HP-Mapper: A High Performance Storage Driver for Docker Containers

Fan Guo[1], **Yongkun Li**[1], Min Lv[1], Yinlong Xu[1], John C.S. Lui[2]

[1]University of Science and Technology of China

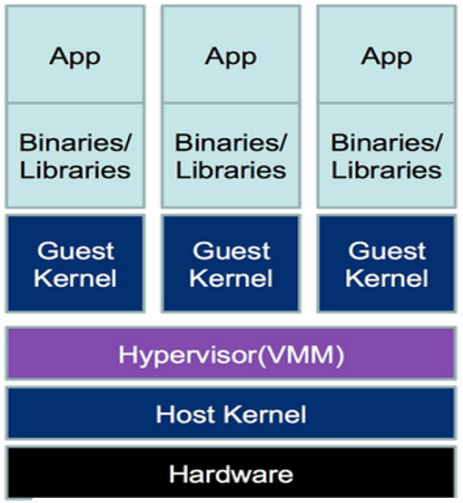
[2]The Chinese University of Hong Kong

Outline

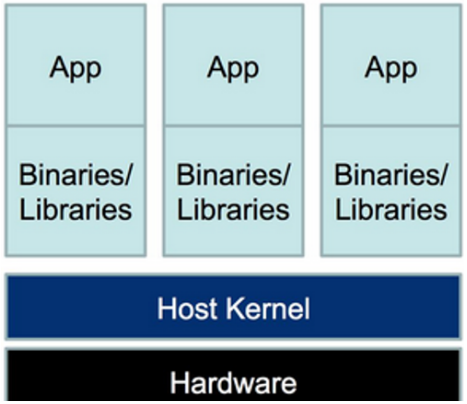
1. Background & Motivation
2. HP-Mapper Design
3. Evaluation
4. Conclusion

Container and Docker

- Container
  - ✓ Process-level virtualization
    - Share host kernel
    - Namespaces, Cgroup
  - ✓ Better performance
    - Fast deployment
    - Low resources usage
    - Near bare-mental performance
- Docker
  - ✓ Most popular container engine
  - ✓ Extensively used in production



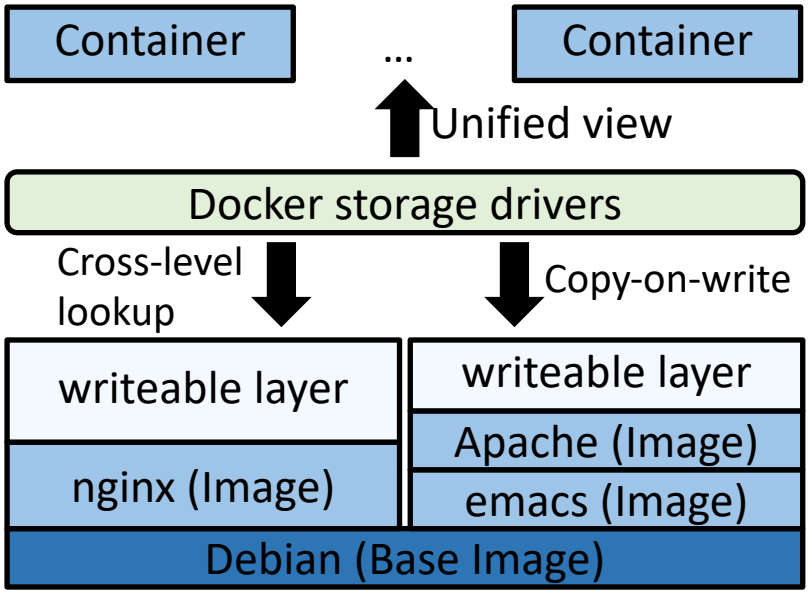
VM



Container

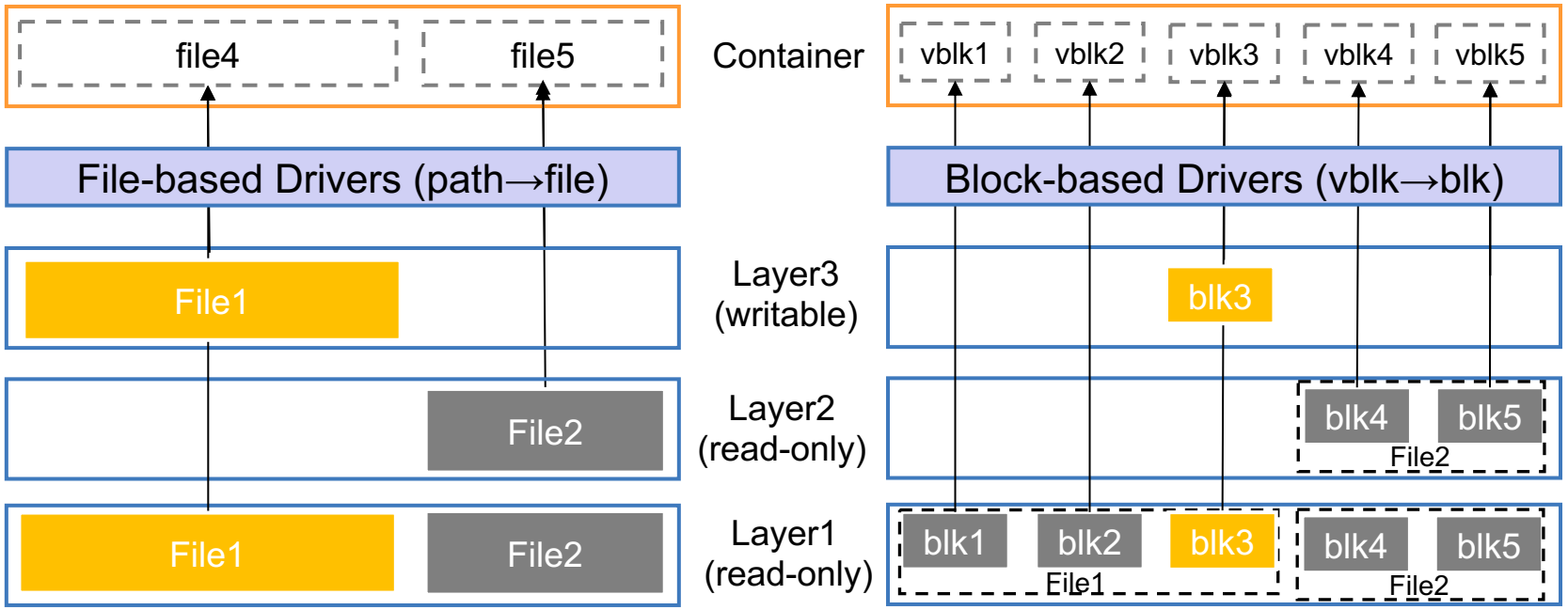
Storage Management of Containers

- Container images
  - ✓ Store all requirements for running the containers
  - ✓ Hierarchical, read-only, sharable
- Storage drivers
  - ✓ Support cross-level lookup and copy-on-write (COW)



Docker Storage Drivers

- File-based Drivers
  - ✓ File-level COW
  - ✓ Share cached data
  - ✓ Overlay2, AUFS
- Block-based Drivers
  - ✓ Block-level COW (≥64KB)
  - ✓ Cannot share cached data
  - ✓ DeviceMapper, ZFS, BtrFS



High COW Latency

- File-based storage drivers incur large COW latency (especially for large files)
  - ✓ Incur large write overhead
  - ✓ Degrade write performance

File Size	4KB	64KB	1MB	16MB
DeviceMapper	0.12	0.74	0.96	1.39
BtrFS	0.09	0.09	0.09	0.10
Overlay2	1.99	2.49	7.14	61.7

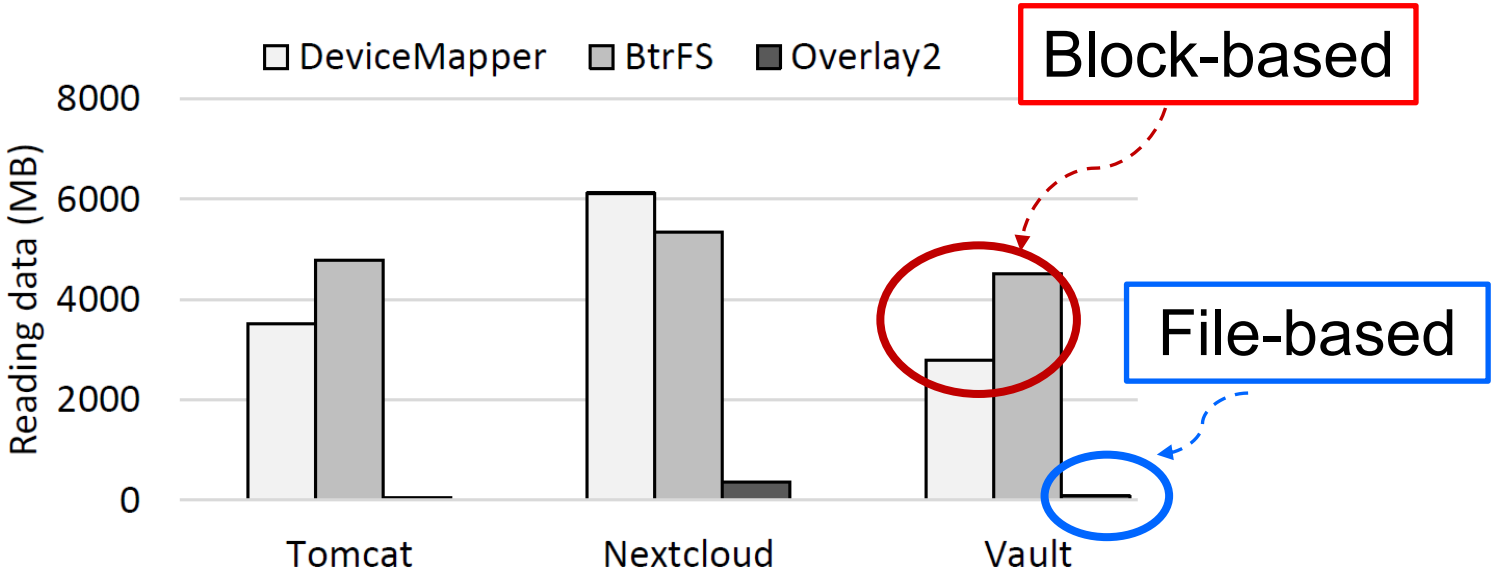
Block-based

File-based

Copy-on-write latency (ms)

Disk I/O Redundancy

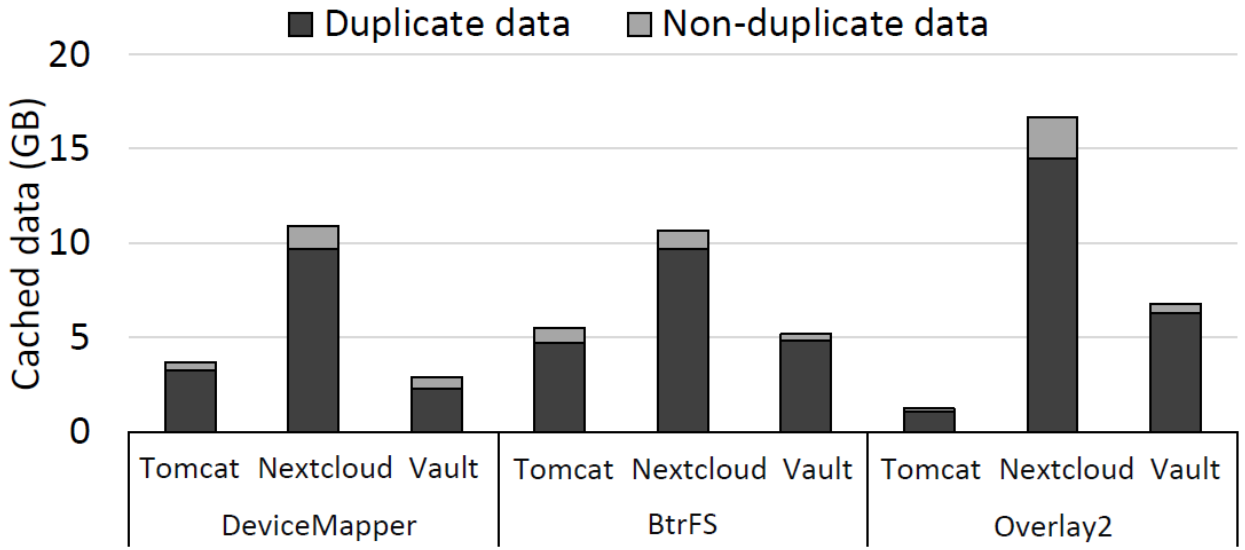
- Block-based drivers introduce many redundant I/Os when reading data from a shared file
  - ✓ Degrade I/O performance
  - ✓ Waste I/O bandwidth



Total amount of reading data during the startup of 64 containers

Cache Redundancy

- Both kinds of storage drivers generate a lot of redundant cached data
  - ✓ Block-based: Read multiple copies of the data (as cache can not be shared)
  - ✓ File-based: Unchanged data in the file are also copied when performing copy-on-write



Motivation

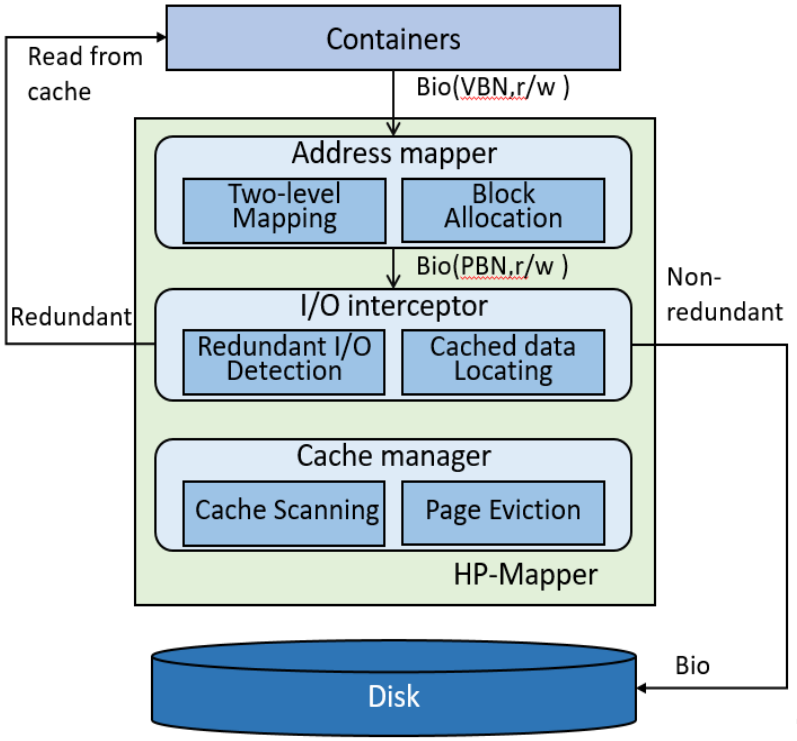
- Limitations of current storage drivers
  - ✓ Tradeoff between write and read performance
  - ✓ Low cache efficiency
- Our goal: Develop a new storage driver for docker containers
  - ✓ Low COW overhead (or high write performance)
  - ✓ High read I/O performance
  - ✓ High cache efficiency

Outline

1. Background & Motivation
2. HP-Mapper Design
3. Evaluation
4. Conclusion

HP-Mapper: A High Performance Storage Driver

- Key idea: HP-Mapper works at the block level and manages physical blocks
  - ✓ Why block level: Low COW overhead
  - ✓ Why physical block: Able to detect redundant I/Os & redundant cached data
- Three modules
  - ✓ Address mapper
  - ✓ I/O interceptor
  - ✓ Cache manager

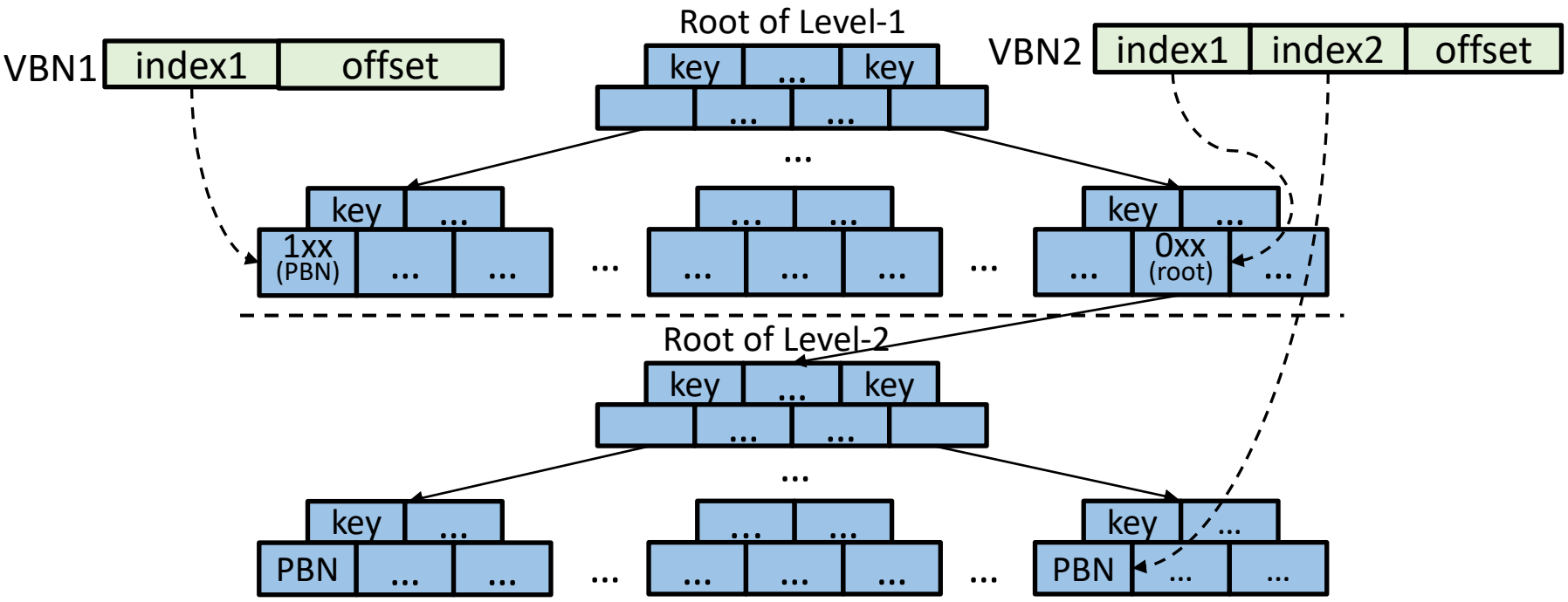


Address Mapper

- Tradeoff exists in block-based management
  - ✓ Large blocks: high COW latency
  - ✓ Small blocks: high lookup and storage overhead due to large metadata size
- HP-Mapper uses a two-level mapping tree
  - ✓ Support two different block sizes
  - ✓ Differentiate different requests w/ on-demand allocation
    - New write: large sequential I/O (large block size)
    - COW: depending on req size

Address Mapper

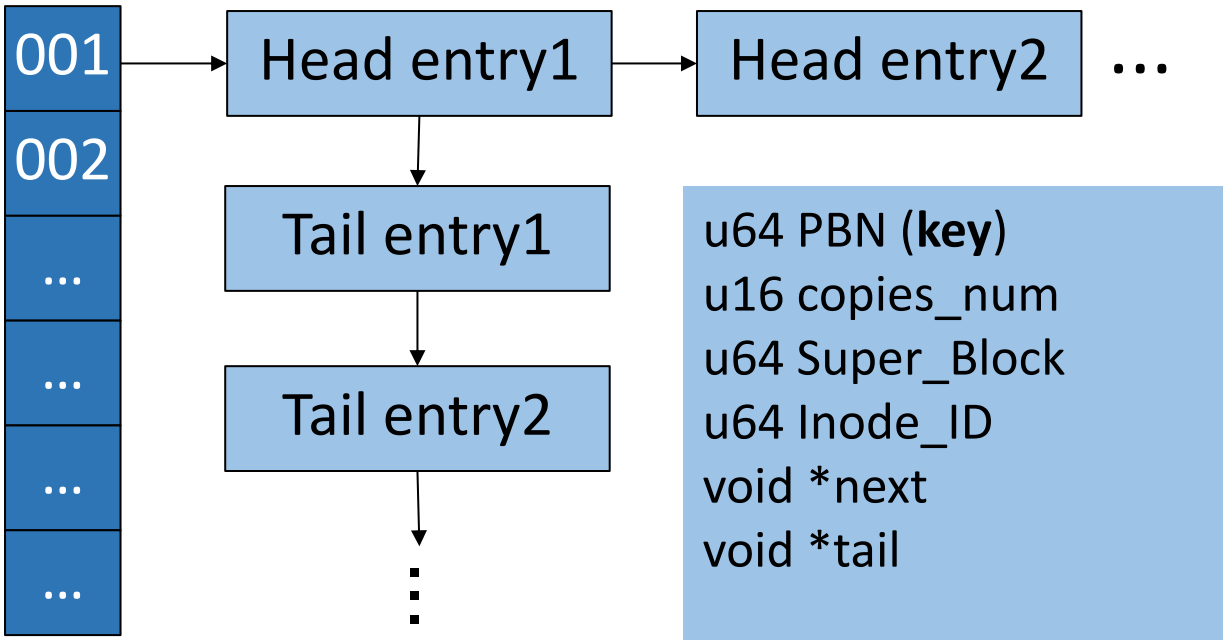
- Two-level mapping tree design



Storage efficiency
Separated block placement + defragmentation

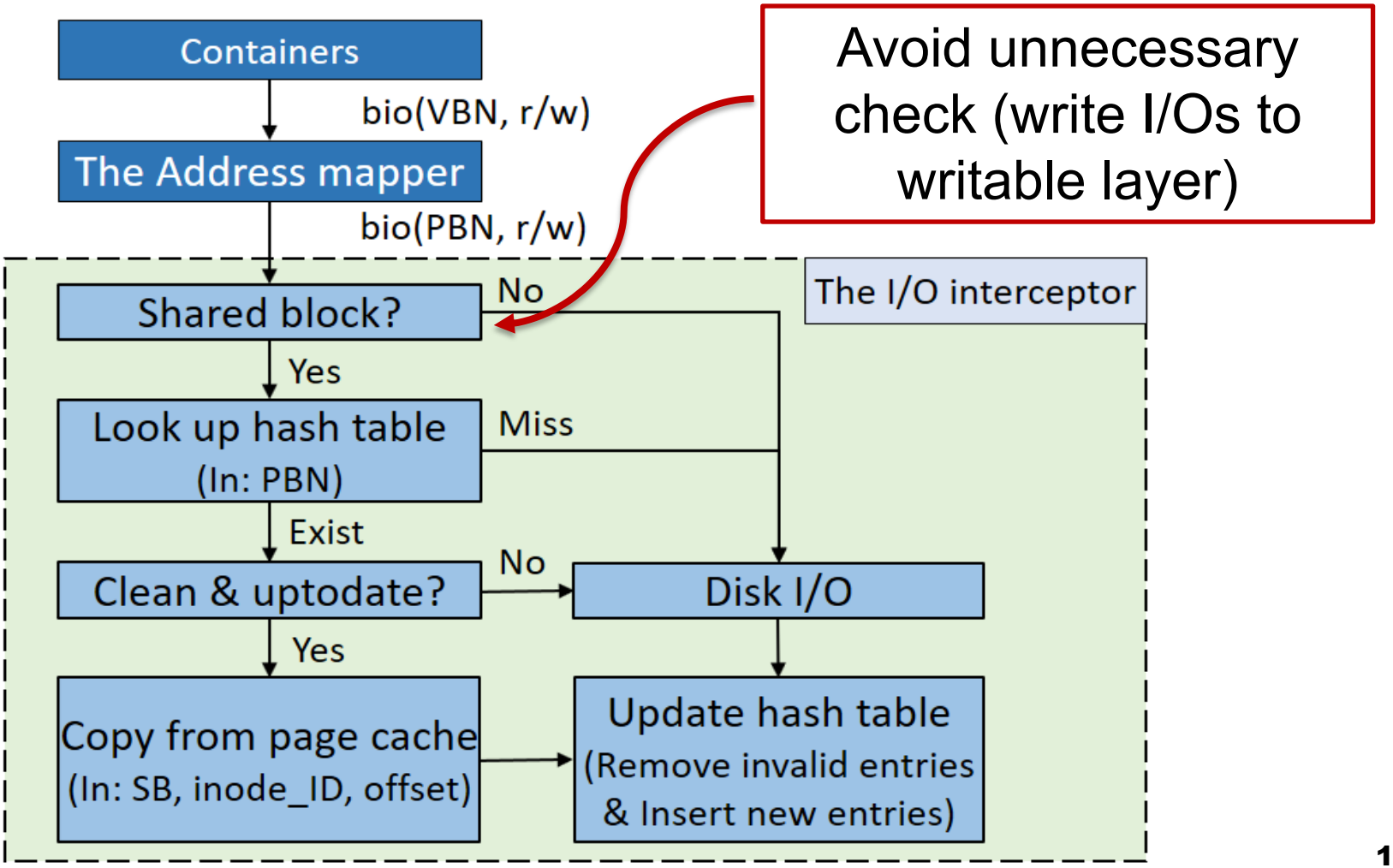
I/O Interceptor – Metadata Management

- How to detect redundant I/O
  - ✓ Indexing with physical block number (PBN)
- PBN-indexing hash table
  - ✓ Entries are linked in a two-dimensional list (LRU)
    - Head entry (latest copy) + Tail entry (other copies)



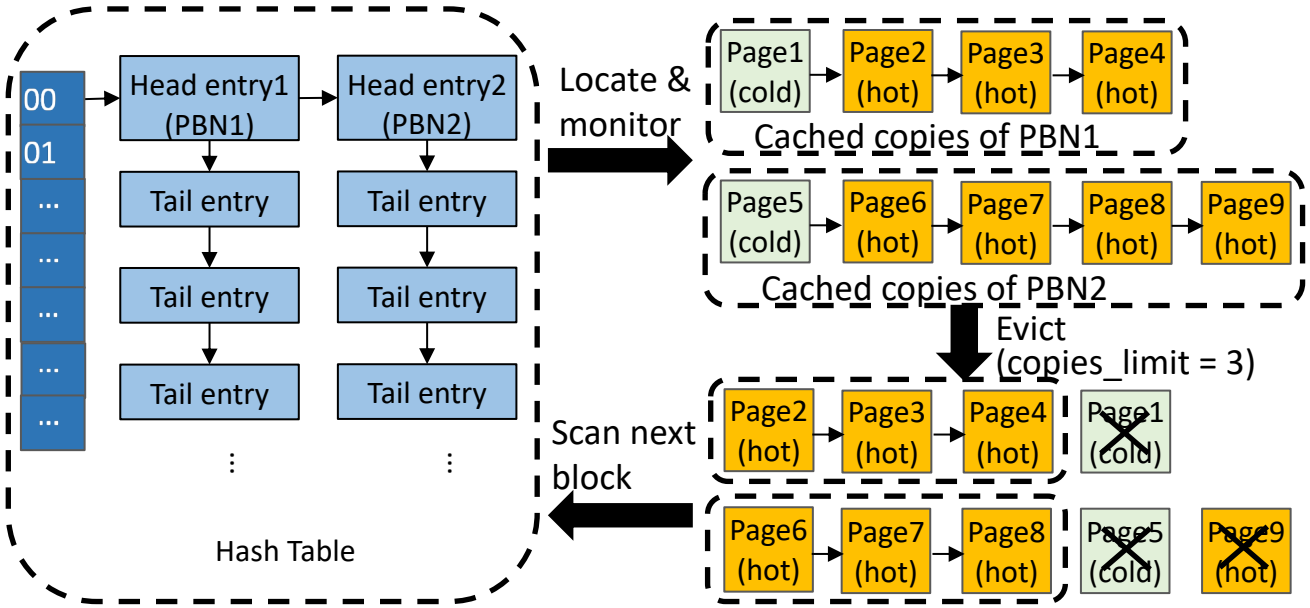
I/O Interceptor - Workflow

- Detect redundant I/O w/ PBN-index



Cache Manager

- How to remove redundant copies in cache
  - ✓ Periodically scan the hash table to locate cached pages
  - ✓ Maintain the hotness of each page (multiple LRU)
- Page eviction: cache hit ratio vs. cache usage
  - ✓ Limit # of copies: utilization-aware adjustment
  - ✓ Hotness-aware eviction



Outline

1. Background & Motivation
2. HP-Mapper Design
3. Evaluation
4. Conclusion

Experiment Setup

- Prototype
  - ✓ Act as a plugin module in Linux kernel 3.10.0
  - ✓ Backing file system: Ext4
- Workloads
  - ✓ Container images: Tomcat, Nextcloud, Vault
- Overhead of HP-Mapper

CPU Overhead	Mem. overhead		Lookup overhead	
	MT	HT	MT	HT
4.1%	3.3MB	10.6MB	1.1 us	< 0.1 us

Overhead of HP-Mapper
(MT represents Mapping Tree, HT represents Hash Table)

Reduction of COW Latency

- Copy-on-write(COW) latency

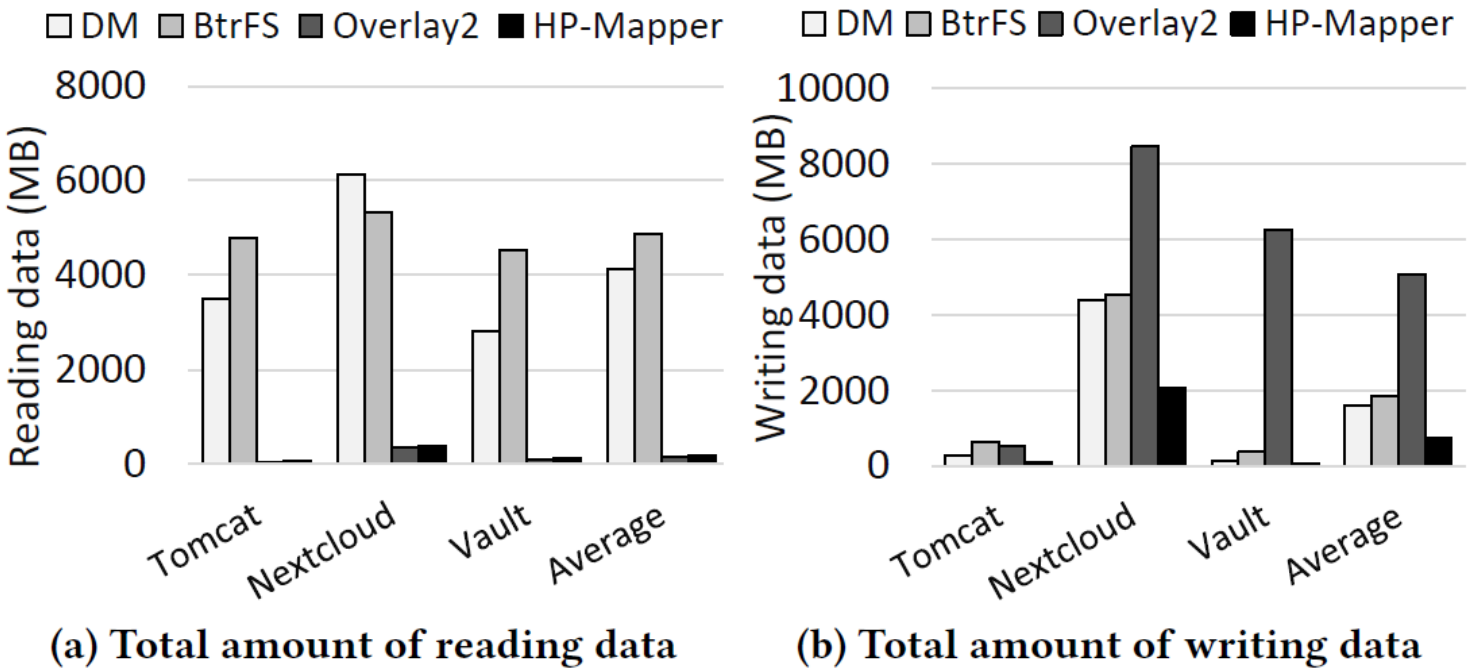
File Size	4KB	64KB	1MB	16MB
DeviceMapper	0.13	0.74	0.96	1.39
BtrFS	0.09	0.09	0.09	0.10
Overlay2	1.99	2.49	7.14	61.7
HP-Mapper	0.07	0.12	0.55	0.57

Copy-on-write latency (ms)

HP-Mapper reduces up to more than 90% COW latency comparing with DeviceMapper and Overlay2

Reduction of Redundant I/Os

- Total amount of reading/writing data when launching 64 containers from a single image

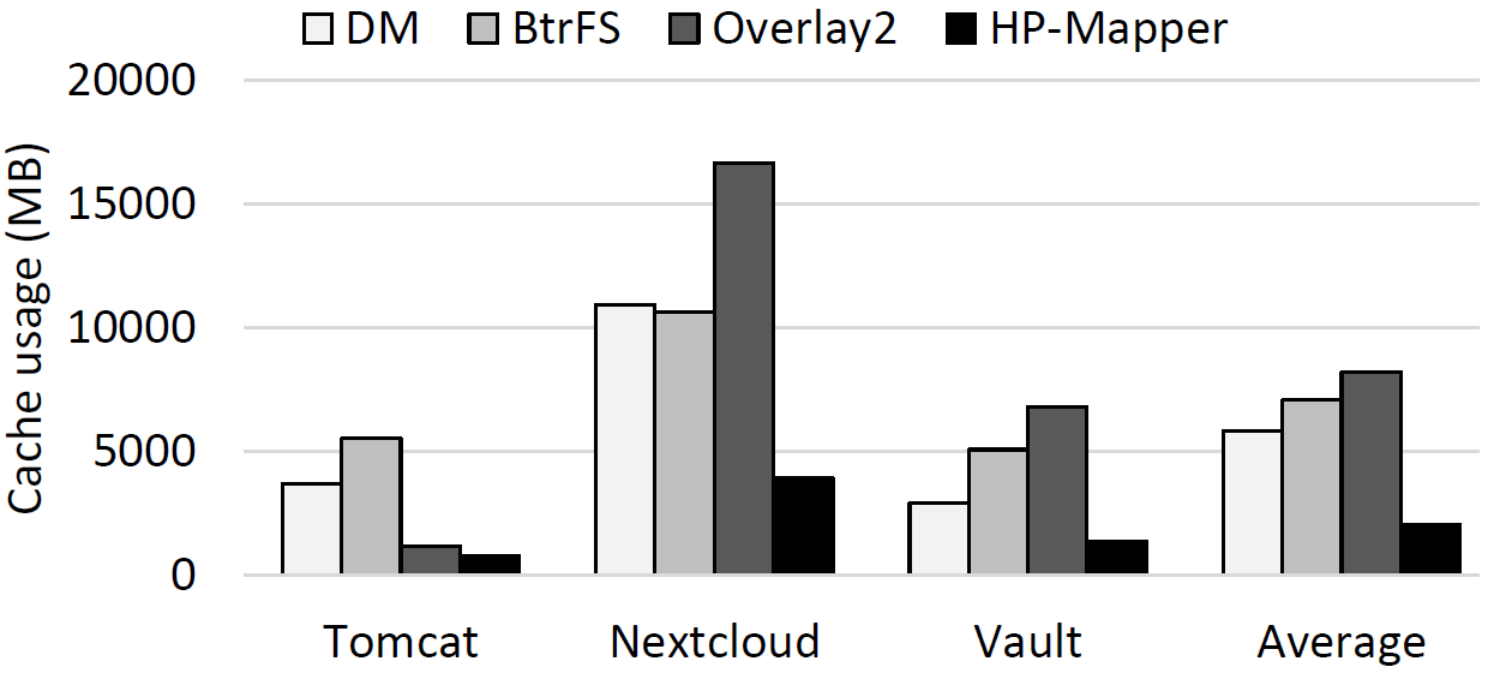


(a) Total amount of reading data (b) Total amount of writing data

HP-Mapper efficiently removes redundant read I/Os, and also reduces more than 50% writing data on average

Improvement of Cache Efficiency

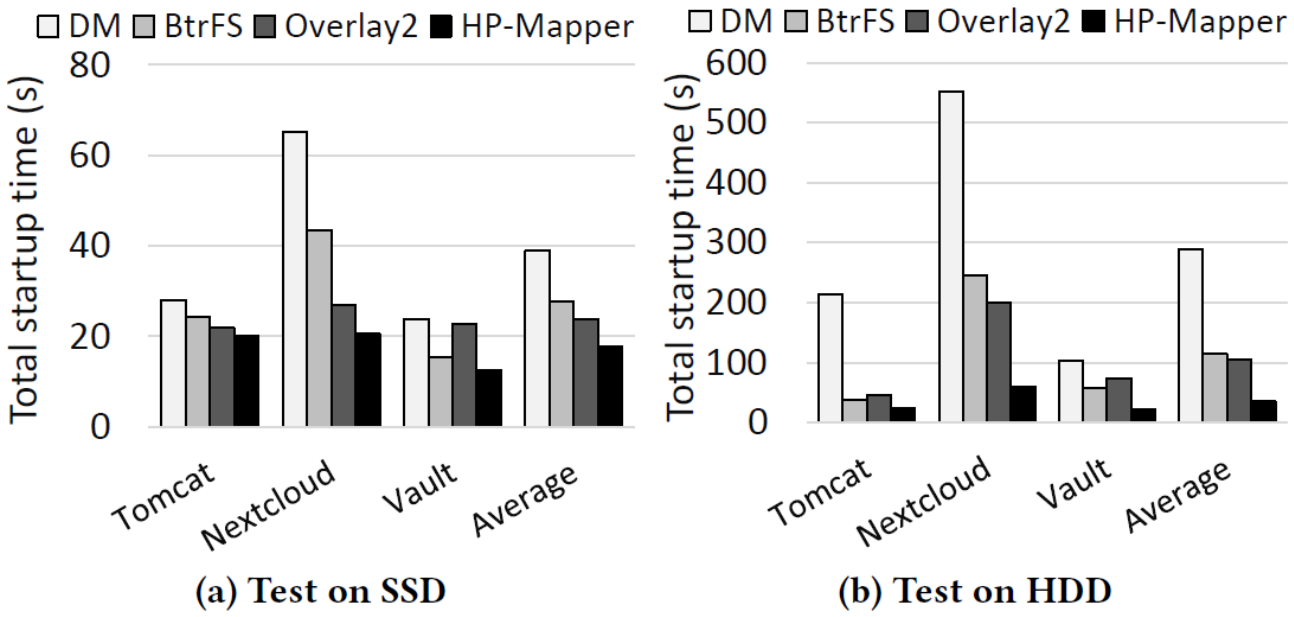
- Cache usage when starting 64 containers from a single image



HP-Mapper reduces more than 65% cache usage on average

Improvement of Startup Time

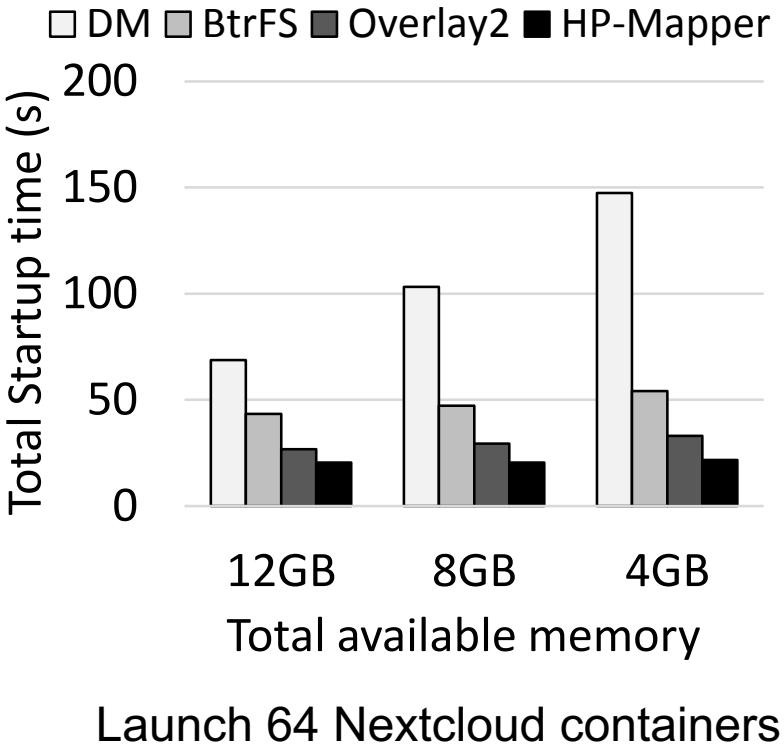
- Total startup time when launching 64 containers from a single image on SSD/HDD



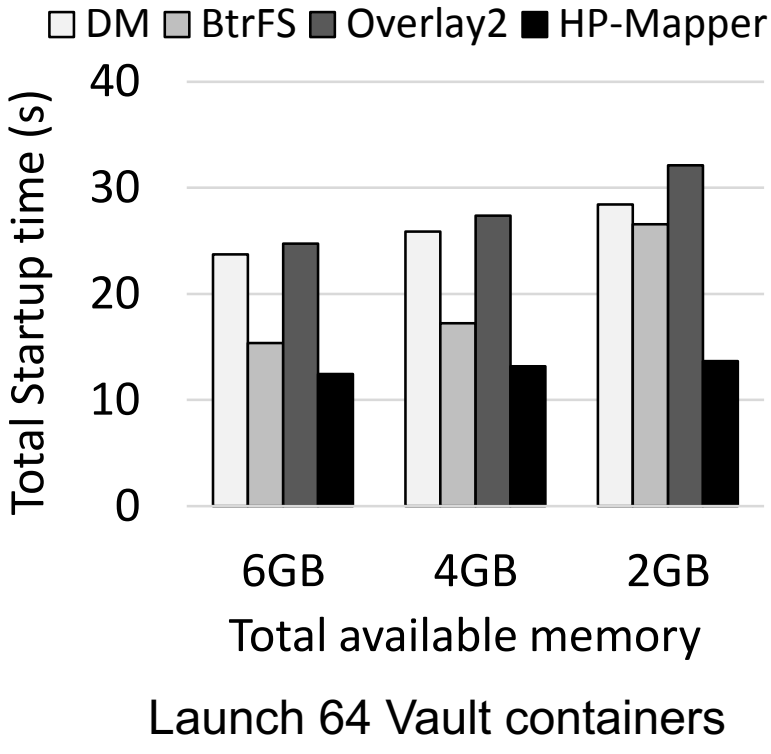
HP-Mapper achieves up to 2.0× - 7.2× faster startup speed than the other three storage drivers

Improvement of Startup Time

- Total startup time when launching 64 containers in memory-scarce systems



Launch 64 Nextcloud containers



Launch 64 Vault containers

HP-Mapper achieves larger improvement as memory size decreases

Outline

1. Background & Motivation
2. HP-Mapper Design
3. Evaluation
4. Conclusion

Conclusion

- Tradeoffs exist for Docker storage drivers
  - ✓ File-based: High COW overhead
  - ✓ Block-based: Low cache efficiency & redundant I/O
- We develop HP-Mapper which achieves
  - ✓ Low COW overhead by following block-based design with differentiated block sizes
  - ✓ High I/O efficiency by intercepting redundant I/Os
  - ✓ High cache efficiency by enabling cache sharing and hotness-aware management

Thanks!

Q&A

Yongkun Li

ykli@ustc.edu.cn

http://staff.ustc.edu.cn/~ykli