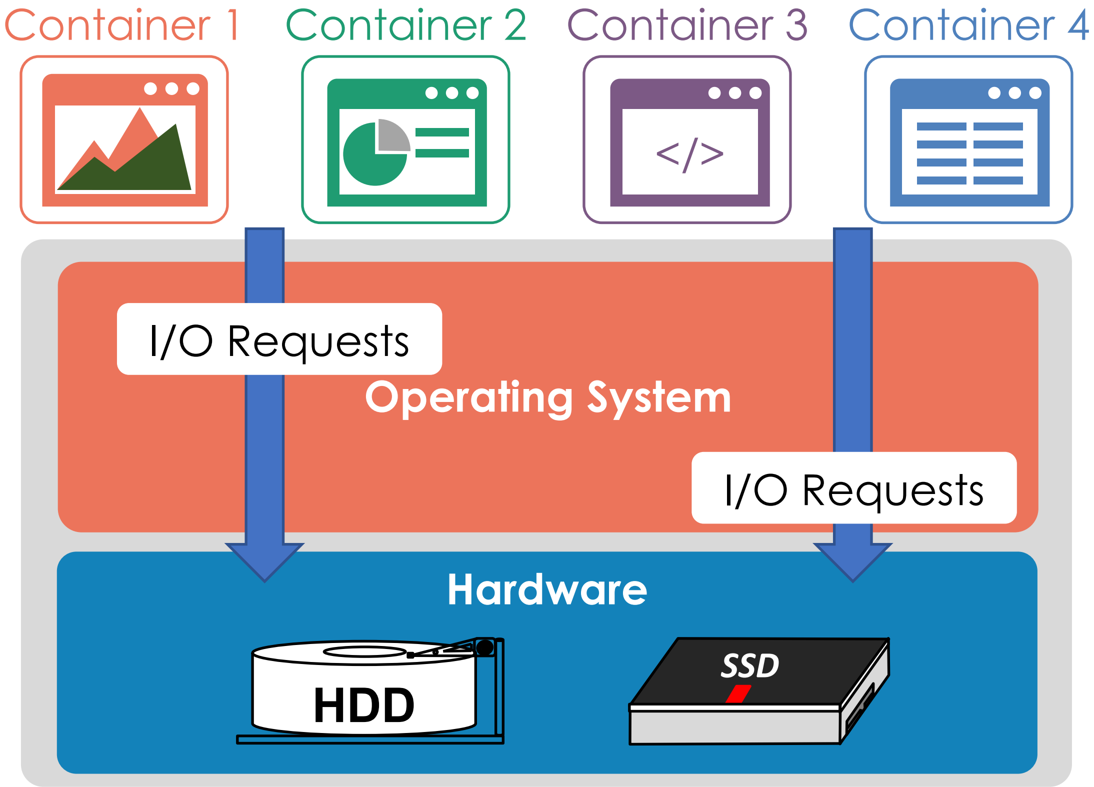
Towards Application-level I/O Proportionality with a Weight-aware Page Cache Management

Jonggyu Park*, Kwonje Oh, and Young Ik Eom

Sungkyunkwan University, South Korea

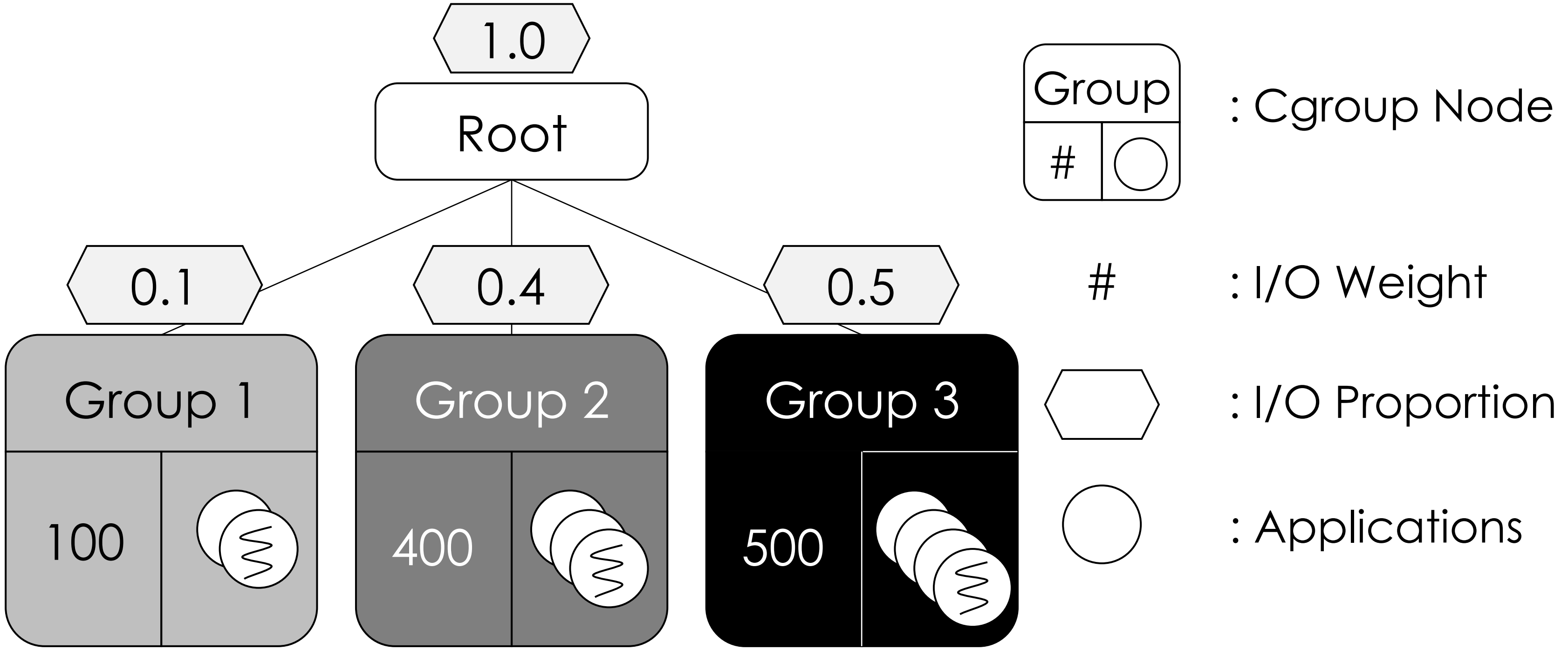
Server Consolidation is Pervasive

- Multiple virtualized instances run on a single host
  - Compete for system resources
  - Efficient resource scheduling is necessary



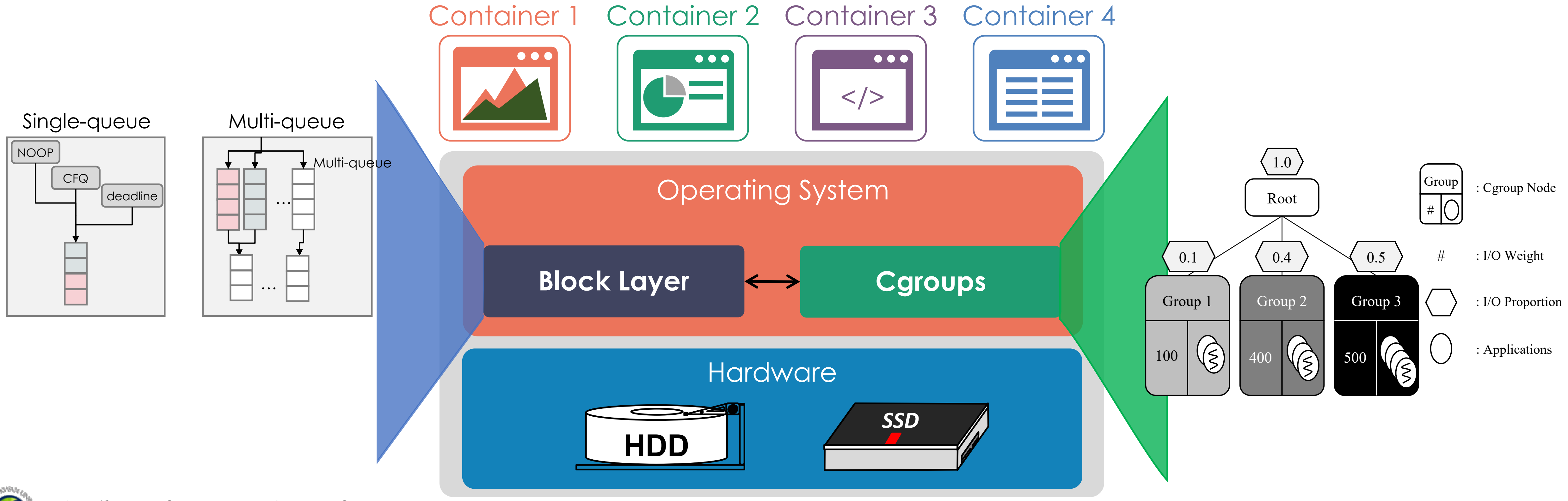
Proportional I/O Sharing by Cgroups

- Cgroups proportionally share I/O resources using I/O weight
    - The I/O bandwidth ratio follows the ratio of I/O weight



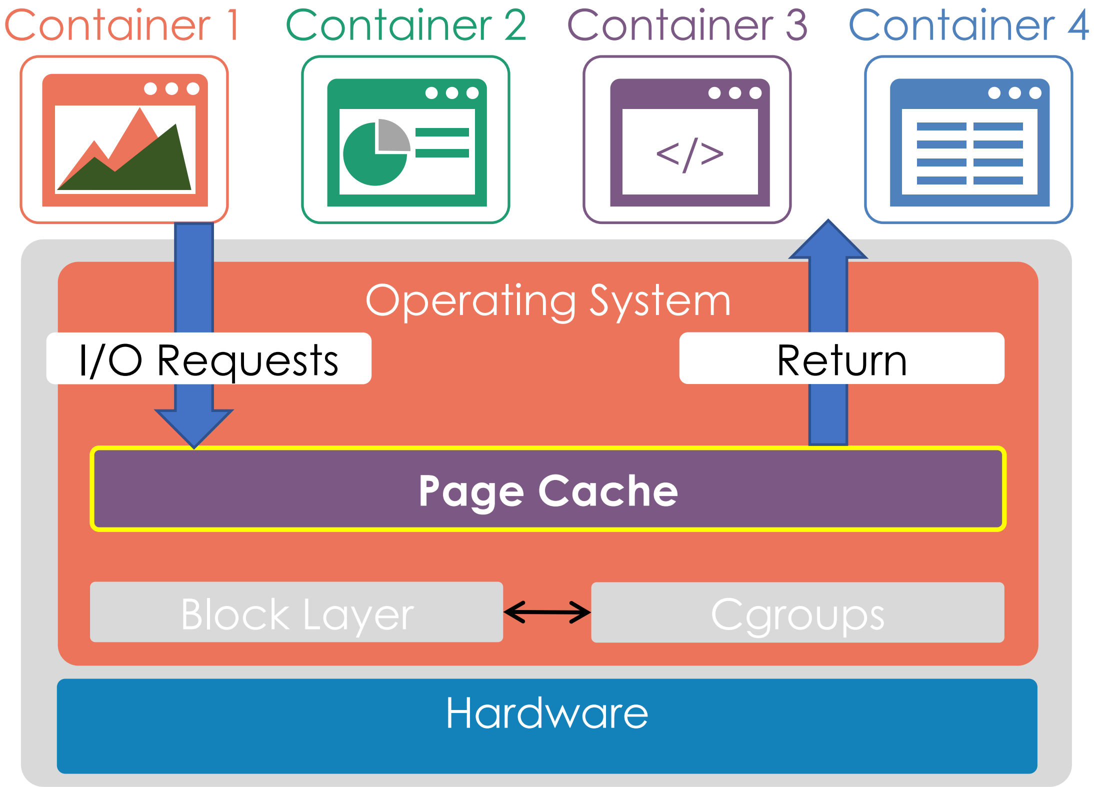
Cgroups and the Block Layer

- The blkio subsystem controls I/O resources collaboratively with the block layer
  - I/O scheduler in the block layer utilizes the I/O weights in scheduling
  - I/O service time (CFQ) or the number of sectors to serve (BFQ)



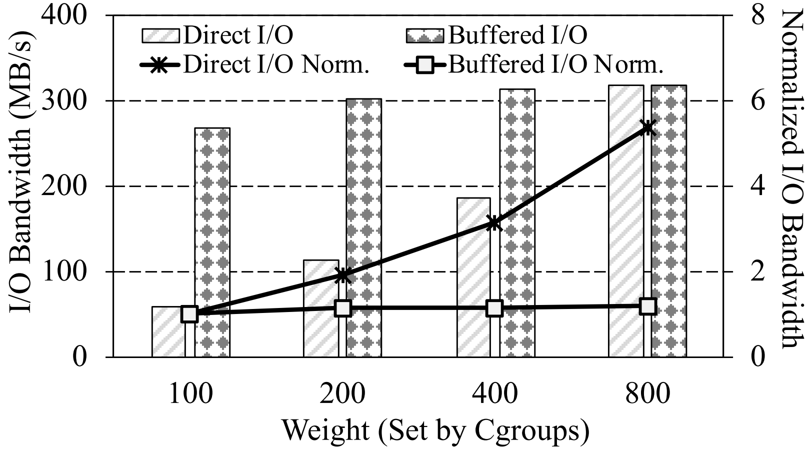
The Page Cache

- The page cache is often utilized to enhance I/O performance.
  - It directly serves I/O requests without delivering them to the block layer, if possible
  - Cgroups cannot control I/O requests that are serviced by the page cache

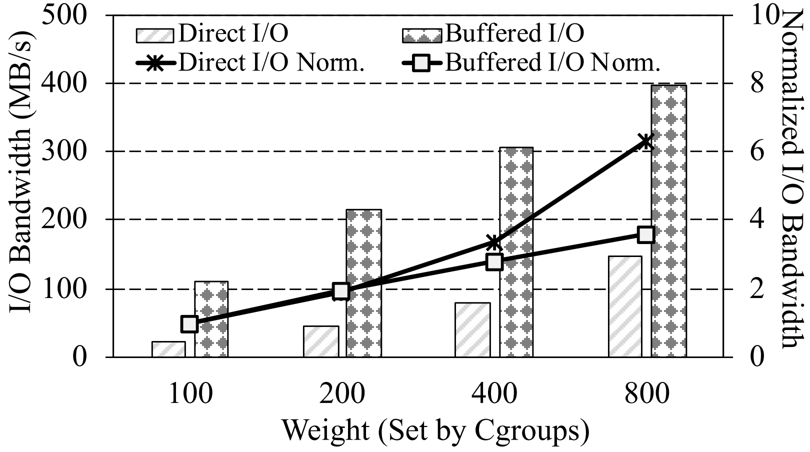


Buffered I/O vs. Direct I/O

- Direct I/O
  - Proportional I/O sharing according to I/O weight
  - Lower performance due to bypassing the page cache
- Buffered I/O
  - Poor proportionality
  - Better performance due to the page cache



Fileserver workload



Re-read workload

The Life of the Page Cache

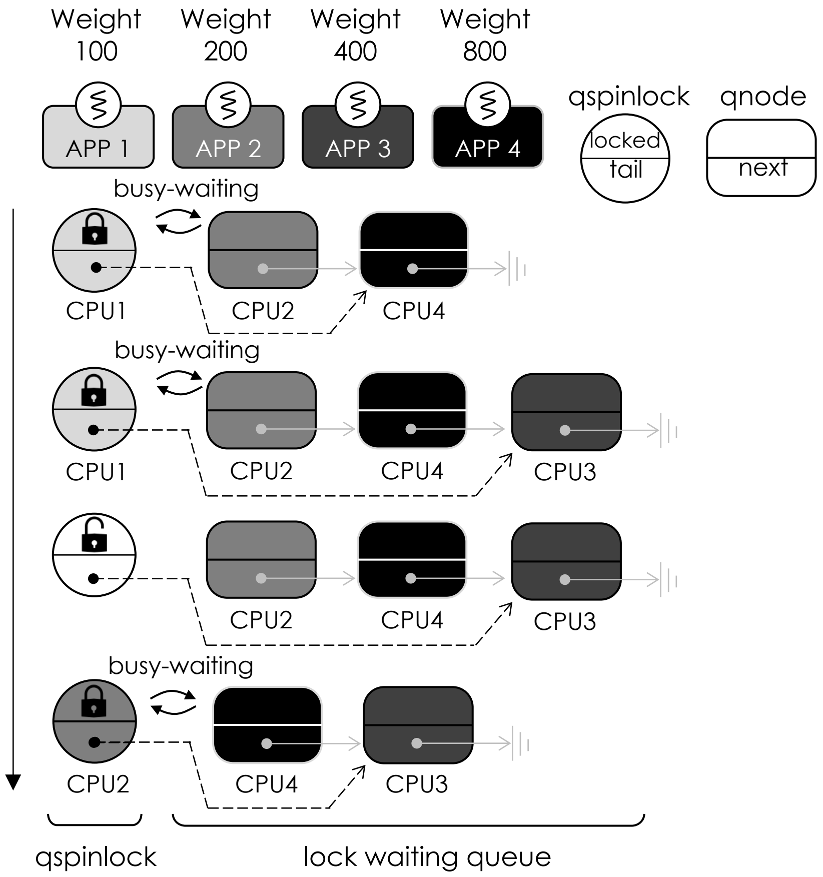
- Page allocation
  - Allocates a new page for the new page cache entry
  - Qspinlock serializes page allocation
  - Critical to the write performance

- Page reclamation
  - Deallocates pages that are not used to secure new pages
  - Reclaims the pages at the tail of the inactive list
  - Decides which pages will reside in the page cache
  - Affects the read performance

Qspinlock of Page Allocation

- Qspinlock prevents race condition
  - Consists of a qspinlock and per-cpu qnodes
  - Allows one CPU holding qspinlock while the head node (CPU2) busy-waits
  - After qspinlock is released, the head node acquires the qspinlock

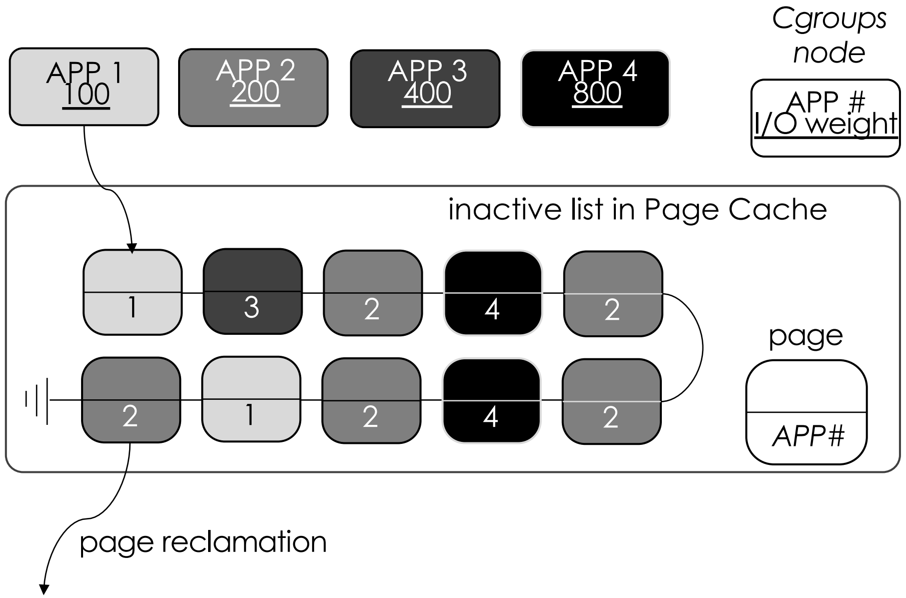
- FIFO-based holder selection
  - The conventional qspinlock for page allocation selects the next holder in a FIFO manner
  - No consideration of I/O weight



An overview of qspinlock

Page Reclamation

- Page cache
  - maintains 2Q LRU
  - Keeps data frequently accessed in the active list, otherwise in the inactive list
  - Reclaims pages at the tail of the inactive list
- Page reclamation
  - Ignores the I/O weight during reclamation
  - Pages used by higher weighted apps can be evicted earlier
  - No scheme to reflect I/O weight



An overview of page reclamation

Justitia

Problem #1: Cgroups focus on block-level I/O proportionality

Problem #2: Page allocation/reclamation do not reflect I/O weight

Weight-awareness!!!

Justitia: new page cache management for application-level I/O proportionality

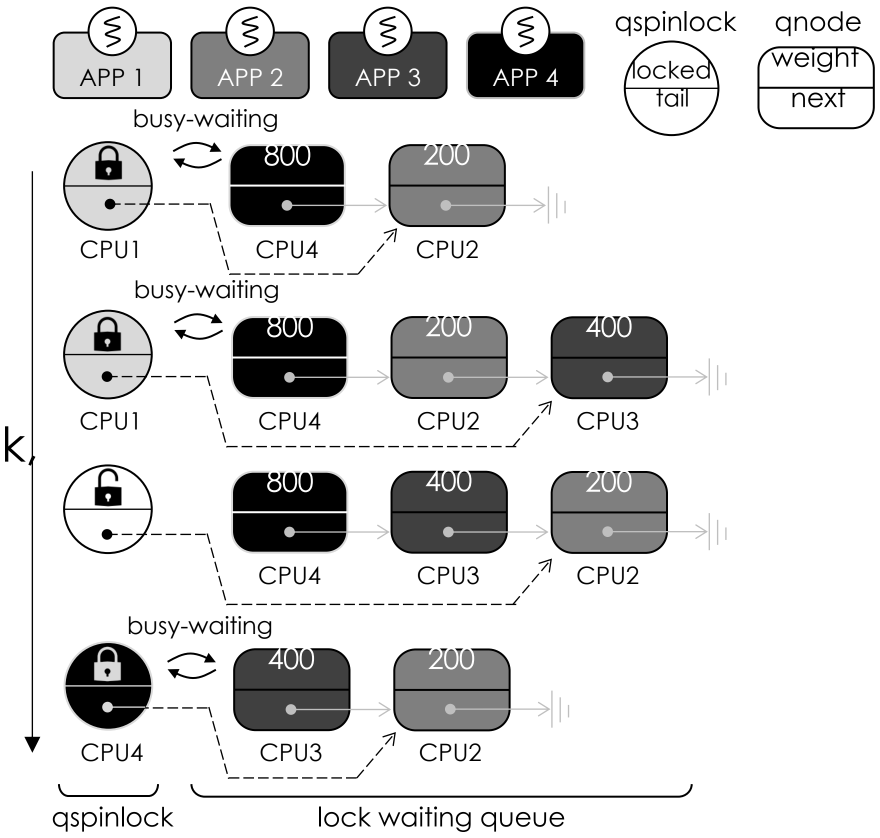
A. Weight-aware Qspinlock for Page Cache Allocation

B. Weight-aware Page Reclamation

Weight-aware Qspinlock for Page Cache Allocation

- Weight-aware Qspinlock
  - Stores weight in the qnode
  - Reflects I/O weight by the following procedure
  1. qspinlock is released
  2. Iterates lock waiting queue to find the qnode (maxNode) with the highest I/O weight
  3. Moves the maxNode next to the head node
  4. Next time, when the head node acquires the qspinlock, the maxNode becomes a head node

In short, Justitia reorders the lock waiting queue based on I/O weight

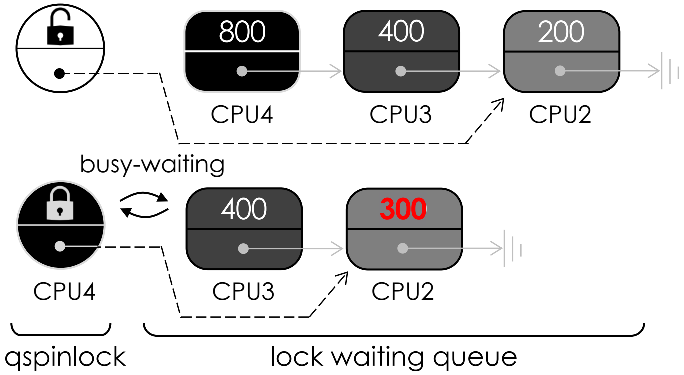


An overview of weight-aware qspinlock

Preventing the Race Condition

- How about the starvation problem?
  - When there are many high-weighted apps, the low-weighted apps can starve

- We adopt aging technique to prevent the starvation problem
  - Whenever reordering occurs, Justitia increases I/O weight of qnodes in the lock waiting queue
  - Justitia considers not only I/O weight but also the waiting time



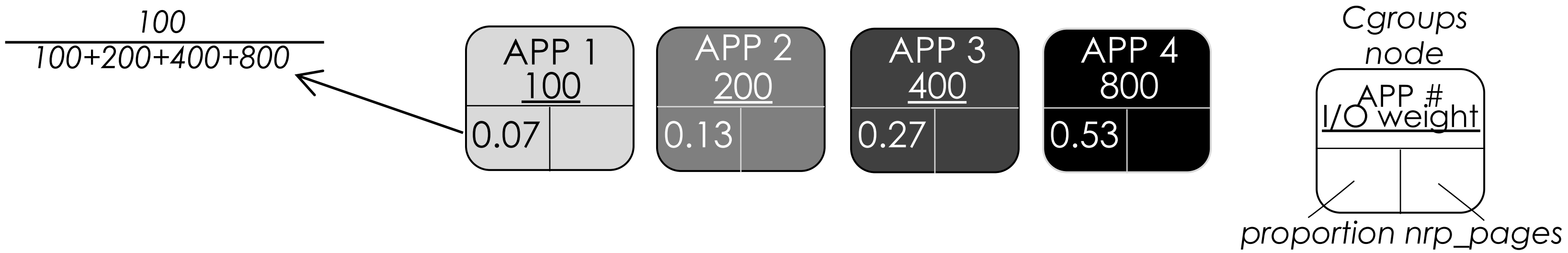
Weight-aware Page Reclamation

Justitia imposes weight-awareness by the following procedures

- Calculating the I/O proportion of each application
- Recording page ownership information on the page structure
- Page reclamation considering the I/O proportion

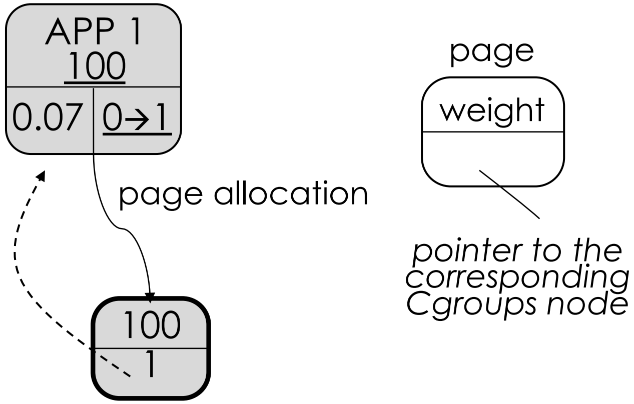
Weight-aware Page Reclamation

- Calculating the I/O proportion of each application
  - New variables in Cgroups are added
    - Proportion: Proportion of I/O weight (weight / total weight)
    - nrp_pages: The number of pages in the page cache that this cgroup is currently using



Weight-aware Page Reclamation

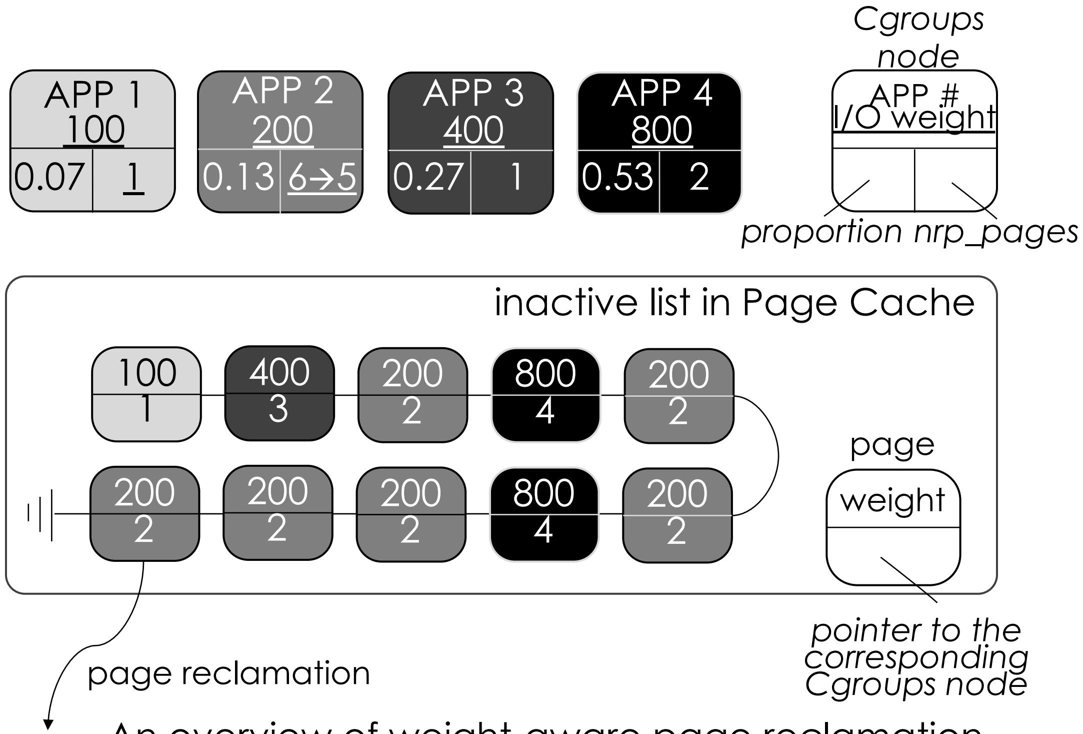
- Recording page ownership information on the page structure
  - New variable in the page structure
    - I/O weight
    - Pointer to the corresponding cgroups node



Weight-aware Page Reclamation

- Page reclamation considering the I/O proportion
  - Justitia reclaims pages whose cgroups hold more pages than its threshold

  *Threshold = proportion * the total # of pages in the page cache



An overview of weight-aware page reclamation

Experimental Setup

- CPU: Intel I7-6700
- Memory: 16GB DRAM
- Storage: SATA SSD 256GB
- Benchmarks: FIO (re-read) and Filebench (fileserver)

Read → Dummy write → **Read**

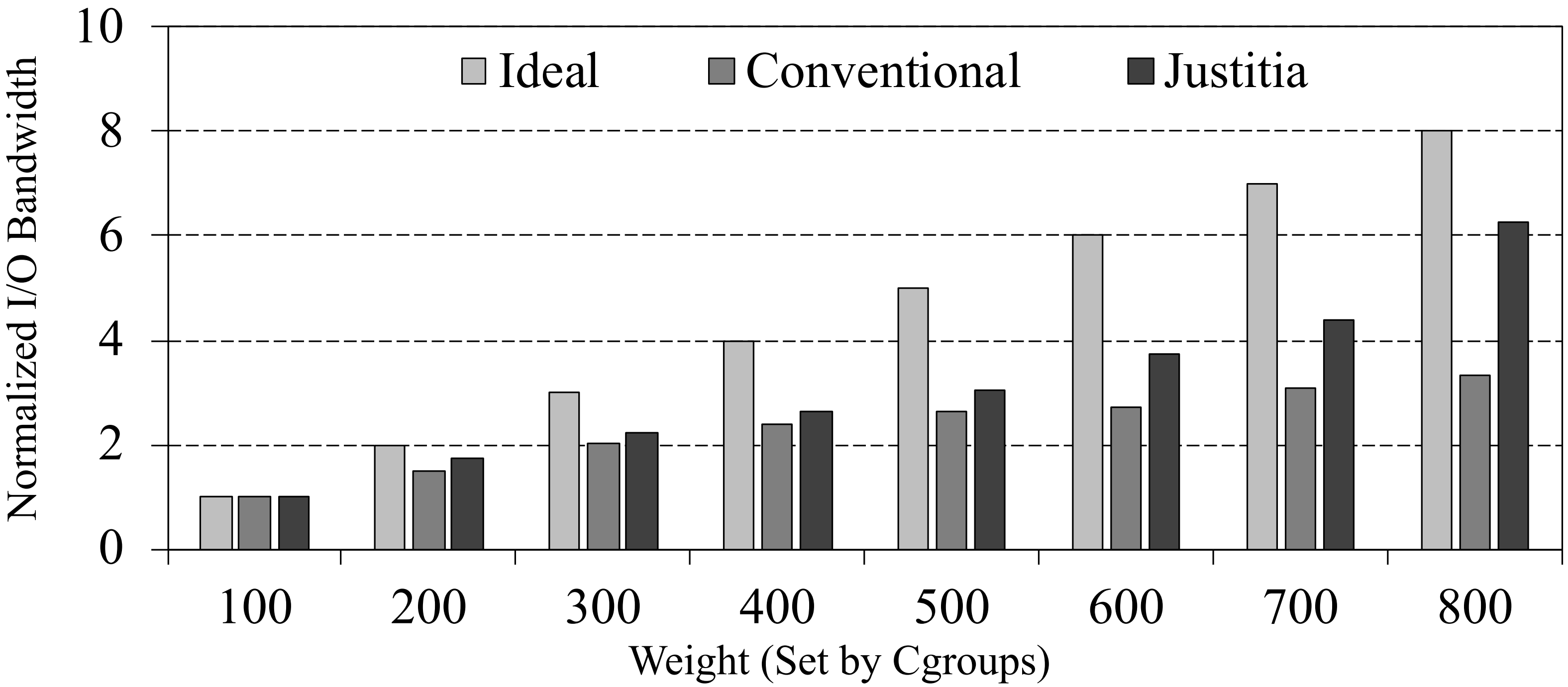
* All applications were containerized by Docker

- A metric to quantitively measure I/O proportionality, introduced in [1]

$$PV = \frac{1}{N} \cdot \sum_{\forall cont} |Ideal - Actual|$$ (Proportionality Variation)

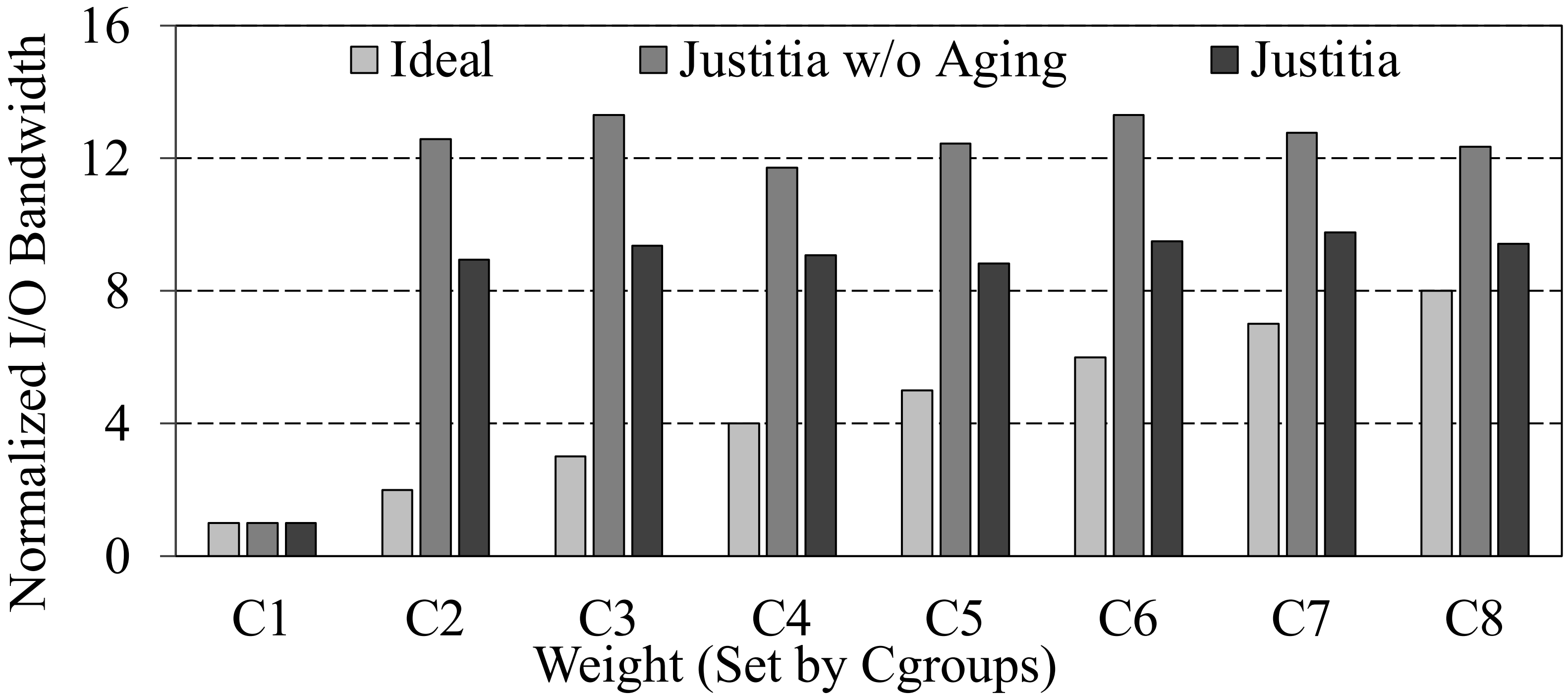
Ref [1] J.Kim et al. “I/O scheduling schemes for better I/O proportionality on flash-based SSDs”

Evaluation (Fileserver)



- Compared with the conventional, Justitia achieves better I/O proportionality
  - Conventional: 1 : 1.51 : 2.02 : 2.40 : 2.63 : 2.71 : 3.07 : 3.31
  - Justitia: 1 : 1.73 : 2.24 : 2.65 : 3.04 : 3.75 : 4.37 : 6.26

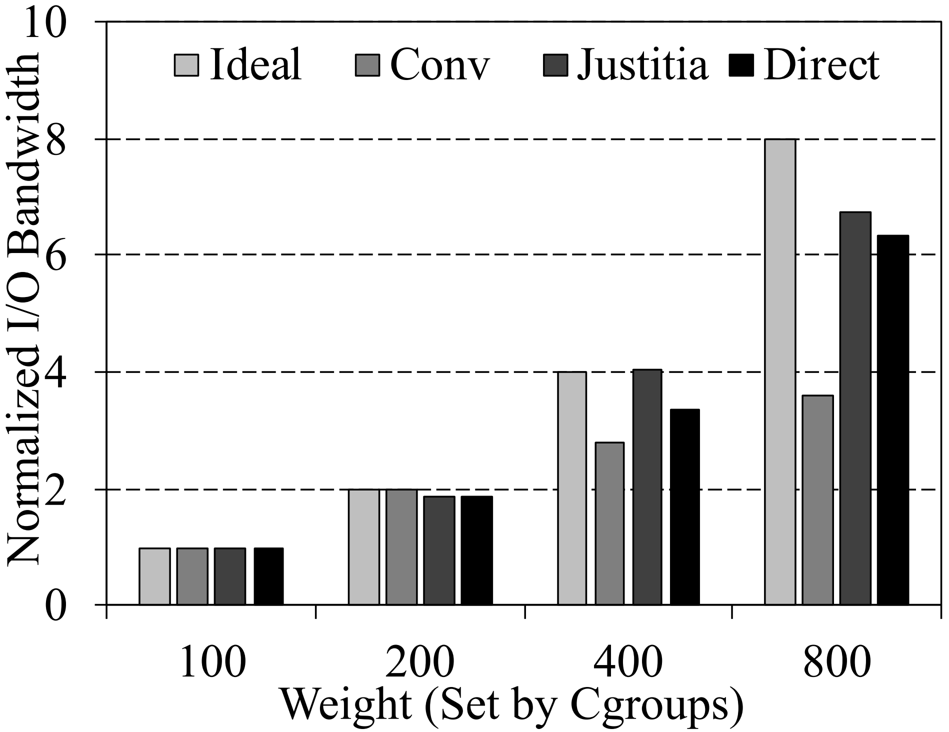
Evaluation (Aging Technique)



Extreme case where C1's weight: 100, C2-C8's weight: 1000

- Justitia without aging: 1 : 12.57 : 13.31 : 11.72 : 12.443 : 13.31 : 12.77 : 13.35 (PV: 2.31)
- Justitia: 1 : 8.94 : 9.36 : 9.08 : 8.83 : 9.49 : 9.77 : 9.43 (PV: 0.64)

Evaluation (Re-read)



- Justitia achieves better I/O proportionality than the other cases
  - PV of Conventional: 1.4
  - PV of Justitia: 0.33
  - PV of Direct I/O: 0.61

Conclusion

- Cgroups support only block-level I/O proportionality, rather than application-level I/O proportionality

- The conventional page cache management do not consider I/O weight either in page allocation and reclamation

- Justitia: a new page cache management for application-level I/O proportionality
    - Weight-aware qspinlock for page allocation
    - Weight-aware page reclamation

- Justitia is available at github.com/kzeoh/Justitia.git

Thank you! Any questions?

Feel free to contact jonggyu@skku.edu