Exploiting SSD-Arrays with Asynchronous I/O

Dwarakanandan B M*
September 30, 2021

*Tobias Ziegler

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Abstract

As data sizes grow, storing them completely in-memory becomes infeasible. This creates a need for secondary storage systems that can keep up with system memory. Solid state drives attached directly using high speed PCI-e interconnects offers high capacity and throughput. In this lab, we explore Asynchronous Input/Output interfaces such as libaio and io_uring to fully exploit the performance and parallelism offered by such SSD-Arrays.

Contents

1 Original Topic: Userspace Network Stacks in Fast Networks

1.1 Motivation

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 as per the guide from Mellanox Technologies [6].

1.2 Challenges

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 [1] . 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.

2 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.

3 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.

4 Contribution

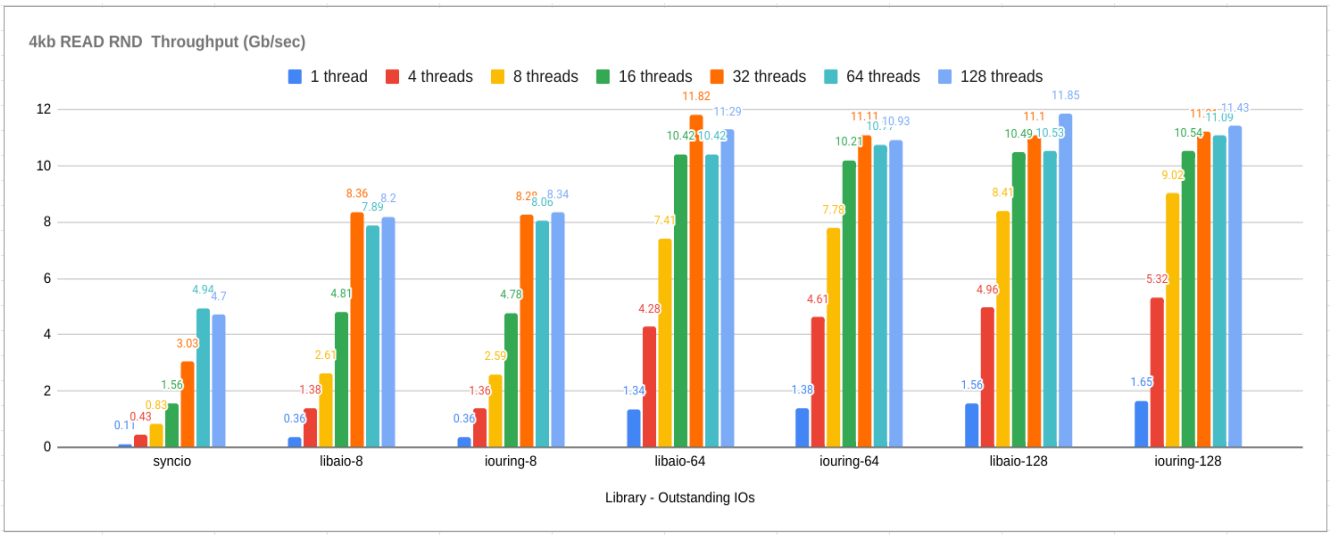
- A benchmarking tool that utilizes traditional IO syscalls, libaio and liburing was written to evaluate these interfaces under similar use-cases. This directly helped us compare the complexity of using these libraries.
- In-depth evaluations of all three I/O interfaces for various Block sizes and I/O 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.

5 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 which offers a max theoretical throughput of 15.754 GB/sec [5].

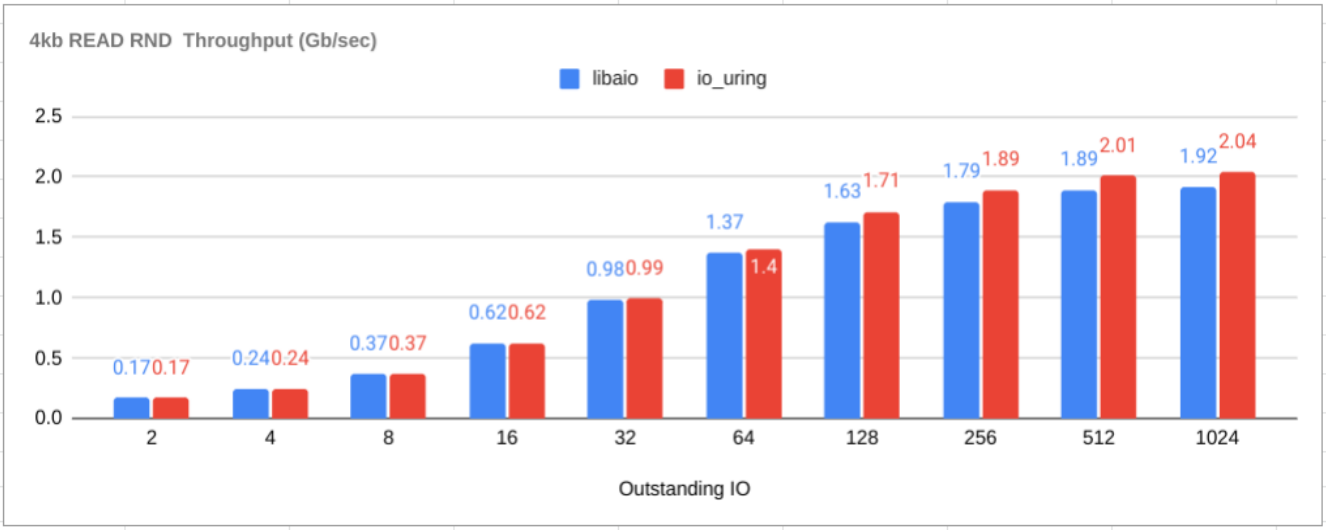
5.1 Throughput comparison

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.



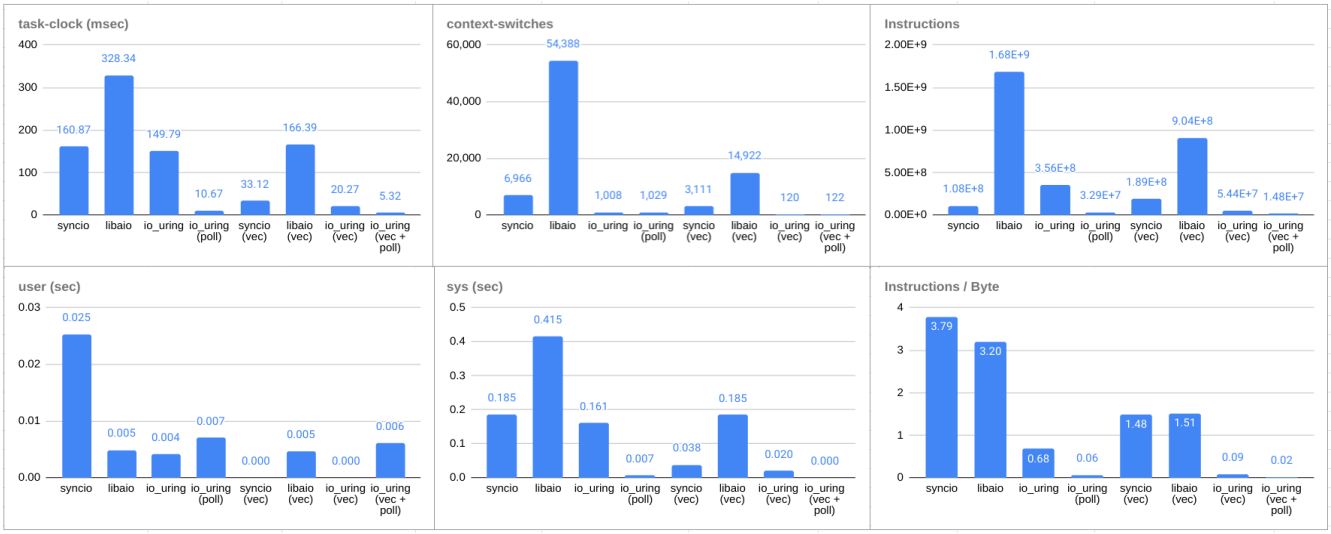
5.2 Optimal Outstanding I/O Count

Comparison of libaio and io_uring at 4kB Random Reads with different outstanding I/O's shows throughput plateauing out at about 128 outstanding I/O's



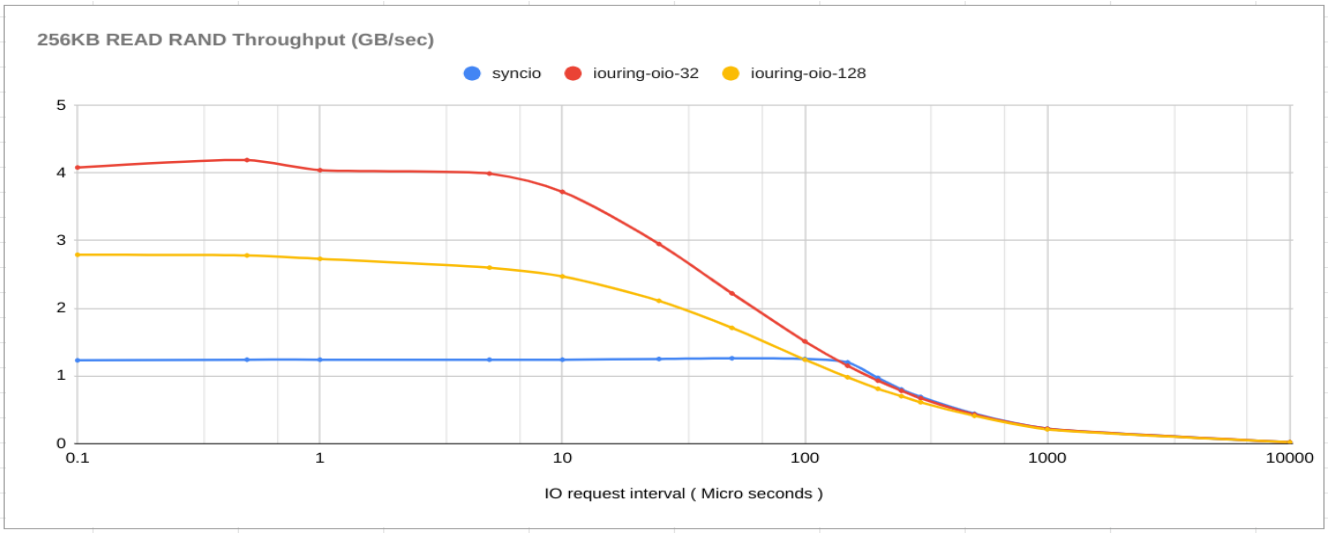
5.3 Performance analysis

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 [4]. Reducing CPU load becomes a major factor in exploiting the bandwidth of SSD-Arrays. [3]



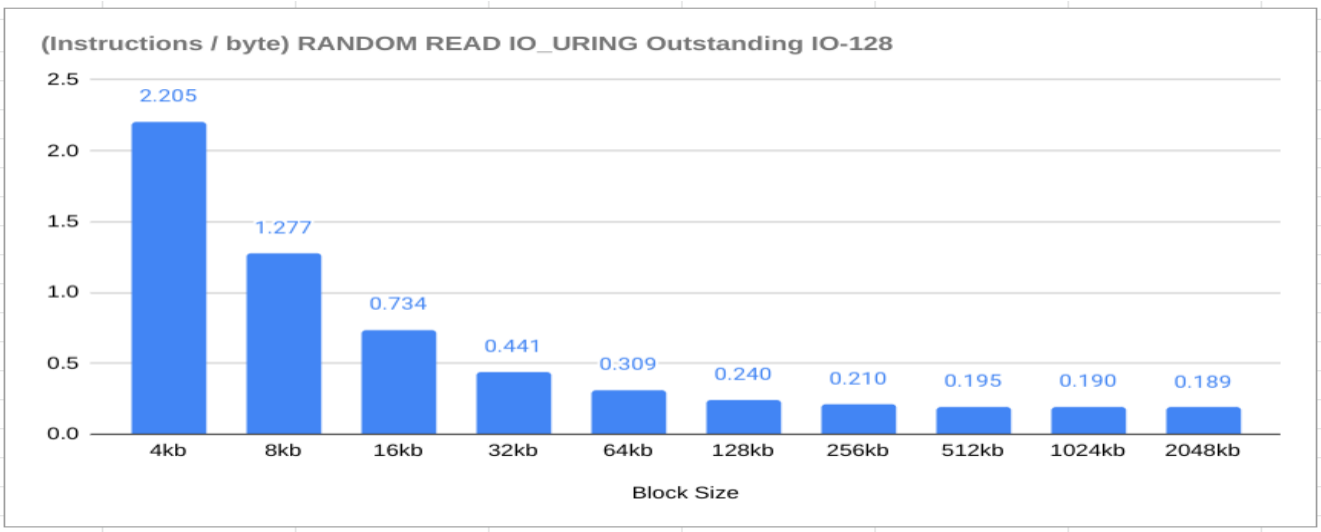
5.4 Optimal IO Request interval

Consider a scenario where one thread acts as an I/O request handler and multiple producers submit I/O requests to it. Our goal was to evaluate the optimal request interval above which we would start losing performance. For asynchronous interfaces an interval of 10 microseconds was found to be optimal. Another interesting observation is that above 100 microseconds asynchronous and synchronous interfaces almost behave identically.



5.5 Instructions per Byte of I/O

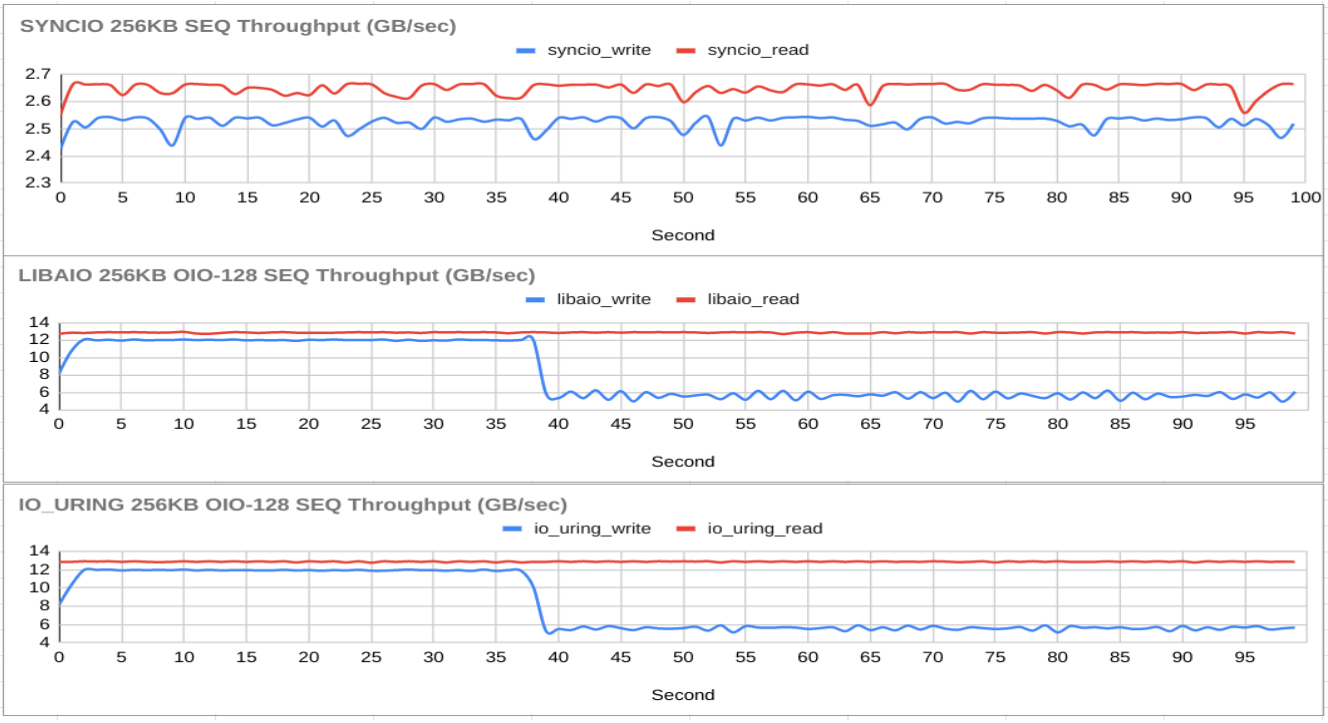
Comparison of io_uring at 128kB RANDOM READS at different block sizes shows that instructions per byte of I/O operation decreases until block size of 64kb, after which we do not see any noticeable improvement in this metric.



5.6 Effects of SLC Cache

SSDs are typically built in a tiered architecture where the fastest layer consists of a small DRAM cache with access speeds similar to system RAM. Below this is a layer of Single Layer Cells (SLC) that offer high throughput but low data density. Bulk of the SSD is made up of Triple layer cells (TLC) that offer high data density at the cost of performance.

Prolonged writes using all three I/O interfaces shows how asynchronous interfaces suffer a sudden loss in performance at about 40 seconds. This is when SLC cache is saturated and I/O is directly performed on the TLC Layer.



6 Documentation

A tool was written in C++ to perform the above described benchmarks and comparisons between syncio, libaio and io_uring. [2]

Prerequisites:

- Linux Kernel above version 5.1
- libaio: Wrapper around the Linux aio
- liburing: Wrapper around the Linux io_uring library

Build tested with:

- gcc/g++ 9.3.0
- Linux kernel 5.4.0

Sample usage:

```
benchmark --file <file_name>
        --threads <thread_count>
        --bsize <block_size_kb>
        --op <read|write>
        --mode <seq|rand>
        --lib <syncio|libaio|liburing>
        --oio <outstanding_io_count>
        --vsize <vectored IO batch size>
        --runtime <runtime in seconds>
        --nodirect (disable O_DIRECT)
        --btype <normal|stress|msg>
        --minterval <message interval in nano seconds>
        --debug (show_debug)
```

References

[1] DPDK. *MLX5 Poll mode drivers, documentation*. URL: https://doc.dpdk.org/guides/nics/mlx5.html (visited on 09/30/2021).

[2] Dwarakanandan. *Benchmarking IO Operations*. URL: https://github.com/dwarakanandan/dmlab (visited on 09/30/2021).

[3] Brendan Gregg. *KPTI/KAISER Meltdown Initial Performance Regressions*. URL: https://www.brendangregg.com/blog/2018-02-09/kpti-kaiser-meltdown-performance.html (visited on 09/30/2021).

[4] Shuveb Hussain. *IO URING Advanced usage*. URL: https://unixism.net/loti/ref-liburing/advanced_usage.html (visited on 09/30/2021).

[5] PCI-SIG. *PCI-Express 3.0 specification*. URL: https://en.wikipedia.org/wiki/PCI_Express (visited on 09/30/2021).

[6] Mellanox Technologies. *Mellanox DPDK Quick start guide*. URL: https://www.mellanox.com/related-docs/prod_software/MLNX_DPDK_Quick_Start_Guide_v16.11_1.5.pdf (visited on 09/30/2021).