PERFORMANCE ANALYSIS OF CONTAINERIZED APPLICATIONS ON LOCAL AND REMOTE STORAGE

Qiumin Xu[1], Manu Awasthi[2], Krishna T. Malladi[3], Janki Bhimani[4], Jingpei Yang[3], Murali Annavaram[1]

[1]USC, [2]IIT Gandhinagar [3]Samsung [4]Northeastern

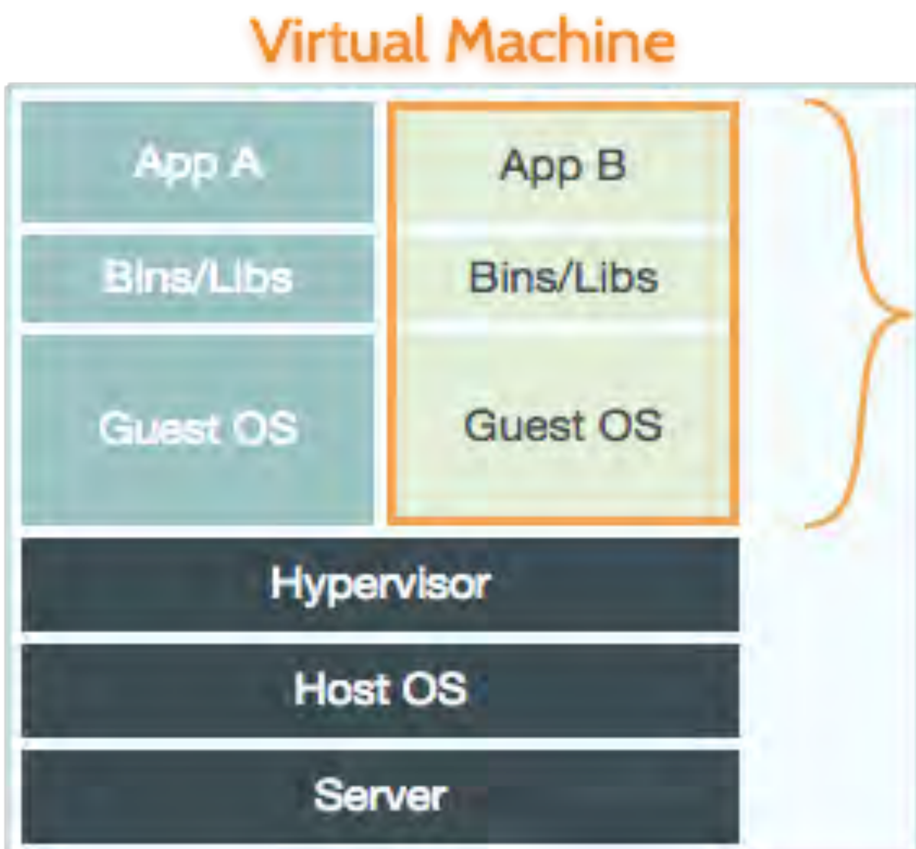
Docker Becomes Very Popular

- Software container platform with many desirable features
  - Ease of deployment, developer friendliness and light virtualization
- Mainstay in cloud platforms
  - Google Cloud Platform, Amazon EC2, Microsoft Azure
- Storage Hierarchy is the key component
  - High Performance SSDs
  - NVMe, NVMe over Fabrics

Agenda

- Docker, NVMe and NVMe over Fabrics (NVMf)
- How to best utilize NVMe SSDs for single container?
  - Best configuration performs **similar to raw performance**
  - Where do the performance anomalies come from?
- Do Docker containers scale well on NVMe SSDs?
  - Exemplify using **Cassandra**
  - Best strategy to divide the resources
- Scaling Docker containers on NVMe-over-Fabrics

What is Docker Container?



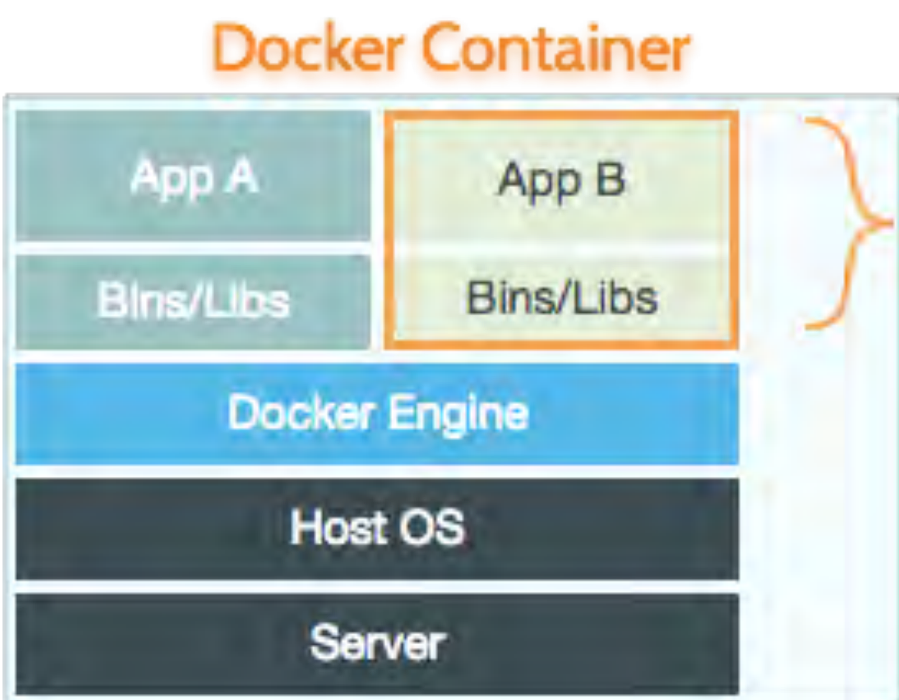
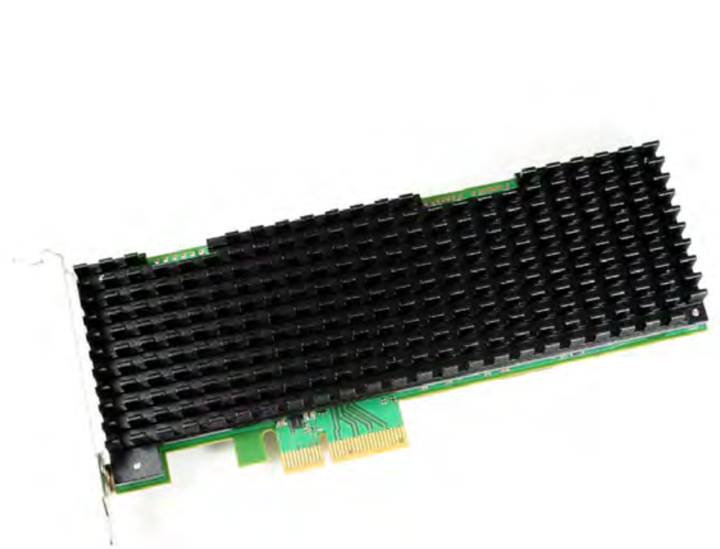


figure from https://docs.docker.com

- Each virtualized application includes an entire OS (~10s of GB)
- Docker container comprises just application and bins/libs
- Shares the kernel with other container
- Much more portable and efficient

Non-Volatile Memory Express (NVMe)

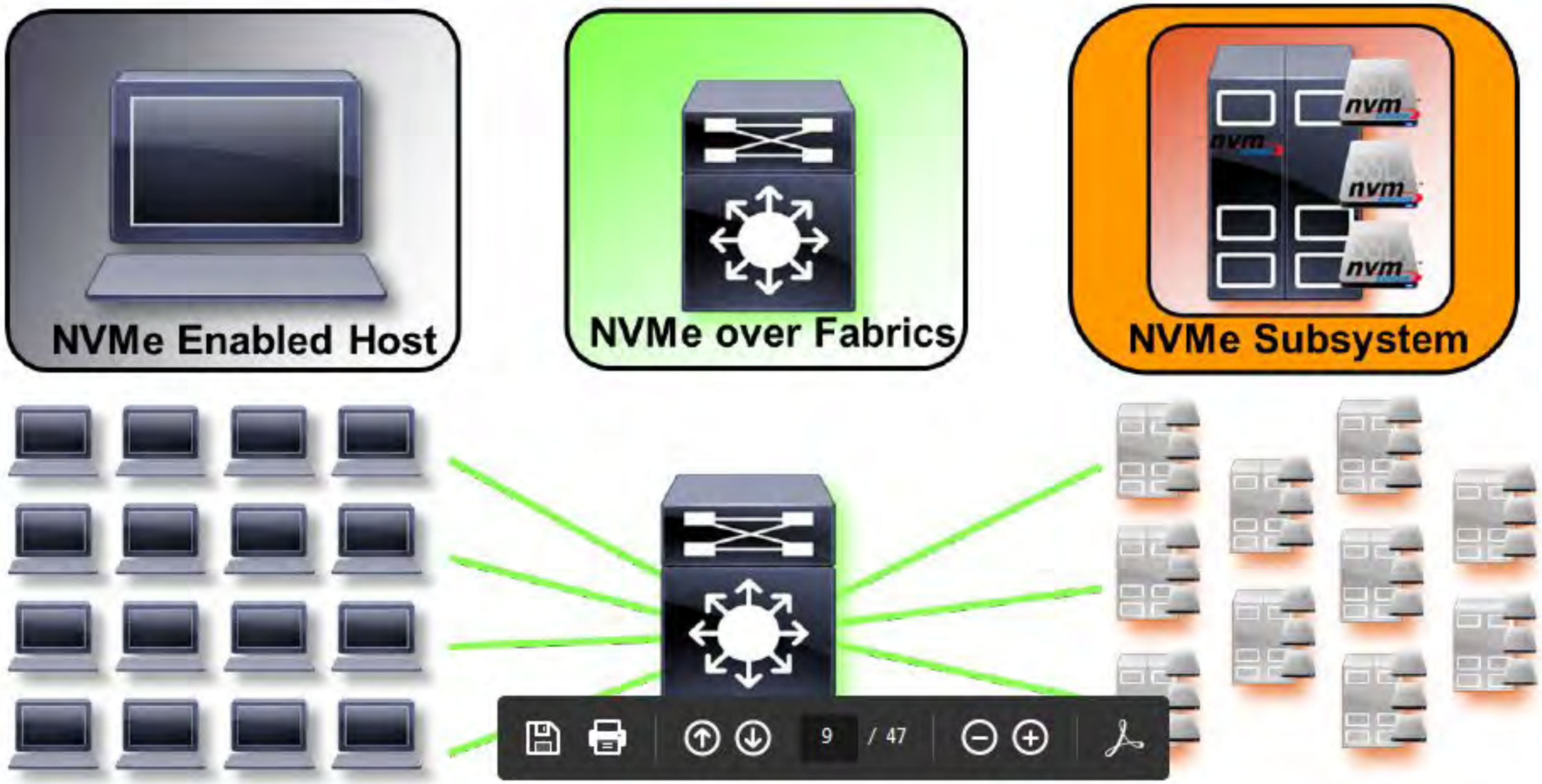
- A storage **protocol standard** on top of PCIe
- NVMe SSDs connect through PCIe and support the standard
  - Since 2014 (Intel, Samsung)
  - Enterprise and consumer variants
- NVMe SSDs leverage the interface to deliver superior perf
  - 5X to 10X over SATA SSD[1]



[1] Qiumin Xu et al. “Performance analysis of NVMe SSDs and their implication on real world databases.” SYSTOR’15

Why NVMe over Fabrics (NVMf)?

- Retains NVMe performance over network fabrics
- Eliminate unnecessary protocol translations
- Enables low latency and high IOPS remote storage

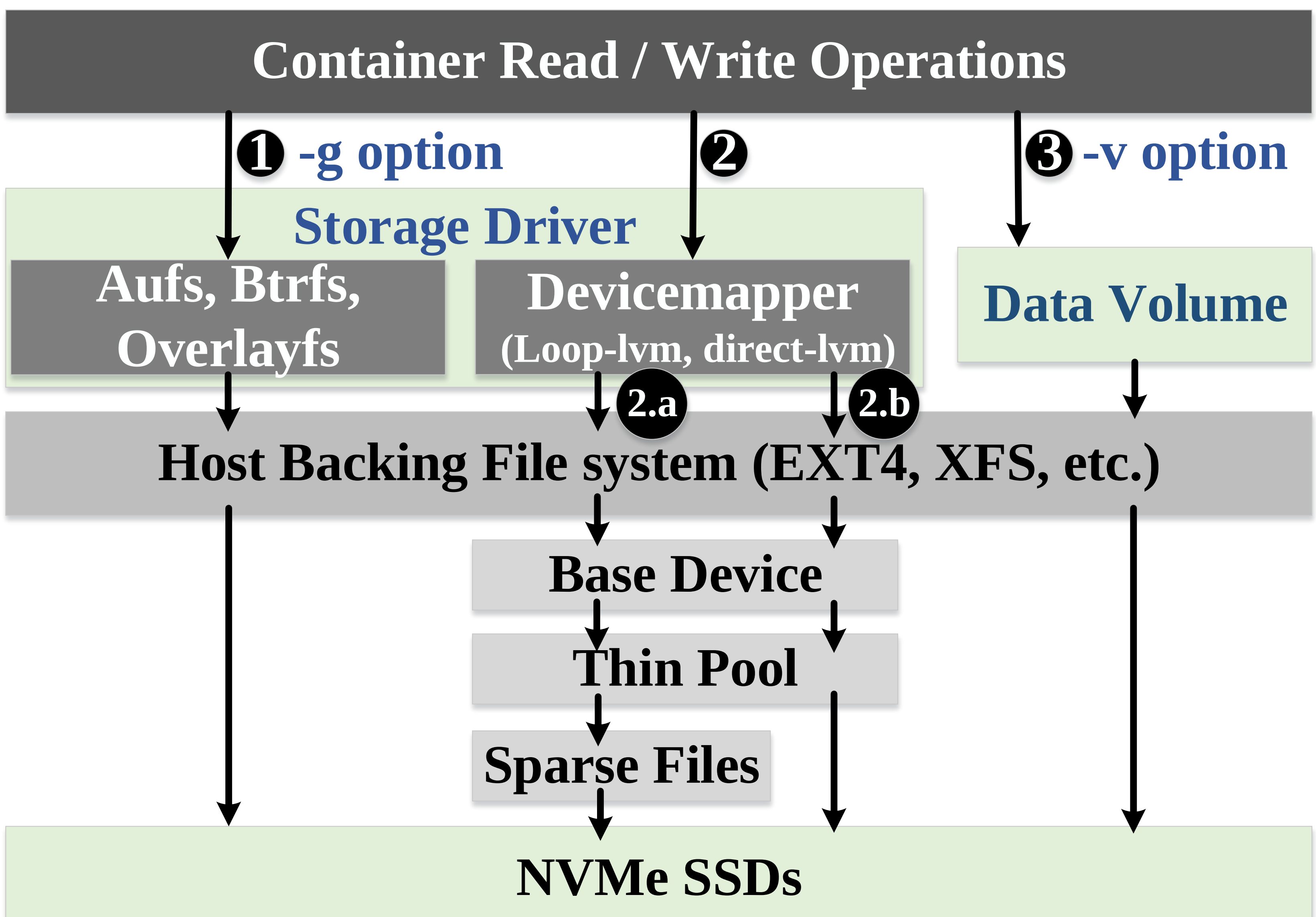


J. M. Dave Minturn, "Under the Hood with NVMe over Fabrics,", SINA Ethernet Storage Forum

Storage Architecture in Docker

Storage Options:

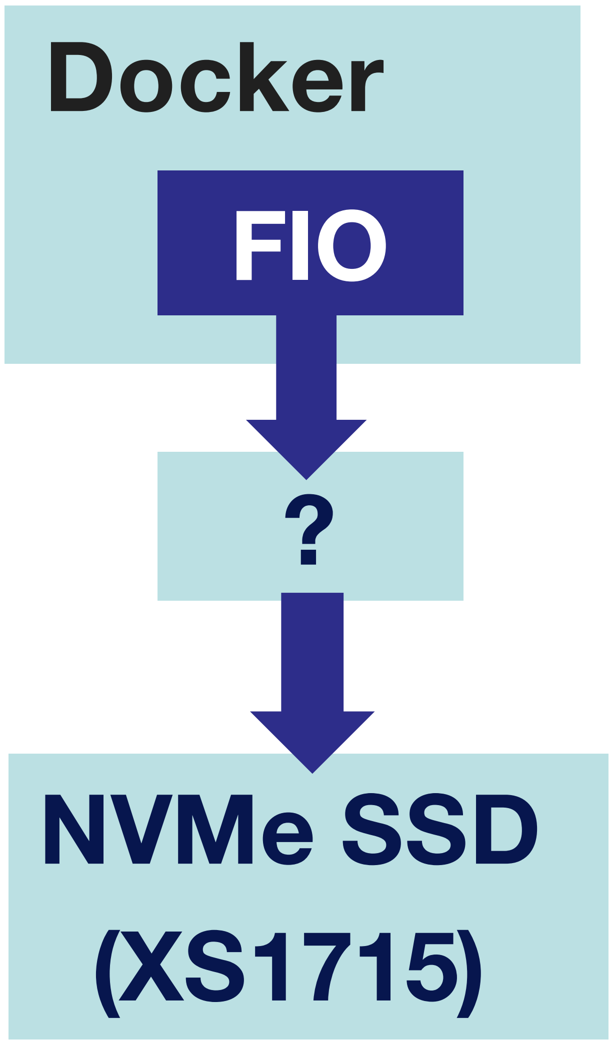
1. Through Docker Filesystem (Aufs, Btrfs, Overlayfs)
2. Through Virtual Block Devices (2.a Loop-lvm, 2.b Direct-lvm)
3. Through Docker Data Volume (-v)



Optimize Storage Configuration for Single Container

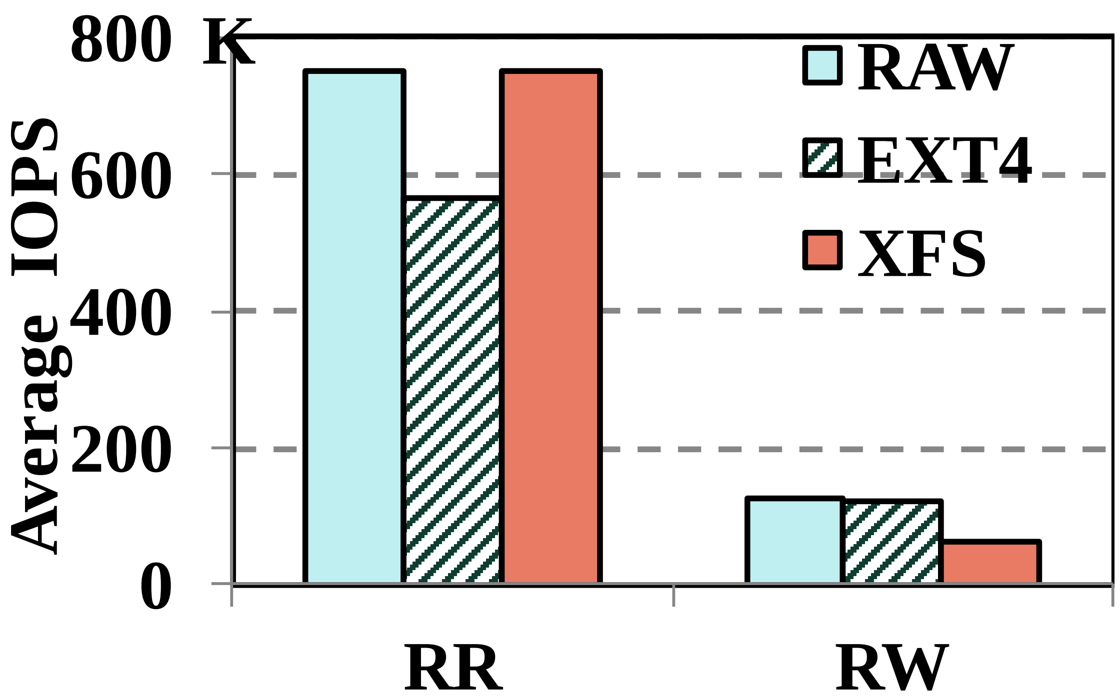
Experimental Environment

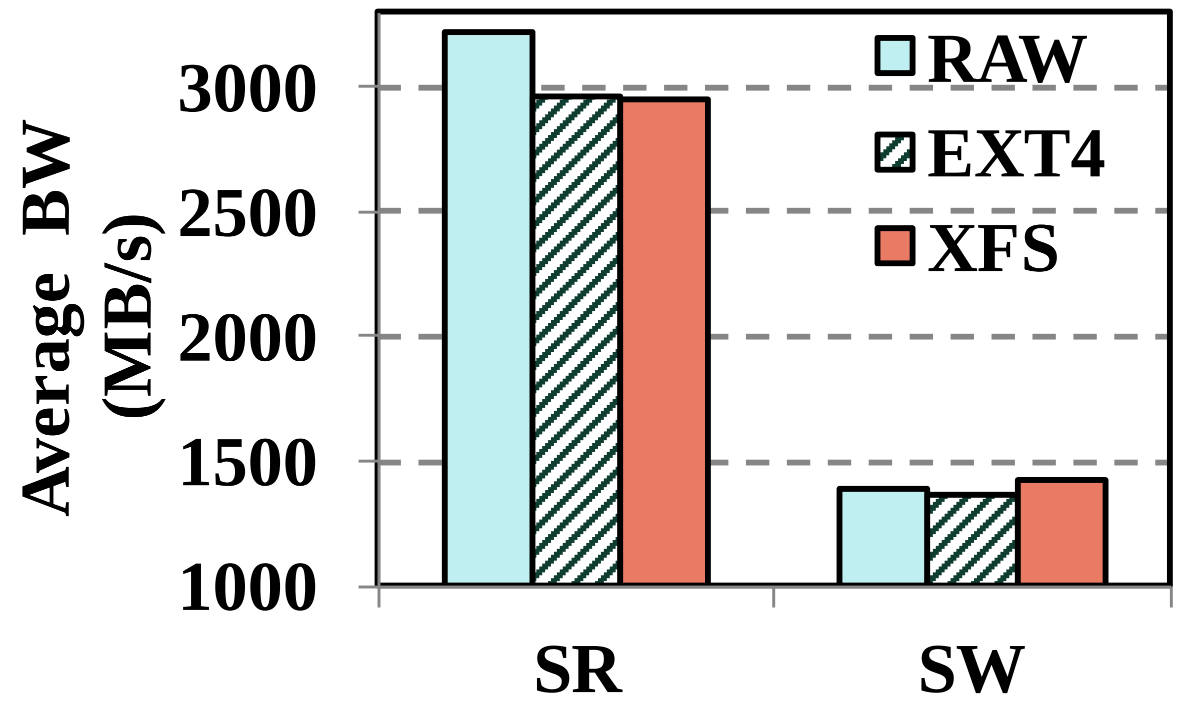
- Dual-socket, 12 HT cores Xeon E5-2670 V3
- enterprise-class NVMe SSD
  - Samsung XS1715
- kernel v4.6.0
- Docker v1.11.2
- fio used for traffic generation
  - Asynchronous IO engine, libaio
  - 32 concurrent jobs and iodepth is 32
  - Measure steady state performance



Performance Comparison

— Host Backing Filesystems

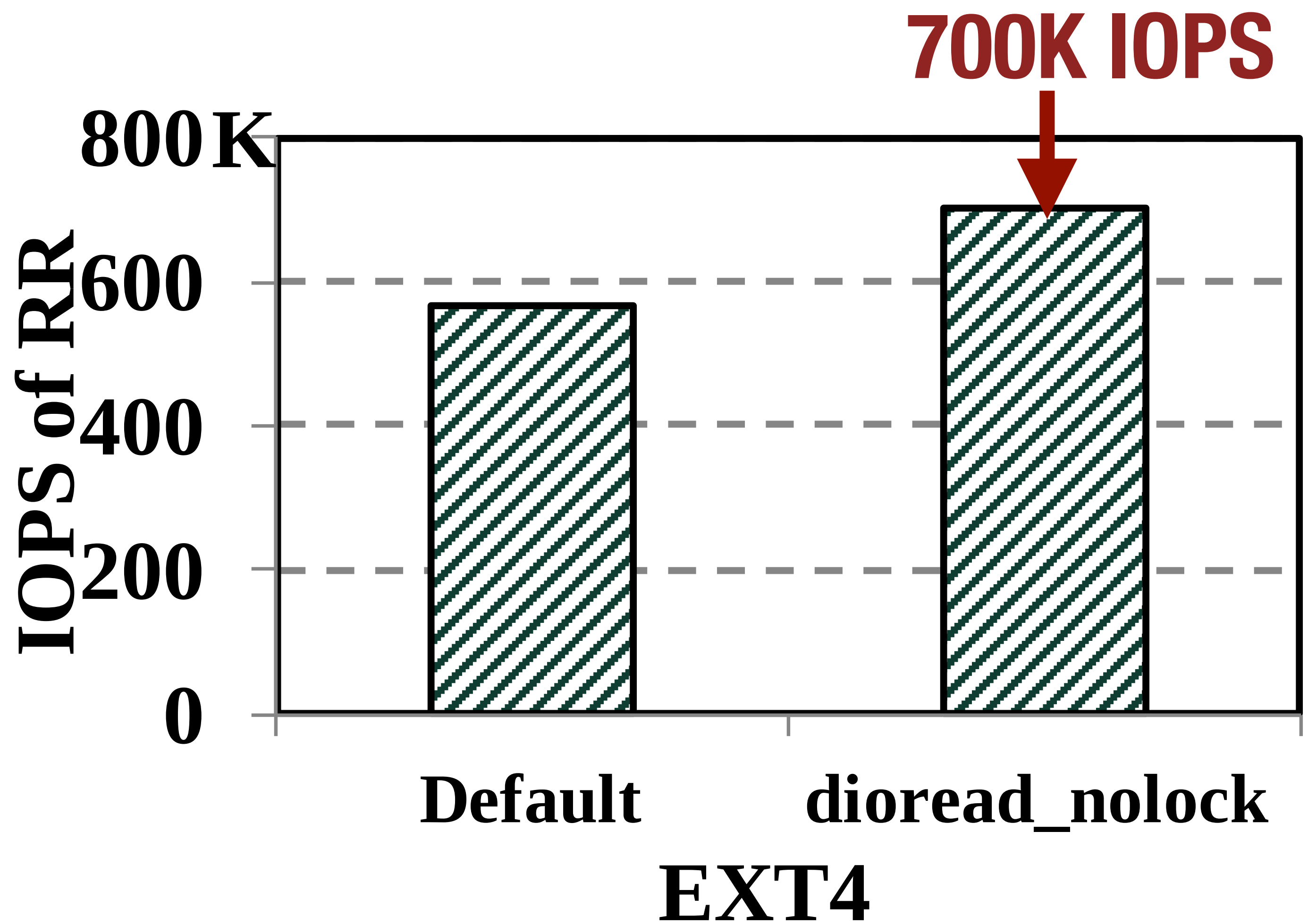




- EXT4 performs 25% worse for RR
- XFS performs closely resembles RAW for all but RW

Tuning the Performance Gap

—Random Reads



- XFS allows multiple processes to read a file at once
  - Uses **allocation groups** which can be accessed independently
- EXT4 requires mutex locks even for read operations

Tuning the Performance Gap

—Random Writes

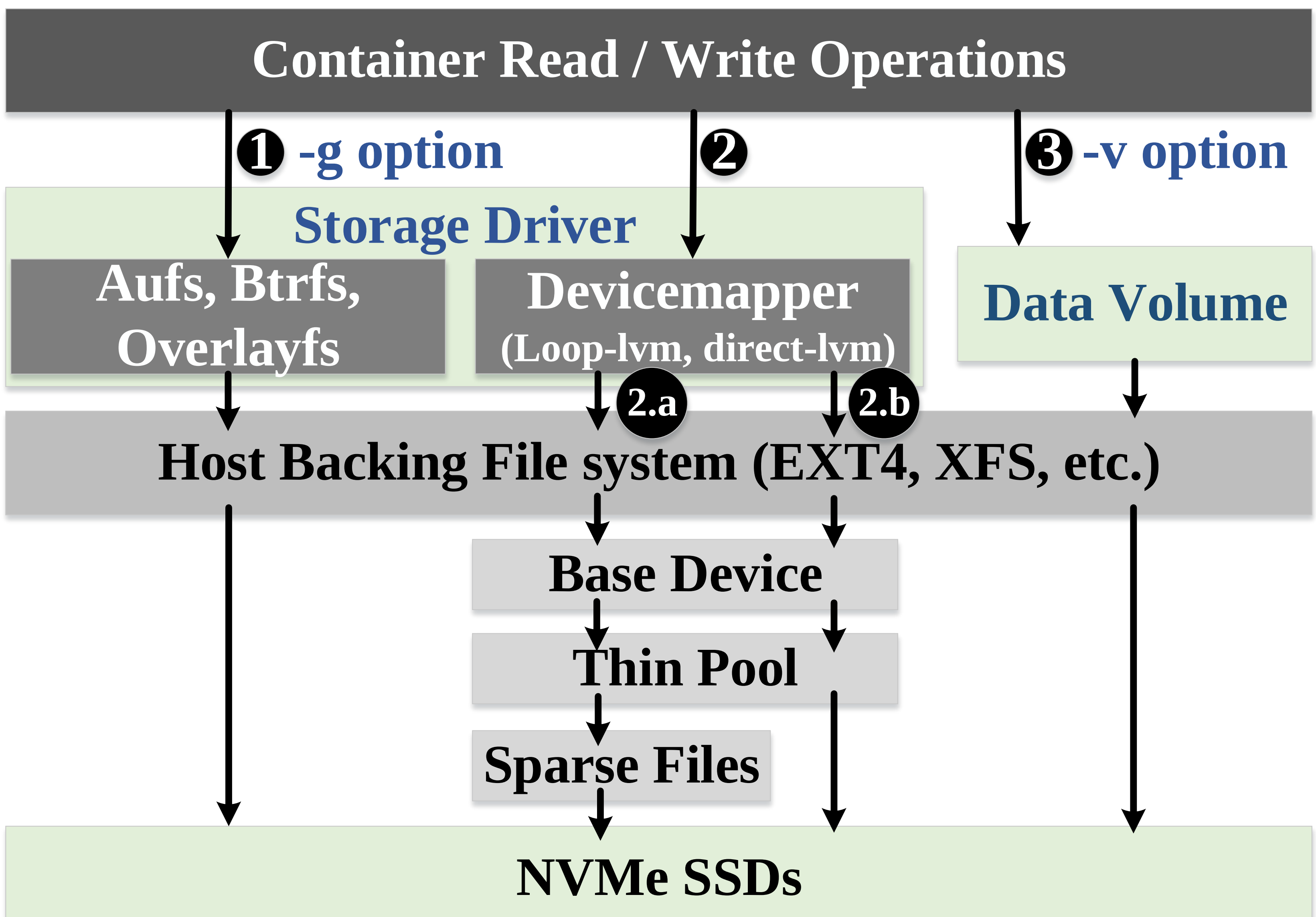
- XFS performs poorly with high thread count
- Contention in exclusive locking kills the write performance
  - Used by extent look up and write checks
  - Patch available but not for Linux 4.6 [1]

[1] https://www.percona.com/blog/2012/03/15/ext4-vs-xfs-on-ssd/

Storage Architecture in Docker

Storage Options:

1. Through Docker Filesystem (Aufs, Btrfs, Overlayfs)
2. Through Virtual Block Devices (2.a Loop-lvm, 2.b Direct-lvm)
3. Through Docker Data Volume (-v)



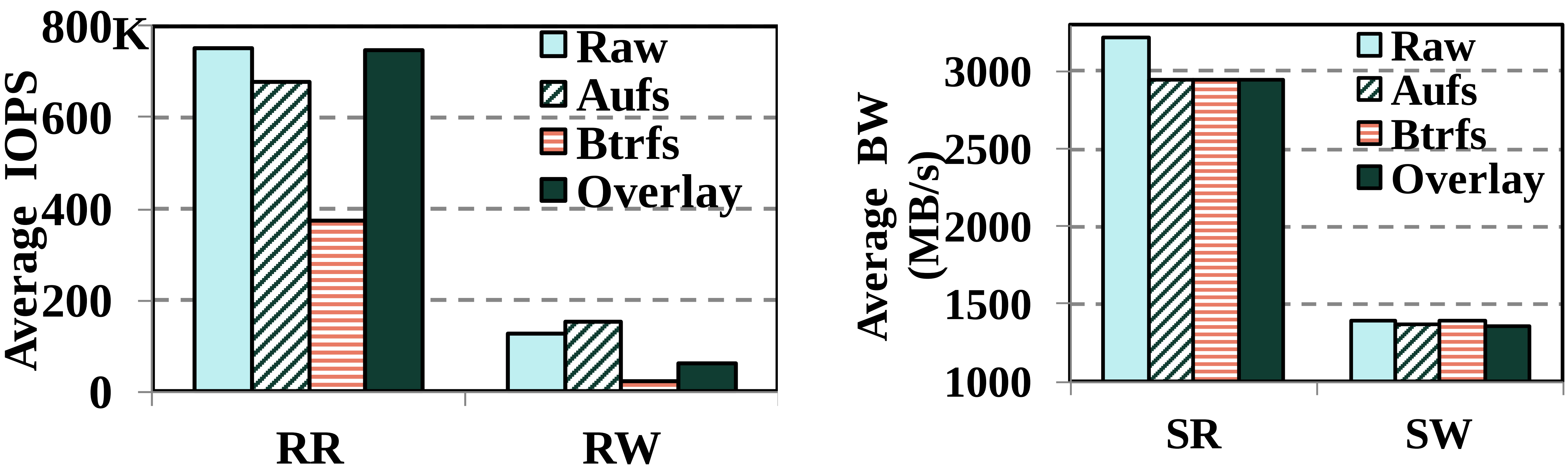
Docker Storage Options

Option 1: Through Docker File System

- **Aufs (Advanced multi-layered Unification FileSystem):**
  - A fast reliable unification file system
- **Btrfs (B-tree file system):**
  - A modern CoW file system which implements many advanced features for fault tolerance, repair and easy administration
- **Overlayfs:**
  - Another modern unification file system which has simpler design and potentially faster than Aufs

Performance Comparison

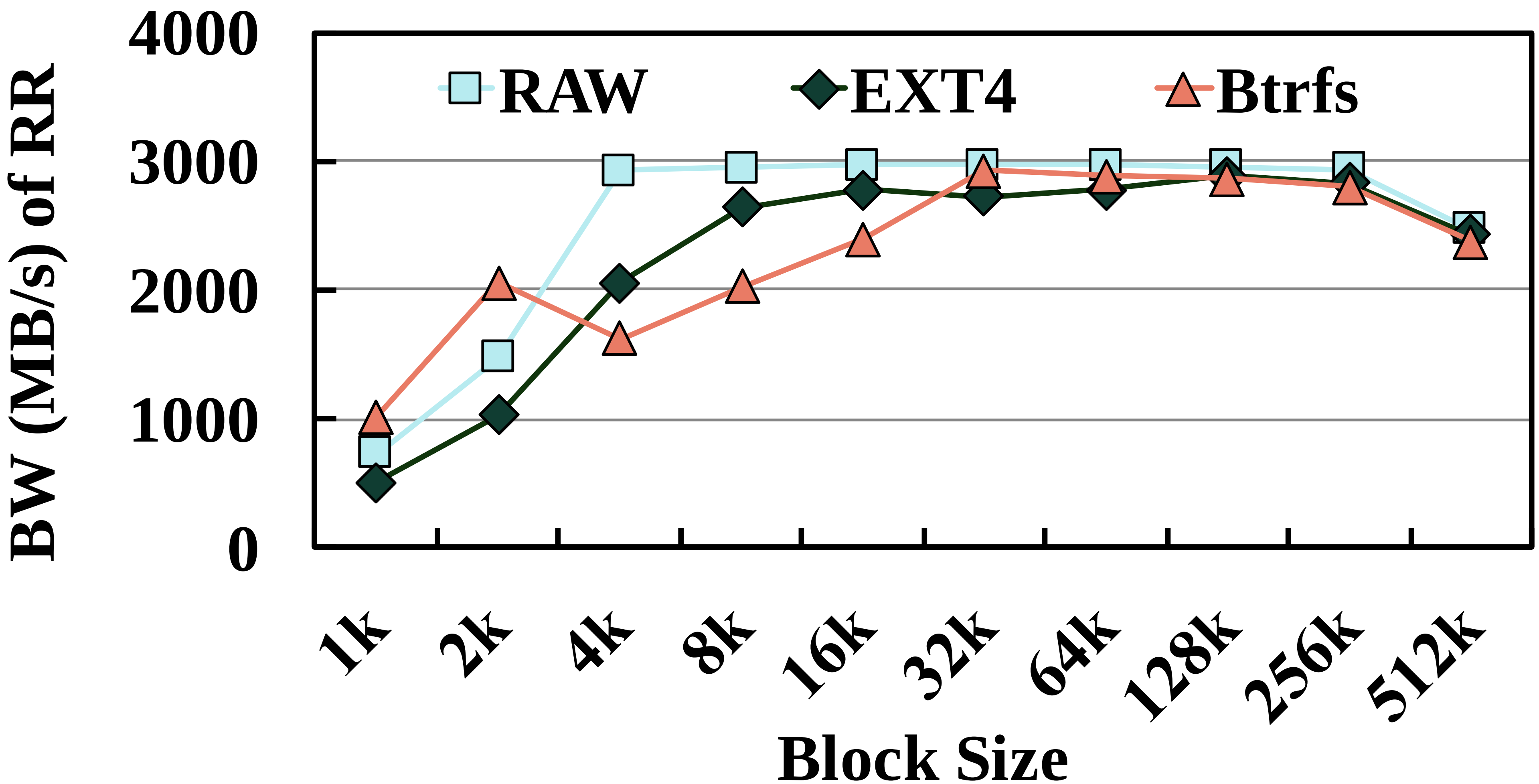
Option 1: Through Docker File System



- Aufs and Overlayfs performs close to raw block device for most cases
- Btrfs has the worst performance for random workloads

Tuning the Performance Gap of Btrfs

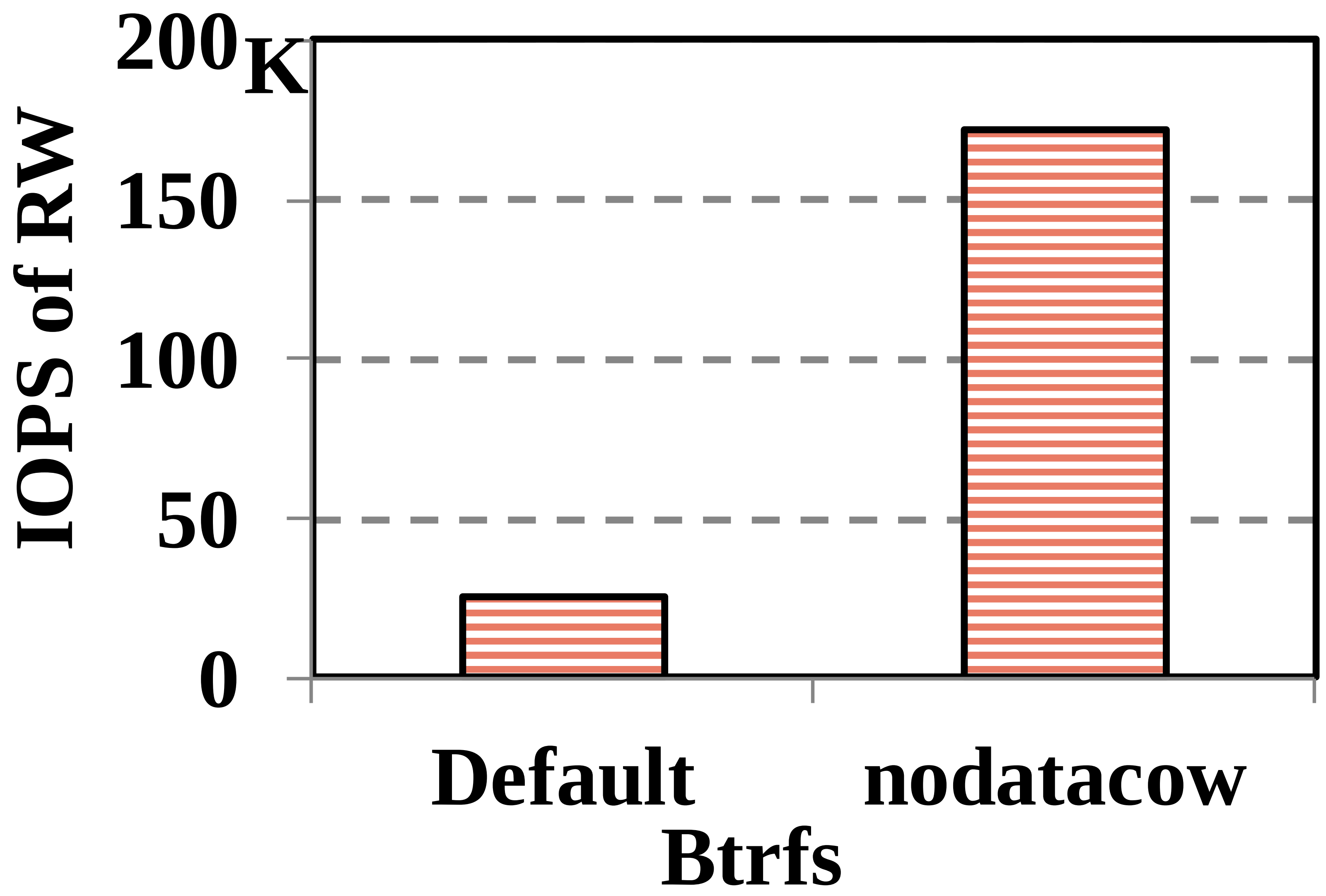
—Random Reads



- Btrfs doesn't work well for small block size yet
- Btrfs must read the file extent before reading the file data.
  - Large block size reduces the frequency of reading metadata

Tuning the Performance Gap of Btrfs

—Random Reads

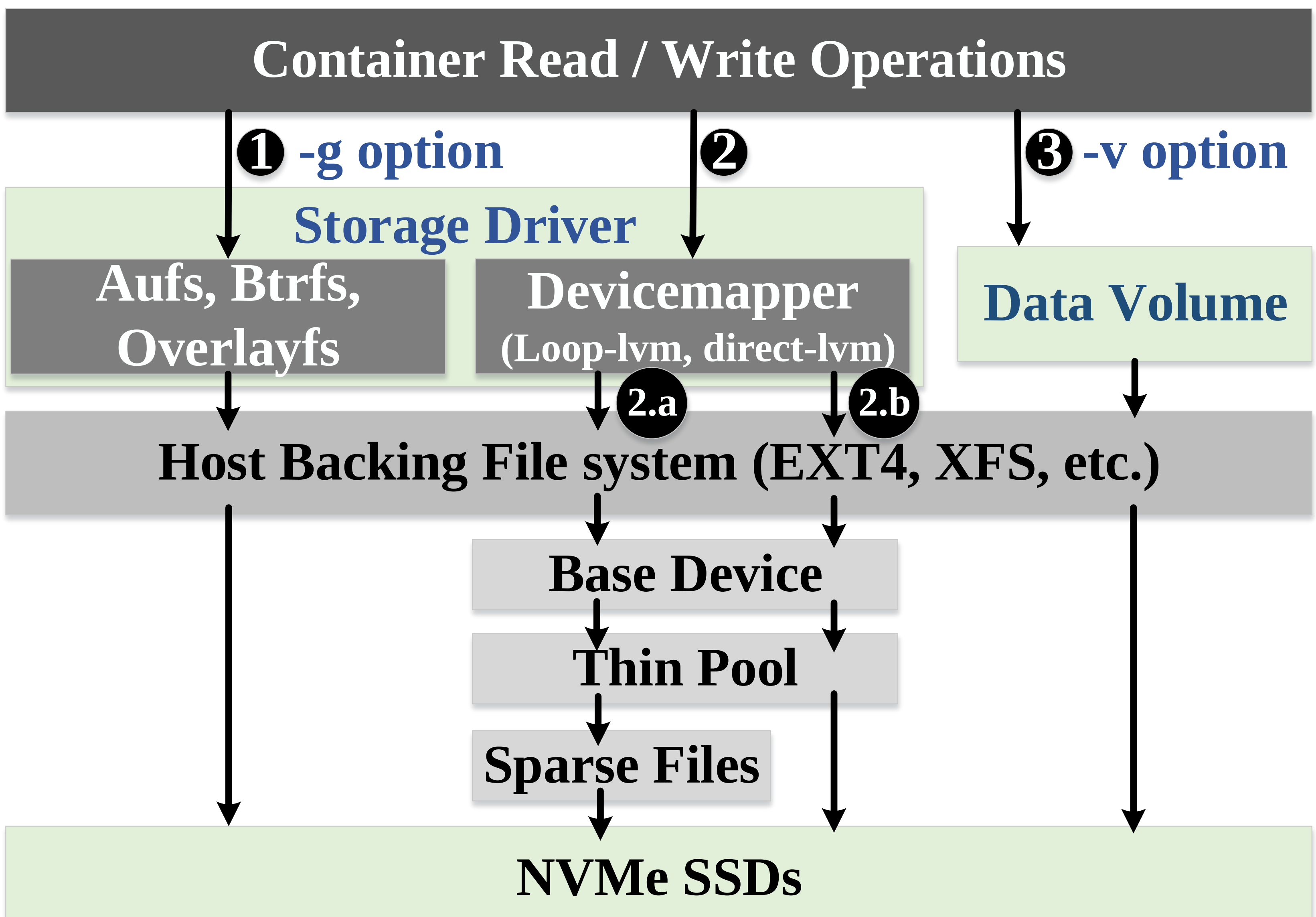


- Btrfs doesn't work well for random writes due to CoW overhead

Storage Architecture in Docker

Storage Options:

1. Through Docker Filesystem (Aufs, Btrfs, Overlayfs)
2. Through Virtual Block Devices (2.a Loop-lvm, 2.b Direct-lvm)
3. Through Docker Data Volume (-v)



Docker Storage Configurations

Option 2: Through Virtual Block Device

- Devicemapper storage driver leverages the **thin provisioning** and **snapshotting** capabilities of the kernel based Device Mapper Framework

- Loop-lvm uses **sparse files** to build the thin-provisioned pools

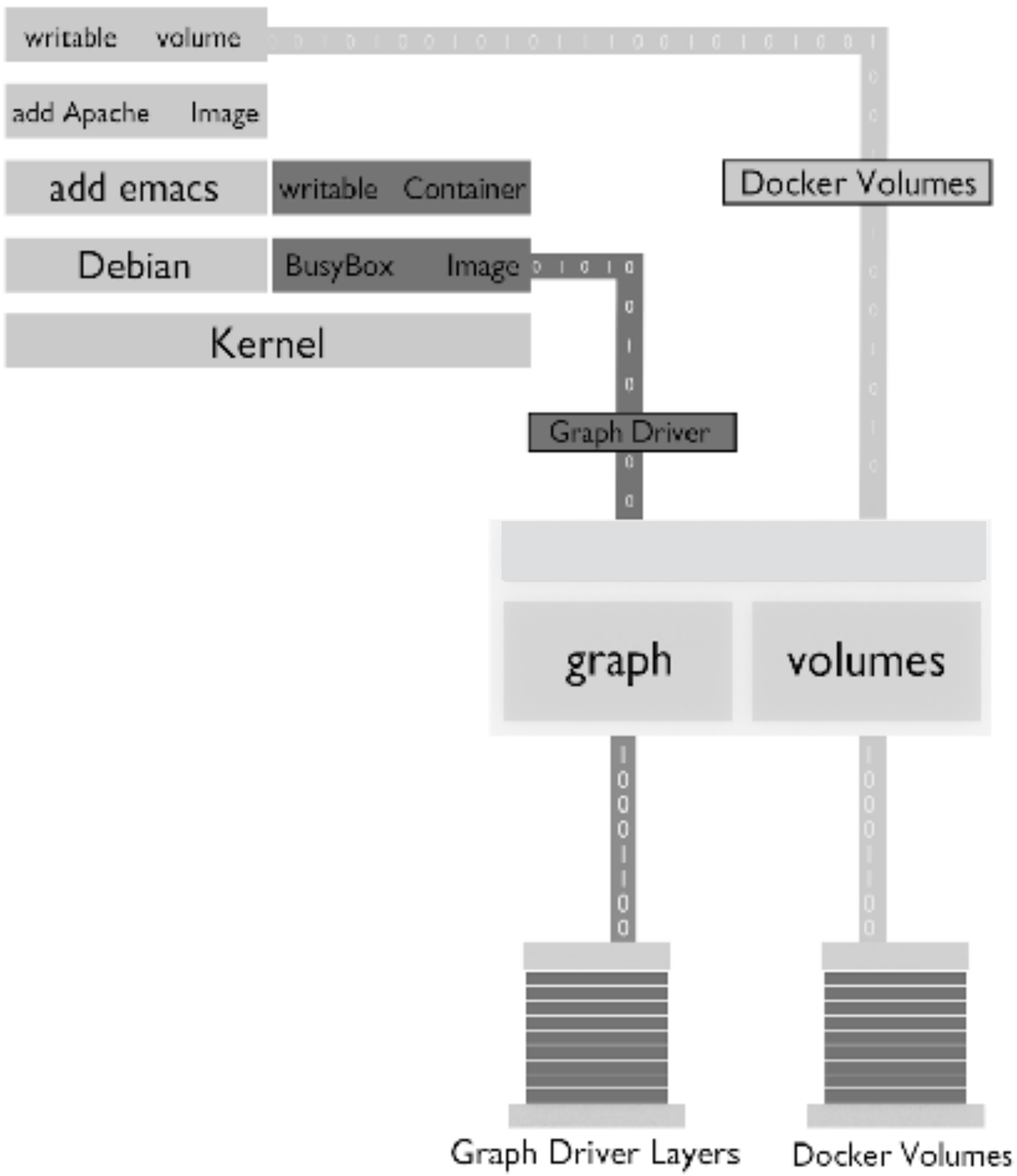
- Direct-lvm uses **block device** to directly create the thin pools

  (Recommended by Docker)

Docker Storage Configurations

Option 3: Through Docker Data Volume (-v)

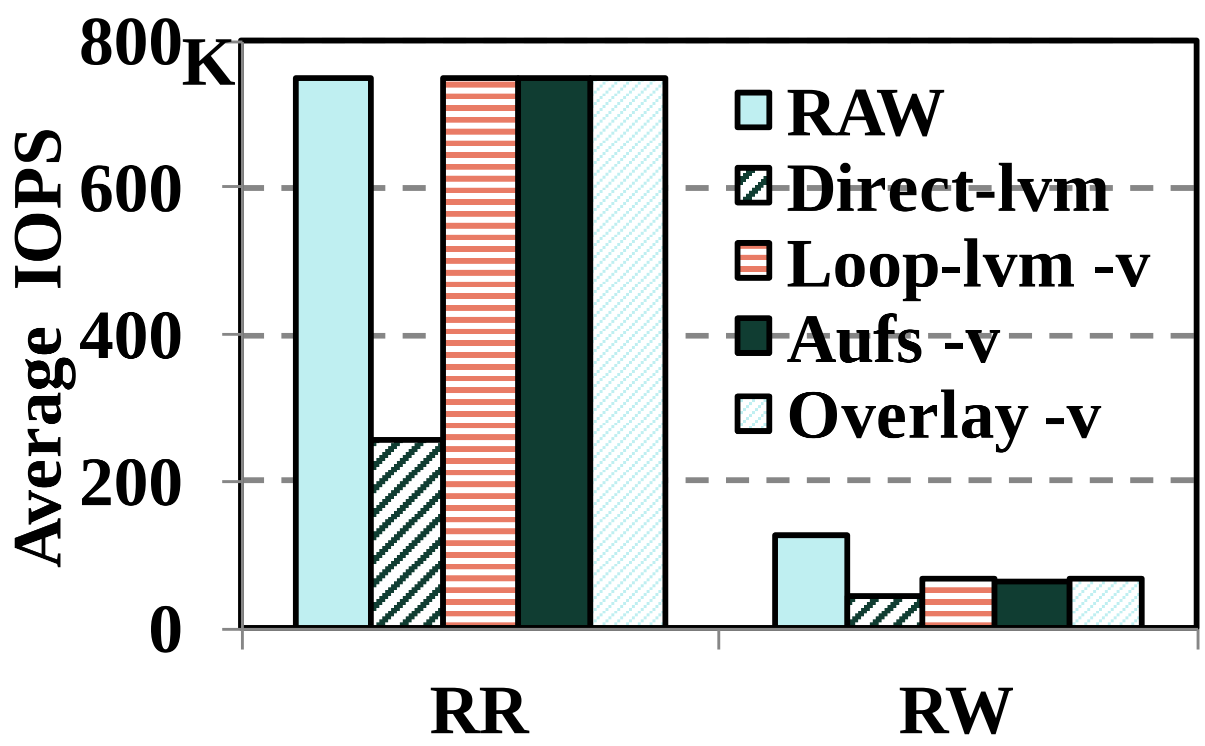
- Data **persists** beyond the lifetime of the container and can be shared and accessed from other containers

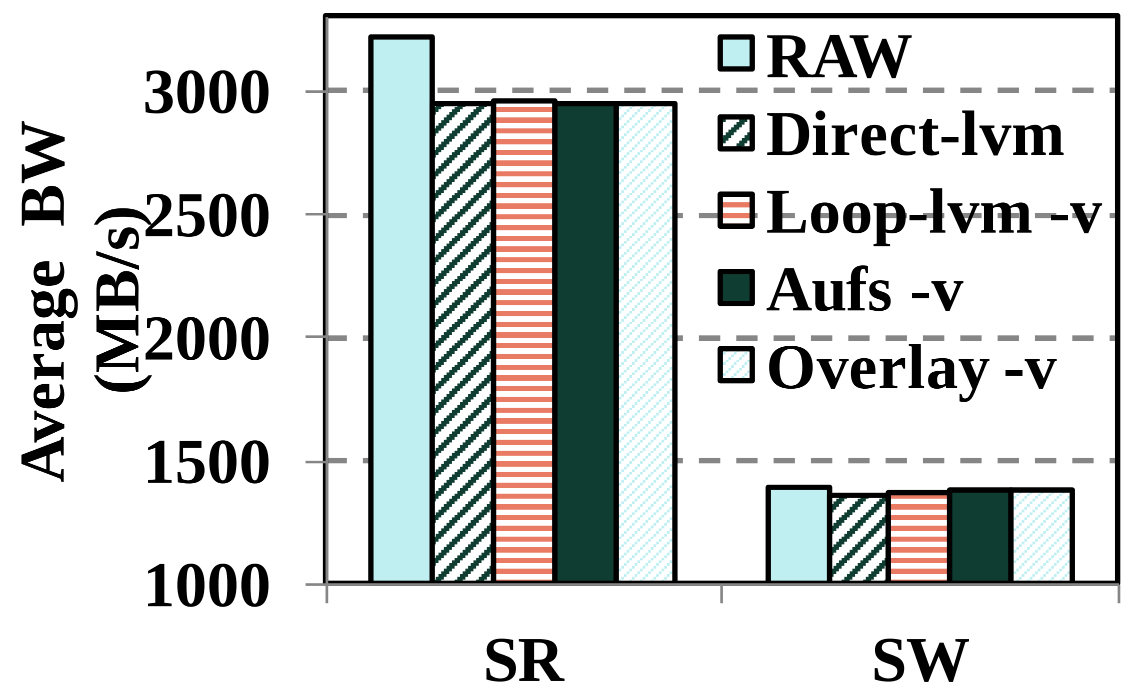


* figure from https://github.com/libopenstorage/openstorage

Performance Comparison

Option 2 & Option 3





- Direct-lvm has worse performance for RR/RW
- LVM, device mapper, and the dm-thinp kernel module introduced **additional code paths and overhead** may not suit IO intensive workloads

Application Performance

Cassandra Database

- NoSQL database
- Scale linearly to the number of nodes in the cluster (theoretically) [1]
- Requires data persistence
  - uses docker volume to store data

[1] Rabl, Tilmann et al. "Solving Big Data Challenges for Enterprise Application Performance Management", VLDB'13

Scaling Docker Containers on NVMe

multiple containerized Cassandra Databases

Experiment Setup

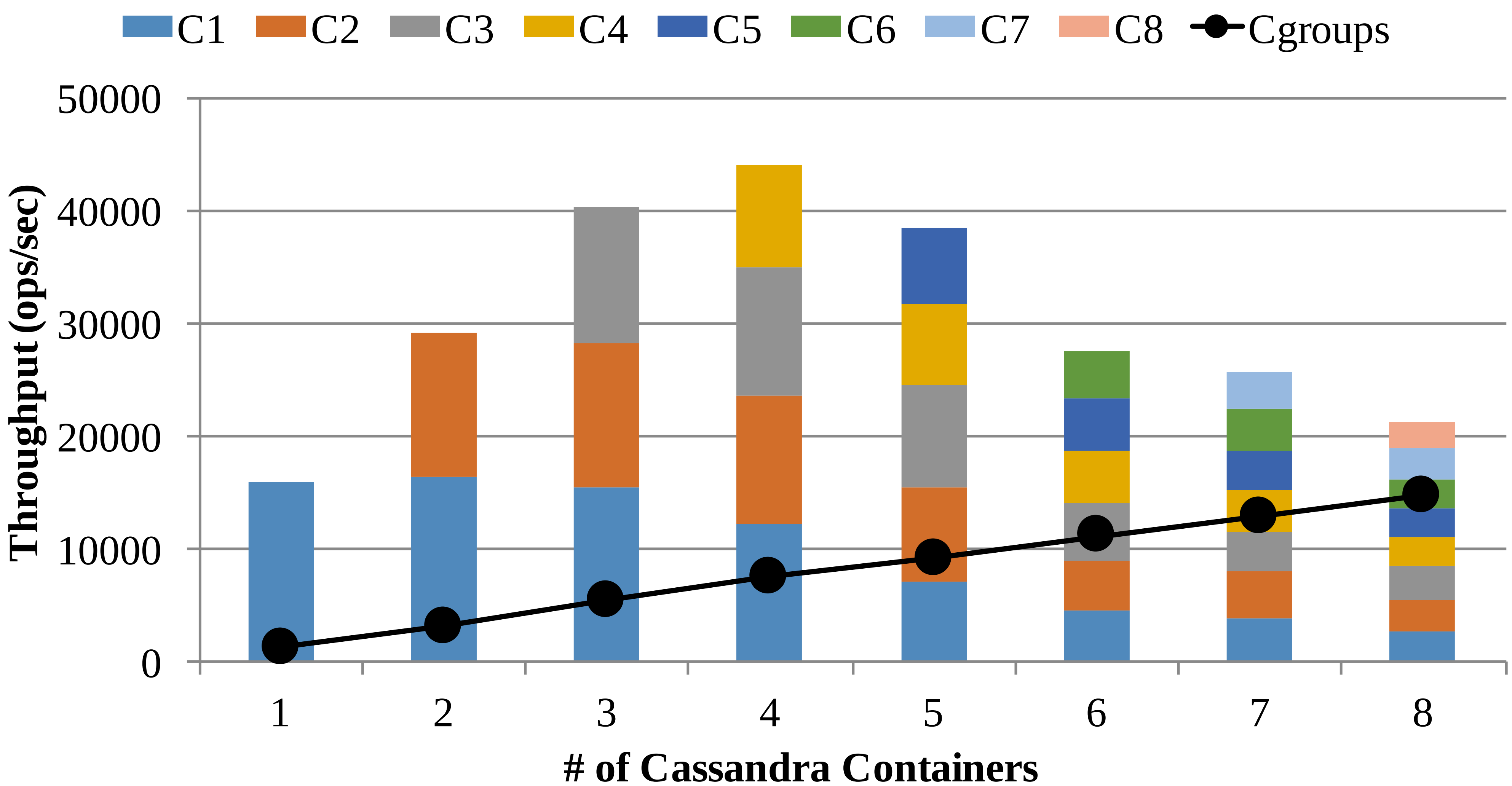
- Dual socket, Xeon E5 server, 10Gb ethernet
- N = 1, 2, 3, … 8 containers
- Each container is driven by a YCSB client
- Record Count: 100M records, 100GB in each DB
- Client thread count: 16

Workloads

- Workload A, 50% read, 50% update, Zipfian distribution
- Workload D, 95% read, 5% insert, normal distribution

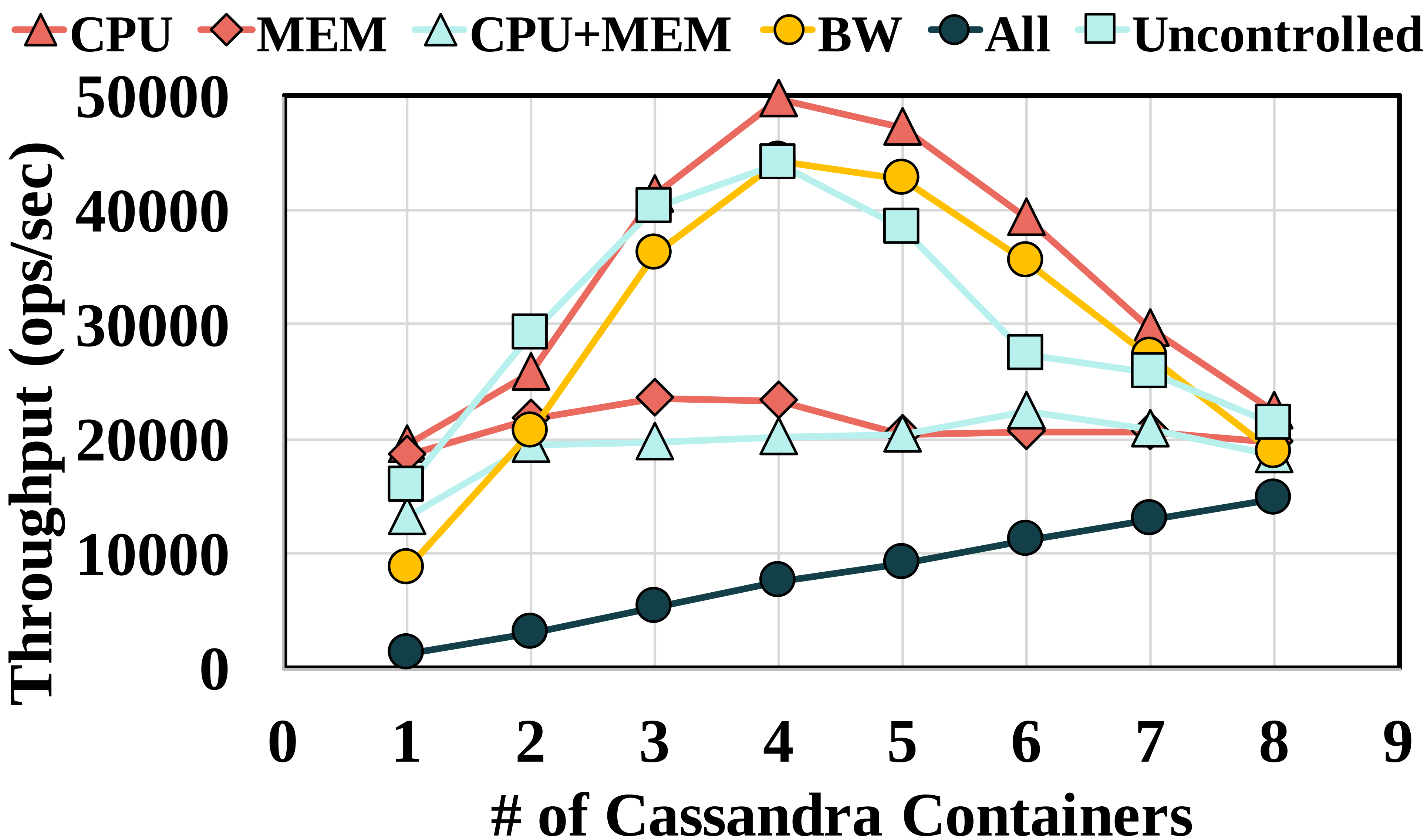
Results-Throughput

Workload D, directly attached SSD



- Aggregated throughput peaks at 4 containers
- Cgroups: 6 CPU cores, 6GB memory, 400MB/s bandwidth

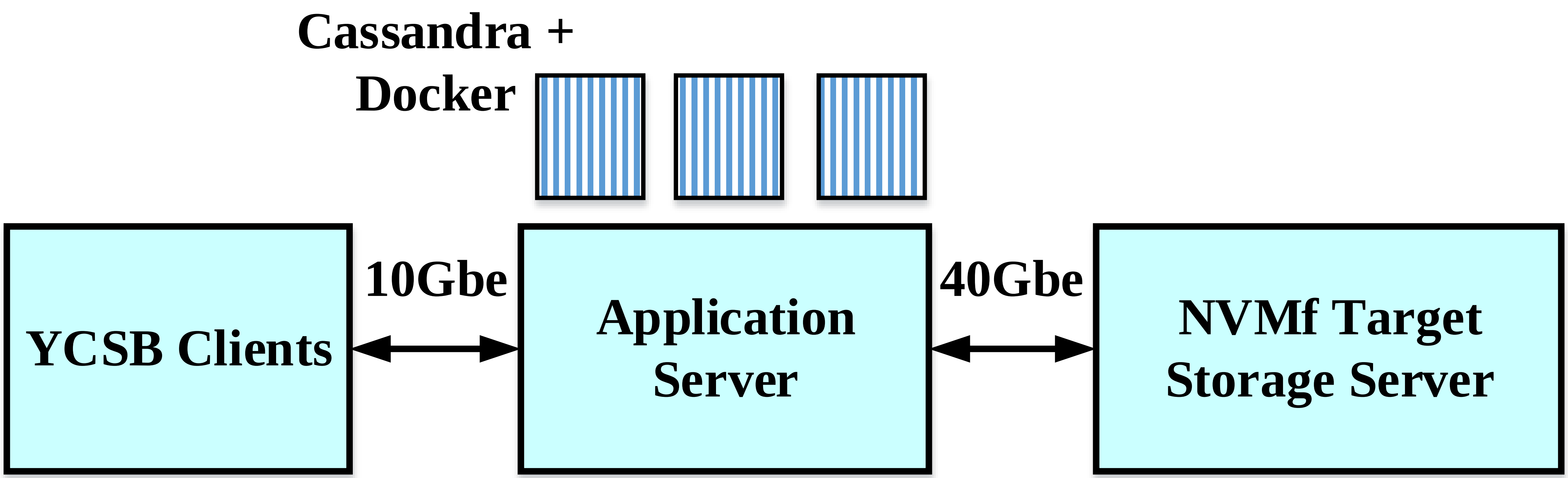
Strategies for Dividing Resources



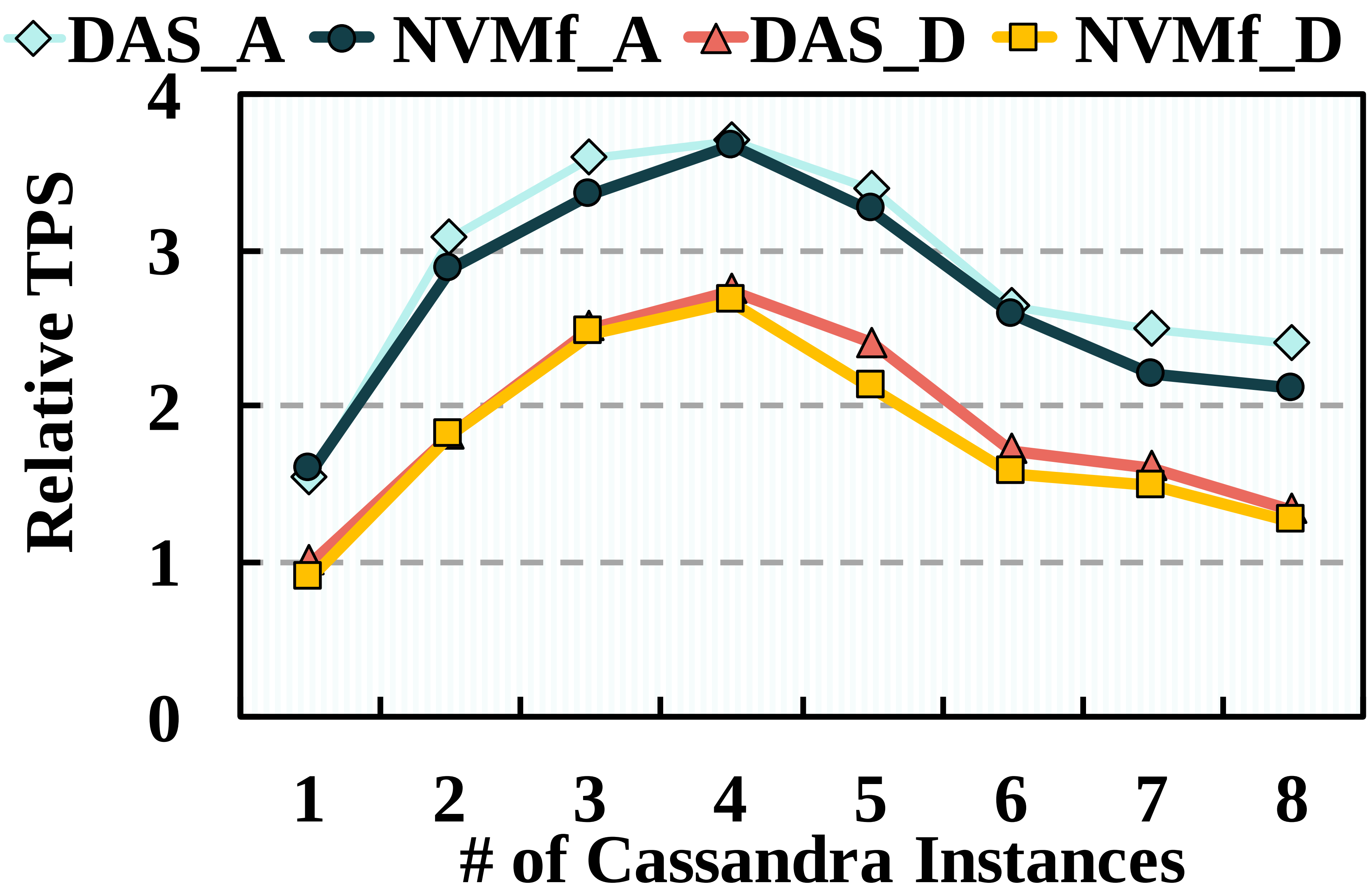
- MEM has the most significant impact on throughput
- Best strategy for dividing resources using cgroups
  - Assign 6 CPU cores for each container, leave other resource uncontrolled

Scaling Containerized Cassandra using NVMf

Experiment Setup

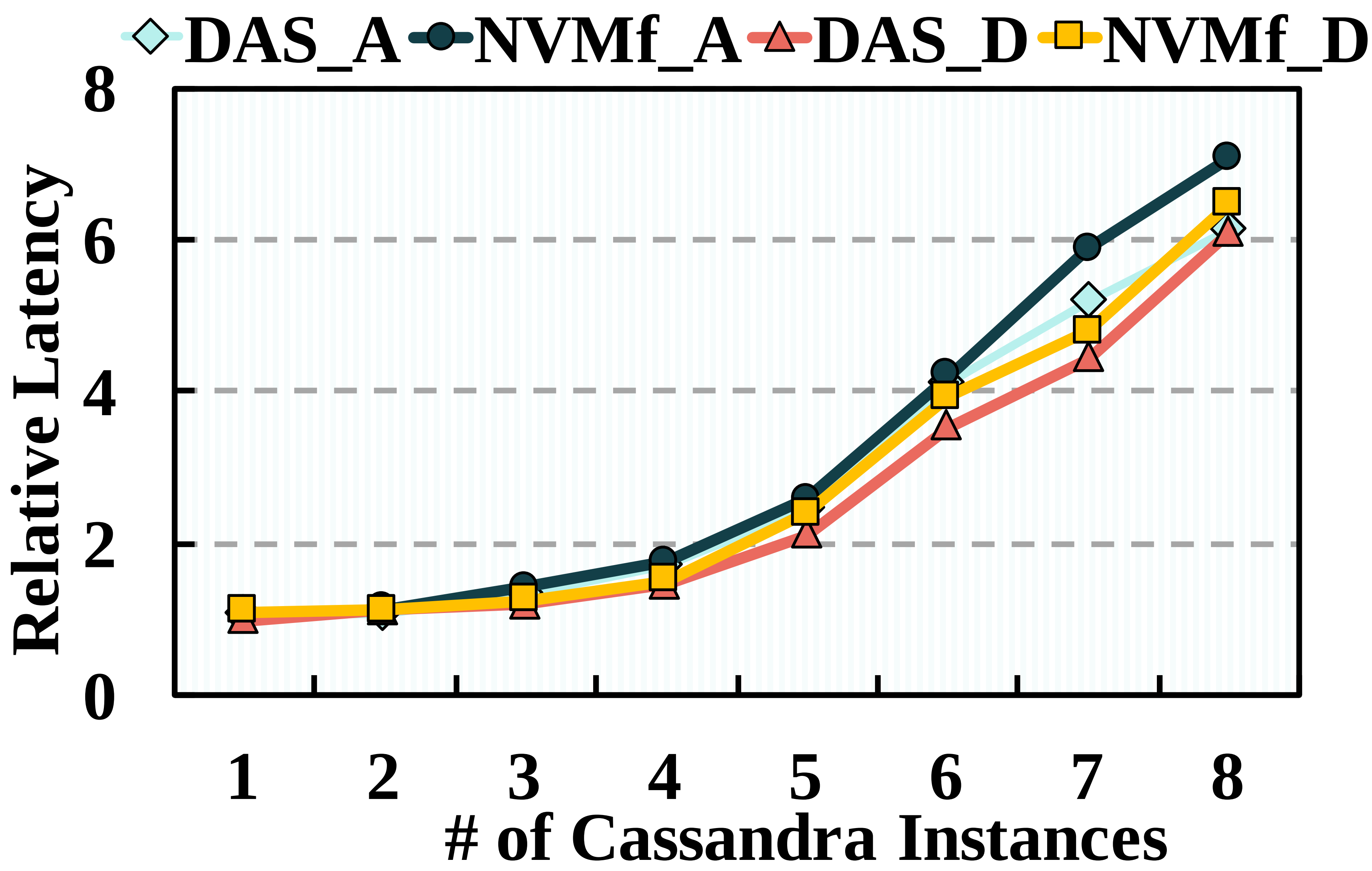


Results-Throughput



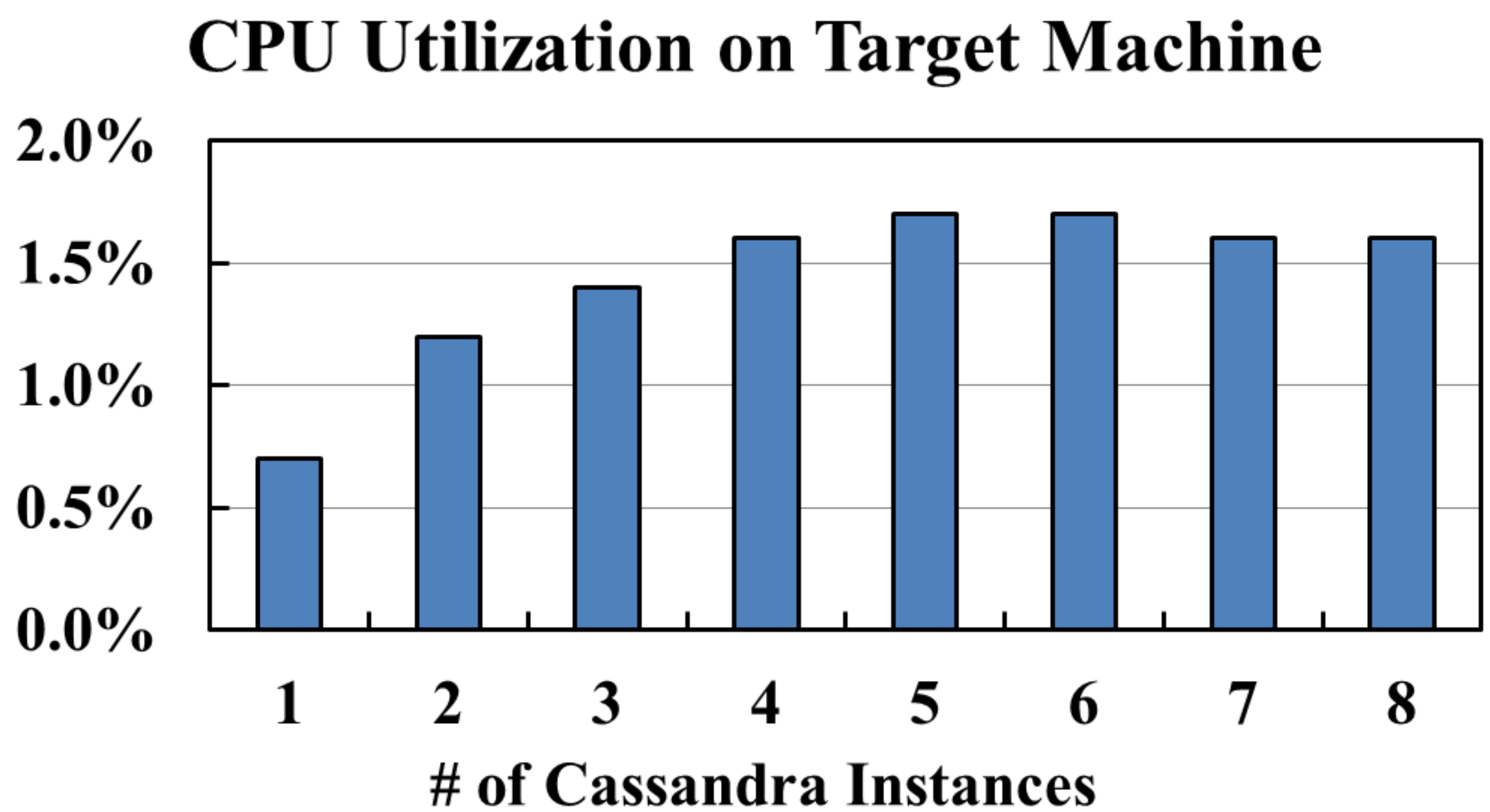
- The throughput of NVMf is within **6% - 12%** compared to directly attached SSDs

Results-Latency



- NVMF incurs only **2% - 15%** longer latency than direct attached SSD.

Results-CPU Utilization



- NVMF incurs less than **1.8%** CPU Utilization on Target Machine

SUMMARY

Best Option in Docker for NVMe Drive Performance

Overlay FS + XFS + Data Volume

Best Strategy for Dividing Resources using Cgroups

Control only the CPU resources

Scaling Docker Containers on NVMf

Throughput: within 6% - 12% vs. DAS

Latency: 2% - 15% longer than DAS

THANK YOU!

QIUMIN@USC.EDU