SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

Caching on PMEM: An Iterative Approach

Yao Yue[†] & Juncheng Yang[†,‡]

[†] Twitter Inc.

[‡] Carnegie Mellon University

Caching on PMEM at Twitter

- **Basic Considerations**
- **Iterations**
  - Testing and modification in lab
  - Testing in prod
  - In-house development for PMEM
- **Lessons Learned**

Incentives, Hypotheses, & Constraints

Caching @ Twitter

Clusters
>300 in prod

Hosts
many thousands

Instances
tens of thousands

Job size
2-6 core, 4-48 GiB

QPS
max 50M (single cluster)

SLO
p999 < 5ms*

Mission critical ⇒ availability

Large resource footprint ⇒ cost

Lots of instances ⇒ fast restart

A large scale analysis of hundreds of in-memory cache clusters at Twitter [OSDI'20]

Why Put Cache on PMEM

- **Cache more data per instance**
  - Reduce TCO if memory bound
  - Improve hit rate

- **Take advantage of data durability**
  - Graceful shutdown and faster rebuild
  - Improve data availability during maintenance

Constraints

- **Maintainable**
  - Same codebase
  - Retain high-level APIs
- **Operable**
  - Flexible invocation and configuration
  - Predictable performance

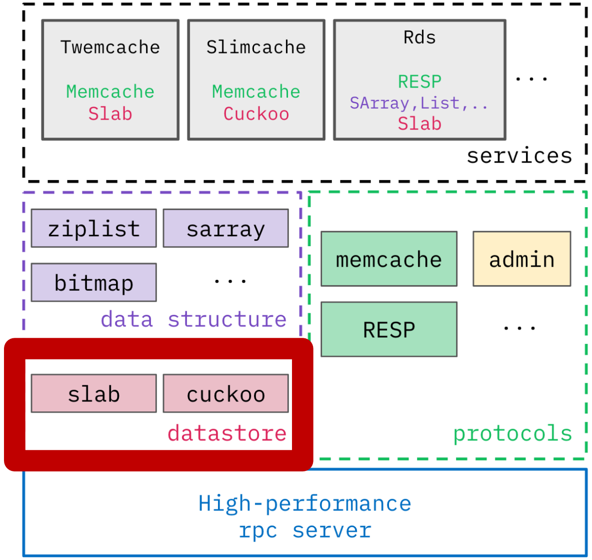
An Iterative Approach

Principles

- ***Show Progress***
- **Be Flexible**
    - Identify issues
    - Modify future plan
- **Be Confident**
    - Verify hypotheses
    - Meet constraints

The Plan

1. Use a modular caching framework
2. PMEM with unaltered cache code (lab, prod)
3. PMEM with minimally altered cache (lab, prod)
4. Design for/with PMEM



Pelikan: A Modular Cache

Test Design

Instance density

18-30 instances / host

Object size

Between 64 and 2048 bytes

Dataset size

Between 4GiB and 32 GiB / instance

# of Connection per server instance

100 / 1000

R/W ratio

Read-only, read heavy, write heavy

Focuses

- Throughput with latency constraints
- PMEM vs. DRAM
- Memory mode vs. AppDirect

- Scalability with dataset size
- Bottleneck analysis

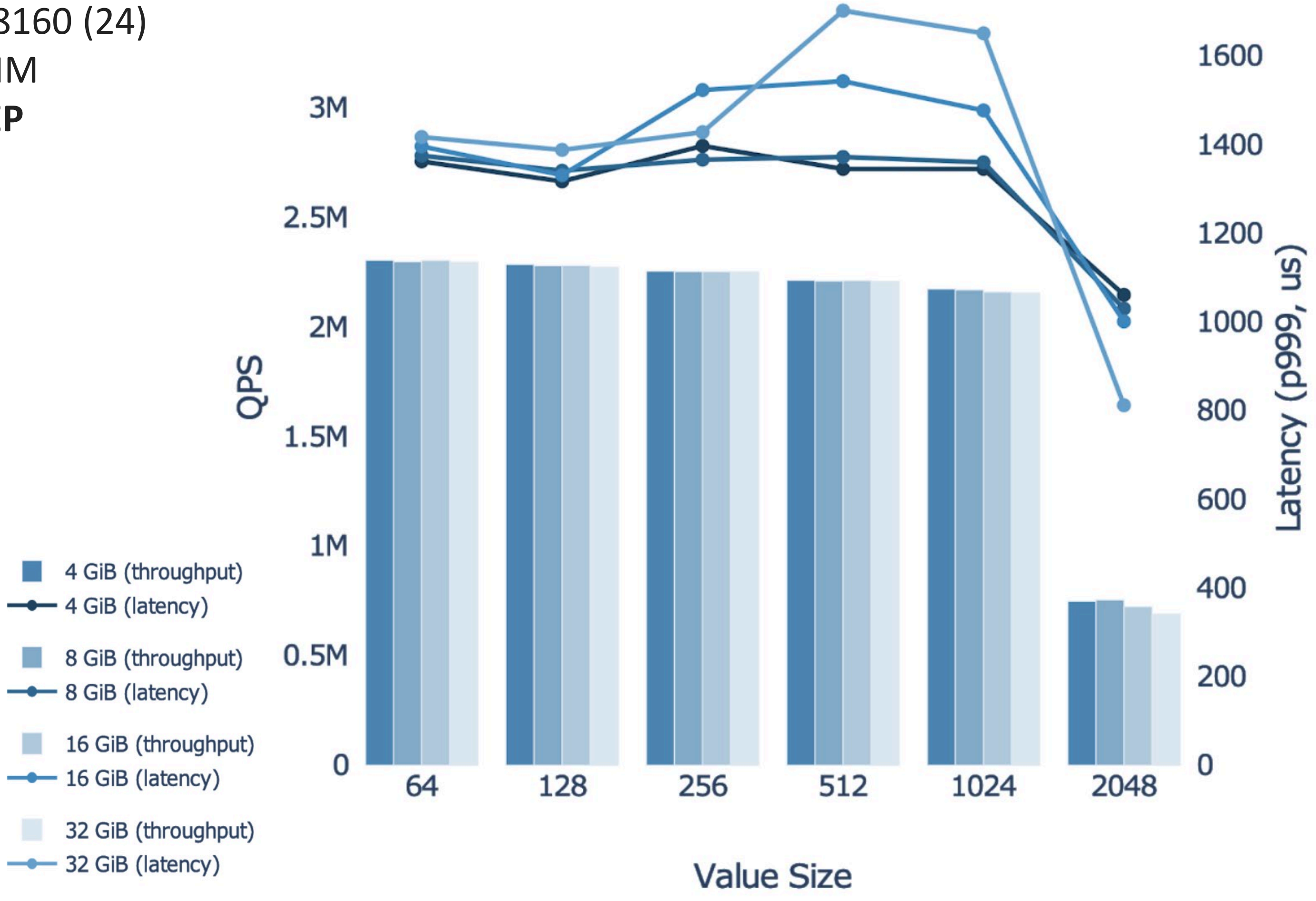
Hardware Config (Intel lab)

- 2 X Intel Xeon 8160 (24)
- 12 X 32GB DIMM
- **12 X 128GB AEP**
- 2-2-2 config
- 1 X 25Gb NIC

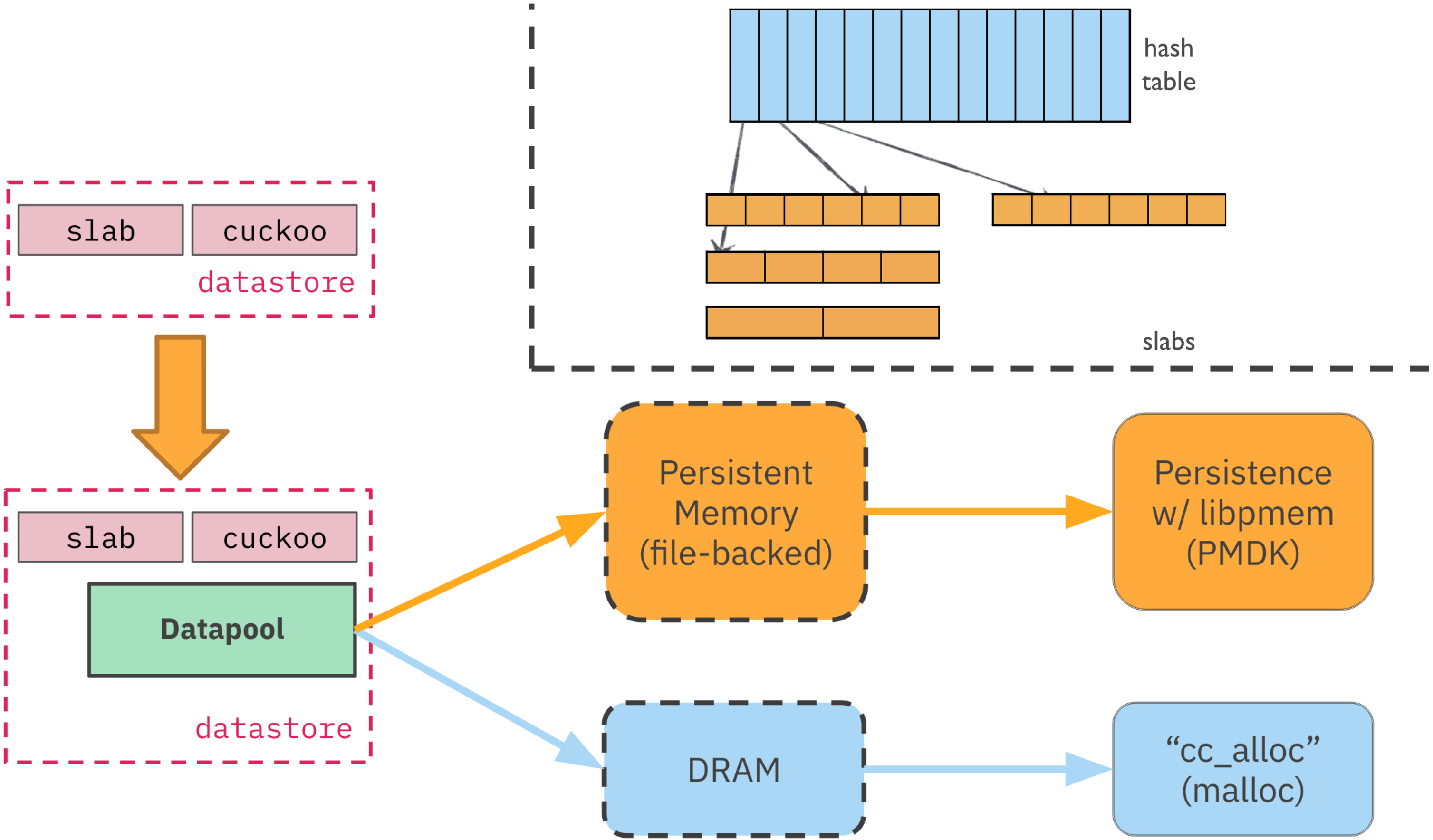
Test Config

- 30 jobs/host
- key size 32B
- 100 conn/job
- 90R:10W

Memory mode



Datapool Abstraction with PMDK



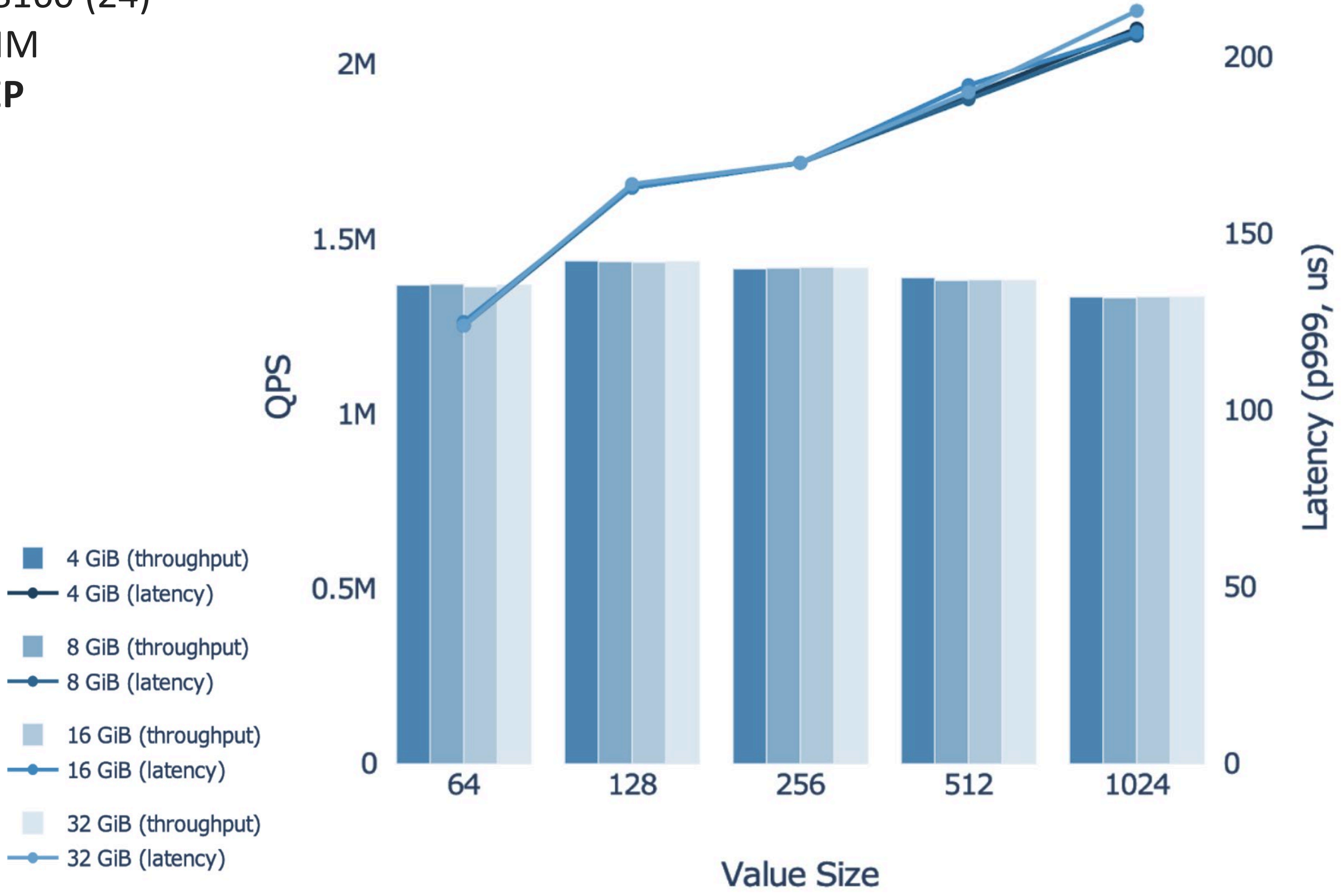
Hardware Config (Intel lab)

- 2 X Intel Xeon 8160 (24)
- 12 X 32GB DIMM
- **12 X 128GB AEP**
- 2-2-2 config
- 1 X 25Gb NIC

Test Config

- 24 jobs/host
- key size 32B
- 100 conn/job
- 90R:10W

AppDirect mode



Rebuild Performance

- **Single instance**
  - 100 GiB of slab data
  - complete rebuild: 4 minutes
- **Concurrent**
  - 18 instances per host
  - complete rebuild: 5 minutes
- **Potential impact**
  - Speed up maintenance by 1-2 orders of magnitude
  - But needs other changes for real adoption

Hardware Config (Twitter prod)

- 2 X Intel Xeon 8160 (20)
- 12 X 16GB DIMM
- **4 X 512GB AEP**
- 2-1-1 config
- 1 X 25Gb NIC

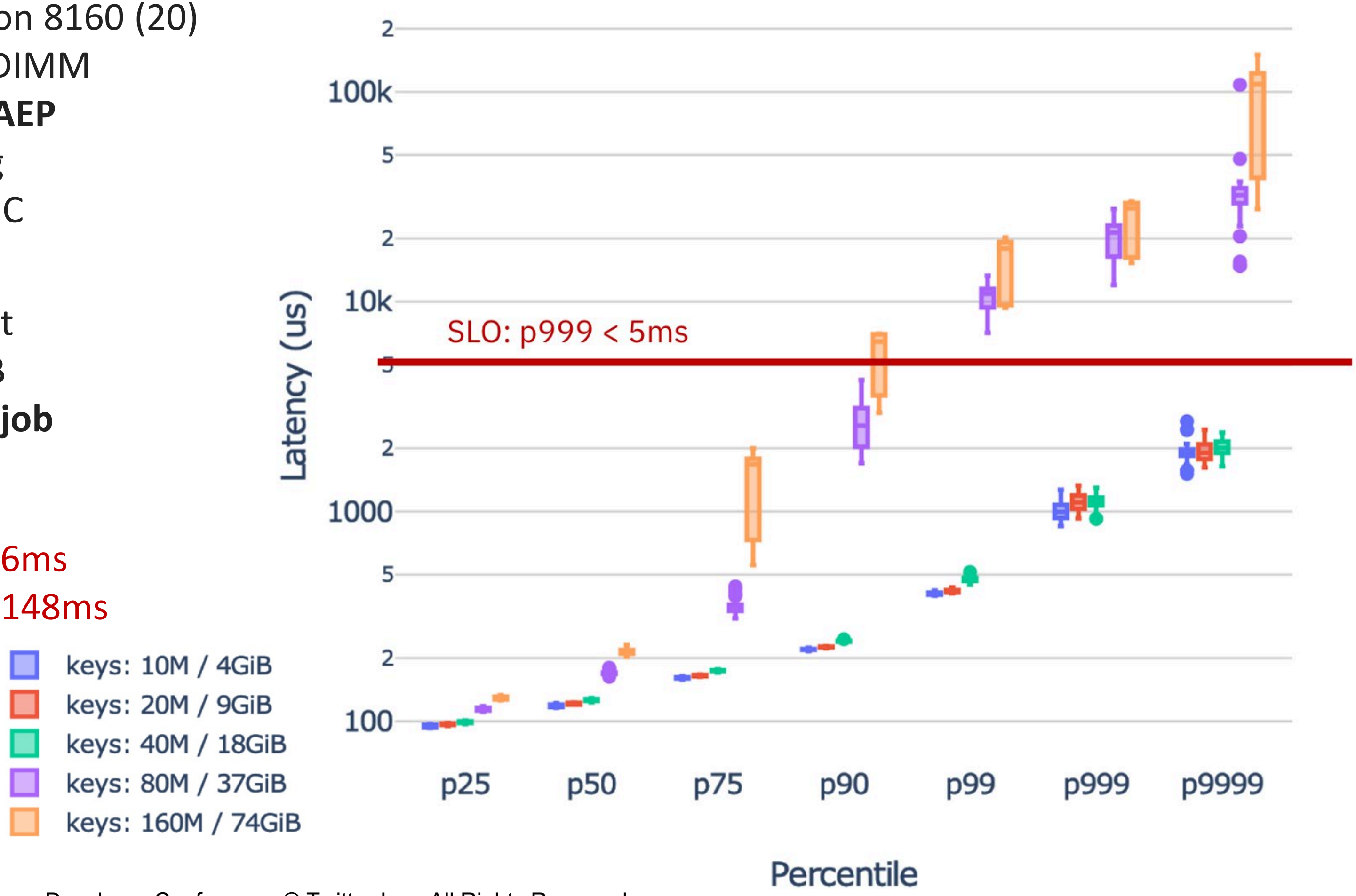
Test Config

- 20 jobs/host
- key size 64B
- **1000 conn/job**
- read-only

p999 max = 16ms

p9999 max = 148ms

Memory mode: throughput 1.08M QPS



Hardware Config (Twitter prod)

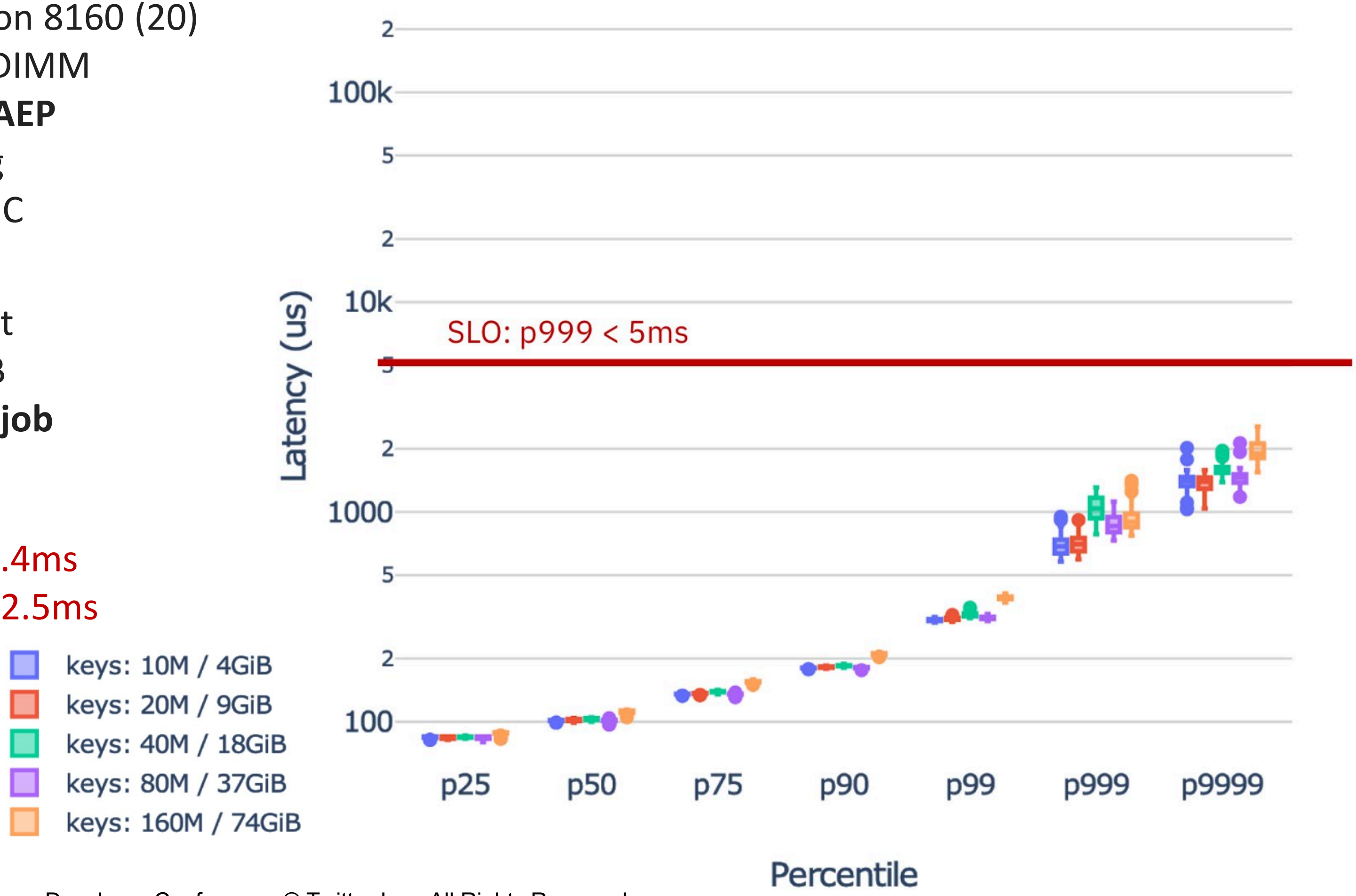
- 2 X Intel Xeon 8160 (20)
- 12 X 16GB DIMM
- **4 X 512GB AEP**
- 2-1-1 config
- 1 X 25Gb NIC

Test Config

- 20 jobs/host
- key size 64B
- **1000 conn/job**
- read-only

p999 max = 1.4ms
p9999 max = 2.5ms

AppDirect mode: throughput 1.08M QPS



A "mid-term" Retrospective

- **What's cache's bottleneck?**
  - Network stack
  - PMEM bandwidth, *if* channel number is small
- **Memory vs AppDirect perf**
  - AppDirect far more predictable
  - Code change is modest
- **How can we improve our story on recovery?**
  - Need to rethink metadata layout
  - Need to rethink direct use of pointers
  - Need to rethink cache operations (future work)

Pelikan Storage Module Redesign

- **What is PMEM good/bad at?**
  - Sequential and large accesses
- **What is a cache's memory access pattern?**
  - Random reads and random writes

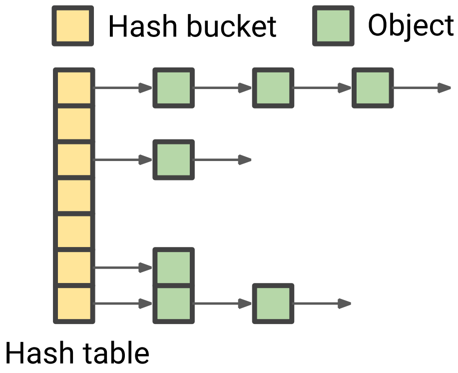
Does this remind you of anything?

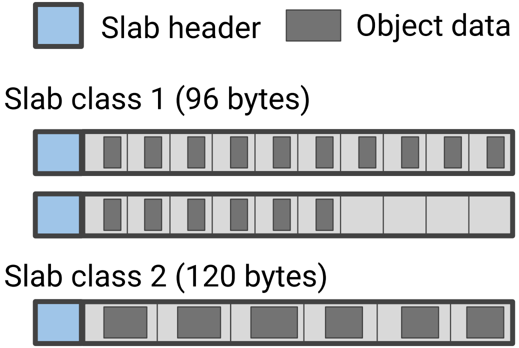
Pelikan Storage Module Redesign

- Log-structured file system/key-value store
- Can we use the same design here?
  - Not really
  - Multiple sources of random memory accesses
- Where are the random memory accesses?
  - Hash table
  - Object storage

Source of Random Memory Access

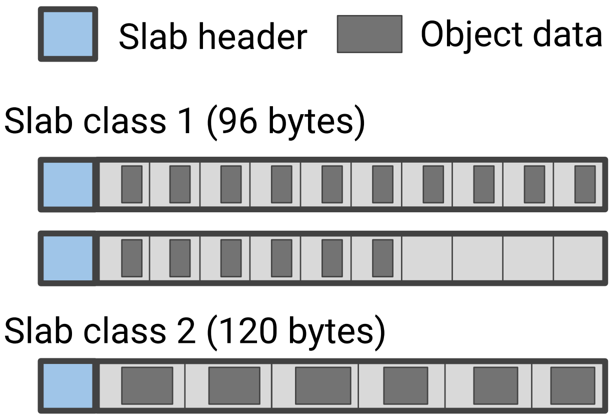
- Object chained hash table
  - Random read and random write
- Slab memory allocation
  - Object write, expiration, deletion, and eviction*





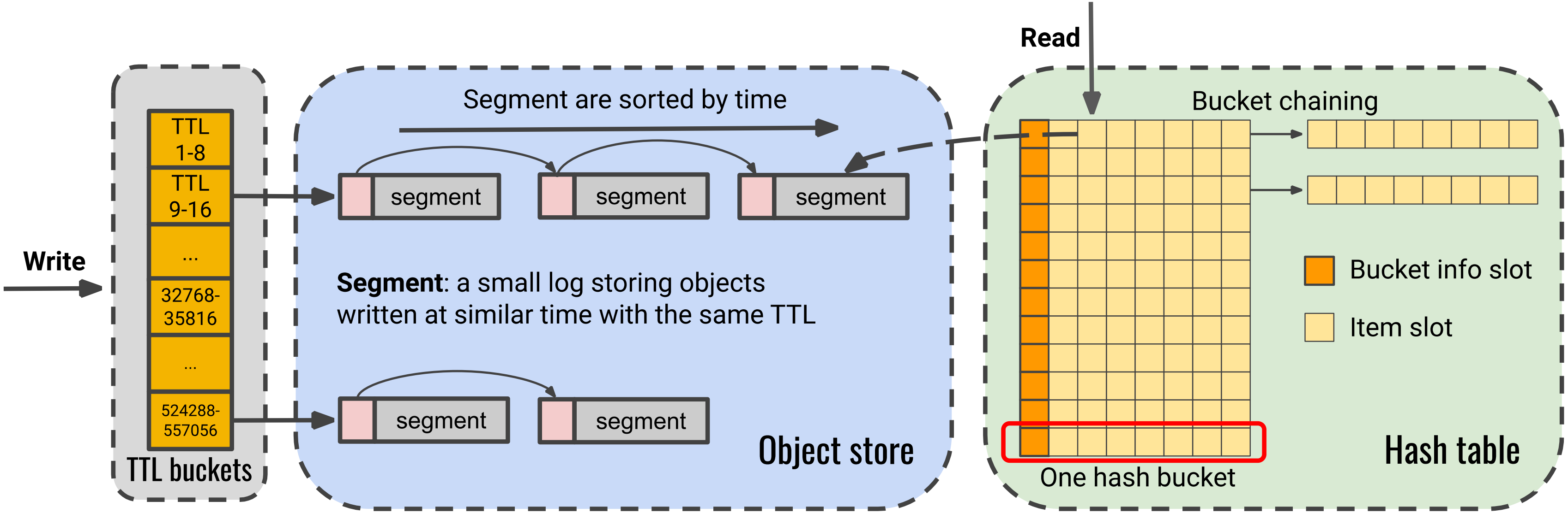
How Pelikan Slab Module Optimizes for PMEM

- Slab eviction
  - Batched evictions without updating metadata for every object
  - Object writes are sequential
- Not enough
  - Object expiration
  - Object delete



Segcache: a Redesign of Storage Module

- Segcache: segment-structured cache



Segcache Overview

- Transform ***all*** *random PMEM writes* into sequential writes
- Move *small random* metadata *reads and writes* into DRAM
- Use PMEM only as object store
  - get request: read only once and no write
  - set request: write once sequentially
  - all bookkeeping: sequentially in batch
- Moreover…

*Some of these have already been partially achieved by Pelikan slab module

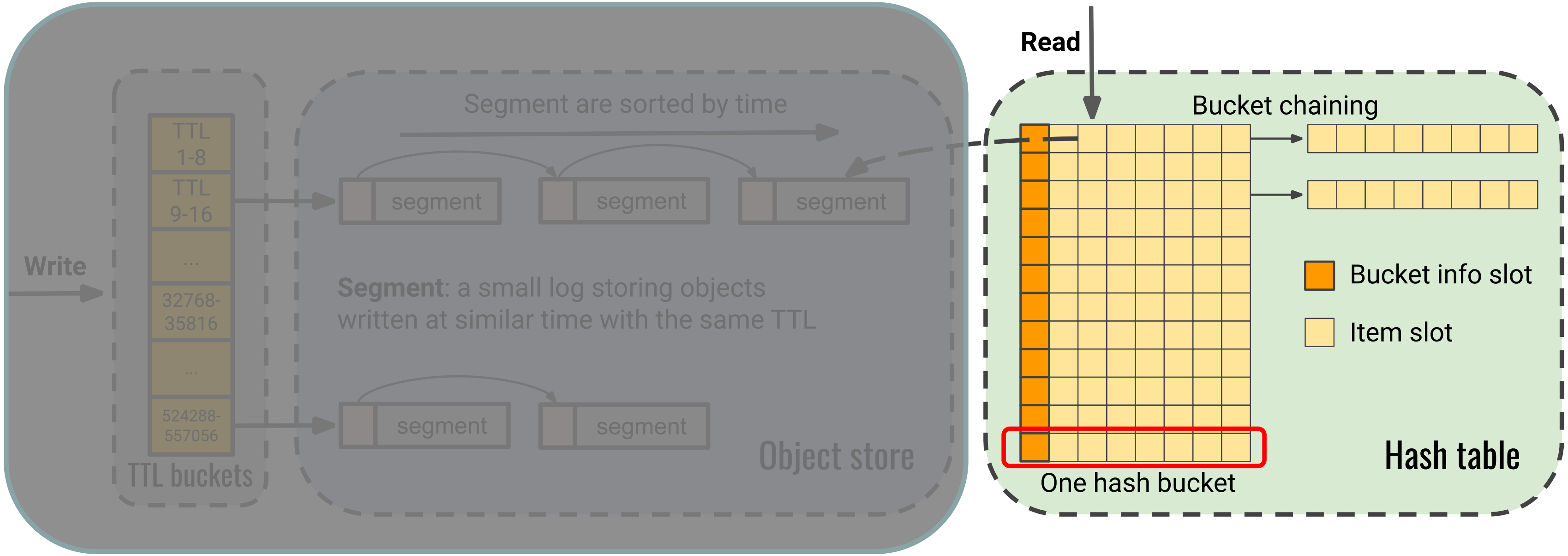
the secret source why Pelikan is better than Memcached on PMEM

Segcache Overview

- **Better memory efficiency**
  - Efficient removal of *all* expired objects
  - Small object metadata (38 bytes -> 5 bytes)
  - Merge-based segment eviction algorithm
  - => 60% memory footprint reduction on Twitter's largest cache cluster

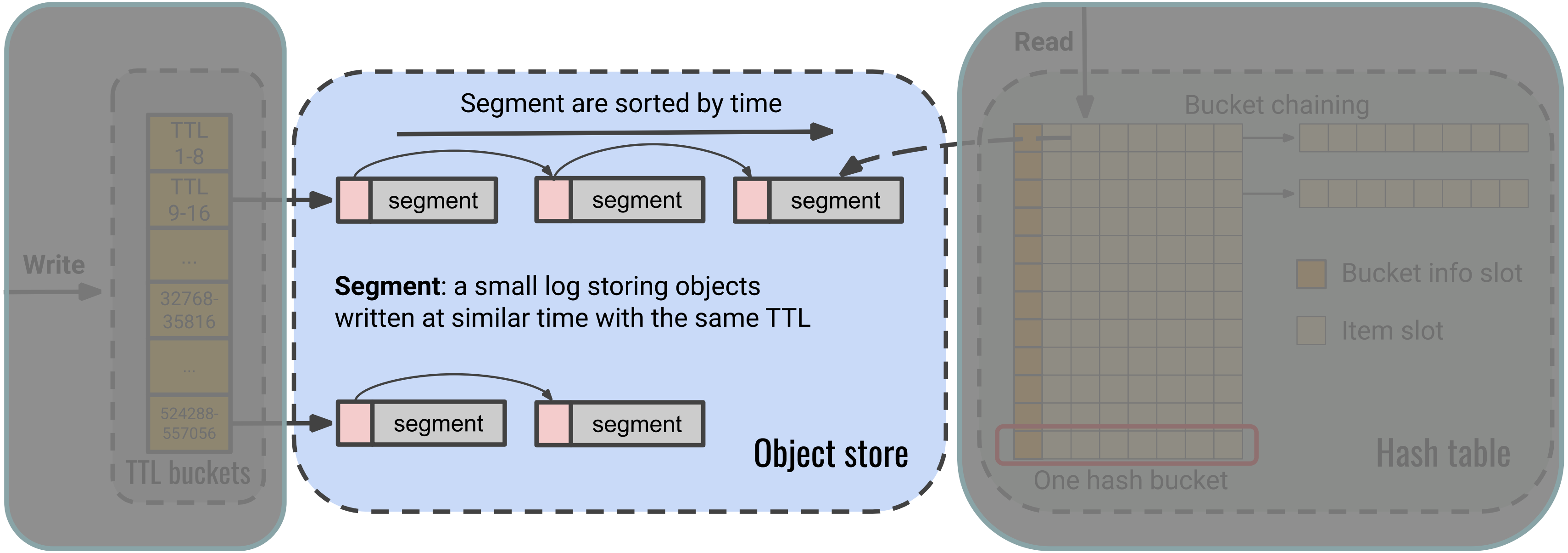
Transform *all random writes* into sequential writes

- Hash table



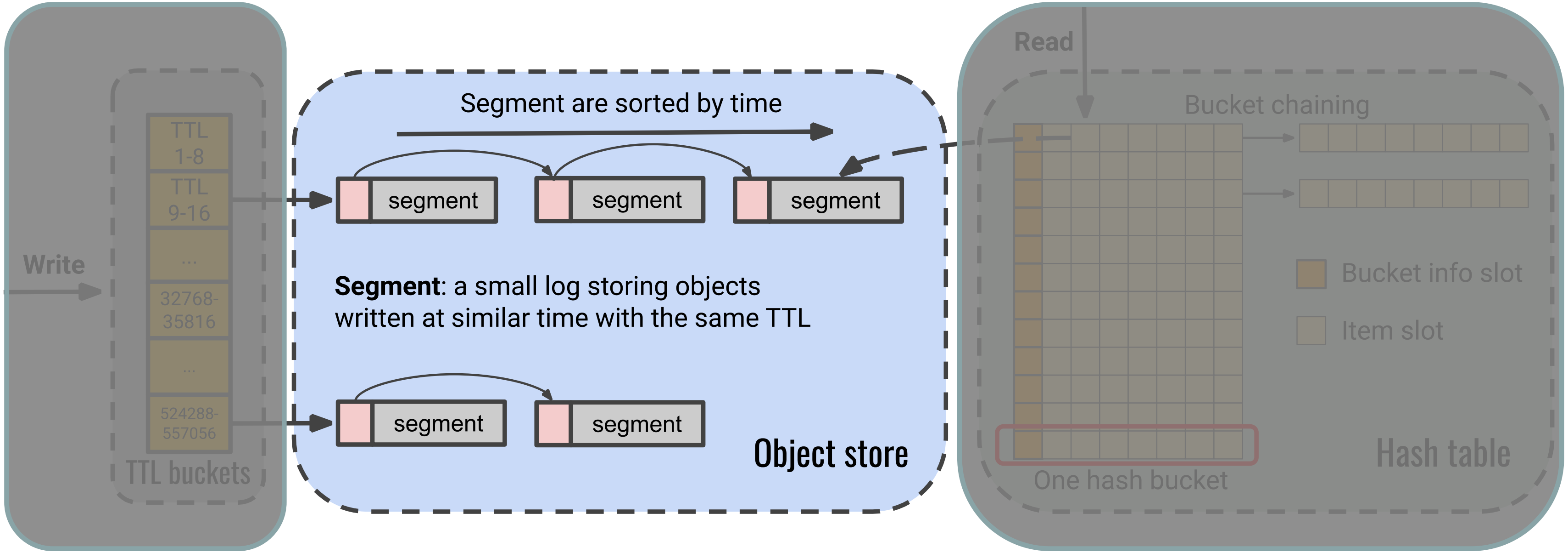
Transform *all random writes* into sequential writes

- Segment: small log, append only
- Segment headers: **shared** object metadata



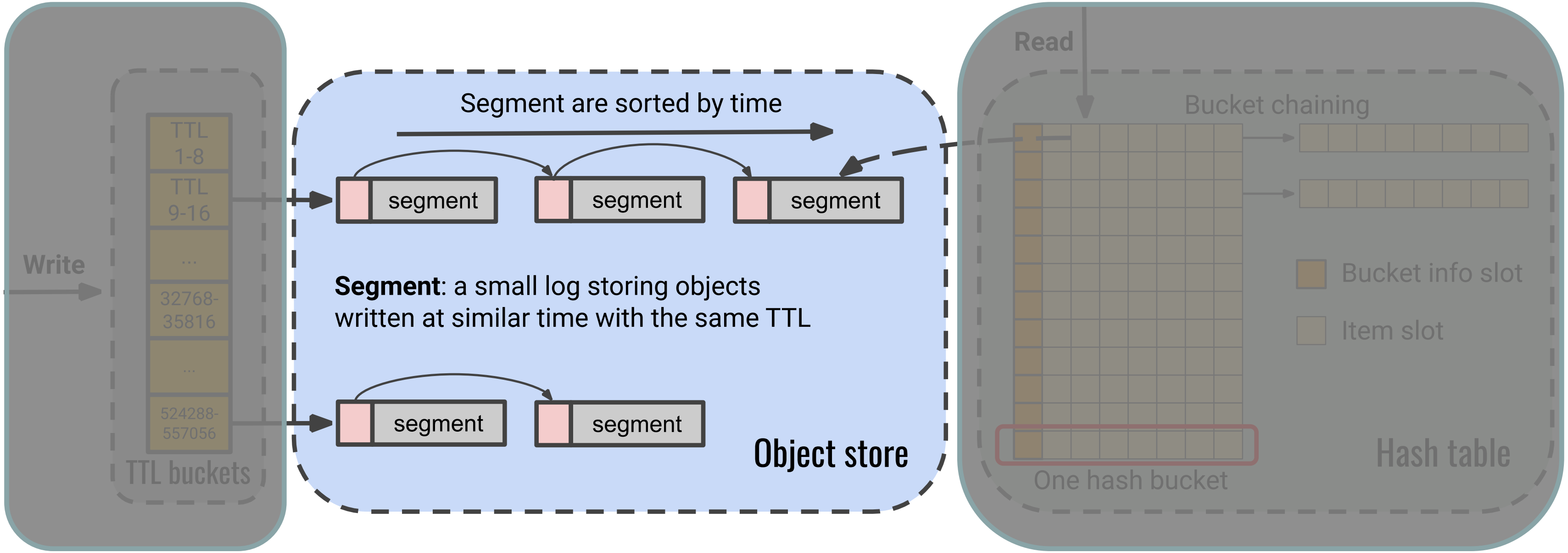
Transform *all random writes* into sequential writes

- Delete: remove hash table entry
- Expire: one segment at a time



Move *small random* metadata *operations* into DRAM

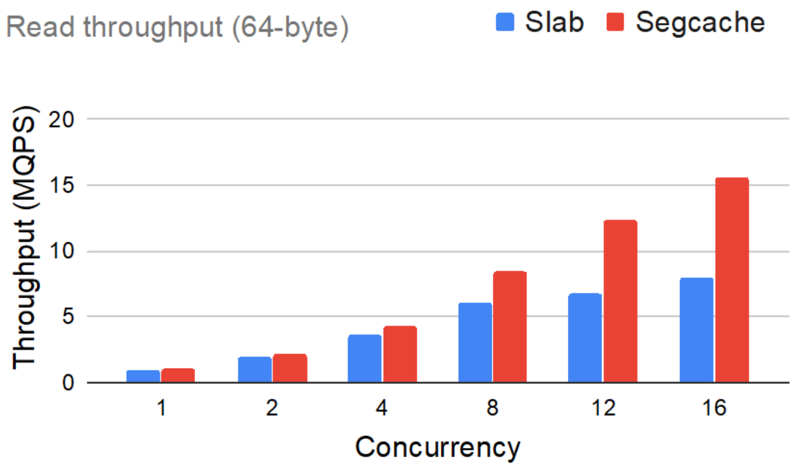
- Move shared segment header into DRAM

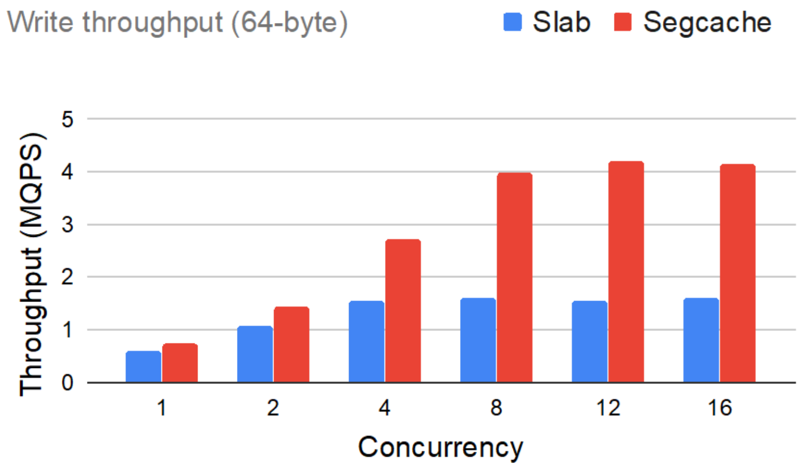


Segcache Microbenchmarks (AppDirect Mode)

Hardware Config (Twitter prod)

- 12 X 16GB DIMM
- **1 X 512GB AEP**
- 2-1-1 config
- 64-byte object





What's Next?

- Segcache
  - Performance on real workloads
  - Recovery performance
- Memory hierarchy
  - How to use PMEM
  - => How to use PMEM + DRAM

Lessons Learned

Takeaway for Caching on PMEM

- Avoid turning PMEM into new bottleneck
- AppDirect is a clear winner
  - But Memory Mode served its purpose along the way
- Due diligence pays off
- Innovate as needed
- Cache as a more durable service is an exciting but major undertaking

Takeaway for PMEM Adoption

- What’s the bottleneck for system at runtime?
- What are the business goals?
- What are the (dev, ops) constraints?
- Is there a path with incremental value gain?
- What are the possible exits?
- Transforming software takes time, too.

Q&A, and References

- [1] Pelikan: http://pelikan.io
- [2] A large scale analysis of hundreds of in-memory cache clusters at Twitter [OSDI'20]