/03/01
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?
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
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).
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
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 (DirectFS)
    • Hybrid FTL (Traditional FTL is expensive, offload metadata consistency to device)
  – Manage multiple devices under a single address space
    • Including garbage collection (Global FTL)
Introducing LightNVM

• Open, Extensible, and Scalable Layer for Open-Channel SSDs in the Linux kernel

• 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></td>
<td></td>
<td>1.34 (+0.16)</td>
<td>1.44 (+0.23)</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>

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

Low overhead compared to hardware overhead
0.16us on reads and 0.23us on writes
### 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 (NVMKV, ...)

![Graph showing Throughput (GB/s)]
Performance Guarantees

Configuration
4K IO Request, Buffered, flash page size 32K, Write timing 1200us

 Writes can be buffered, Reads take full time
Conclusion

• Enable data placement and garbage collection on host
• Enable global garbage collection
• Enable application-driven FTLs
• Industry traction
  – MemBlaze, IIT Madras, Stealth startups, and others
• Soon to be submitted to the Linux kernel
• Source and interface specification available at
  https://github.com/OpenChannelSSD