D2FQ: Device-Direct Fair Queueing for NVMe SSDs

Jiwon Woo, Minwoo Ahn, Gyusun Lee and Jinkyu Jeong
Sungkyunkwan University (SKKU)
Computer Systems Laboratory
SSDs can deliver \textbf{Million IOPS}

Being able to handle requests from multi-tenants
Conventional I/O Scheduling

SSDs can deliver **Million IOPS**

High CPU overhead

- CFQ [Linux]
- BFQ [Linux]
- FlashFQ [ATC ‘13]
- MQFQ [ATC ‘19]
Device-side I/O Scheduling

Saving host CPU cycles by offloading I/O scheduling function to device
Device-side I/O Scheduling

Some storage protocols already have device-side scheduling features

HIOS [SIGARCH ‘14]

FLIN [ISCA ‘18]
Device-side I/O Scheduling

App1  App2  App3  AppN

High
Medium
Low

RR
RR
RR

NVMe Weighted Round-Robin

RR
WRR

OS
Device-side I/O Scheduling

1. I/O handling frequency is only adjustable parameter
2. No consideration on I/O size
3. Supporting only three priority classes

App1 -> App2 -> App3 -> AppN

High

Low

NVMe Weighted Round-Robin

RR

RR
Our Approach

D2FQ: Device-Direct Fair Queueing for NVMe SSDs

A low CPU overhead fair queueing I/O scheduler built on top of NVMe WRR

NVMe Weighted Round-Robin
Virtual Time-based Fair Queueing

Virtual time = $\frac{\sum I/O \text{ size}_{\text{completed}}}{I/O \text{ weight}}$
Virtual Time-based Fair Queueing

Satisfy fairness by equalizing virtual time of flows

Virtual time = \sum \frac{I/O size_{completed}}{I/O weight}

App1
App2
App3
App4
Virtual Time-based Fair Queueing

虚時の公平なキュー・ミング

Virtual time = \( \frac{\sum \text{I/O size}_{\text{completed}}}{\text{I/O weight}} \)
Virtual Time-based Fair Queueing – D2FQ

Virtual time = \[ \frac{\sum I/O \text{ size}_{\text{completed}}}{I/O \text{ weight}} \]
Virtual Time-based Fair Queueing – D2FQ

Virtual time

\[ \text{Virtual time} = \frac{\sum I/O \text{ size}_\text{completed}}{I/O \text{ weight}} \]

Submission = Dispatch

Directly dispatch to device
Virtual Time-based Fair Queueing – D2FQ

Need to slowdown

Global virtual time ($gvt$): Minimum virtual time value among active flows

Need to catch up others

Throttle flows whose virtual time is far ahead of $gvt$
Virtual Time-based Fair Queueing – D2FQ

NVMe WRR

RR

RR

RR

RR
Virtual Time-based Fair Queueing – D2FQ

Virtual time

- Direct request dispatch

\[ gvt \]

\[ \tau_m \]

\[ \tau_l \]

\[ Q_{high} \]

\[ Q_{mid} \]

\[ Q_{low} \]

App1

App2

App3

App4

Virtual time

\[ if, vt_{flow} - gvt < \tau_m \]

else if, \( vt_{flow} - gvt < \tau_l \)

Otherwise
# D2FQ Challenges

## How to obtain sufficient I/O processing speed difference

*Dynamic HL ratio adjustment*

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which flow should be selected for I/O throttling?</td>
<td>Setting the queue class thresholds ((\tau_m, \tau_l))</td>
</tr>
<tr>
<td></td>
<td>Please see the paper</td>
</tr>
<tr>
<td>How to manage gvt scalably?</td>
<td>Sloppy minimum tracking</td>
</tr>
</tbody>
</table>

*Please see the paper*
Dynamic HL Ratio Adjustment

- **HL ratio**
  - Ideal ratio of I/O processing speed between high and low queues
  - Ability to regulate virtual time process

  e.g.)

  ![Diagram showing HL ratio with different weights]

  - High weight: 4
  - Medium weight: 2
  - Low weight: 1
Dynamic HL Ratio Adjustment

- HL ratio
  - Ideal ratio of I/O processing speed between high and low queues
  - Ability to regulate virtual time process

  e.g.)

  High weight : 4

  Most important factor to achieve I/O fairness

  Low weight : 1
Dynamic HL Ratio Adjustment

- HL ratio
  - Ideal ratio of I/O processing speed between high and low queues
  - Ability to regulate virtual time process

**Low HL ratio**
- Small ability to regulate virtual time progression
- Need to set a proper HL ratio value dynamically
- may violate fairness

**High HL ratio**
- Takes too long to process requests in low priority queues
- may incurs high tail latency

Need to set a proper HL ratio value dynamically
Dynamic HL Ratio Adjustment

Increasing HL ratio

- Detect unfairness with $\tau_w$
- Calculate the additional I/O throttling capability to provide fairness
  
  - Calculate the delta of virtual time ($\Delta vt$) last time period
  - Current system requires at least \( \frac{\Delta vt_{max}}{\Delta vt_{min}} \) times additional throttling capability

\[
HL Rationext = \left\lfloor \frac{\Delta vt_{max}}{\Delta vt_{min}} \times HL Ratio_{prev} \right\rfloor + 1
\]
Dynamic HL Ratio Adjustment

**Decreasing HL ratio**

- Occur when fairness is satisfied
  - Maximum virtual time gap is below $\tau_w$
- Calculate slowdown of each flow
  - Required throttling capability of system to satisfy fairness between a flow and the slowest flow
- Set next HL ratio as the largest slowdown among all active flows

\[
\text{slowdown}(f) = \frac{\text{Estimated bandwidth of flow } f \text{ when using high queues only}}{\text{Actual bandwidth of flow } f}
\]

\[
H/Lratio_{next} = \text{MAXIMUM}(\text{slowdown}(f))
\]
# Evaluation

## Experimental configuration

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
</table>
| **CPU**        | Intel Xeon Gold 5112 3.6 GHz  
8 physical cores (Hyperthreading off) |
| **OS**         | Ubuntu 18.04.4 |
| **Base kernel**| Linux 5.3.10 |
| **Memory**     | DDR4 192 GB |
| **Storage device** | Samsung SZ985 800 GB Z-SSD |
| **Target fair I/O schedulers** | None / **D2FQ** / **MQFQ[ATC’19]** / BFQ [Linux] |
| **Workloads**  | Microbenchmark: FIO (libaio engine)  
Realistic workload: YCSB on RocksDB |
MQFQ and D2FQ achieve fairness while fully utilizing device bandwidth
D2FQ reduced CPU utilization by up to 45% compared to MQFQ
Dynamic HL Ratio Adjustment

- Compare I/O performance with three HL ratio setups
  - Static-3, Static-128, dynamic (D2FQ-default)
  - # of flows increase with event1 and event2
  - Flows have different weights (1 vs 3)

<table>
<thead>
<tr>
<th>Run time (sec)</th>
<th>I/O weight</th>
<th># of flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>base 0 - end</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>event1 10 - end</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>event2 20 - end</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
- Static-128 and our scheme (dynamic) achieve fairness
- Static-3 fails to achieve fairness because HL ratio of 3 is too small

• Runtime change of HL ratio in our scheme (dynamic)
Dynamic HL Ratio Adjustment

- Tail latency

- Dynamic shows low tail latency as compared to static-128
- **Single thread high queue-depth I/O performance**

- **D2FQ shows low CPU usage & high I/O performance (latency and bandwidth)**
- **D2FQ can be combined with AIOS [ATC'19], low-latency block-layer bypassing scheme**
  - LL-D2FQ shows lowest CPU usage and highest I/O performance
D2FQ Conclusion

▪ A low CPU overhead fair queueing I/O scheduler built on top of NVMe WRR

▪ Fair queueing with high scheduling performance
  • Reducing CPU utilization by up to 45%
  • Fully utilizing bandwidth & showing low latency
  • Enhanced scalability

▪ Vitalizing block-layer-bypass schemes (e.g., AIOS [ATC’19])
  • Their low-latency I/O performance is now augmented with fair I/O scheduling

Source Code:
https://github.com/skkucsl/d2fq
Thank you

Contact:
Jiwon Woo - jiwon.woo@csi.skku.edu
Minwoo Ahn - minwoo.ahn@csi.skku.edu
Gyusun Lee - gyusun.lee@csi.skku.edu
Jinkyu Jeong - jinkyu@skku.edu

Source Code:
https://github.com/skkucsld2fq