Exploiting SSD-Arrays with Asynchronous I/O

TECHNISCHE UNIVERSITÄT DARMSTADT

Dwarakanandan B M
September 22, 2021

Original Topic

Userspace Network Stacks in Fast Networks:

As high speed networks such as Infiniband grow increasingly popular, the overhead of using traditional TCP/IP stacks become evident. Our goal was to evaluate and benchmark various userspace alternatives such as DPDK, mTCP, eRPC.

Our test-bed consisted of two Linux machines each with Mellanox ConnectX-5 network adapters capable of 100G networking on which a KVM guest was setup. SR-IOV virtualization was enabled on the adapter and it was exposed to the VM using PCI passthrough. We were able to setup DPDK 20.11.1 inside the VM with MLX5 Poll mode drivers.

A major caveat with the current MLX5 Poll mode drivers for Infiniband adapters is that they must be configured to use Ethernet as their link layer. Unfortunately we discovered that the switch interconnecting these nodes only supports Infiniband link layer. Since modifying the infrastructure within the time frame of this Lab was infeasible, we decided to change topics. This issue has been resolved now by inter-connecting the adapters using an Ethernet switch.

1 Topic

Flash storage has seen tremendous growth in the past few years, not only in capacity but also in performance metrics such as throughput and latency. Replacing the old SATA interface the advent of NVMe M.2 SSDs have popularized SSD-Arrays directly attached using PCIe lanes. This exposes the large scale parallelism inherent to flash storage, at which point the traditional Operating System's I/O stack becomes a performance bottleneck. Our goal is to evaluate the feasibility of Asynchronous I/O stacks to exploit this parallelism.

2 Research Question

Our main focus is on benchmarking the differences between traditional Synchronous I/O *(syncio)*, Async I/O using the Linux AIO Interface *(libaio)* and Async I/O using io_uring interface *(io_uring)*.

- Identifying optimal I/O workloads to fully exploit the bandwidth using the aforementioned libraries on parameters such as how many threads are needed at minimum, how many outstanding I/O operations are needed.
- Perform an in-depth comparison of (libaio) and (io_uring) on factors like efficiency of I/O request submission/completion, code complexity, etc.

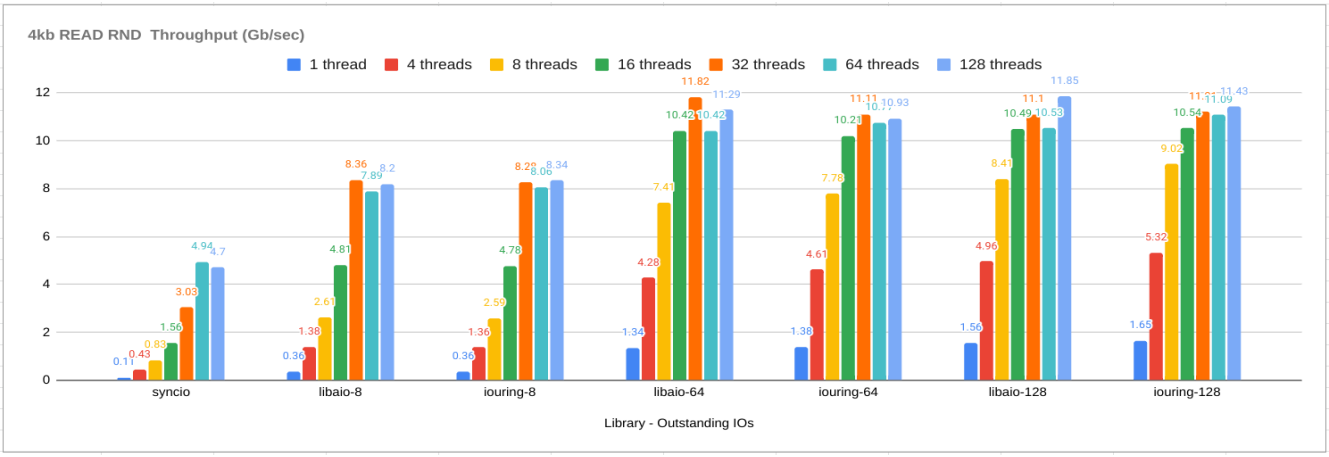
3 Contribution

- A benchmarking tool that utilizes traditional IO syscalls, libaio and liburing was written to evaluate these interfaces under similar use-cases. This directly helped us compare the complexity of using these libraries.
- In-depth evaluations of all three I/O interfaces for various Block sizes and I/O configurations such as Read/Write operations, Sequential/Random access, Vectored/Non-vectored IO, Buffered/Direct IO were performed.
- Performance implications of certain flash storage characteristics such as SLC caching, Over-provisioning and garbage collection were evaluated.

4 Evaluation

Our evaluation was performed on four Samsung 980 PRO 1TB M.2 SSDs that were mounted on an ASRock Hyper Quad M.2 controller. The SSDs were configured in RAID-0 and the controller itself was connected using 16 PCIe 3.0 lanes.

Throughput comparison of the three interfaces at varying thread counts and outstanding I/O operations clearly shows the advantages of Asynchronous I/O. Traditional I/O stacks *(syncio)* are not capable of exploiting the large bandwidth and parallelism of SSD-Arrays.



Performance analysis of these libraries at 4kB Random Reads, 128 Outstanding IO Ops, shows the efficiency of *io_uring* as compared to *libaio* in terms of context-switches and instructions executed per byte of data access. Various optimizations of *io_uring* such as Fixed buffers, Kernel Queue polling have been taken into consideration. Reducing CPU load becomes a major factor in exploiting the bandwidth of SSD-Arrays.