Multi-Tenant I/O Isolation with Open-Channel SSDs

Javier González   Matias Bjørling
{javier, matias}@cnexlabs.com

Abstract

Multi-tenancy environments rely at their core on fair resource distribution by the host system. For storage, this translates into the host managing throughput and latency needs for each tenant. In Solid State Drives (SSDs), throughput expectations are mostly met thanks to more powerful controllers and faster media. Latency guarantees, on the other hand, remain a challenge due to the unpredictability intrinsic to the design of current SSDs and the narrow interfaces available to the host. We propose the use of Open-Channel SSDs to partition the device into its physical units, accessible in parallel by the host, and mapped onto tenants via traditional block devices in order to provide I/O isolation. We describe our architecture and provide an experimental evaluation to show the benefits.

1. Introduction

Solid State Drives (SSDs) are the dominant form of secondary storage in web-scale data centers, hyperscale infrastructures and shared storage arrays. While their internal parallelism has been enough to outperform spinning disks, SSDs have a widely accepted and well documented Achilles' heel: unpredictable, tail latencies [2,3]. This is mainly caused by two factors: (i) the need for performing background garbage collection (GC), and more importantly, (ii) the latency of a given I/O being determined by the order in which it is scheduled. Programming operations (write and erase) exhibit an order of magnitude higher base latencies than read operations. Therefore, reads are highly disturbed by I/O collisions on the physical media. This problem only aggravates in multi-tenancy setups, where a single SSD is shared among several tenants, each implementing its own scheduling on a different workload.

A straightforward solution is to use dedicated devices for each tenant. However, the increasing density of non-volatile memories and the associated cost makes this an expensive solution. Moreover, the increasing parallelism provided by more powerful SSD-controllers makes this solution also inefficient. Indeed, next generation SSDs support +1M IOPS in a single device. Another solution is to partition the device into several namespaces and dedicate them to each tenant. This tackles the cost issue and allows to take advantage of the device parallelism. However, namespaces, as defined in current protocols such as SCSI and more recently NVMe, are not guaranteed to be mapped onto physical parallel units in the device. This means that I/Os stemming from different namespaces could be addressed to the same physical die, therefore colliding when they reach the media. A different way of partitioning the SSD is by logically tagging I/O streams [3] and letting the device doing the physical separation. The problem is that current multi-stream solutions are limited in

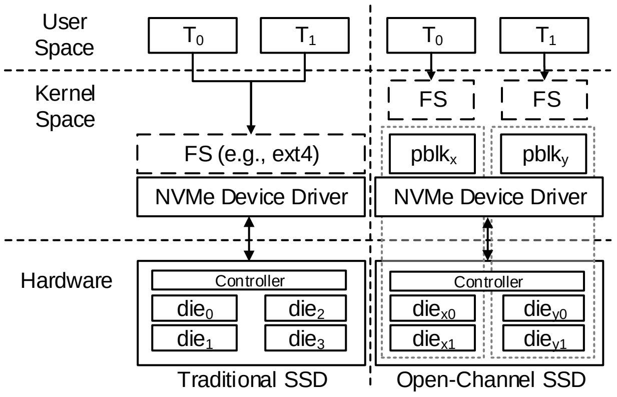

Figure 1: Physical Isolation Archicture

the number of streams exposed to the host. More importantly, the streams are static and do not resemble the device's parallel units either. Partitioning is then best-effort and is conditioned by the device's embedded data placement strategy.

In general, the problem is that current abstractions presented to the host by the OS do not capture the parallelism intrinsic in the SSD, thus making it impossible to guarantee that applications will not disturb each other on a multi-tenant environment. In this paper, we describe how I/O isolation can be achieved with Open-Channel SSDs without modifying current applications. We describe our architecture and report evaluation results.

2. Architecture

Our goal is to provide inter-application I/O isolation in multi-tenancy environments, where tenants can be database instances, VMs, containers, etc. To achieve this, we need to guarantee that each tenant is only able to issue I/O to an exclusive and restricted number of physical parallel units in the device (LUNs). By doing so, I/Os that emanate from different tenants and are addressed to the same device would not collide when reaching the physical media, thus eliminating the unpredictability added by neighbor tenant I/Os. The price is (i) dedicating parts of the device's parallelism to specific tenants and (ii) limiting the bandwidth of each tenant to the number of LUNs that it uses. Our hypothesis is then that by physically separating data belonging to different tenants on the device, inter-tenant indeterminism is mitigated. Being the observable result lower tail latencies on mixed workloads, specially on environments where some tenants are write heavy and others are read heavy. The underlying assumption is that SSDs have enough intrinsic parallelism to provide the bandwidth applications expect even when their LUNs are dedicated. For applications with high bandwidth needs, the assumption is that throughput is worth trading for inter-application I/O determinism on a multi-tenancy setup. Note that we focus on eliminating added I/O indeterminism across tenants sharing the same physical storage device; how to optimize I/O scheduling withing a tenant is a different issue, not addressed in this paper.

For our system design, we assume the use of an Open-Channel SSD (OCSSD), i.e., an SSD that exposes the internal parallelism of the controller and the non-volatile memory attached to it to the host. LightNVM administers the OCSSD in a Linux system. The Flash Translation Layer (FTL) is then partly managed by the host and partly by the device. The physical block device LightNVM target, or *pblk*, implements host-side generic FTL functionality: data placement, I/O scheduling and GC; and exposes a traditional block device to user space. Media-centric metadata, error handling, scrubbing and wear-leveling are handled by the controller [1]. In order to partition the OCSSD, we extend LightNVM with multi-target support, which enables several FTL targets (e.g., *pblk*) to be instantiated on a single device. These changes have been merged in the Linux kernel 4.10 and *pblk* is in the process of being upstreamed [1].

Using *pblk* and LightNVM's multi-target support as building blocks, we propose that I/O isolation on a single SSD can be achieved by tenants using dedicated *pblk* instances as their storage backends. Each *pblk* instance is created on top of a collection of physical LUNs, which are exclusively owned by the instance. This vision of a partition differs from the traditional one in that partitions on a traditional block device do not map physical LUNs. A *pblk* instance is then characterized by three metrics: (i) its **capacity**, which is given by the number of blocks and planes within a LUN and the block page size; (ii) its top **bandwidth**, which is naturally rate-limited by the read/write throughput capabilities of a physical LUN; and (iii) its dedicated **I/O channel**, which is guaranteed by the fact that LUNs are owned exclusively. As a result, each tenant implements its own data placement, GC and I/O scheduling independently on different, isolated parallel partitions within the device. Disturbances across tenants from a media perspective are therefore eliminated. This architecture is depicted in Figure 1.

3. Experimental Evaluation

The purpose of our evaluation is to test the hypothesis that partitioning the device can help minimizing tail latencies in tenants with mixed workloads. To do so, we design an experiment where we concurrently issue I/O from 2, 4, and 8 tenants. In each case, one of the tenants issues 4K random reads and the rest issue writes. All workloads have queue depth 1 and use the whole bandwidth of the used block device. We measure latency for the reader. Figure 2 contains the result for this experiment on a CNEX Labs Open-Channel SSD. On the left side, we see the latencies when a single partition is used across all the 128 LUNs in the device and all tenants share it. In this case, we observe how reads are being disturbed by concurrent writes. Note that when no writes are being issued, latencies are similar to the ones

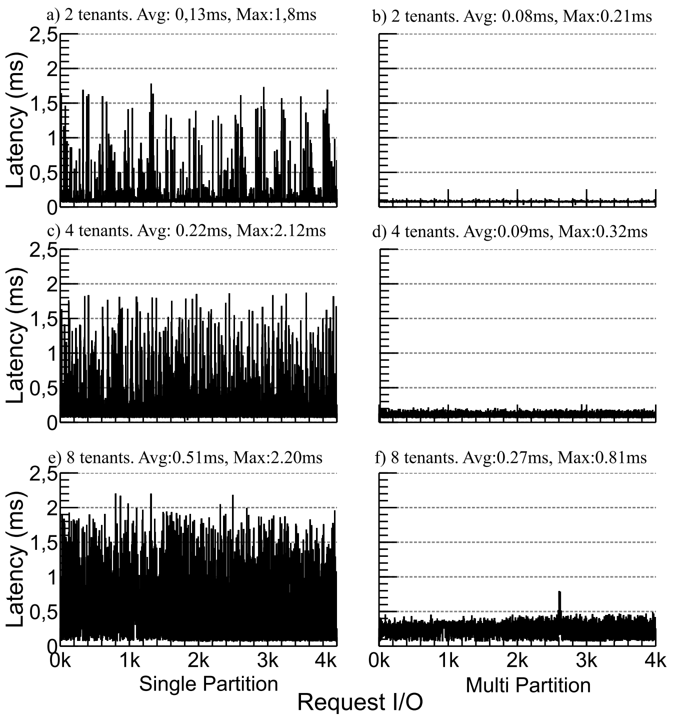

Figure 2: Random read latency comparison (bs=4k, qd=1) for an OC-SSD using the whole bandwidth of the device on a single partition (left side) and using multiple partitions (right side).

in sub-figure *b*. On the right size, we see the latencies when each tenant uses a dedicated partition. Here, we observe how latencies are still affected, but to a much lesser extent. This disturbances come from the fact that the number of inflight I/Os is proportional to the number *pblk* instances, but the device submission and completion queues are still shared. As a reference, we provide the results for the same experiment run on a traditional enterprise-class NVMe SSD: 2 tenants. *Avg:0.30ms, Max:22.92ms*; 4 tenants. *Avg:1.85ms, Max:19.76ms*; 8 tenants. *Avg:2.02ms, Max:29,97ms*.

4. Conclusion

While the extent of the benefits is clearly dependent on the actual workload, we show that using LUN partitions on an Open-Channel SSD is a viable approach to provide I/O isolation on multi-tenancy environments. Which workloads and applications benefit the most is a topic that we will tackle in future research.

References

[1] BJØRLING, M. Open-Channel SSD NVMe Specification 1.3.

[2] HAO, M., SOUNDARARAJAN, G., KENCHAMMANA-HOSEKOTE, D., CHIEN, A. A., AND GUNAWI, H. S. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (2016), pp. 263–276.

[3] KANG, J.-U., HYUN, J., MAENG, H., AND CHO, S. The Multi-Streamed Solid-State Drive. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)* (Philadelphia, PA, June 2014), USENIX Association.

[1]Development: `https://github.com/OpenChannelSSD/linux`. *pblk*-specific documentation: /Documentation/lightnvm/pblk.txt