Interrupt Support and Coalescing Enhancement for SPDK When Online Applications Co-running with Offline Applications

Binyao Jiang

INTRODUCTION

A modern datacenter server often run a wide range of online applications while these applications usually have a low utilization of modern NVMe SSDs whose peak throughput can be 3000MB/s which seems impossible to be saturated by the lightweight online applications. Thus, in order to improve utilization and energy efficiency, some people suggest co-running throughput-driven offline applications with these online applications[1]. Furthermore, NVMe 1.4 protocol provides NVM sets which can physically separate SSD resources for performance isolation, which makes this co-running mechanism much more realistic.

SPDK is a user-space, polled-mode, asynchronous, lockless NVMe driver which supports kernel-bypass, a technique to transfer data directly from user-space without kernel's overhead. With kernel-bypass, 512B random read average latency can be reduced from 6.01us to 3.16us[2], which is significant for online applications. However, SPDK cannot coexist with traditional NVMe driver, if co-running is employed, those offline applications have to use SPDK as backing driver, where forcibly 100% CPU usage is required due to polling.

In this project, we aim to integrate interrupt support into SPDK and further add dynamic interrupt coalescing support where each user can control the frequency or the batch size of each interrupt event. This dynamic attributes highly match the SPDK's design philosophy: one-to-one SW queue to HW queue mapping instead of multiplexing.

OBJECTIVES

Add interrupt support into SPDK tool. For offline applications which is not latency-sensitive, switch them to interrupt-mode to save CPU resources.

Design and implement dynamic run-time interrupt coalescing support into interrupt-mode SPDK and NVMe SSD. In this case, users can control the frequency or the batch size of each interrupt event in a more fine-grained manner for better CPU resource saving.

Modify FIO (Flexble I/O tester), a well-known disk workload benchmark tool, to show performance data under sequential and random read/write pattern:

1) Correctness
2) Peak throughput
3) CPU usage under peak throughput
4) Unloaded latency

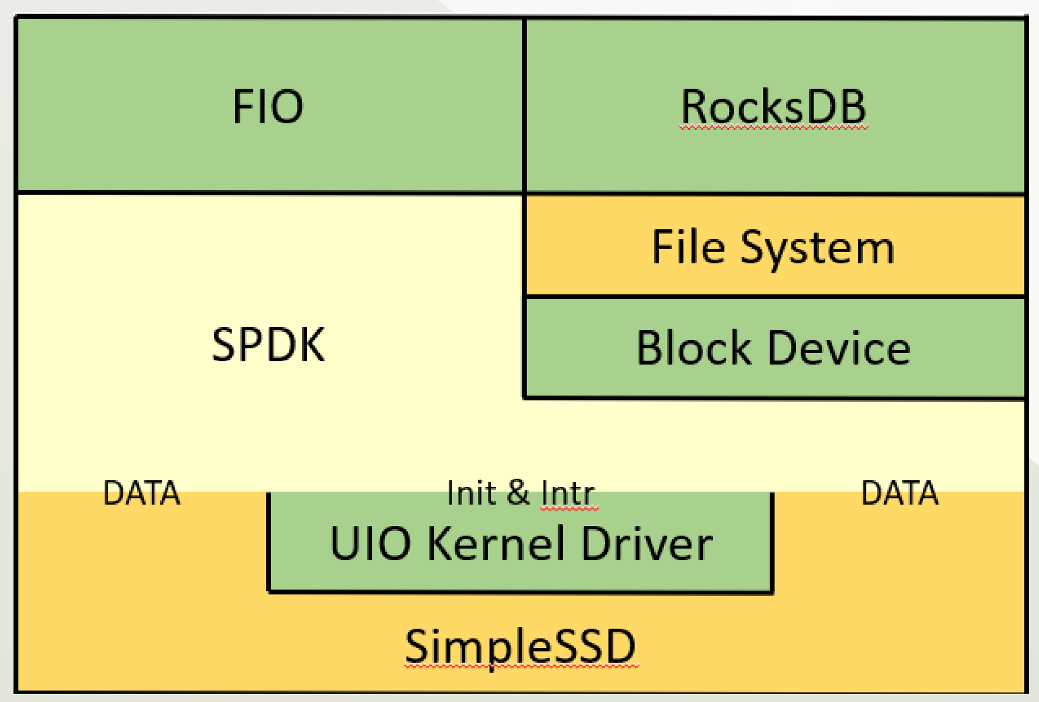
Modify RocksDB, an embedded key-value store open sourced by Facebook, to show end-to-end industrial application level performance.

SYSTEM DESIGN

System Overview:

The whole system we proposed ranges from hardware-level, kernel-level and user-level, and it runs upon gem5 simulator:

1) SimpleSSD: an open-source full-system SSD Simulator built upon gem5.
2) UIO Kernel Driver: SPDK relies on this driver to initialize hardware resources such as memory map its base-address-register memory to user space.
3) SPDK: a user-space, polled-mode, asynchronous, lockless NVMe driver
4) FIO: flexble I/O tester. Serve as microbenchmark application.
5) RocksDB: key-value store. Serve as end-to-end industrial application.



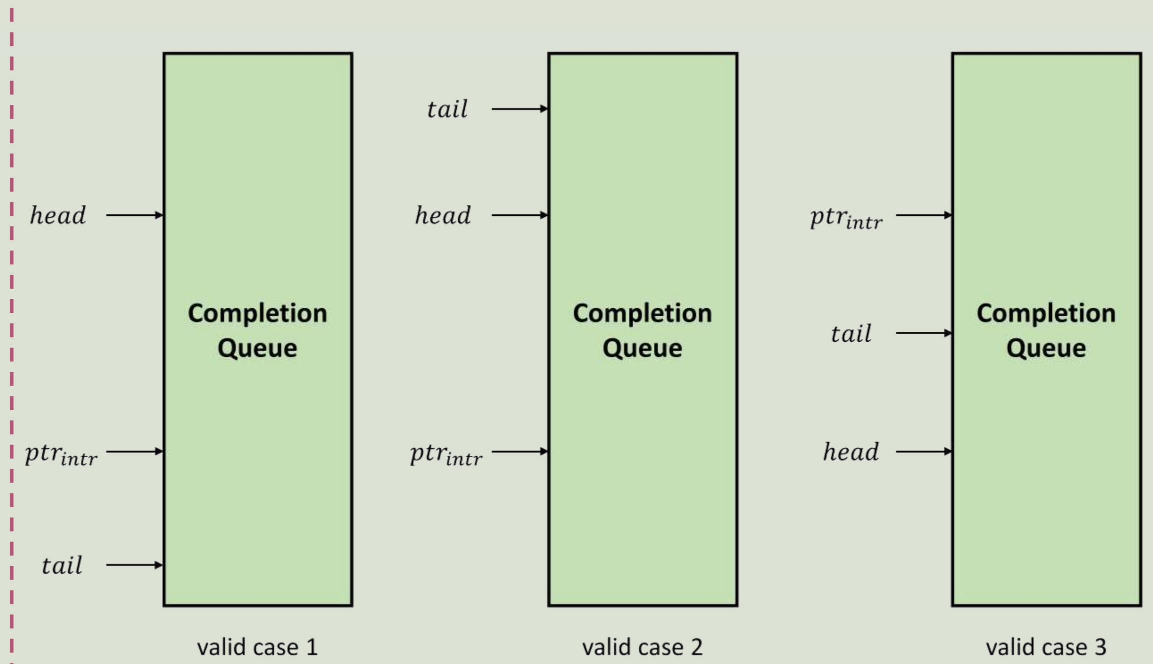
Software Interfaces:

SPDK can check and collect the command completions through polling interface and interrupt interface. Data access is kernel-bypassed. Initialization and interrupt support requires kernel's involvement. Here, two additional functions are added into SPDK for interrupt support:

1) int nvme_ctrlr_set_intr(struct spdk_nvme_ctrlr *ctrlr): set I/O queue to be interruptible before allocating in hardware.
2) int32_t spdk_nvme_qpair_interrupt_completions(struct spdk_nvme_qpair *qpair, uint32_t num_completions): sleep and wait until <num_completions> completions finished.

Dynamic Interrupt Coalescing Design:

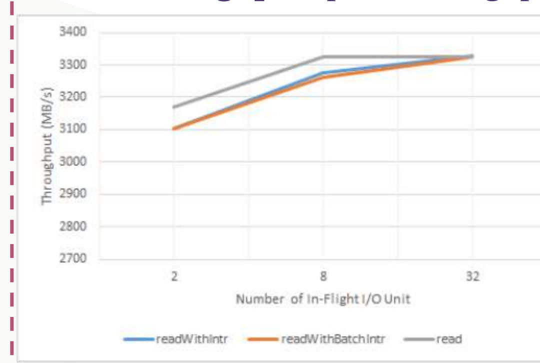
Information representing <num_completions> is transformed into absolute completion queue buffer pointer, and sent to SSD to deal with hidden in-flight command/completion issue: user has no idea hardware's current processing status when interrupt command arrives at SSD's side. Following is three valid cases where SSD should trigger an interrupt event directly:



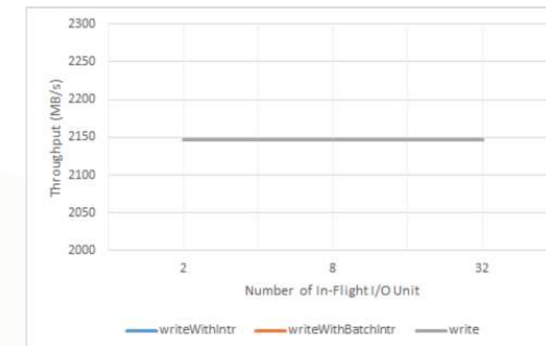
EXPERIMENTAL RESULTS

Microbenchmark (FIO):

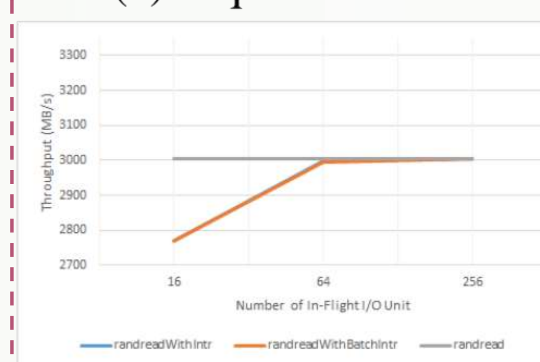
Peak throughput: peak throughput could still be saturated.



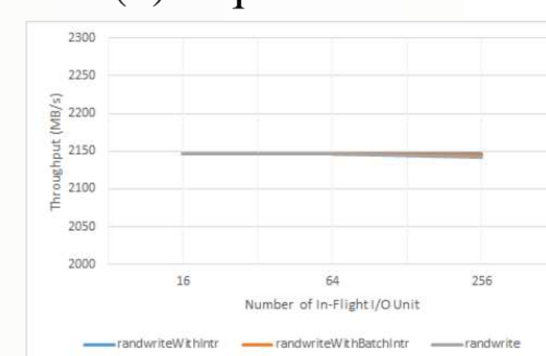
(a) sequential read



(b) sequential write

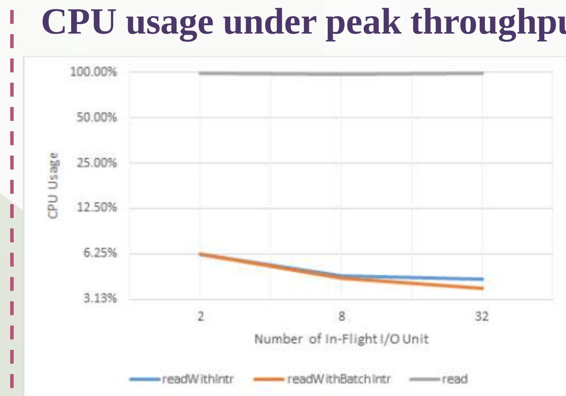


(c) random read

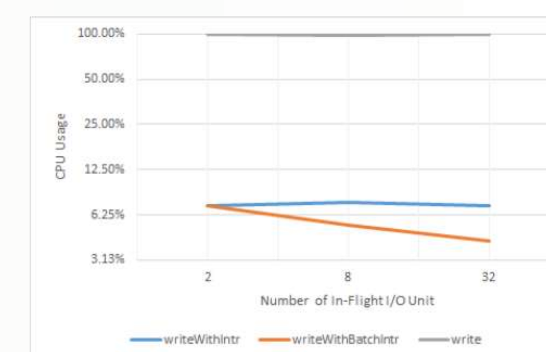


(d) random write

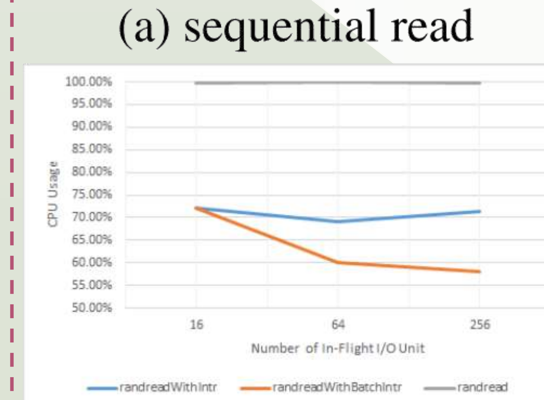
CPU usage under peak throughput: save 50% and even 96% CPU cores.



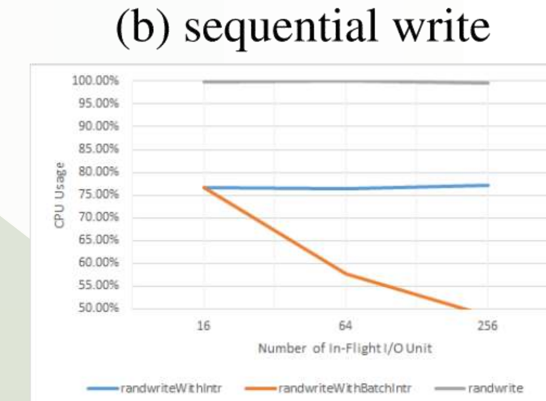
(a) sequential read



(b) sequential write

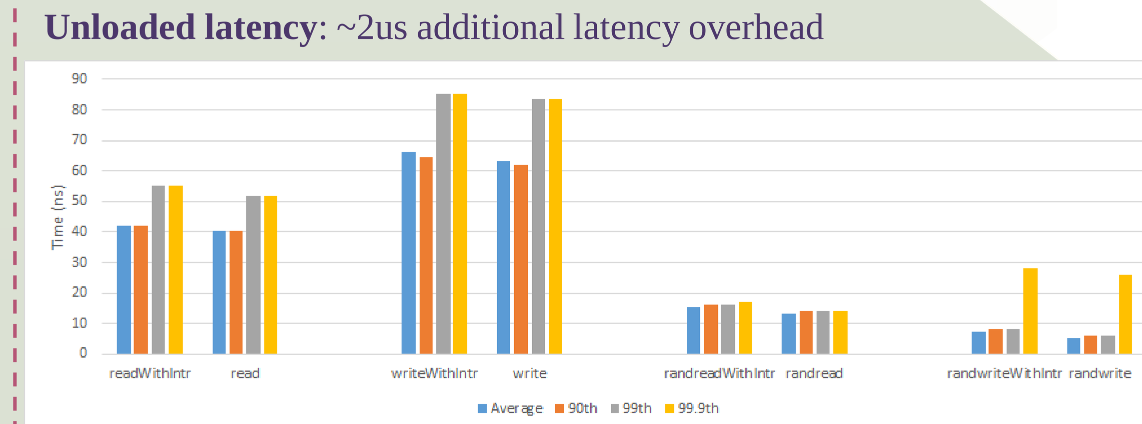


(c) random read



(d) random write

Unloaded latency: ~2us additional latency overhead



Application-level Benchmark (RocksDB):

1) 32.5%-73.3% CPU usage saved.
2) 1.7%-21.3% performance degradation mainly brought by incompatible SPDK threading model instead of interrupt itself.

Benchmark	Poll-mode		Interrupt-mode	
	ops/s	CPU (%)	ops/s	CPU (%)
Insert	126569	151	124410	61
Overwrite	99616	158	97109	68
Readwrite	120	155	94	49
Writesync	185	131	154	35
Randread	22515	148	20563	100

CONCLUSIONS

This paper has presented interrupt support and interrupt coalescing enhancement for existing SPDK. We have introduced two additional functions into SPDK for these supports. Moreover, we have designed a dynamic interrupt coalescing mechanism at both software side and hardware side. At last, it turns out that our system can save 50% and even 96% CPU cores while still saturating SSD's throughput under sequential and random read/write pattern, and only introduce 2us latency overhead in unloaded scenarios. In industrial application RocksDB, we have shown our system can save 32.5%-73.3% CPU usage compared with original results.

REFERENCES

[1] Zhang, Jie, et al. "FLASHSHARE: Punching through server storage stack from kernel to firmware for ultra-low latency SSDs." *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 2018.

[2] Damien Le Moal. I/o latency optimization withpolling.Vault–Linux Storage and Filesystem, 2017.

[3] Gouk, Donghyun, et al. "Amber*: Enabling Precise Full-System Simulation with Detailed Modeling of All SSD Resources." *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018.

[4] Jung, Myoungsoo, et al. "Simplessd: modeling solid state drives for holistic system simulation." *IEEE Computer Architecture Letters* 17.1 (2017): 37-41.

[5] Binkert, Nathan, et al. "The gem5 simulator." *ACM SIGARCH Computer Architecture News* 39.2 (2011): 1-7.

[6] Yang, Ziye, et al. "Spdk: A development kit to build high performance storage applications." *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017.

CONTACT

Binyao Jiang
Email: binyaoj2@illinois.edu
Linkedin: linkedin.com/in/byjiang1996/
Facebook: facebook.com/byjiang1996
Wechat: jby-2015