XRP: In-Kernel Storage Functions with eBPF

Yuhong Zhong[1], Haoyu Li[1], Yu Jian Wu[1], Ioannis Zarkadas[1], Jeffrey Tao[1], Evan Mesterhazy[1], Michael Makris[1], Junfeng Yang[1], Amy Tai[2], Ryan Stutsman[3], and Asaf Cidon[1]

[1]Columbia University, [2]Google, [3]University of Utah

With the rise of new high performance memory technologies, such as 3D XPoint and low latency NAND, new NVMe storage devices can now achieve up to 7 GB/s bandwidth and latencies as low as 3 µs [2–4]. At such high performance, the kernel storage stack becomes a major source of overhead impeding both application-observed latency and IOPS. For the latest 3D XPoint devices, the kernel's storage stack *doubles* the I/O latency, and it incurs an even greater overhead for throughput. As storage devices become even faster, the kernel's relative overhead is poised to worsen.

Existing approaches to tackle this problem tend to be radical, requiring intrusive application-level changes or new hardware. Complete kernel bypass through libraries such as SPDK [10] allows applications to directly access underlying devices, but such libraries also force applications to implement their own file systems, to forgo isolation and safety, and to poll for I/O completion which wastes CPU cycles when I/O utilization is low. Others have shown that applications using SPDK suffer from high average and tail latencies and severely reduced throughput when the schedulable thread count exceeds the number of available cores [8].

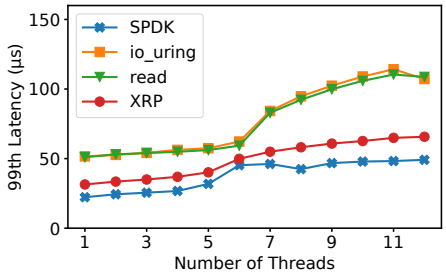
In contrast to these approaches, we seek a readily-deployable mechanism that can provide fast access to emerging fast storage devices that requires no specialized hardware and no significant changes to the application while working with existing kernels and file systems. To this end, we rely on BPF (Berkeley Packet Filter [9]) which lets applications offload simple functions to the Linux kernel [1]. Similar to kernel bypass, by embedding application-logic deep in the kernel stack, BPF can eliminate overheads associated with kernel-user crossings and the associated context switches. Unlike kernel bypass, BPF is an OS-supported mechanism that ensures isolation, does not lead to low utilization due to busy-waiting, and allows a large number of threads or processes to share the same core, leading to better overall utilization.

The support of BPF in the Linux kernel makes it an attractive interface for allowing applications to speed up storage I/O. However, using BPF to speed up storage introduces several unique challenges. Unlike existing packet filtering and tracing use cases, where each BPF function can operate in a self-contained manner on a particular packet or system trace — for example, network packet headers specify which flow they below to — a storage BPF function may need to synchronize with other concurrent application-level operations or require multiple function calls to traverse a large on-disk data structure, a workload pattern we call "resubmission" of I/Os. Unfortunately the state required for resubmission such as access-control information or metadata on how individual storage blocks fit in the larger data structure they belong to is not available at lower layers.

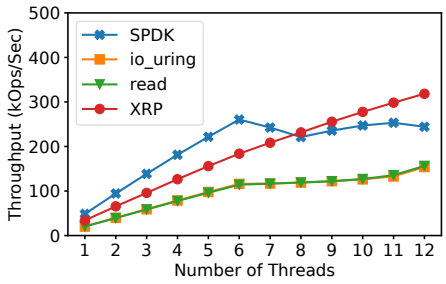
To tackle these challenges, we design and implement XRP (eXpress Resubmission Path) [11], a high-performance storage data path using Linux eBPF. XRP is inspired by XDP, the recent efficient Linux eBPF networking hook [6]. In order to maximize its performance benefit, XRP uses a hook in the NVMe driver's interrupt handler, thereby bypassing the kernel's block, file system and system call layers. This allows XRP to trigger BPF functions directly from the NVMe driver as each I/O completes, enabling quick resubmission of I/Os that traverse other blocks on the storage device.

The key challenge in XRP is that the low-level NVMe driver lacks the context that the higher levels provide. Those layers contain information such as who owns a block (file system layer), how to interpret the block's data, and how to traverse the on-disk data structure (application layer).

Our insight is that many storage-optimized data structures that power real-world databases [5, 7] – such as on-disk B-trees, log-structured merge trees, and log segments – are typically implemented on a small set of large files, and they are updated orders of magnitude less frequently than they are read. Hence, we exclusively focus XRP on operations contained within one file and on data structures that have a fixed layout on disk. Consequently, the NVMe driver only requires a minimal amount of the file system mapping state, which we term the *metadata digest*; this information is small enough that it can be passed from the file system to the NVMe driver so it can safely perform I/O resubmissions. This allows XRP to safely support some of the most popular on-disk data struc-



(a) 99th-percentile latency.



(b) Throughput.

Figure 1: Tail latency and throughput of XRP and SPDK against `read` and io_uring with BPF-KV with random key lookups.

tures.

We present a design and implementation of XRP on Linux, with support for ext4, which can easily be extended to other file systems. XRP enables the NVMe interrupt handler to resubmit storage I/Os based on user-defined BPF functions.

We augment two key-value stores with XRP: BPF-KV, a B^+-tree based key-value store that is custom-designed for supporting BPF functions, and WiredTiger's log-structured merge tree, which is used as one of MongoDB's storage engines [5]. With random 512 B object reads on BPF-KV with multiple threads using a B^+-tree that has six index levels on disk, XRP has 61%–120% higher throughput and 16%–41% lower p99 latency than `read()` (Figure 1). XRP also enables more efficient sharing of cores among applications than kernel bypass: it is able to provide 30% better throughput than SPDK with two threads sharing the same core. In addition, XRP is able to consistently improve WiredTiger's performance by up to 24% under YCSB.

We make the following contributions.

1. **New Datapath**. XRP is the first datapath that enables the use of BPF to offload storage functions to the kernel.
2. **Performance**. XRP improves the throughput of a B-tree lookup by up to 2.5× compared to normal system calls.
3. **Utilization**. XRP provides latencies that approach kernel bypass, but unlike kernel bypass, it allows cores to be efficiently shared by the same threads and processes.
4. **Extensibility**. XRP supports different storage use cases, including different data structures and storage operations (e.g., index traversals, range queries, aggregations).

References

[1] eBPF. https://ebpf.io/.

[2] Intel® Optane™ SSD DC P5800X Series. https://ark.intel.com/content/www/us/en/ark/products/201859/intel-optane-ssd-dc-p5800x-series-1-6tb-2-5in-pcie-x4-3d-xpoint.html.

[3] Toshiba memory introduces XL-FLASH storage class memory solution. https://business.kioxia.com/en-us/news/2019/memory-20190805-1.html.

[4] Ultra-Low Latency with Samsung Z-NAND SSD. https://www.samsung.com/semiconductor/global.semi.static/Ultra-Low_Latency_with_Samsung_Z-NAND_SSD-0.pdf.

[5] WiredTiger storage engine. https://docs.mongodb.com/manual/core/wiredtiger/.

[6] XDP. https://www.iovisor.org/technology/xdp.

[7] Siying Dong, Mark Callaghan, Leonidas Galanis, Dhruba Borthakur, Tony Savor, and Michael Strum. Optimizing space amplification in RocksDB. In *CIDR*, volume 3, page 3, 2017.

[8] Jaehyun Hwang, Midhul Vuppalapati, Simon Peter, and Rachit Agarwal. Rearchitecting linux storage stack for µs latency and high throughput. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 113–128. USENIX Association, July 2021.

[9] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *USENIX Winter 1993 Conference (USENIX Winter 1993 Conference)*, San Diego, CA, January 1993. USENIX Association.

[10] Ziye Yang, James R Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E Paul. SPDK: A development kit to build high performance storage applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 154–161. IEEE, 2017.

[11] Yuhong Zhong, Haoyu Li, Yu Jian Wu, Ioannis Zarkadas, Jeffrey Tao, Evan Mesterhazy, Michael Makris, Junfeng Yang, Amy Tai, Ryan Stutsman, and Asaf Cidon. XRP: In-Kernel storage functions with eBPF. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 375–393, Carlsbad, CA, July 2022. USENIX Association. https://www.usenix.org/system/files/osdi22-zhong_1.pdf.