Improving I/O Resource Sharing of Linux Cgroup for NVMe SSDs on Multi-core Systems

USENIX HotStorage 2016

Sungyong Ahn*, Kwanghyun La*, Jihong Kim**
*Memory Business, Samsung Electronics Co., Ltd.
**Seoul National University
Outline

- Introduction
- Motivation
- Contributions
- Weight-based dynamic throttling (WDT) scheme
- Experimental Results
- Conclusion
Outline

- Introduction
- Motivation
- Contributions
- Weight-based dynamic throttling (WDT) scheme
- Experimental Results
- Conclusion
OS-level Virtualization

Multiple isolated instances (containers) running on a single host.
Multiple isolated instances (containers) running on a single host.

- Hardware resources should be isolated and allocated to containers
Multiple isolated instances (containers) running on a single host.

- Hardware resources should be isolated and allocated to containers
- Different resource requirements should be satisfied

I have CPU-bound jobs

I have I/O-bound jobs
Linux Control Groups (Cgroups)

Kernel-level resource manager of Linux

I have CPU-bound jobs

I have I/O-bound jobs
I/O bandwidth is shared according to I/O weights

Proportional I/O scheme in Linux Cgroups

Ideal proportional I/O sharing

Normalized I/O bandwidth
Outline

- Introduction
- Motivation
- Contributions
- Weight-based dynamic throttling (WDT) scheme
- Experimental Results
- Conclusion
Questions

When Linux Cgroups work with NVMe SSD,

1. Can they share the I/O resource in proportion to I/O weights?
2. Is it scalable?
When Linux Cgroups work with NVMe SSD,

1. Can they share the I/O resource in proportion to I/O weights?
2. Is it scalable?
Proportional I/O with NVMe SSDs

Existing Cgroups cannot support the proportional I/O to NVMe SSDs

[Diagram showing normalized I/O bandwidth with different weights for containers A, B, C, and D.]

- Container A: weight=10
- Container B: weight=5
- Container C: weight=2.5
- Container D: weight=1

Linux Cgroups

Hardware

Normalized I/O bandwidth

BASELINE

- C(10): 10
- C(5): 5
- C(2.5): 2.5
- C(1): 1.0
NVMe SSDs have different I/O stack from SATA storage

- Existing proportional I/O scheme is implemented in single queue block layer.
First Attempt: Using the Existing Static Throttling

**Upper limit of I/O bandwidth**
- Limit the maximum number of bytes or I/O requests for particular time interval (throttling window)
Static throttling is not enough to support the proportional I/O

First Attempt: Using the Existing Static Throttling

- **Static Throttling**
  - Upper limit = 10 (Container A)
  - Upper limit = 5 (Container B)
  - Upper limit = 2.5 (Container C)
  - Upper limit = 1 (Container D)

- **Linux**
  - Groups I/O throttling
  - Static limit

- **Single-queue Block Layer**
  - Proportional I/O (CFQ)

- **Multi-queue Block Layer**

- **Hardware**
  - Serial ATA
  - NVM EXPRESS

**Chart:**
- **Normalized I/O Bandwidth**
  - C(10)
  - C(5)
  - C(2.5)
  - C(1)

**Graph:**
- Static Throttling
  - 9.9
  - 2.2
  - 1.2
  - 1.0
Because...

I/O workloads fluctuate with time

Maximum read bandwidth of container A

Container A cannot fully use the allocated bandwidth

Maximum read bandwidth of container B
I/O workloads fluctuate with time

Because...

Maximum read bandwidth of container A

Container A cannot use the allocated bandwidth

Maximum read bandwidth of container B
Outline

- Introduction
- Motivation
- **Contributions**
- Weight-based Dynamic throttling (WDT) scheme
- Experimental Results
- Conclusion
We achieved the proportional I/O for NVMe SSDs.

We achieved the scalable performance of Linux Cgroups.
Outline

- Introduction
- Motivation
- Contributions
- Weight-based Dynamic throttling (WDT) scheme
- Experimental Results
- Conclusion
Overview of WDT Scheme

Distributing the credits to containers according to I/O weights

To update TotalCredit, future I/O demand is predicted
All containers are allocated credits in proportion to their I/O weight.

Container A
I/O weight = 10

Container B
I/O weight = 5

Total Credit = 15
All containers are allocated credits in proportion to their I/O weight. Credits are replenished periodically.
All containers are allocated credits in proportion to their I/O weight. Credits are replenished periodically. If a container has no available credit, it is throttled.
All containers are allocated credits in proportion to their I/O weight. Credits are replenished periodically. If a container has no available credit, it is throttled.

Budget Distributor

---

- Container A: I/O weight = 10
- Container B: I/O weight = 5
In order to remove storage idle time, TotalCredit is adjusted.
Technical Challenge

✓ How to predict **TotalCredit** required for the next interval?
Technical Challenge

✓ How to predict future I/O demand for the next interval?
Future I/O Demand Predictor

- Monitoring I/O demand of each container for every interval
  - Prediction of the future I/O demand from cumulative distribution function
    - 80th percentile of a cumulative distribution of I/O demand (assuming normal distribution)

```
weight \( w_1 \)
Container \( C_1 \)

weight \( w_2 \)
Container \( C_2 \)

weight \( w_3 \)
Container \( C_3 \)

Data flow

- Credit allocation
- CPM Monitoring

WDT

\[ B_1^j + R_1^j > U_1^j \]
\[ B_2^j + R_2^j > U_2^j \]
\[ B_3^j + R_3^j > U_3^j \]

Monitoring \( CPM_1 \)
Monitoring \( CPM_2 \)
Monitoring \( CPM_3 \)

Budget Distributor

TotalCredit

TotalCredit Updater

Residual Credits Carryover

Future I/O Demand Predictor

Block Layer

Figure 2
The 80th Percentile

\( z \)
When Linux Cgroups work with NVMe SSD,

1. Can they share the I/O resource in proportion to I/O weights?
2. Is it scalable?
Scalability problem of the existing Cgroups throttling layer

Scalability of the Existing Cgroups on NUMA

The number of NUMA nodes

1 node 2 nodes 3 nodes 4 nodes

KIOPS

C1 C2 C3 C4
All containers share a single request_queue lock across NUMA nodes

- Lock contention
- Remote memory accesses to the lock state
- Cacheline invalidations caused by cache coherence protocol

The number of NUMA nodes

CPU cache miss ratio (%)
We adopt fine-grained per-container locks

The cache miss ratio decreases to **12.8%** from 38.2%
Outline

- Introduction
- Motivation
- Contributions
- Weight-based dynamic throttling (WDT) scheme
- Experimental Results
- Conclusion
Experiment Setup

- Linux kernel 4.0.4 (modified)
- Dell 4-nodes NUMA machine
- Samsung NVMe SSDs
- Block I/O traces replayer
  - UMass trace repository
  - SNIA IOTTA repository

Dell R920 Server
(4-nodes NUMA machine)

Samsung XS1715 NVMe SSDs
Result 1: Proportional I/O Support

WDT scheme satisfies the proportional sharing requirements
Result 2: Performance Scalability

- WDT- : Using single spin lock
- WDT : Using per-container locks

![Bar chart showing performance comparison between WDT- and WDT.](chart)

- **WDT-** using single spin lock shows I/O bandwidths of:
  - C1: 1333 MB/s
  - C2: 670 MB/s
  - C3: 334 MB/s
  - C4: 133 MB/s

- **WDT** using per-container locks shows I/O bandwidths of:
  - C1: 1762 MB/s
  - C2: 881 MB/s
  - C3: 440 MB/s
  - C4: 176 MB/s

**30% performance improvement**
Outline

- Introduction
- Motivation
- Contributions
- Weight-based dynamic throttling (WDT) scheme
- Experimental Results
- Conclusion
Conclusion

- Proposed the weight-based dynamic throttling scheme to support proportional I/O sharing for NVMe SSDs.
- Proposed the per-container locks for scalable performance.
Thank you