Improved Storage Performance Using the New Linux Kernel I/O Interface

John Kariuki (Software Application Engineer, Intel)
Vishal Verma (Performance Engineer, Intel)
Agenda

- Existing Linux IO interfaces
- `io_uring`: The new efficient IO interface
- Liburing library
- Performance
- Upcoming features
- Summary
**Existing Linux Kernel IO Interfaces**

- **Synchronous I/O interfaces:**
  - Thread starts an I/O operation and immediately enters a wait state until the I/O request has completed
  - `read(2)`, `write(2)`, `pread(2)`, `pwrite(2)`, `preadv(2)`, `pwritev(2)`, `preadv2(2)`, `pwritev2(2)`

- **Asynchronous I/O interfaces:**
  - Thread sends an I/O request to the kernel and continues processing another job until the kernel signals to the thread that the I/O request has completed
  - Posix AIO: `aio_read`, `aio_write`
  - Linux AIO: `aio`
Existing Linux User-space IO Interfaces

- SPDK: Provides a set of tools and libraries for writing high performance, scalable, user-mode storage applications
- Asynchronous, polled-mode, lockless design
- https://spdk.io

This talk will cover Linux Kernel IO Interfaces
The Software Overhead Problem

Over 30% SW overhead with most of I/O interfaces vs. pvsync2 when running single I/O to an Intel® Optane™ P4800X SSD

Test configuration details: slide 24
io_uring: The new IO interface

- High I/O performance & scalable:
  - Zero-copy: Submission Queue (SQ) and Completion Queue (CQ) place in shared memory
  - No locking: Uses single-producer-single-consumer ring buffers
- Allows batching to minimize syscalls: Efficient in terms of per I/O overhead.
- Allows asynchronous I/O without requiring O_DIRECT
- Supports both block and file I/O
- Operates in interrupted or polled I/O mode
Introduction to Liburing library

- Provides a simplified API and easier way to establish io_uring instance

- Initialization / De-initialization:
  - `io_uring_queue_init()`: Sets up io_uring instance and creates a communication channel between application and kernel
  - `io_uring_queue_exit()`: Removes the existing io_uring instance

- Submission:
  - `io_uring_get_sqe()`: Gets a submission queue entry (SQE)
  - `io_uring_prep_readv()`: Prepare a SQE with readv operation
  - `io_uring_prep_writev()`: Prepare a SQE with writev operation
  - `io_uring_submit()`: Tell the kernel that submission queue is ready for consumption
Introduction to Liburing library

- Completion:
  - `io_uring_wait_cqe()`: Wait for completion queue entry (CQE) to complete
  - `io_uring.peek_cqe()`: Take a peek at the completion, but do not wait for the event to complete
  - `io_uring_cqe_seen()`: Called once completion event is finished. Increments the CQ ring head, which enables the kernel to fill in a new event at that same slot

- More advanced features not yet available through liburing

- For further information about liburing
  - [http://git.kernel.dk/cgit/liburing](http://git.kernel.dk/cgit/liburing)
## I/O Interfaces comparisons

<table>
<thead>
<tr>
<th>SW Overhead</th>
<th>Synchronous I/O</th>
<th>Libaio</th>
<th>io_uring</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Calls</td>
<td>At least 1 per I/O</td>
<td>2 per I/O batch.</td>
<td>1 per batch, zero when using SQ submission thread.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Batching reduces per I/O overhead</td>
</tr>
<tr>
<td>Memory Copy</td>
<td>Yes</td>
<td>Yes – SQE &amp; CQE</td>
<td>Zero-Copy for SQE &amp; CQE</td>
</tr>
<tr>
<td>Context Switches</td>
<td>Yes</td>
<td>Yes</td>
<td>Minimal context switching polling</td>
</tr>
<tr>
<td>Interrupts</td>
<td>Interrupt driven</td>
<td>Interrupt driven</td>
<td>Supports both Interrupts and polling I/O</td>
</tr>
<tr>
<td>Blocking I/O</td>
<td>Synchronous</td>
<td>Asynchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Buffered I/O</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Performance
Single Core IOPS: libaio vs. io_uring

4K Rand Read IOPS at QD=128
4x Intel® Optane™ SSD, 1 Xeon CPU Core, FIO

IO Submission and Completion batch sizes [1,32]
Test configuration details: slide 24

io_uring: 1.87M IOPS/core
libaio: ~900K IOPS/core
Single Core IOPS: libaio vs io_uring vs SPDK

4K Rand Read IOPS at QD=128
21x Intel® Optane™ P4800X SSDs, 1 Xeon CPU Core

<table>
<thead>
<tr>
<th>IOPS (Millions)</th>
<th>libaio</th>
<th>io_uring</th>
<th>SPDK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.90</td>
<td>1.87</td>
<td>10.38</td>
</tr>
</tbody>
</table>

io_uring: 2x more IOPS/core vs libaio
SPDK: 5.5x more IOPS/core vs io_uring

IO Submission/Completion batch sizes 32 for libaio & io_uring with 4x Intel® Optane™ P4800X SSDs. libaio data collected with fio, io_uring data collected with fio t & SPDK with perf. Test configuration details: slide 24

2019 Storage Developer Conference. © Intel Corporation. All Rights Reserved.
I/O Latency: libaio vs. io_uring vs. SPDK

Submission/Completion Latency (4K Read, QD=1) With Intel® Optane™ SSD

- **Libaio**: Avg. Latency = 1526 ns (Submission) + 489 ns (Completion)
- **IO_uring (without fixedbufs)**: Avg. Latency = 704 ns (Submission) + 155 ns (Completion)
- **IO_uring (with fixedbufs)**: Avg. Latency = 647 ns (Submission) + 154 ns (Completion)
- **SPDK**: Avg. Latency = 150 ns (Submission) + 160 ns (Completion)

**Submission + Completion SW latency**:
- io_uring: 60% lower vs. libaio
- SPDK: 60% lower vs. io_uring

**Submission Latency**: Captures TSC before and after the I/O submission.
**Completion Latency**: Captures TSC before and after the I/O completion check.

Test configuration details: slide 24
libaio vs io_uring I/O path

libaio

- 85.17% fio
- 1.24% entry_SYSCALL_64
- 1.24% do_syscall_64
- 45.28% __x64_sys_io_submit
- 32.03% aio_read
- 30.38% blkdev_read_iter
- 30.13% generic_file_read_iter
- 29.30% blkdev_direct_IO

io_uring

- 81.46% entry_SYSCALL_64
- 75.93% do_syscall_64
- 73.32% __x64_sys_io_uring_enter
- 31.24% io_ring_submit
- 30.83% io_submit_sqe
- 23.37% __io_submit_sqe
- 22.39% io_read
- 20.68% blkdev_read_iter
- 35.62% io_iopoll_check
- 33.80% io_iopoll_getevents
- 28.61% blkdev_iopoll
- 0.87% nvme_poll
- 1.34% blkdev_iopoll

io_uring: submission + completion in 1 syscall
## Interrupt and Context Switch

<table>
<thead>
<tr>
<th>METRICS</th>
<th>libaio</th>
<th>io_uring</th>
<th>RATIONALE</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW Interrupts</td>
<td>172,417.78</td>
<td>251.80</td>
<td>io_uring polling eliminates Interrupts</td>
</tr>
<tr>
<td>Context Switch</td>
<td>112.27</td>
<td>1.47</td>
<td>Reduces context switches by 99%</td>
</tr>
</tbody>
</table>

Workload: 4K Rand Read, 60 sec, 4 P4800, no batching.
HW interrupts & Context Switch metrics are per sec. We used fio for libaio test and fio t for io_uring.
Test configuration details: slide 24
Top-down Microarchitecture Analysis Methodology (TMAM) Overview

io_uring with batching:

- 32% reduction in backend bound stalls vs. libaio
- 32% improvement in μOps retired vs. libaio. 66% lower CPI for io_uring vs. libaio
TMAM Level-3 Analysis
Cache, Branch & TLB: libaio vs. io_uring

io_uring reduces icache & iTLB misses by over 60% vs. libaio

Workload: 4K Rand Read, 60 sec, 4 P4800
Test configuration details: slide 24

2019 Storage Developer Conference. © Intel Corporation. All Rights Reserved.
TMAM Level-3 Analysis
Cache, Branch & TLB: SPDK vs. IO_URING

SPDK
90% less iTLB and L1-icache misses
6x better IOPS/core

Workload: 4K Rand Read, 60 sec
Test configuration details: slide 24
What’s Next for IO_URING

- io_uring for socket based I/O
  - Support already added for sendmsg(), recvmsg()
- Support for devices like RAID (md), Logical Volumes(dm)
- Async support for more system calls
  - Eg: open+read+close in a single call
io_uring is the latest high performance I/O interface in the Linux Kernel (available since 5.1 release)

Eliminates limitations of current Linux kernel async I/O interfaces

Building an application for next generation of NVMe SSDs? io_uring enables
  - Less than 1 usec SW latency to submit/complete I/Os
  - 1 – 2 million IOPS/Core
NOTICES AND DISCLAIMERS

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration.
- No product or component can be absolutely secure.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit [http://www.intel.com/benchmarks](http://www.intel.com/benchmarks).
- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit [http://www.intel.com/benchmarks](http://www.intel.com/benchmarks).
- Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at [http://www.intel.com/go/turbo](http://www.intel.com/go/turbo).
- Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
- Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.
- Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
- © Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.
Performance Configuration

Performance configuration for slide 5 data:
**Relative Latency:** SuperMicro SYS-2029U-TN24R4T, Intel(R) Xeon(R) Platinum 8270 CPU @ 2.70GHz, 384GB DDR4, Ubuntu 18.04 LTS, Linux Kernel 5.2.0, 1x Intel® Optane™ DC SSD P4800X 375GB SSD, fio-3.14-6-g97134, 4K 100% Random Reads, iodepth=1, ramp time = 30s, direct=1, runtime=300s, Data collected at Intel Storage Lab 07/17/2019

**Throughput:** SuperMicro SYS-2029U-TN24R4T, Intel(R) Xeon(R) Platinum 8270 CPU @ 2.70GHz, 384GB DDR4, Ubuntu 18.04 LTS, Linux Kernel 5.2.0, 1x Intel® SSD DC P4610 1.6TB, fio-3.14-6-g97134, 4K 100% Random Reads, iodepth=1 to 256 varied (exponential 2), ramp time=30s, direct=1, runtime=300s, Data collected at Intel Storage Lab 07/17/2019

Performance configuration for slide 11, 12 & 19 data: Intel Server S2600WFT, Intel(R) Xeon(R) Platinum 8280L CPU @ 2.70GHz, 192GB DDR4, Fedora 27, Linux Kernel 5.0.0-rc6, 4x Intel® Alderstream 503GB SSD, SPDK commit 41b7f1ca2189, SPDK bdevperf, runtime = 60s, Data collected at Intel Storage Lab 09/12/2019

Performance configuration for slide 14 data: SuperMicro SYS-2029U-TN24R4T, Intel(R) Xeon(R) Platinum 8270 CPU @ 2.70GHz, 384GB DDR4, Ubuntu 18.04 LTS, Linux Kernel 5.2.0, 4x Intel® Optane™ DC SSD P4800X 375GB SSD, fio-3.14-6-g97134, t/fio app used with varied batching sizes, Data collected at Intel Storage Lab 07/17/2019

Performance configuration for slide 15, 17 &18 data: SuperMicro SYS-2029U-TN24R4T, Intel(R) Xeon(R) Platinum 8270 CPU @ 2.70GHz, 384GB DDR4, Ubuntu 18.04 LTS, Linux Kernel 5.2.0, 4x Intel® Optane™ DC SSD P4800X 375GB SSD, SPDK commit c223ba3b0f, fio-3.14-6-g97134, runtime = 60s, Data collected at Intel Storage Lab 09/6/2019

Performance configuration for slide 25 data: SuperMicro SYS-2029U-TN24R4T, Intel(R) Xeon(R) Platinum 8270 CPU @ 2.70GHz, 384GB DDR4, Ubuntu 18.04 LTS, Linux Kernel 5.2.0, 2x Intel® Optane™ DC SSD P4800X 375GB SSD, 2x Intel® SSD DC P4610 fio-3.14-6-g97134, runtime = 300s, Data collected at Intel Storage Lab 07/17/2019
Relative IOPS Performance: Single Core: IO_Uring vs. Libaio

- Up to 10-15% improvement with io_uring on Intel® SSD DC P4610 at lower queue depths
- io_uring performs up to 1.8x better at lower queue depths on Intel® Optane™ SSDs