Linux Kernel Abstractions for Open-Channel Solid State Drives

Matias Bjørling Jesper Madsen Javier González Philippe Bonnet
IT University of Copenhagen

Abstract

The widespread adoption of SSDs is due to the combination of two factors: (i) superior performance, and (ii) identical block device interface. However, the combination of these two factors is becoming a problem as SSD performance keeps on improving, revealing performance bottlenecks throughout the I/O stack. To tackle this issue, an increasingly popular solution for cloud providers is to consider open-channel SSDs, i.e., custom firmware SSDs that expose the physical storage space directly to the host and its operating system. In this paper, we report on the design and implementation of an open-channel SSD management layer for Linux. We discuss how our open, extensible and scalable design efficiently supports various data-intensive applications (e.g., file systems, database systems, or key-value stores), and thus constitutes a new platform for developing software-defined storage. We provide a performance evaluation of our current implementation on top of NVMe devices and show the low overhead of our design.

1. Introduction

Modern SSDs perform complex decisions about mapping and scheduling, independently from the applications and operating systems that write and read data. In particular, the Flash Translation Layer embedded in flash-based SSDs guesses I/O locality in order to avoid contention, optimize its mapping, and reduce garbage collection overhead. Conversely, applications map their data structures onto the logical address space, exposed by the SSDs without robust performance models to motivate their decisions. This legacy double-blind design leads to waste of resources, unpredictable performance and bottlenecks [6, 1, 5, 3].

There is a need for a tighter form of collaboration between data-intensive applications, operating system and SSD to reconcile the complexity of storage management with the high-performance goals of modern applications. Examples of tight integration can be found in the context of database systems, with database appliances (e.g., Exascale or Netezza) or custom-firmware SSD for SQL Server [4]. But, how about applications running on warehouse-scale computers? How can warehouse-scale SSDs implement optimized Flash Translation algorithms to cater to the (ever changing) idiosyncrasies of a given application workload? A solution is to decouple SSD management from physical storage. More concretely, a solution is to consider open-channel SSDs, i.e., custom firmware SSDs that expose the physical storage space directly [6, 7]. The problem is then how to organize SSD management, in a way that efficiently supports high-performance, data-intensive applications. This is the problem we tackle in this paper.

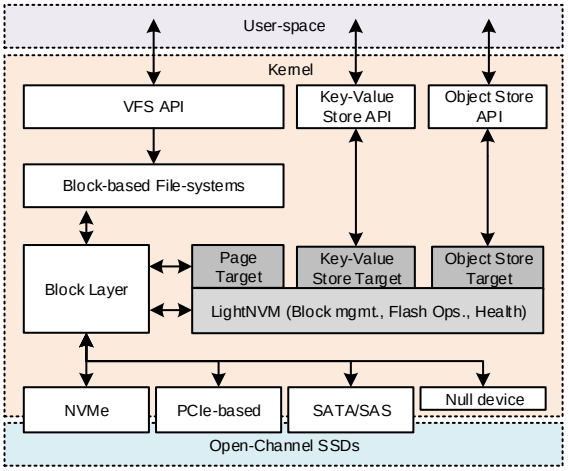

Figure 1: LightNVM Architecture.

2. Design

We propose LigthNVM, a new Linux kernel framework that extends the block layer as shown in Figure 1. Originally conceived as an evaluation platform [2], LightNVM provides:

- Generic core features for flash-based SSD management such as: list of free and in-use blocks, generic function for handling flash characteristics, and information about the global state of the device;
- *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), and rely on the generic core features to organize its mapping onto the underlying physical address space. Application-specific parameters include for example data placement strategies (how many flash chips to write to at a time), choice of garbage collector, and wear-leveling policies. Each target can then be specifically optimized to a given workload based on (i) its characteristics (with possibly some form of deduplication, compression or storage security), and (ii) the performance it expects from the underlying storage device. Targets do not handle work such as ECC, or more advance durability techniques, such as data scrubbing, read and write disturb, and similar optimizations that have to be implemented within the open-channel SSD to extend the lifetime of a device.

The internal and external I/O request flow management is handled through the native Linux block layer. This design simplifies the code necessary to control both external and internal requests. Underneath, the device drivers register themselves as open-channel SSDs and indicate the features exported by their firmware, such as bad block tables or powercap features.

Component	Description	Native Latency(us)		LightNVM Latency(us)	
		Read	Write	Read	Write
Kernel and fio overhead	Submission and completion to/from device driver	1.18	1.21	1.34	1.44
Time to complete request by HW	Null block device	0.060			
	Null NVMe hardware device	35			
	OpenSSD with LightNVM firmware	350			

Table 1: Time spent at completing a 4K IO request within the block layer, LightNVM (page-level target), NVMe and OpenSSD hardware.

Metric	Native	LightNVM-Page	Key-value
Throughput	29GB/s	28.1GB/s	44.7GB/s
Latency	32.04μs	33.02μs	21.20μs
Kernel Time %	66.03%	67.20%	50.01%

Table 2: Table summarizing the performance differences observed between the page-based block device interface, LightNVM page target and LightNVM key-value target issuing 1MB writes.

3. Evaluation

We evaluate the overhead of LightNVM using a NVMe null device with a constant write speed of 380MB/s. This experiment considers the CPU overhead of LightNVM, using a fully-associative logical to physical page-level target. The results are shown in Figure 2 At 4KB IO request sizes, LightNVM increases the CPU load by 6%, declining to 1.3% for 512KB IO requests.

Table 1 shows an analysis of where time is spent within the kernel, LightNVM (still with the page-level target), and three devices for a single request. With an SSD with embedded FTL (denoted native on the table), the host requires 1.18μs for reads and 1.21μs for writes for 4K IO access. In contrast, LightNVM takes 1.34μs for reads, and 1.44μs for writes. Put differently, for high-performance NVMe devices, where response time is 35μs, the overhead introduced by LightNVM accounts to 0.66% of the overall response time for reads and 0.10% for writes.

The overhead is largely contributed by locks around in-flight tracking and allocation of flash pages and blocks. This can be improved by using multi-queue devices (Both the OpenSSD device and the NVMe null device is single-queue) and per-cpu data structures that preallocate flash pages and blocks.

In order to show the benefits of using application-specific targets, we implemented a simple block-based key-value store (KV) with LightNVM managing the storage. Lookups are made by translating a given key into the corresponding physical block. Inserts are handled by (i) allocating a new block from the address space manager; (ii) associating a key to it; and finally (iii) writing the payload to the physical block. Since KV store is intended to be used directly by applications, we create a user-space API to handle requests and notifications.

Table 2 shows the the performance differences observed between the native page-based block device interface, LightNVM block device interface using page-based mapping, and key-value target where each value is being assigned a unique physical block. The experiments are performed using a null block device as backend device. With 1MB sequential writes, latency is reduced 14.29% (from 33.02μs to 22.20μs). This has a major impact in throughput, which improves 55.8% (from 28.1GB/s to 44.7GB/s). It also reduces kernel CPU-usage by 17.2%. These experiments illustrate the potential performance benefits of application-specific software-defined storage.

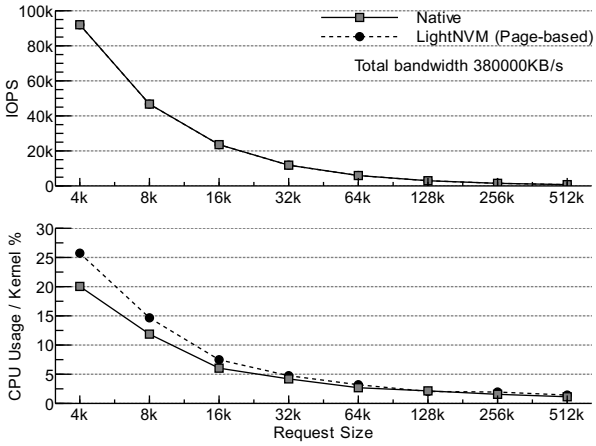

Figure 2: IOPS and CPU kernel usage of 4KB to 512KB random write IO sizes with the null NVMe hardware device (Max 380MB/s).

References

[1] BJØRLING, M., AXBOE, J., NELLANS, D., AND BONNET, P. Linux block IO: Introducing multi-queue SSD access on multi-core systems. In *Proceedings of the 6th International Systems and Storage Conference* (New York, NY, USA, 2013), SYSTOR '13, ACM, pp. 22:1–22:10.

[2] BJØRLING, M., MADSEN, J., BONNET, P., ZUCK, A., BANDIC, Z., AND WANG, Q. Lightnvm: Lightning fast evaluation platform for non-volatile memories.

[3] CAULFIELD, A. M., DE, A., COBURN, J., MOLLOW, T. I., GUPTA, R. K., AND SWANSON, S. Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories. *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture* (Dec. 2010), 385–395.

[4] DO, J., KEE, Y., PATEL, J., AND PARK, C. Query processing on smart SSDs: opportunities and challenges. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (2013).

[5] OUYANG, J., LIN, S., JIANG, S., AND HOU, Z. SDF: Software-defined flash for web-scale internet storage systems. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems* (2014).

[6] OUYANG, X., NELLANS, D., WIPFEL, R., FLYNN, D., AND PANDA, D. Beyond block I/O: Rethinking traditional storage primitives. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on* (2011), IEEE, pp. 301–311.

[7] SAXENA, M., ZHANG, Y., SWIFT, M. M., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Getting Real: Lessons in Transitioning Research Simulations into Hardware Systems. *FAST* (2013), 215–228.