Performance Characterization of Modern Storage Stacks

Zebin Ren and Animesh Trivedi

VU Amsterdam

z.ren@vu.nl
a.trivedi@vu.nl

GitHub

@Large Research
Massivizing Computer Systems

VU VRIJE UNIVERSITEIT AMSTERDAM

Paper: https://atlarge-research.com/pdfs/2023-cheops-iostack.pdf

Source code: https://github.com/atlarge- research/Performance-Characterization-Storage-Stacks

The Development of Storage Devices

New Devices

Less than **1k** I/O per Second
Latency: **~5ms**

More than 1000x speed up

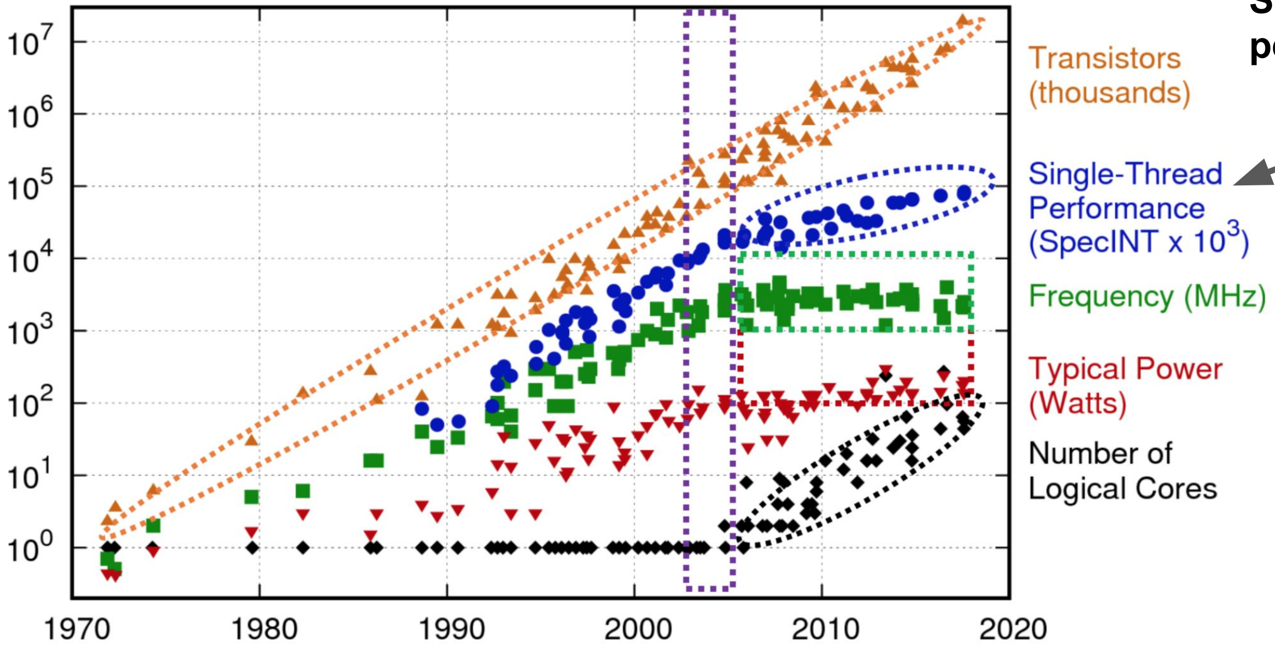
550-1000K I/O per Second
Latency: **~7us**

New Interfaces

SERIAL ATA

NVM EXPRESS

CPU is the Bottleneck



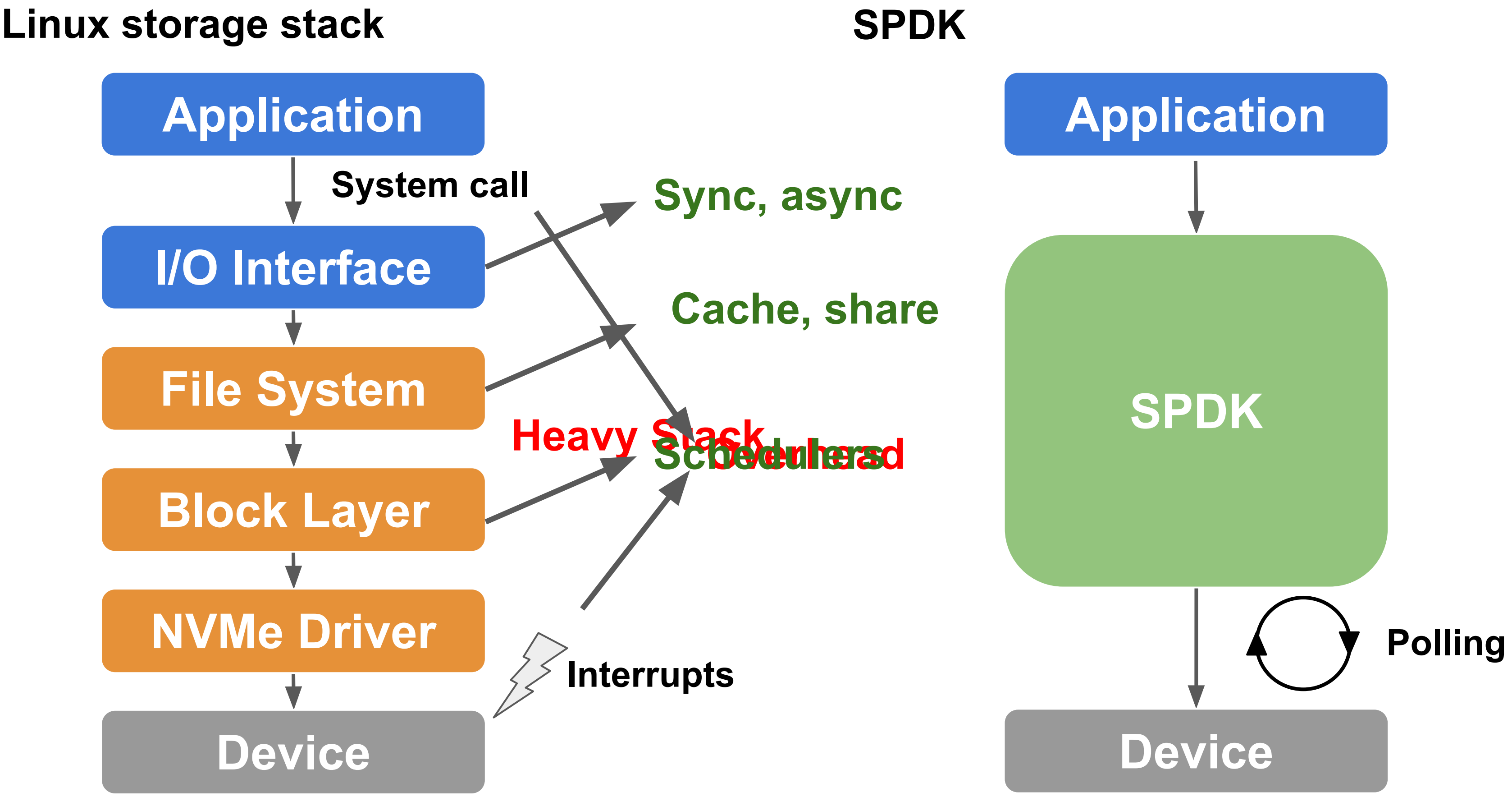
Single-thread performance

Frequency

CPU has become the bottleneck !

https://www.researchgate.net/figure/42-Years-of-Microprocessor-Trend-Data-6-Orange-Moores-Law-trend-Purpule-Dennard_fig1_336577121

Storage Stack



I/O Interfaces

POSIX IO (**psync**)

- Synchronous interface
- Widely used

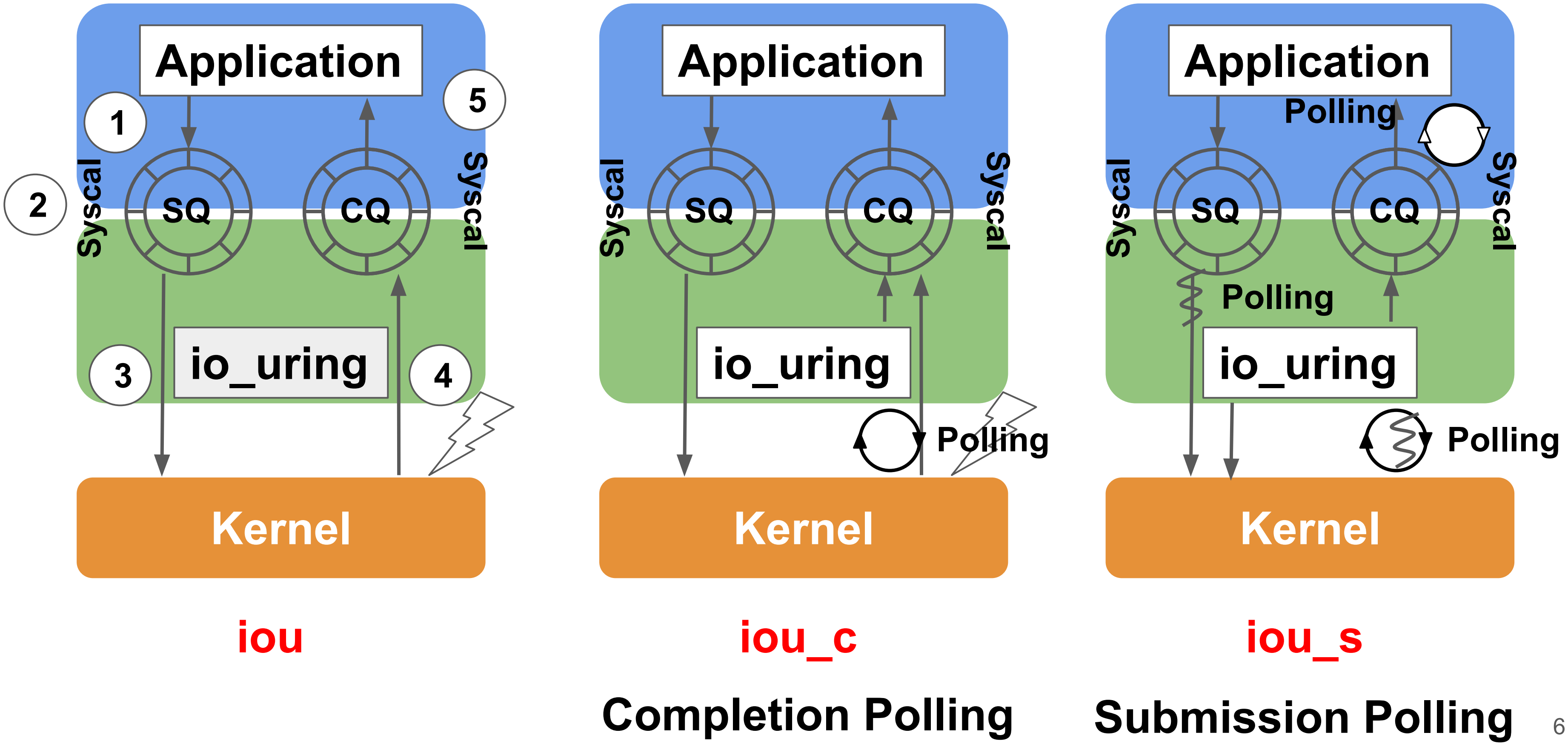
Asynchronous I/O (**libaio**)

- Asynchronous I/O interface for Linux

io_uring (**iou**)

- A new asynchronous I/O interface
- Designed for performance

io_uring



Application
1
5
2
Syscal
SQ
CQ
Syscal
3
io_uring
4
Kernel
iou
Application
Syscal
SQ
CQ
Syscal
io_uring
Polling
Kernel
iou_c
Completion Polling
Application
Polling
Syscal
SQ
CQ
Syscal
Polling
io_uring
Polling
Kernel
iou_s
Submission Polling

Research Problems

Q1: What is the **performance gap** between different **I/O API** and **storage stacks**?

Q2: What is the **cause** of the **performance gap**?

Q3: How does the performance gap **scale** with the number of processes?

Setup

Devices

Intel Optane * 7 → **3.8 Million** IOPS

Workload generator

fio → Widely used + flexible

Workload

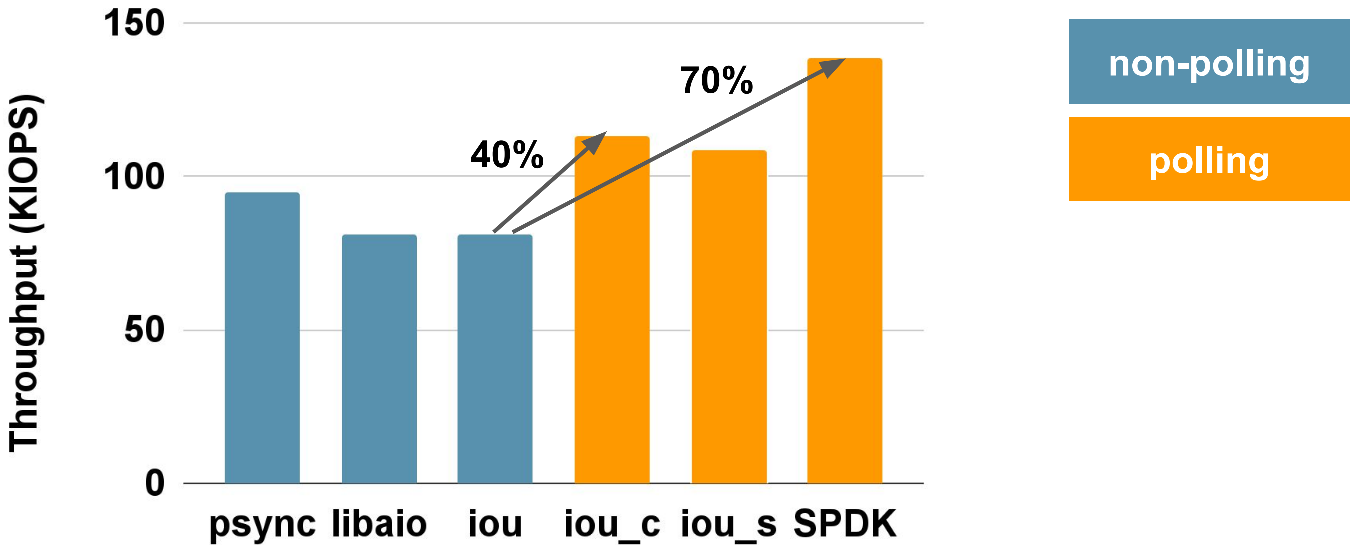
4KB random read → to maximize software overhead

Low workload → **1** outstanding request

High workload → **128** outstanding request

What is the performance gap between different I/O APIs and storage stacks?

Performance: Low Workload (Queue Depth = 1)

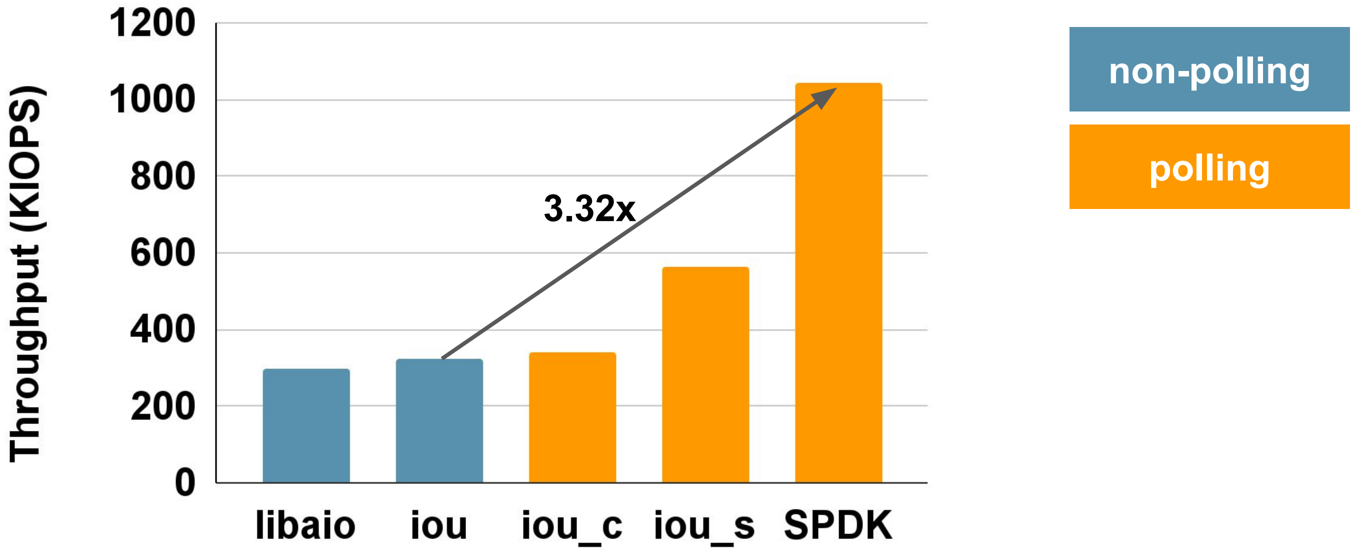


psync has better throughput than libaio and iou

Polling improves the throughput

SPDK has better throughput than the Linux storage stack

Performance: High Workload (Queue Depth = 128)



iou is better than libaio

Polling improves throughput, slightly

SPDK has much better throughput than the linux storage stack

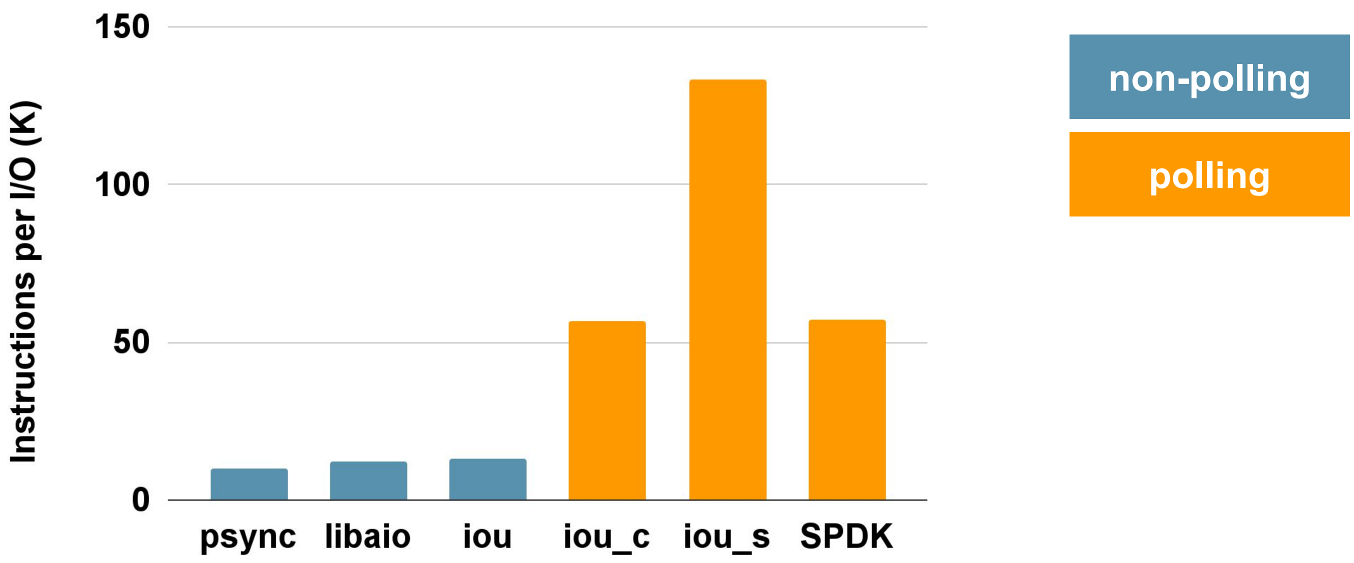
Why there is a performance gap?

Number of instructions per I/O

Instructions per cycle (IPC)

Micro-architectural Efficiency: # Instructions per I/O

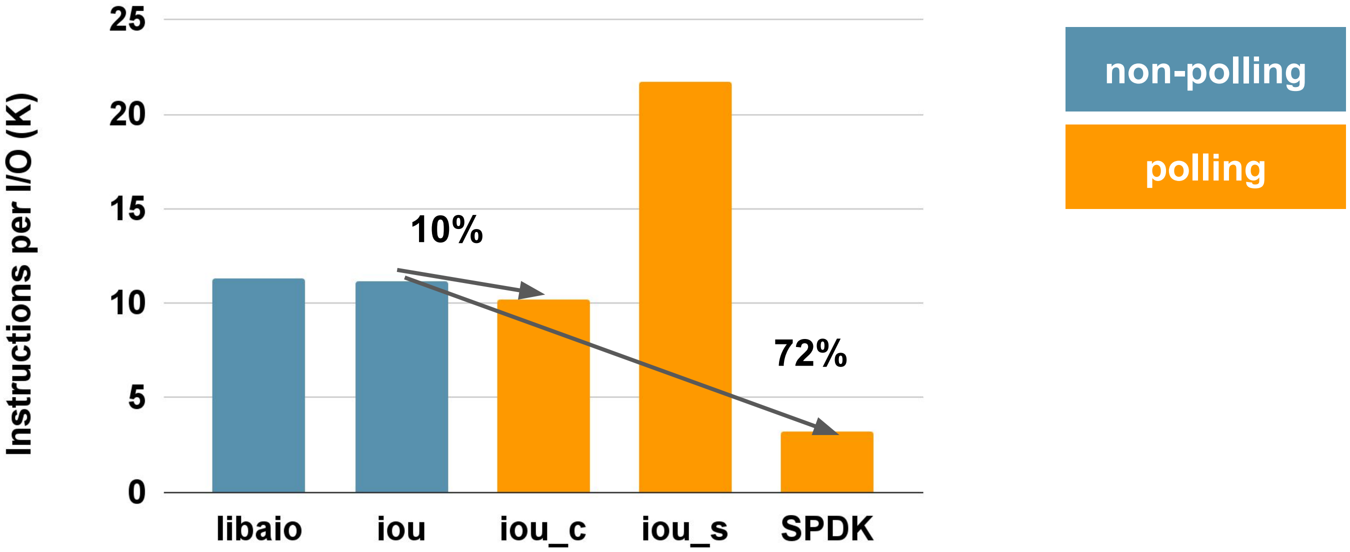
Low Workload



psync is more efficient than libaio and io_uring

Polling wastes instructions at low workload

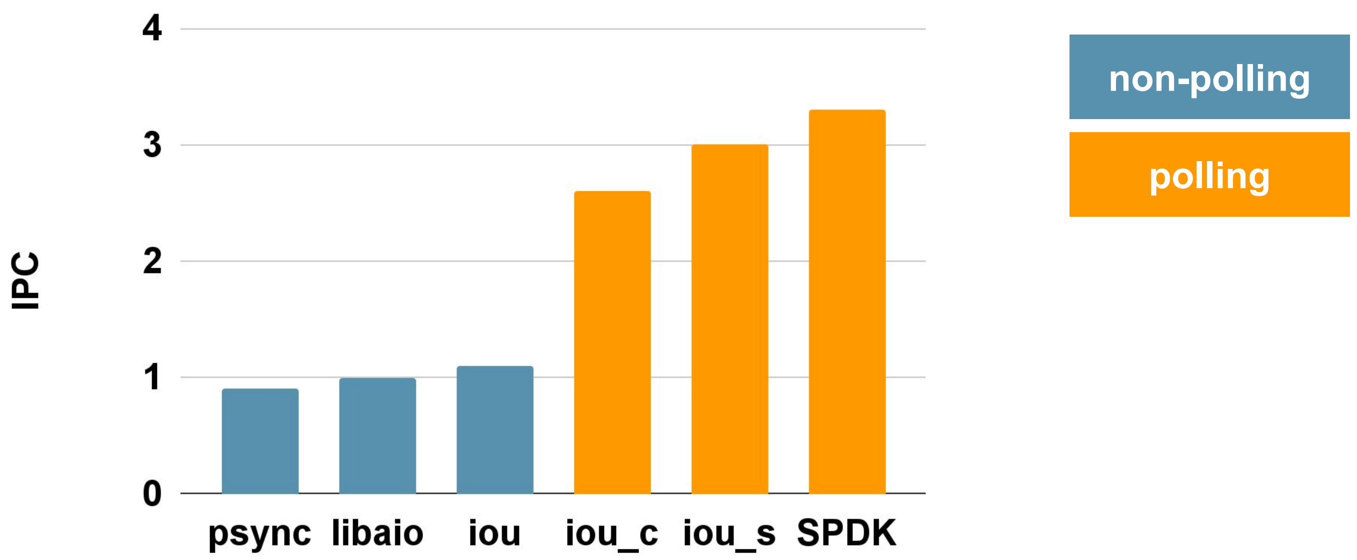
Micro-architectural Efficiency: # Instructions per I/O
High Workload



Polling is efficient at high workload

SPDK is much more efficient than the Linux storage stack

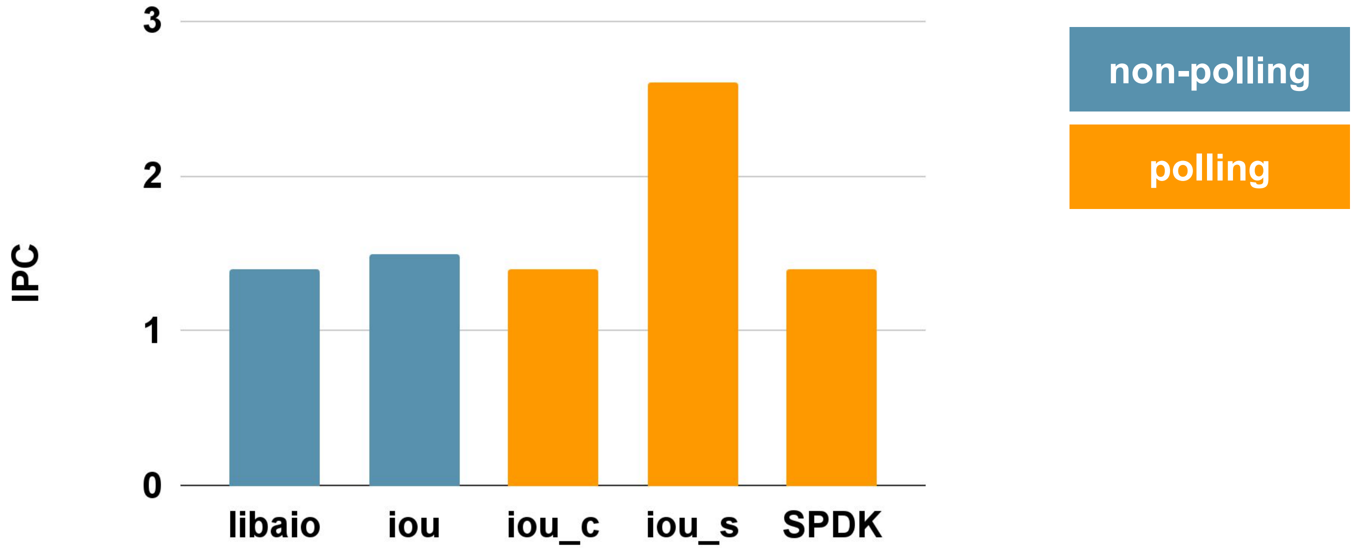
Micro-architectural Efficiency: IPC
Low
Workload



iou has higher IPC

Polling leads to high IPC at low workload

Micro-architectural Efficiency: IPC High Workload



Non-polling delivers to high IPC than low workload

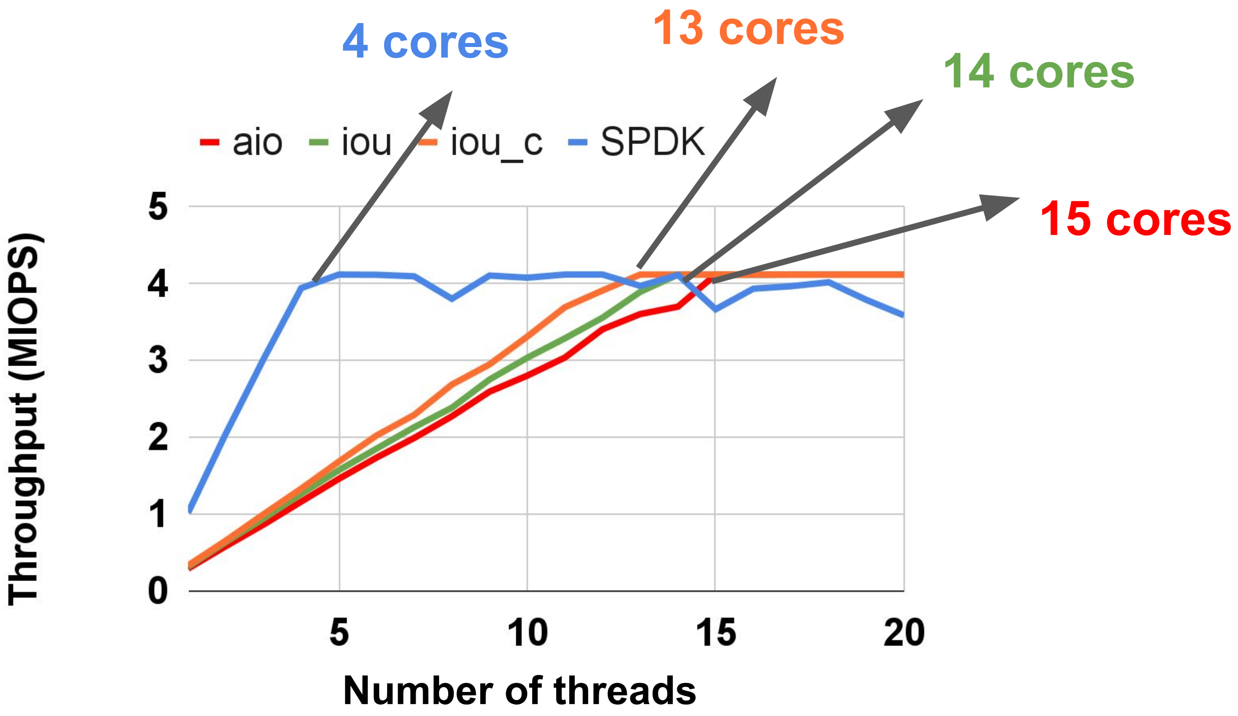
Non-polling and polling APIs have comparable results

How does the performance gap scale with the number of processes?

Scalability of performance

Impact of I/O schedulers

Scalability

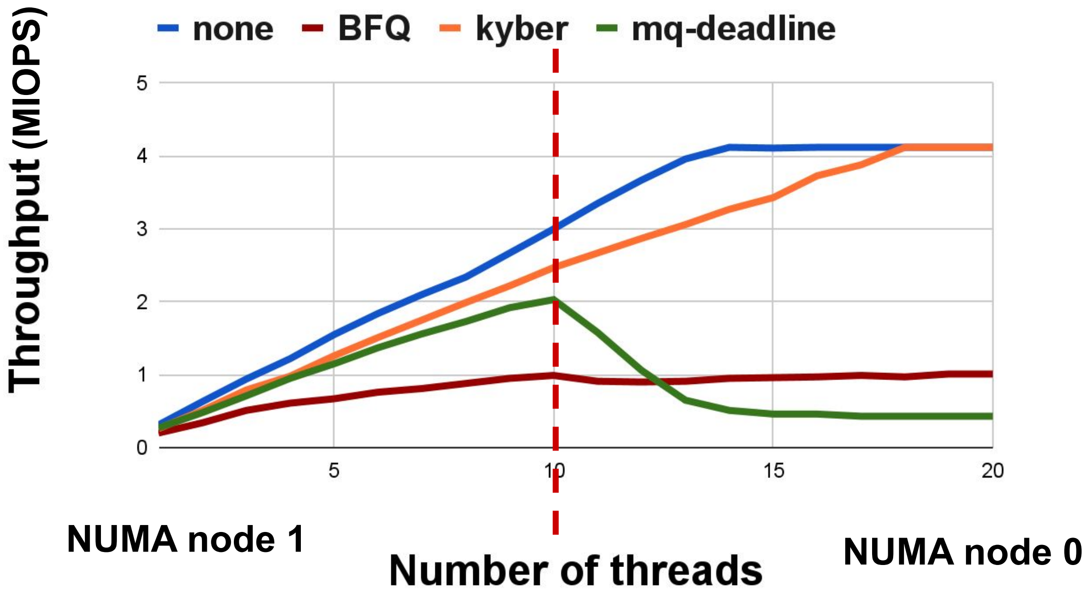


Performance scales linearly for the Linux kernel I/O APIs

io_uring has better performance

SPDK has much higher efficiency than the Linux storage stack

I/O Schedulers



All the I/O schedulers has overhead than the none scheduler

kyber can saturate all the devices with enough CPU resource

mq-deadline and BFQ has bad performance for cross-NUMA access

Take-Home Messages

1. ***Use polling, but carefully***
   Polling wastes CPU time at low I/O workload

2. ***Big gap between Linux storage stack and SPDK***
   SPDK is lightweight and can deliver higher throughput when CPU is the bottleneck

3. The problem of ***Linux I/O stack*** is ***inefficiency***
   Reduce software overhead, scalability of I/O schedulers

GitHub

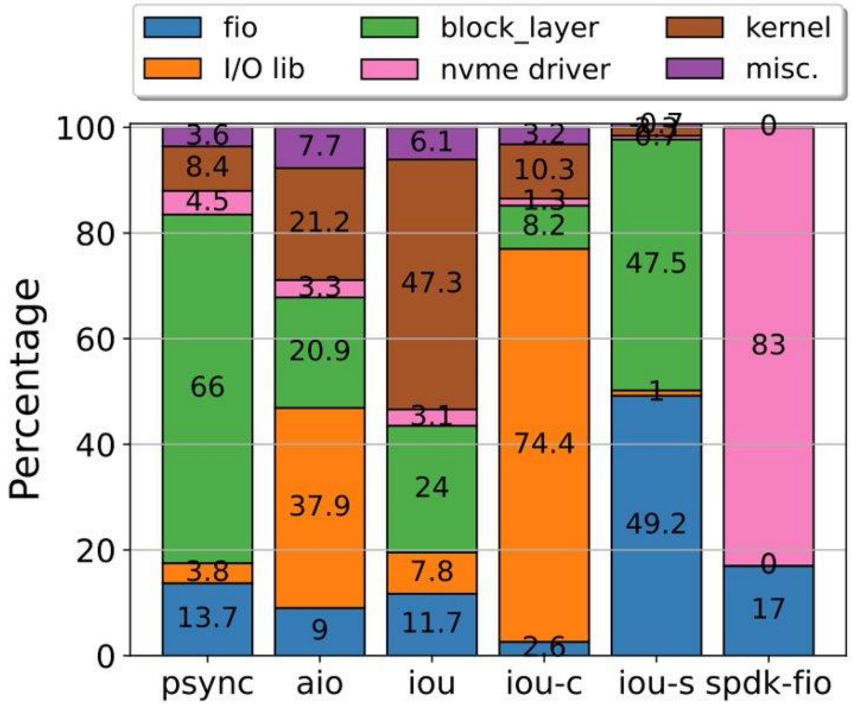
Source code: https://github.com/atlarge-research/Performance-Characterization-Storage-Stacks

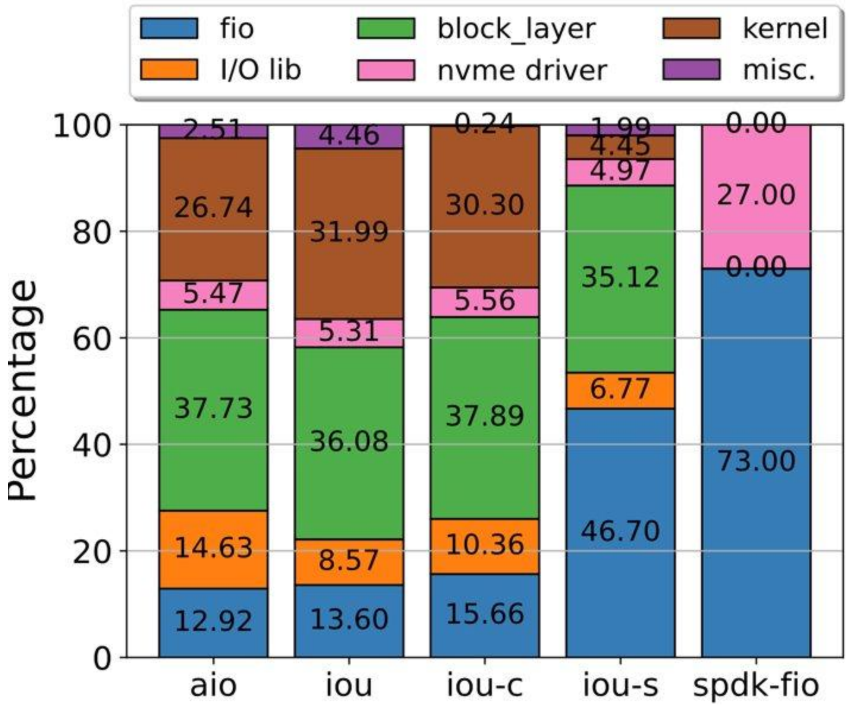
Further Reading

[1] libaio https://man7.org/linux/man-pages/man7/aio.7.html
[2] io_uring https://man.archlinux.org/man/io_uring.7.en
[3] spdk https://spdk.io/
[4] Multi-Queue Block IO Queueing Mechanism (blk-mq). https://www.kernel.org/doc/html/latest/block/blk-mq.html
[5] Efficient IO with io_uring. https://kernel.dk/io_uring.pdf
[6] Matias Bjørling, Jens Axboe, David W. Nellans, and Philippe Bonnet. Linux block IO: introducing multi-queue SSD access on multi-core systems. SYSTOR 2013.
[7] Diego Didona, Jonas Pfefferle, Nikolas Ioannou, Bernard Metzler, and Animesh Trivedi. 2022. Understanding modern storage APIs: a systematic study of libaio, SPDK, and io_uring. SYSTOR 2022.
[8] Sungjoon Koh, Junhyeok Jang, Changrim Lee, Miryeong Kwon, Jie Zhang, and Myoungsoo Jung. Faster than Flash: An In-Depth Study of System Challenges for Emerging Ultra-Low Latency SSDs. IISWC 2019.
[9] Gyusun Lee, Seokha Shin, Wonsuk Song, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. Asynchronous I/O Stack: A Low-latency Kernel I/O Stack for Ultra-Low Latency SSDs. ATC 2019.
[10] Woong Shin, Qichen Chen, Myoungwon Oh, Hyeonsang Eom, and Heon Y. Yeom. OS I/O Path Optimizations for Flash Solid-state Drives. ATC 2014.
[11] Athanasios Stratikopoulos, Christos Kotselidis, John Goodacre, and Mikel Luján. FastPath: Towards Wire-Speed NVMe SSDs. FPL 2018.

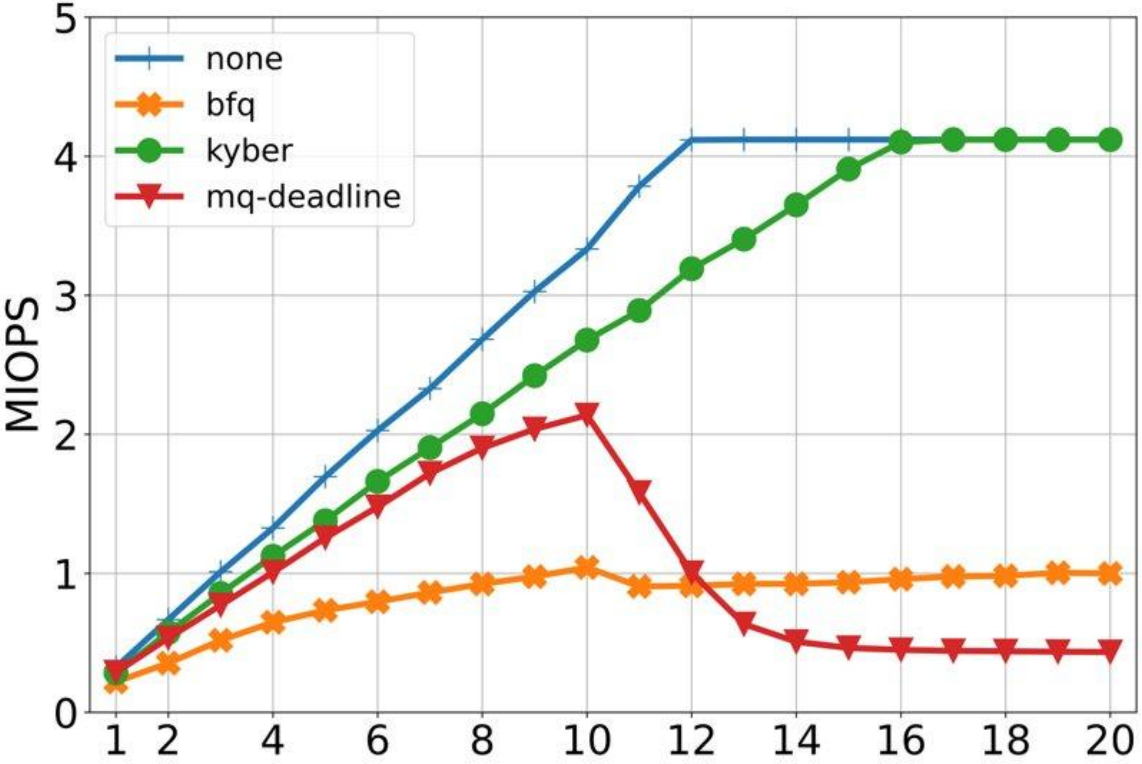
Thank you!
Questions?

Backup Slides: Work Breakdown

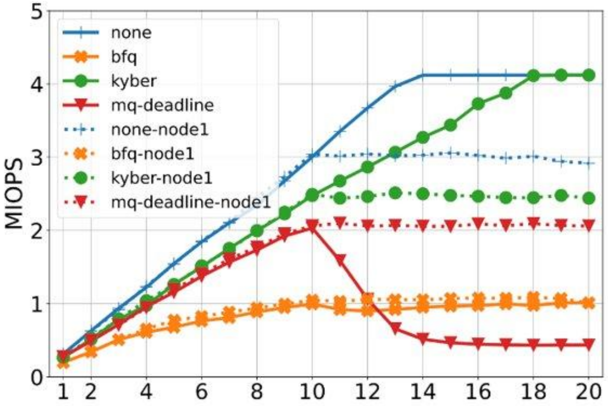




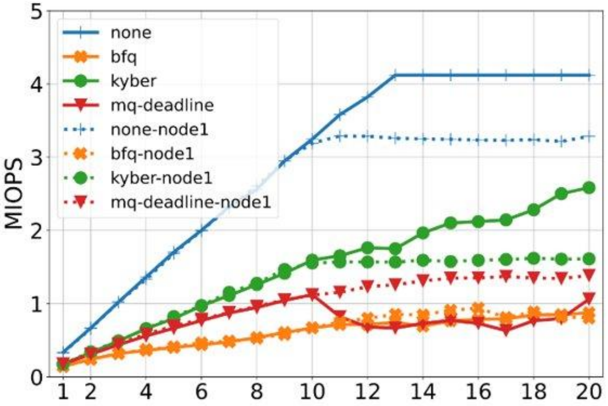
Backup Slides: I/O Scheduler



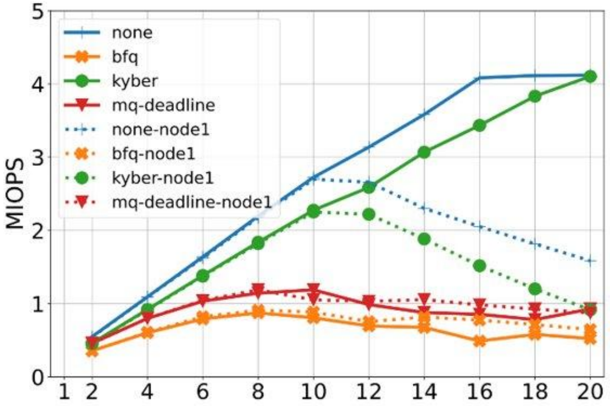
Backup Slides: I/O Scheduler



(a) default `io_uring` (iou).



(b) with completion polling (iou-c).



(c) with submission polling (iou-s).