Improving Performance by Bridging the Semantic Gap between Multi-queue SSD and I/O Virtualization Framework

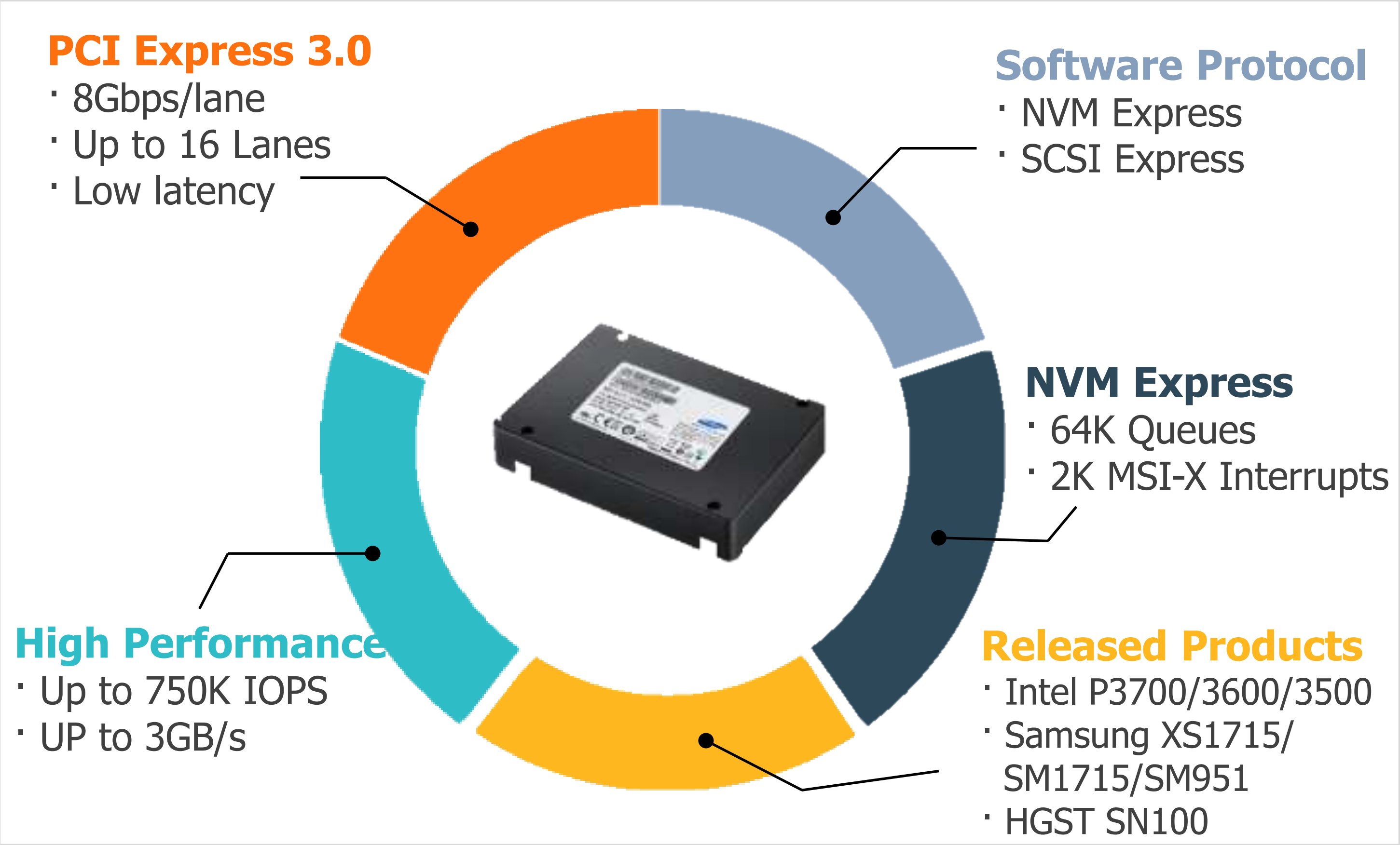
Tae Yong Kim[†*], Dong Hyun Kang[†], Dongwoo Lee[†], Young Ik Eom[†]

[†]Sungkyunkwan University, South Korea
[*]Samsung Electronics, South Korea

MSST 2015

Background

- **Multi-queue SSD**



PCI Express 3.0
- 8Gbps/lane
- Up to 16 Lanes
- Low latency

Software Protocol
- NVM Express
- SCSI Express

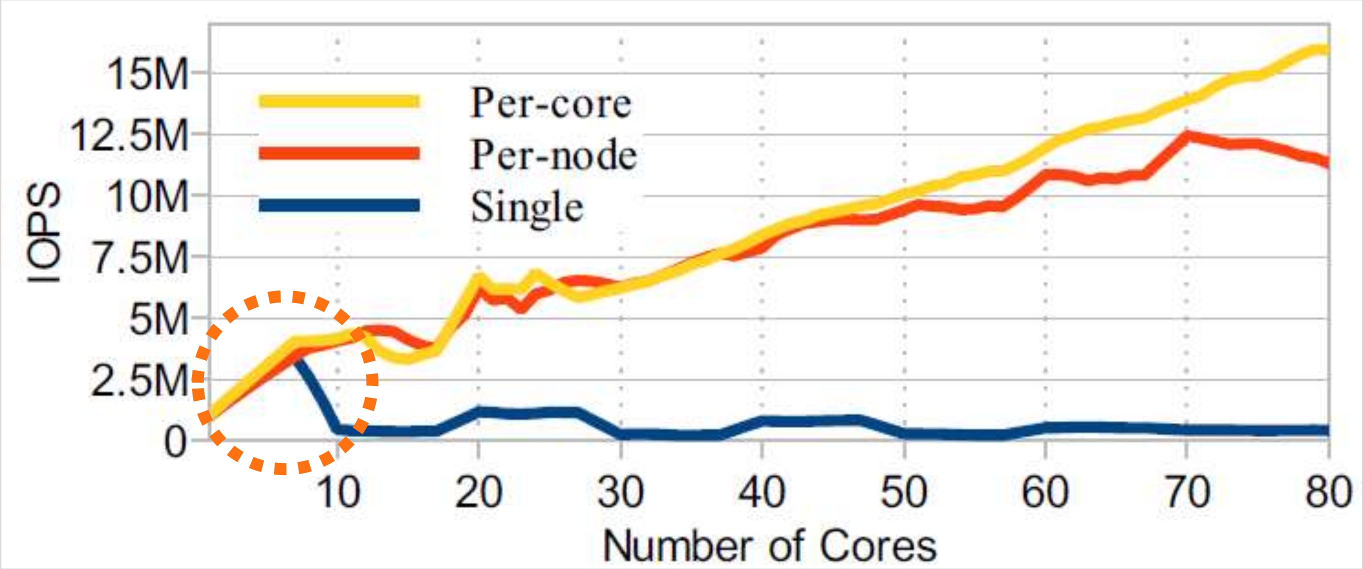
NVM Express
- 64K Queues
- 2K MSI-X Interrupts

High Performance
- Up to 750K IOPS
- UP to 3GB/s

Released Products
- Intel P3700/3600/3500
- Samsung XS1715/ SM1715/SM951
- HGST SN100

Motivation (1/3)

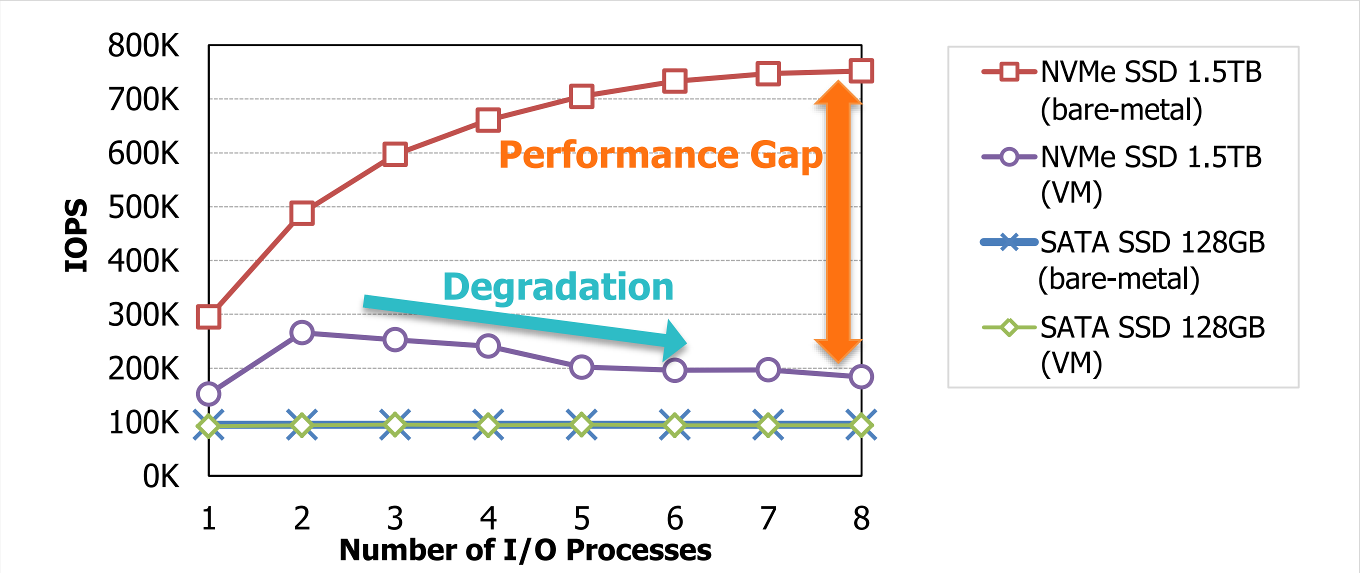
- **Problem in previous Linux block layer with multi-queue SSD**
  - Performance degradation on I/O scalability
  - Lock contention problem due to a single request queue
  - Proposed per core software queues (Updated in ver. 3.13)



※ **M. Bjørling et al.,** "Linux block io: introducing multi-queue ssd access on multi-core systems" , SYSTOR, 2013

Motivation (2/3)

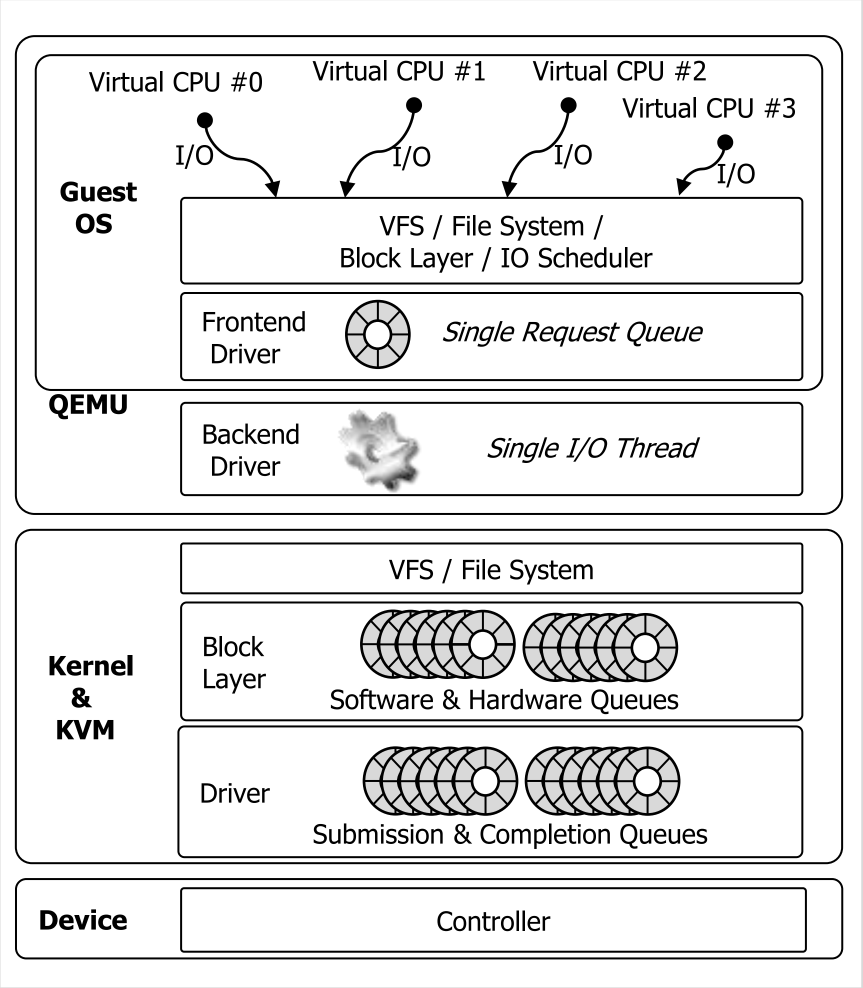
- **I/O scalability issue in KVM/QEMU**
  - 4KB random read performance varying the number of I/O processes
  - Severe performance gap: up to 74%
  - Performance increasingly degraded



※ **Host PC :** Intel i7 3.5GHz * 4, 8GB RAM, Ubuntu 14.04 64bit(Kernel version 3.13)
※ **VM :** 8 vCPU, 8GB RAM, Ubuntu 14.04, QEMU 2.0.0, KVM Accel., Virtio-Blk-Data-Plane
※ **Benchmark :** FIO (Direct I/O, libaio : native async. I/O, Queue Depth : 32)

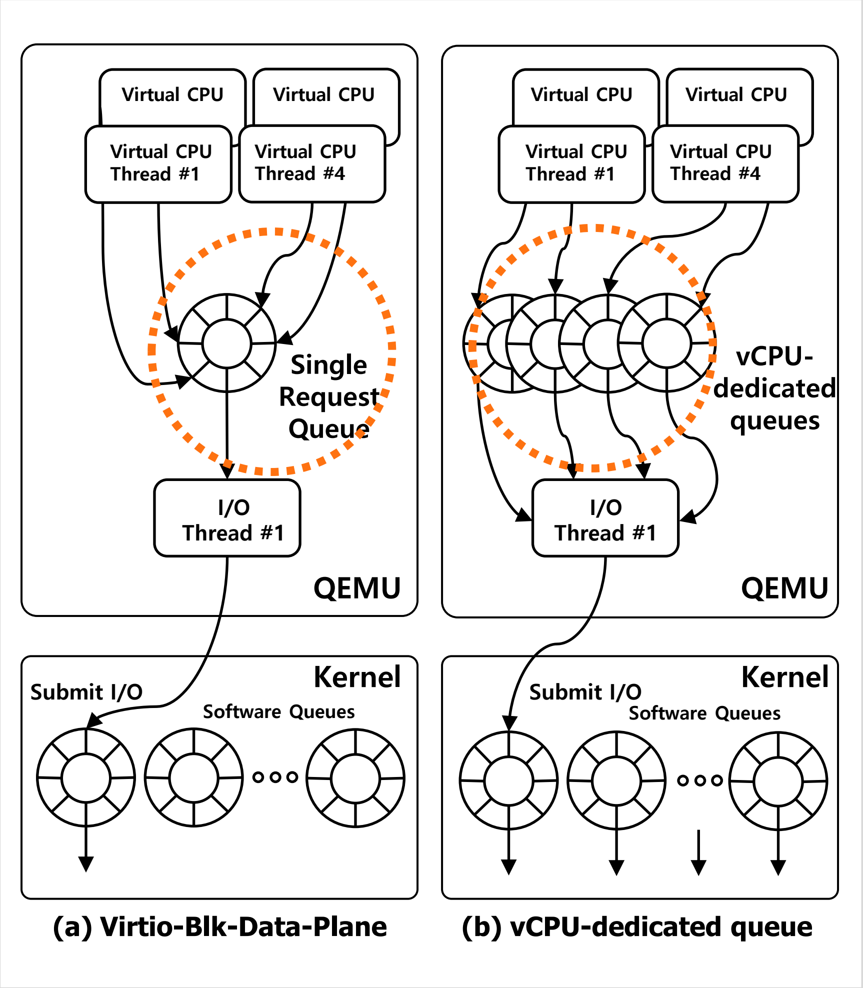
Motivation (3/3)

- **KVM/QEMU**
  - Virtio-Blk-Data-Plane
    - Para-virtualized I/O technique
  - Layers & Data Structures
    - Per-virtual-core threads
    - Various I/O layers
    - Numerous queues
    - I/O thread
  - Single Request queue
    - Shared by all Virtual CPUs
    - Frequent lock contentions
  - Single I/O thread
    - Executed by single core
    - Significant bottleneck



Architecture (1/5)

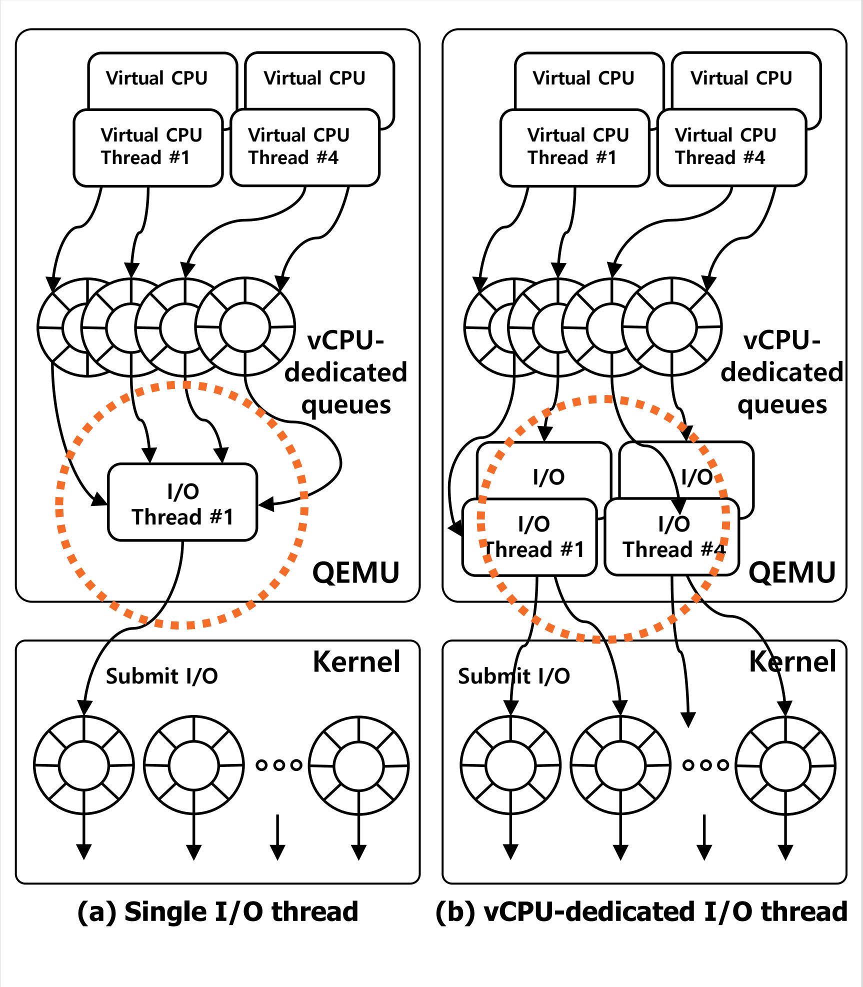
- **Virtio-Blk-Data-Plane (a)**
  - Single shared request queue
  - **Cause**: Single global mutex
  - Severe lock contentions among vCPUs
  - Wastes time for acquiring the lock
- **vCPU-dedicated queue (b)**
  - **Key**: Dedicated request queue per vCPU
  - Minimizes the lock contentions
  - Waiting time decreases by up to 80%



(a) Virtio-Blk-Data-Plane (b) vCPU-dedicated queue

Architecture (2/5)

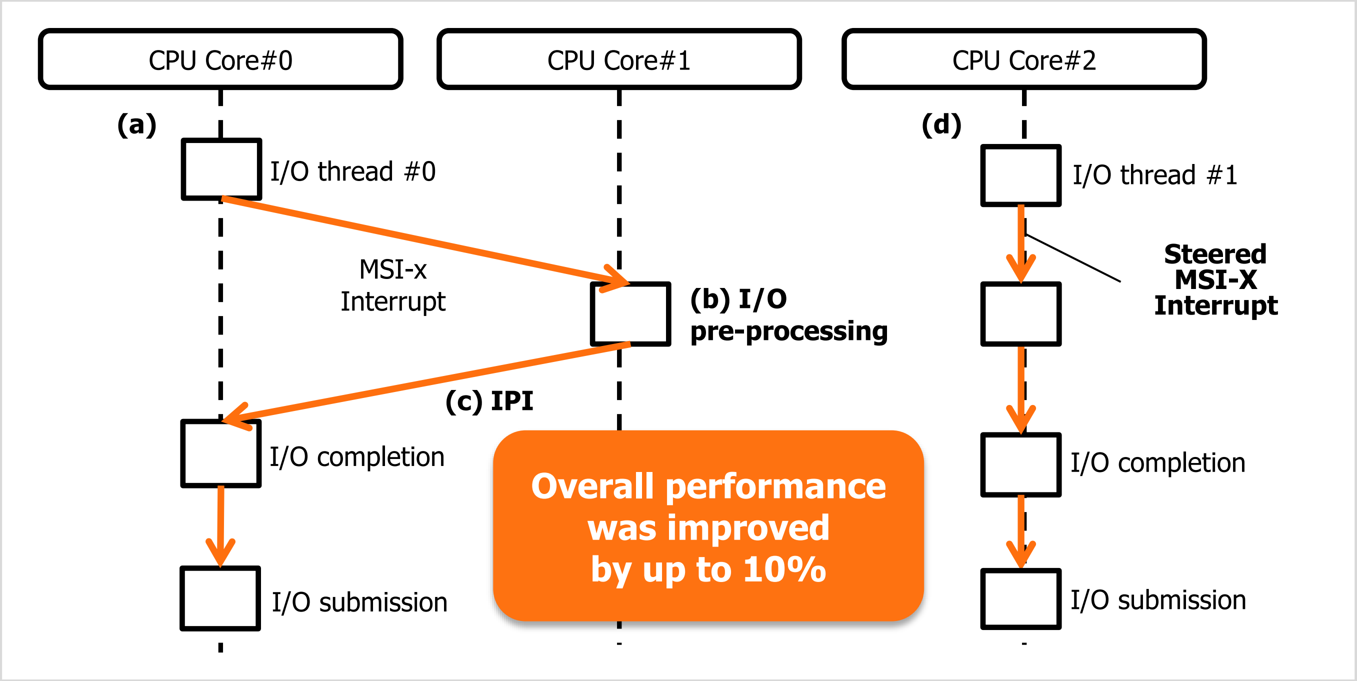
- **Single I/O thread (a)**
  - All I/O requests are inserted into one queue in the host
  - **Cause**: inefficient distribution caused by the single I/O thread
  - Disturbs I/O parallelism
- **vCPU-dedicated I/O thread (b)**
  - **Key**: Dedicated I/O thread per vCPU
  - I/O threads are executed by non-overlapping CPU core
  - Improves I/O parallelism



(a) Single I/O thread (b) vCPU-dedicated I/O thread

Architecture (3/5)

- **Configuring CPU Affinity for I/O Completion**
  - MSI-X is useless in single I/O thread architecture
  - Unnecessary context switches: (a) – (b) – (c)
  - Assign a single non-overlapping CPU per I/O thread
  - Improves cache hit rates and reduces scheduling overheads



CPU Core#0

CPU Core#1

CPU Core#2

(a) I/O thread #0

MSI-x Interrupt

(b) I/O pre-processing

(c) IPI

I/O completion

I/O submission

(d) I/O thread #1

Steered MSI-X Interrupt

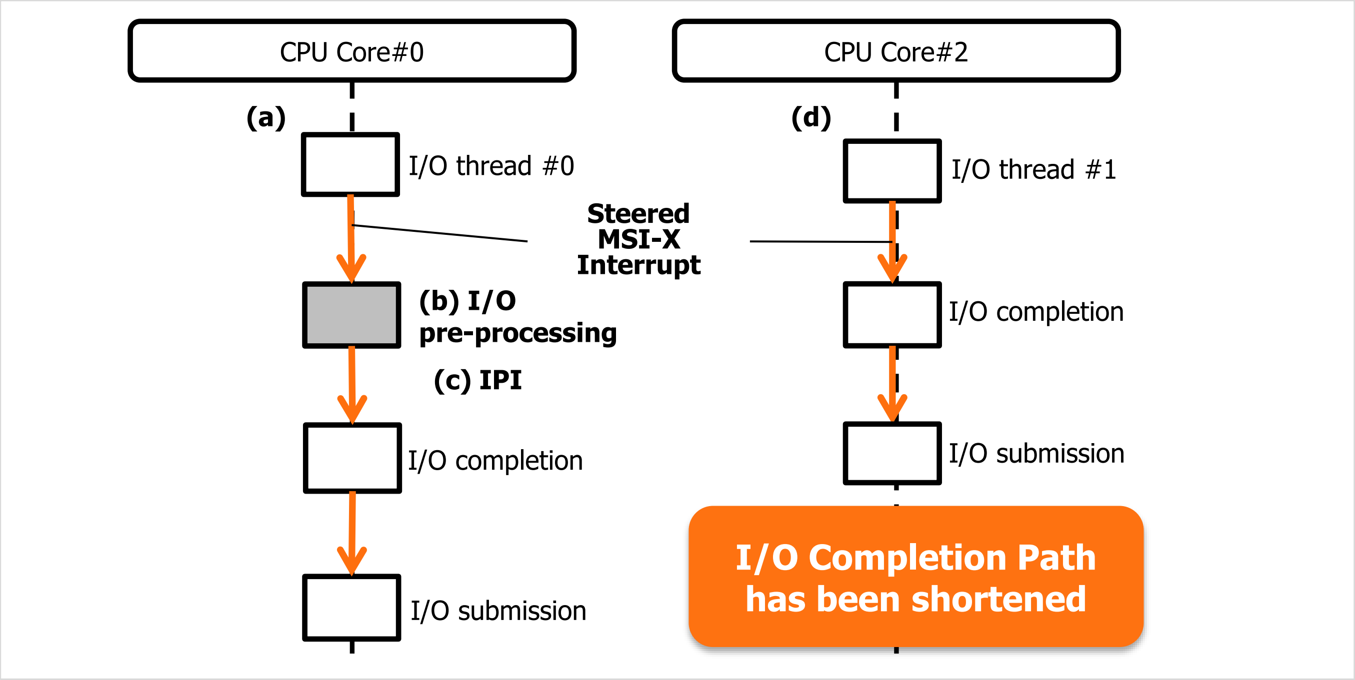
I/O completion

I/O submission

Overall performance was improved by up to 10%

Architecture (4/5)

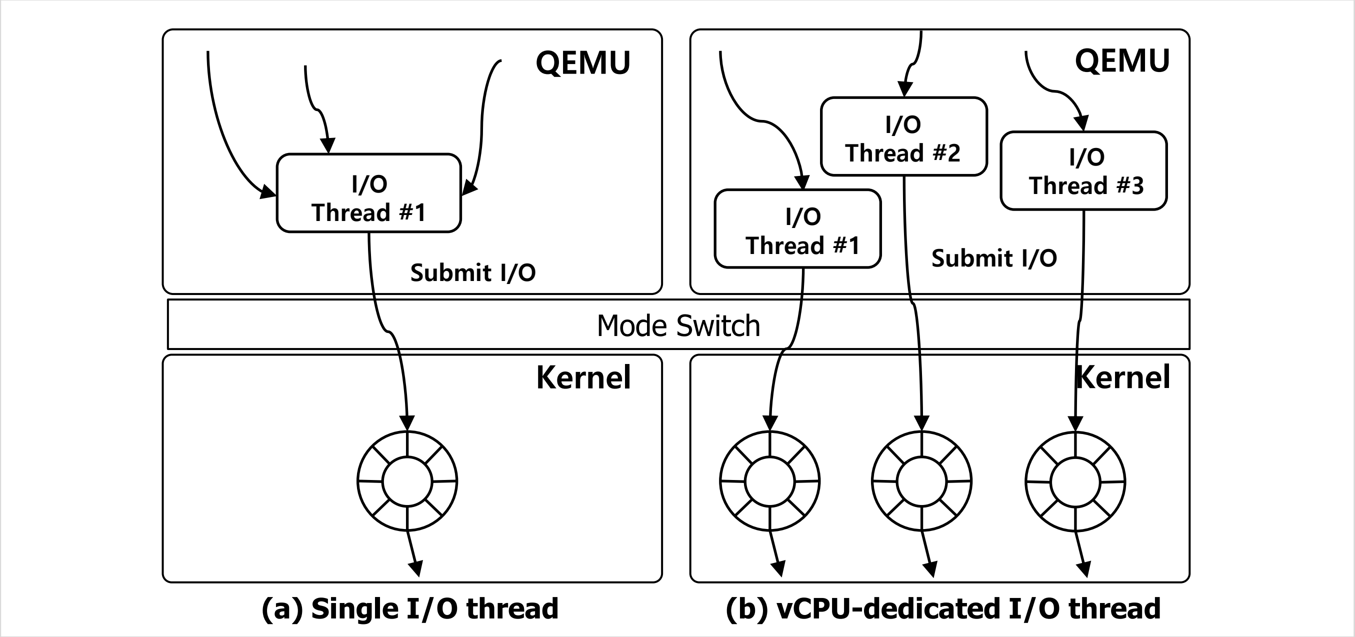
- **Eliminating Inter-process Interrupts (IPI)**
  - IPI steers I/O completion to the particular CPU
  - IPI requires additional interrupt scheduling
  - IPI can be entirely eliminated by steered MSI-X interrupt



I/O Completion Path has been shortened

Architecture (5/5)

- **I/O batch submission technique**
  - batches all I/O request and submits through one system call
- **Workload-aware I/O batch submission**
  - vCPU-dedicated I/O threads pollute the existing technique
  - Estimates intensiveness of the I/O workloads by the history
  - Waits for time to batch more, if intensive



(a) Single I/O thread (b) vCPU-dedicated I/O thread

Experiment (1/5)

- **Experimental Group**

Denotation	Features	Information
Baseline	Single queue, Single I/O thread	Unmodified QEMU 2.1.2
MQ	Multi-queue, Single I/O thread	Previous work by Ming Lei※
MIOT	vCPU-dedicated queues and I/O threads	Our approach

※ **Ming Lei,** “Virtio blk multi-queue conversions” , KVM Forum, 2014

- **Experimental Setup**

Setup	Contents
Host Machine	Intel i7-2600 quad-core CPU 3.40GHz, 16GB RAM
Target Device	Null block device, Samsung XS1715 1.5TB
Guest Machine	8 Virtual CPUs, 14GB RAM, Virtio-Blk-Data-Plane

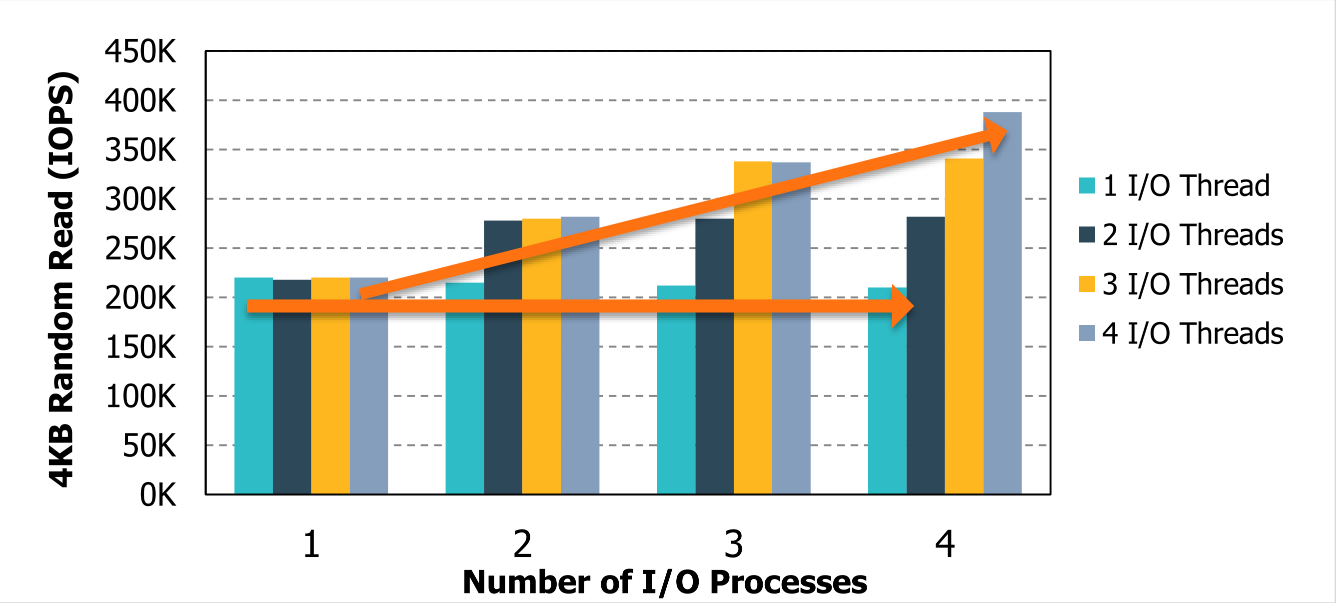
- **FIO benchmark for I/O workload**

FIO	Contents
Workload	4KB random read, 4KB random write, 32KB seq. read, 32KB seq. write
Config.	libaio, I/O depth: 32, non-cache mode, I/O processes: 1-8, 1GB data

Experiment (2/5)

- **Impact of the Number of I/O Threads**
  - Single VM with 4 virtual CPUs and 4 request queues
  - I/O threads is varied from 1 to 4
  - Specifying a non-overlapping CPU affinity for each I/O thread

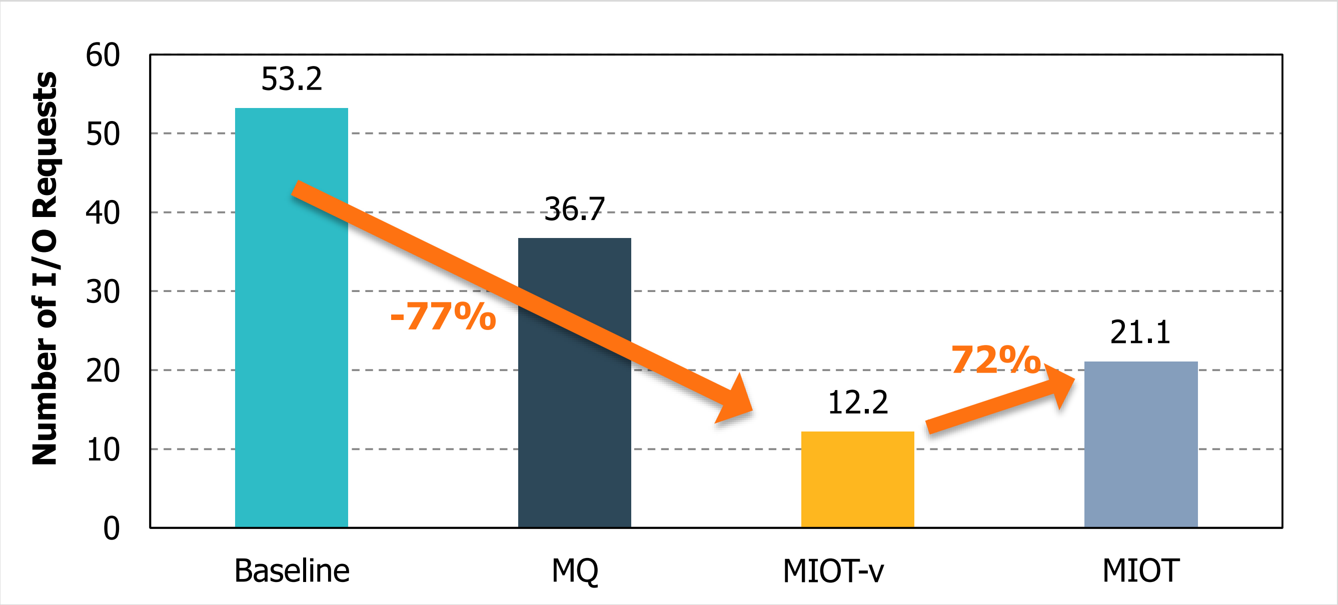
More I/O threads obviously contribute to higher performance



Experiment (3/5)

- **Effect of Workload-aware I/O batch submission**
  - Measure the number of I/O requests per system call
  - MIOT-v degraded the number of I/O request up to 77%
  - MIOT improves it up to 72% compared to MIOT-v

The overall performance was improved by up to 10%

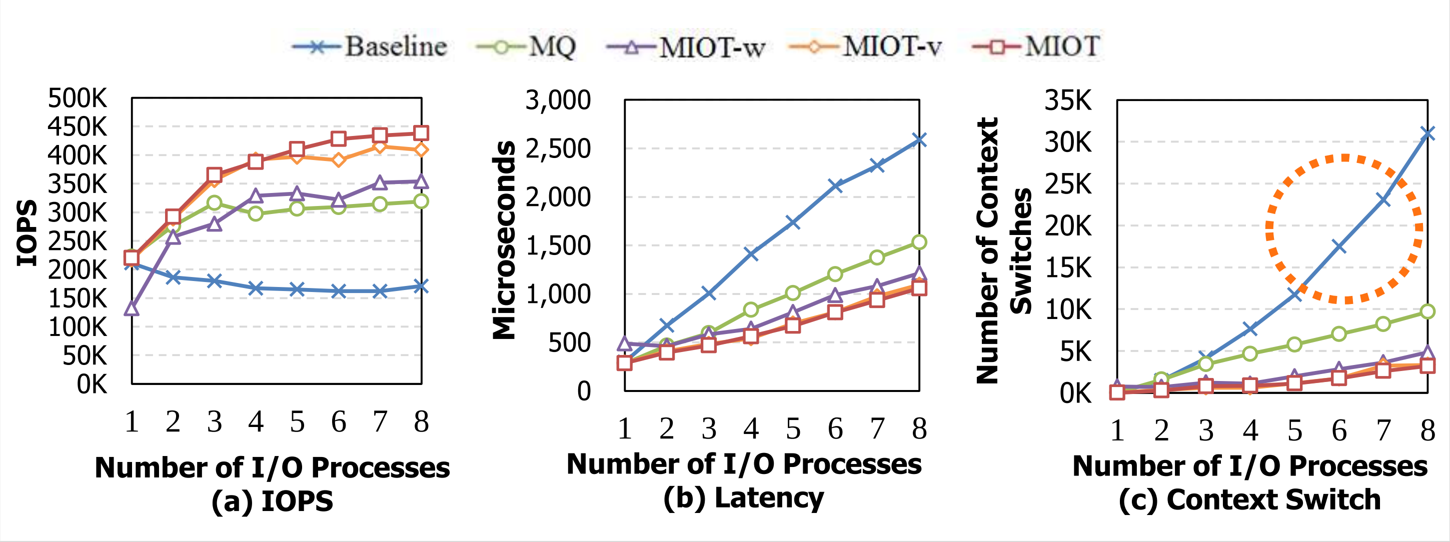


※ **MIOT-v:** MIOT without the Workload-aware I/O batch submission technique

Experiment (4/5)

- **Analysis of the Effect of Three Optimizations**
  - Measure IOPS, latency, and the number of context switches
  - Through 4KB random read of FIO microbenchmark
  - Each optimization has a positive impact on IOPS

The excessive context switches are the major cause of performance degradation



(a) IOPS

(b) Latency

(c) Context Switch

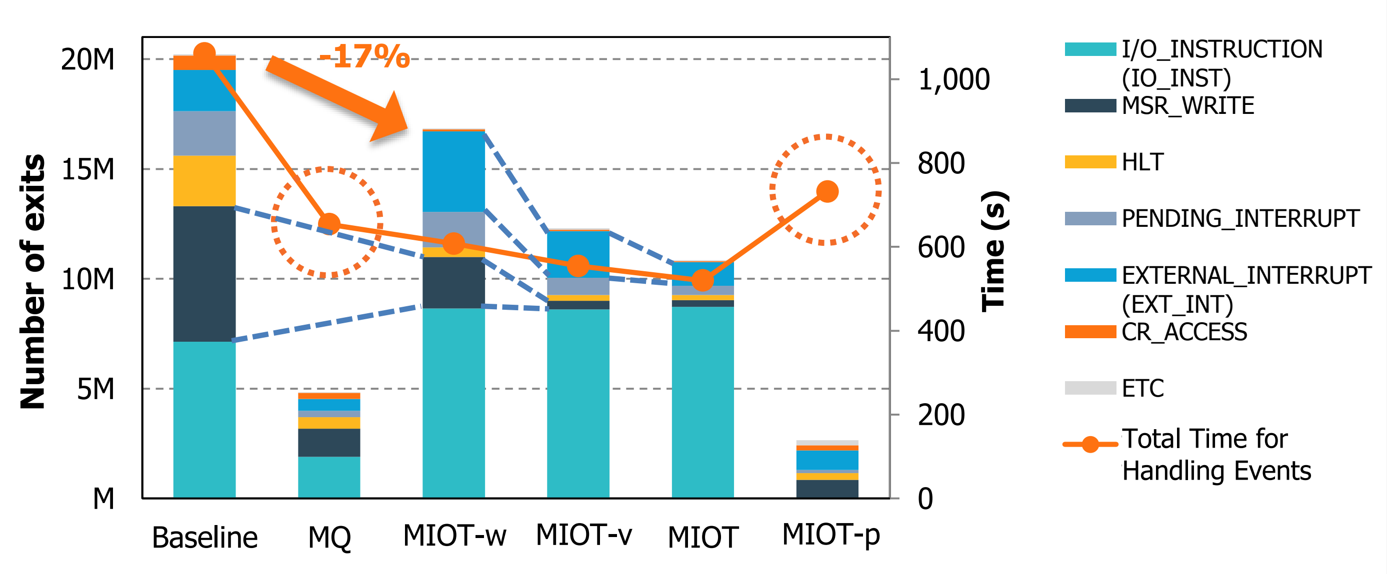
※ **MIOT-w**: MIOT without optimizations

※ **MIOT-v:** MIOT without Workload-aware I/O batch submission

Experiment (5/5)

- **Analysis of the Effect of Three Optimizations**
  - Measure the number of exits and total time for handling events through perf
  - The number of exits is mostly proportional to the total time
  - But not absolute in all cases such as MQ and MIOT-p

45% decrease of the number of exits by our approach

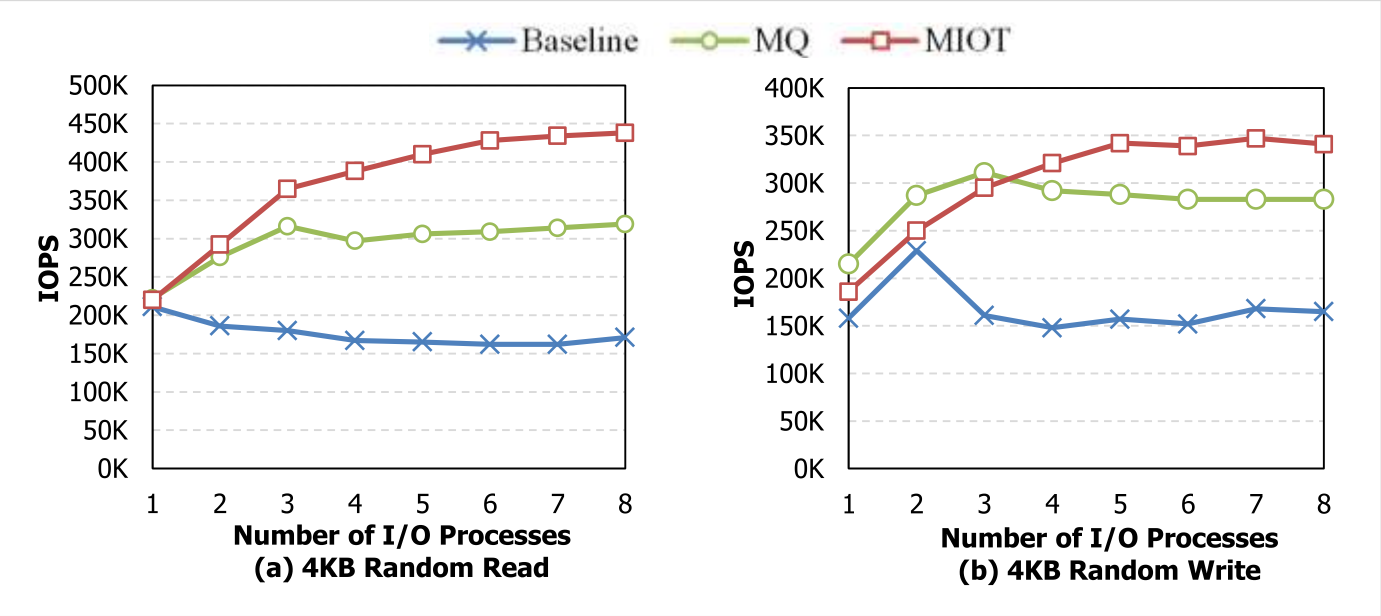


※ **MIOT-w**: MIOT without optimizations
※ **MIOT-v:** MIOT without Workload-aware I/O batch submission
※ **MIOT-p**: MIOT with polling mechanism

Evaluation (1/4)

- **IOPS on Null Block Device**
  - Both IOPS of Baseline were gradually decreased
  - Both IOPS of MQ were limited and still have I/O scalability issue
  - MIOT achieved up to 440K / 350K IOPS (Random read / Random write)

MIOT improved IOPS by up to 2.67x compared to Baseline
by up to 38% compared to MQ



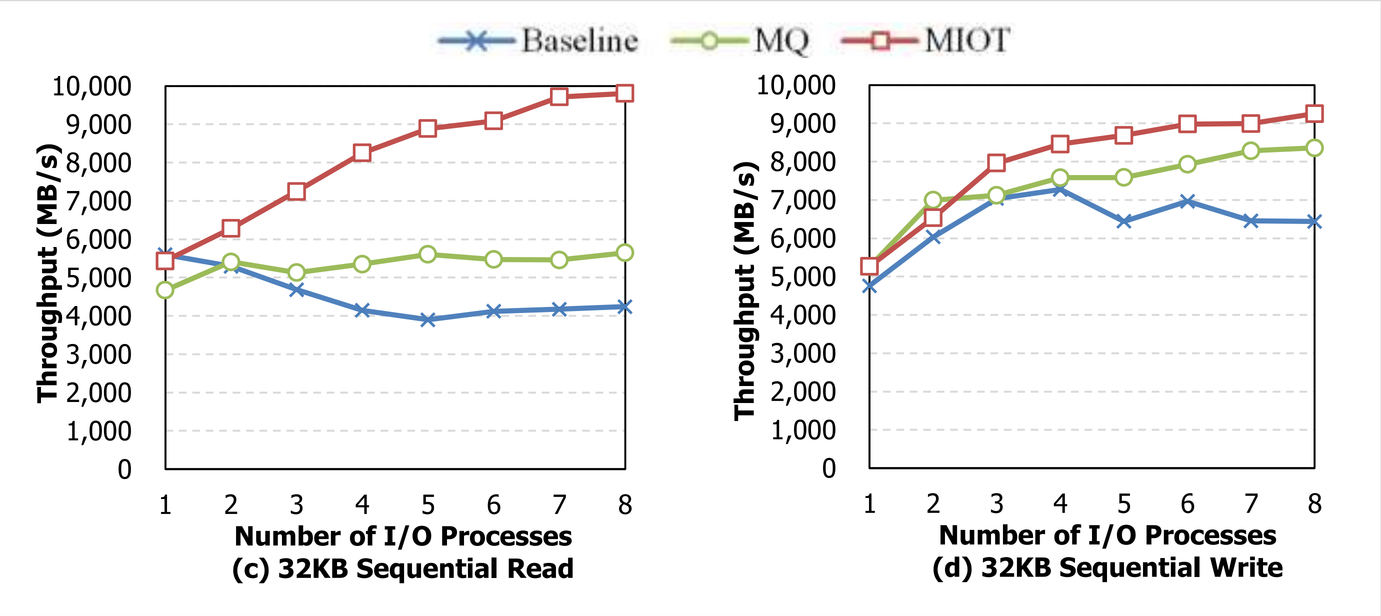
(a) 4KB Random Read

(b) 4KB Random Write

Evaluation (2/4)

- **Throughput on Null Block Device**
  - Seq. read throughput of Baseline was gradually decreased
  - The throughput of MQ on seq. read achieved little improvement
  - MIOT reached up to 9800 MB/s / 9200 MB/s (Seq. read / Seq. write)

MIOT improved throughput by up to 2.32x compared to Baseline
by up to 77% compared to MQ



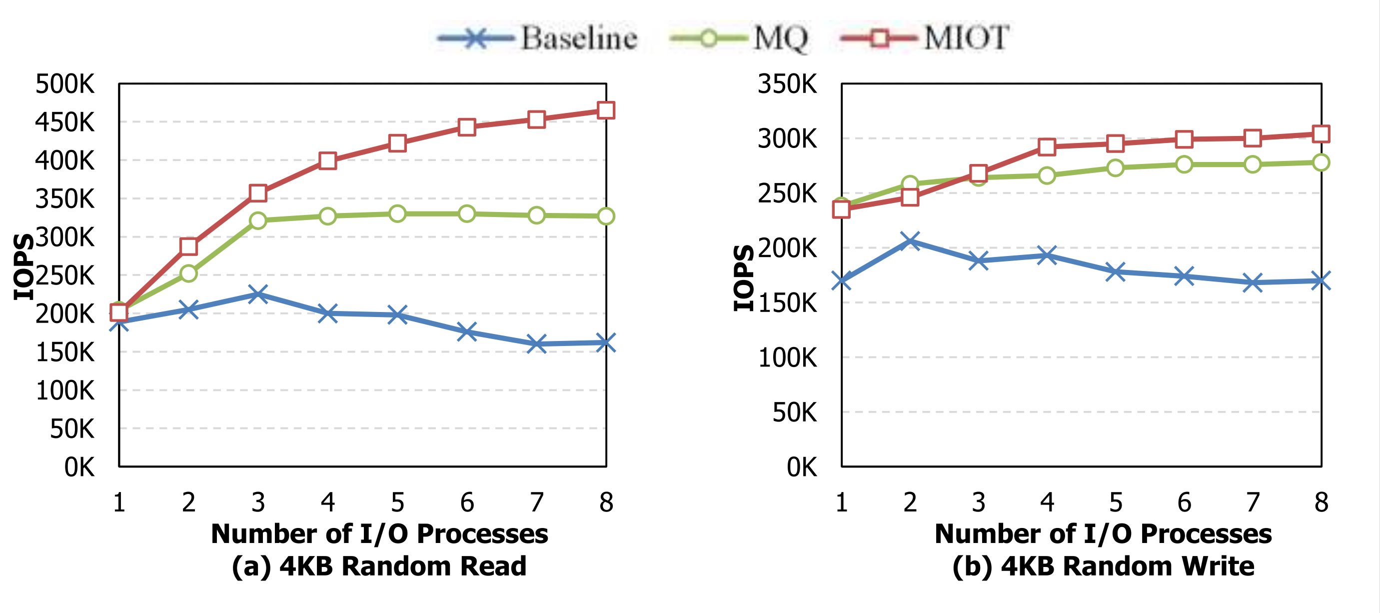
(c) 32KB Sequential Read

(d) 32KB Sequential Write

Evaluation (3/4)

- **IOPS on NVMe SSD**
  - Both IOPS of Baseline were gradually decreased
  - IOPS on random write is limited by native IOPS (350K IOPS) of the SSD
  - MIOT achieved up to 460K / 300K IOPS (Random read / Random write)

MIOT improved IOPS by up to 2.87x compared to Baseline
by up to 42% compared to MQ



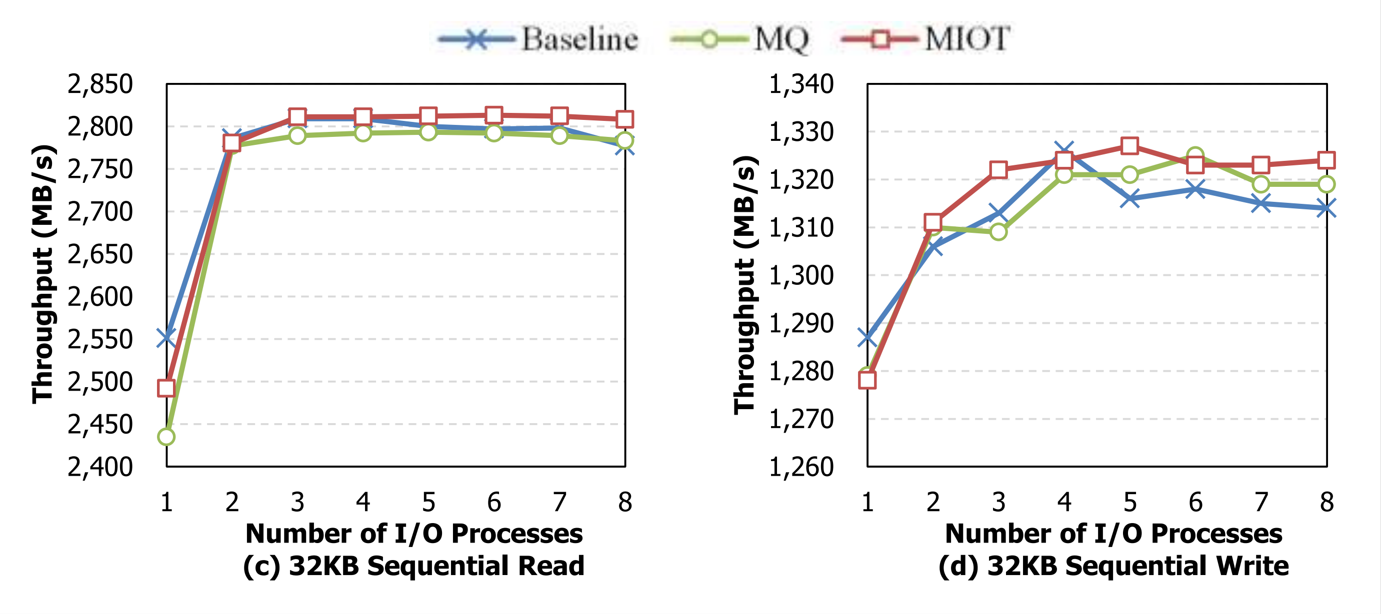
Number of I/O Processes
(a) 4KB Random Read

Number of I/O Processes
(b) 4KB Random Write

Evaluation (4/4)

- **Throughput on NVMe SSD**
  - All throughput is limited by native performance (3000 MB/s, 1400 MB/s)
  - MIOT gained little achievement in Sequential workloads
  - MIOT reached up to 2800 MB/s / 1300 MB/s (Seq. read / Seq. write)

The throughput improvements are now concealed on the NVMe SSD



(c) 32KB Sequential Read

(d) 32KB Sequential Write

Conclusion

Motivation

Existing QEMU cannot guarantee the performance of guest machines when a multi-queue SSD is used

Analysis

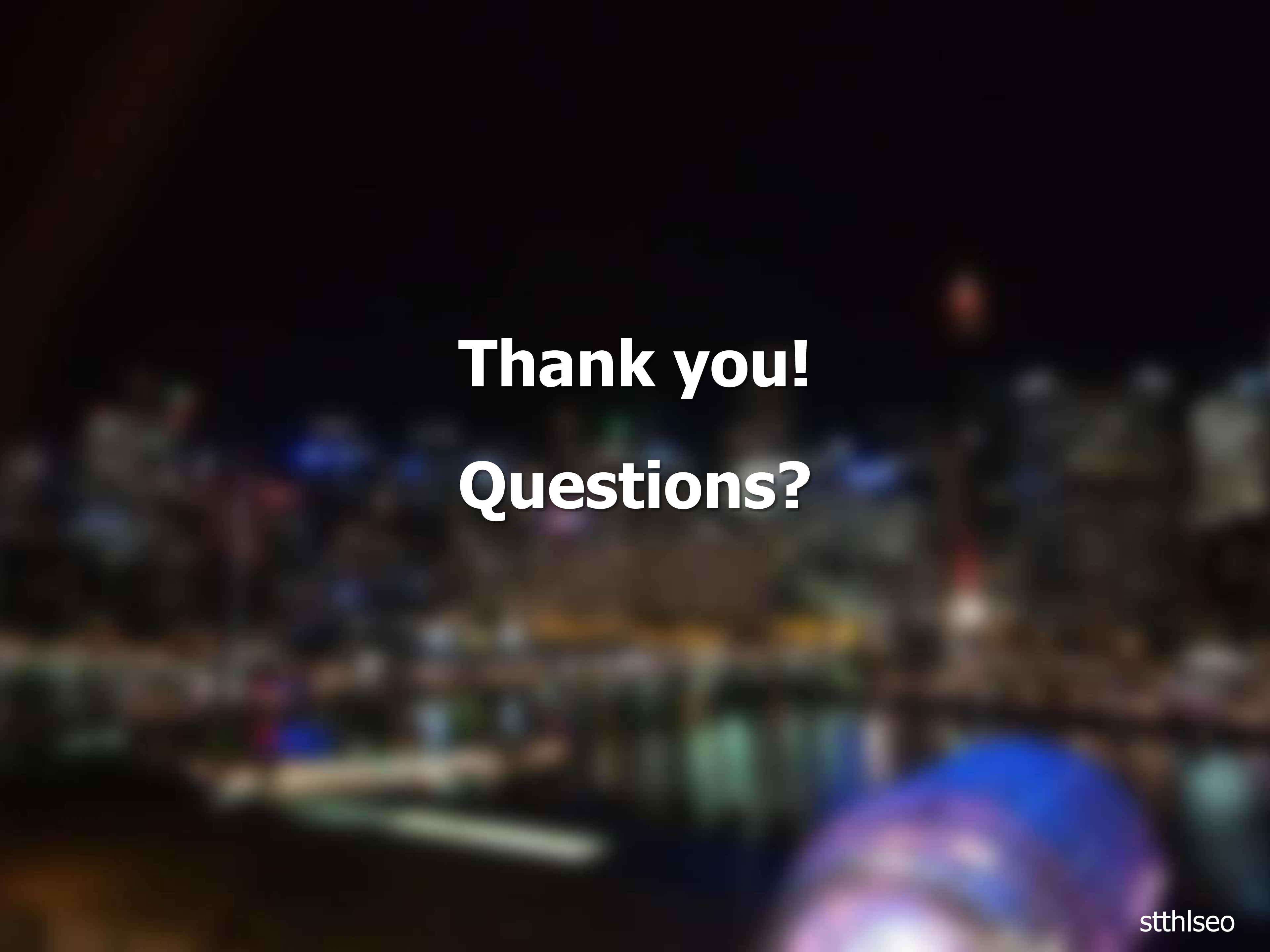
Guest machines suffered from lock contentions and parallelism issue

Solution

We proposed a novel approach, the design of vCPU-dedicated queues and I/O threads with three optimizations

Evaluation

IOPS performance was significantly improved by up to 2.67x, and the throughput was enhanced by up to 132%



Thank you!

Questions?

stthlseo