NyxCache: Flexible and Efficient Multi-Tenant Persistent Memory Caching

Kan Wu, Kaiwei Tu, Yuvraj Patel, Rathijit Sen, Kwanghyun Park, Andrea Arpaci-Dusseau and Remzi Arpaci-Dusseau
In-Memory Key-Value Caches are Crucial
In-Memory Key-Value Value Caches are Crucial
In-Memory Key-Value Value Caches are Crucial
A Cache Server is Usually Multi-Tenant

Cache (Single Server)

Cache
Cache
Cache

Consolidated instances
A Cache Server is Usually Multi-Tenant

Consolidated instances
Contention -> regulation required
A Cache Server is Usually Multi-Tenant

Consolidated instances
Contention -> regulation required
Example sharing policies
• resource limit based on price tier,
• QoS
• proportional sharing, …
Persistent Memory for In-Memory KV Caches

Persistent Memory (PMEM)
- Intel Optane DC PMM (byte-addressable, memory bus, comparable performance to DRAM)
Persistent Memory for In-Memory KV Caches

Persistent Memory (PMEM)
- Intel Optane DC PMM (byte-addressable, memory bus, comparable performance to DRAM)

Appealing building blocks for in-mem KV caches
- Large capacity -> high hit rate
- Low cost per byte -> cheap, scale
- Energy-efficiency -> operational cost
- ...
Challenges: Multi-tenancy over PMEM
Challenges: Multi-tenancy over PMEM

Can we not regulate PM access?
Challenges: Multi-tenancy over PMEM

Read Throughput
(single thread 256B, GB/s)
Challenges: Multi-tenancy over PMEM

Lessons
- We must regulate PMEM access; small PMEM traffic can have a big effect

Read Throughput
(single thread 256B, GB/s)

Collocate 1GB/s 64B writes
Device Max BW: 14 GB/s

2.7x
Challenges: Multi-tenancy over PMEM

Lessons

- We must regulate PMEM access; small PMEM traffic can have a big effect
- We need new PMEM sharing mechanisms; existing DRAM/storage mechanisms can be ineffective due to PMEM’s unique characteristics
Challenges: Multi-tenancy over PMEM

Lessons

- **We must regulate PMEM access**: small PMEM traffic can have a big effect
- **We need new PMEM sharing mechanisms**: existing DRAM/storage mechanisms can be ineffective due to PMEM’s unique characteristics
  - **Example**: memory bandwidth limiting for “limiting impact to others”
  - **Setup**: Cache A and B (B limit: 1GB/s PMEM traffic)
Challenges: Multi-tenancy over PMEM

Lessons

- **We must regulate PMEM access**: small PMEM traffic can have a big effect
- **We need new PMEM sharing mechanisms**: existing DRAM/storage mechanisms can be ineffective due to PMEM’s unique characteristics
  - **Example**: memory bandwidth limiting for “limiting impact to others”
  - **Setup**: Cache A and B (B limit: 1GB/s PMEM traffic)
Challenges: Multi-tenancy over PMEM

Lessons
● We must regulate PMEM access; small PMEM traffic can have a big effect
● We need new PMEM sharing mechanisms; existing DRAM/storage mechanisms can be ineffective due to PMEM’s unique characteristics
  - Example: memory bandwidth limiting for “limiting impact to others”
  - Setup: Cache A and B (B limit: 1GB/s PMEM traffic)
  - Memory bandwidth limiting is ineffective due to PMEM 256B internal access granularity

![Graph showing Cache A, P99 Latency (256B read, us) and B (256B writes) achieving 1GB/s all time]
Goal: Design New PMEM Sharing Mechanisms
Goal: Design New PMEM Sharing Mechanisms

What mechanisms should we focus?
Many Sharing Goals ...

Resource Limiting
- $ \rightarrow \text{Resources}
Many Sharing Goals …

**Resource Limiting**
- $ \rightarrow $ Resources

**Quality of Service (QoS)**
- Latency-critical clients have latency guarantee
- Best-effort clients
Many Sharing Goals ...

Resource Limiting
- $ -> Resources

Quality of Service (QoS)
- Latency-critical clients have latency guarantee
- Best-effort clients

Proportional Sharing
- Weight -> Allocation
Many Sharing Goals …

.aws

Resource Limiting
• $ \rightarrow $ Resources

Quality of Service (QoS)
• Latency-critical clients have latency guarantee
• Best-effort clients

Fair Slowdown
• Equalize client slowdowns
• Slowdown: $\frac{p_{alone}}{p_{share}}$

Proportional Sharing
• Weight $\rightarrow$ Allocation
Focus: Basic Mechanisms

Resource Limiting
- $ \rightarrow $ Resources

Quality of Service (QoS)
- Latency-critical clients have latency guarantee
- Best-effort clients

Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1: Request Regulation</td>
<td>Control the rate a client can access PM</td>
</tr>
<tr>
<td>M2: Resource Usage Accounting</td>
<td>How much PMEM resource (not bandwidth) does a client use?</td>
</tr>
<tr>
<td>M3: Interference Analysis</td>
<td>Who interferes client A the most?</td>
</tr>
<tr>
<td>M4: Slowdown Estimation</td>
<td>How much has a client been slowed as a result of sharing?: $\frac{P_{alone}}{P_{share}}$</td>
</tr>
</tbody>
</table>

Fair Slowdown
- Equalize client slowdowns
- Slowdown: $\frac{P_{alone}}{P_{share}}$

Proportional Sharing
- Weight $\rightarrow$ Allocation
Focus: Basic Mechanisms

Resource Limiting
- $ \rightarrow $ Resources

Quality of Service (QoS)
- Latency-critical clients have latency guarantee
- Best-effort clients

Mechanisms

M1: Request Regulation
- Control the rate a client can access PM

M2: Resource Usage Accounting
- How much PMEM resource (not bandwidth) does a client use?

M3: Interference Analysis
- Who interferes client A the most?

M4: Slowdown Estimation
- How much has a client been slowed as a result of sharing?: $\frac{p_{alone}}{p_{share}}$

Fair Slowdown
- Equalize client slowdowns
- Slowdown: $\frac{p_{alone}}{p_{share}}$

Proportional Sharing
- Weight $\rightarrow$ Allocation
Focus: Basic Mechanisms

Resource Limiting
- $ \rightarrow $ Resources

Quality of Service (QoS)
- Latency-critical clients have latency guarantee
- Best-effort clients

Mechanisms

M1: Request Regulation
- Control the rate a client can access PM

M2: Resource Usage Accounting
- How much PMEM resource (not bandwidth) does a client use?

M3: Interference Analysis
- Who interferes client A the most?

M4: Slowdown Estimation
- How much has a client been slowed as a result of sharing?: $\frac{P_{alone}}{P_{share}}$

Fair Slowdown
- Equalize client slowdowns
- Slowdown: $\frac{P_{alone}}{P_{share}}$

Proportional Sharing
- Weight $\rightarrow$ Allocation
Focus: Basic Mechanisms

Resource Limiting
- $ \rightarrow $ Resources

Quality of Service (QoS)
- Latency-critical clients have latency guarantee
- Best-effort clients

aws

Mechanisms

M1: Request Regulation
- Control the rate a client can access PM

M2: Resource Usage Accounting
- How much PMEM resource (not bandwidth) does a client use?

M3: Interference Analysis
- Who interferes client A the most?

M4: Slowdown Estimation
- How much has a client been slowed as a result of sharing? $\frac{P_{\text{alone}}}{P_{\text{share}}}$

Fair Slowdown
- Equalize client slowdowns
- Slowdown: $\frac{P_{\text{alone}}}{P_{\text{share}}}$

Proportional Sharing
- Weight $\rightarrow$ Allocation
Focus: Basic Mechanisms

Resource Limiting
- $ \rightarrow $ Resources

Quality of Service (QoS)
- Latency-critical clients have latency guarantee
- Best-effort clients

M1: Request Regulation
- Control the rate a client can access PM

M2: Resource Usage Accounting
- How much PMEM resource (not bandwidth) does a client use?

M3: Interference Analysis
- Who interferes client A the most?

M4: Slowdown Estimation
- How much has a client been slowed as a result of sharing? $\frac{p_{\text{alone}}}{p_{\text{share}}}$

Fair Slowdown
- Equalize client slowdowns
- Slowdown: $\frac{p_{\text{alone}}}{p_{\text{share}}}$

Proportional Sharing
- Weight $\rightarrow$ Allocation

Mechanisms
Focus: Basic Mechanisms

Resource Limiting
- $ -> Resources

Quality of Service (QoS)
- Latency-critical clients have latency guarantee
- Best-effort clients

Mechanisms

M1: Request Regulation
- Control the rate a client can access PM

M2: Resource Usage Accounting
- How much PMEM resource (not bandwidth) does a client use?

M3: Interference Analysis
- Who interferes client A the most?

M4: Slowdown Estimation
- How much has a client been slowed as a result of sharing?: $\frac{p_{\text{alone}}}{p_{\text{share}}}$

Fair Slowdown
- Equalize client slowdowns
- Slowdown: $\frac{p_{\text{alone}}}{p_{\text{share}}}$

Proportional Sharing
- Weight -> Allocation
Contributions

Re-evaluate Key Mechanisms
● Analyze problems with existing mechanisms on PMEM

NyxCache: a flexible access regulation framework for any sharing goal
● Design new software mechanisms for PMEM sharing
● Revise four policy implementations based on new mechanisms
Contributions

Re-evaluate Key Mechanisms
● Analyze problems with existing mechanisms on PMEM

NyxCache: a flexible access regulation framework for any sharing goal
● Design new software mechanisms for PMEM sharing
● Revise four policy implementations based on new mechanisms

This talk:
Interference Analysis
QoS Policy
Interference Analysis

Use Case: Quality-of-Service policy
● **Latency-critical** clients (with tail latency guarantee) + Best-effort clients

Example Target:
P999 < 5ms

Latency-Critical Client

PMEM
Interference Analysis

**Use Case:** Quality-of-Service policy
- **Latency-critical** clients (with tail latency guarantee) + Best-effort clients

Example Target: P999 < 5ms
Interference Analysis

**Use Case:** Quality-of-Service policy

- **Latency-critical** clients (with tail latency guarantee) + Best-effort clients

![Diagram](image)

- Latency target violated

**Example Target:**

- P999 < 5ms
Interference Analysis

**Use Case:** Quality-of-Service policy

- **Latency-critical** clients (with tail latency guarantee) + Best-effort clients

Latency target violated

Example Target: P999 < 5ms

Throttle best-effort clients

![Diagram](image)
Interference Analysis

**Use Case:** Quality-of-Service policy
- **Latency-critical** clients (with tail latency guarantee) + Best-effort clients
- **Question:** Who should we throttle? **interference analysis** to find out the most interfering client -> quick rescue and high utilization

Latency target violated

Throttle best-effort clients

Example Target: P999 < 5ms
Problems: Interference Analysis on PMEM

**DRAM method**: use clients’ BW as indicator; higher BW -> more interference

**Problems**: PMEM Bandwidth is not a good indicator of interference
Problems: Interference Analysis on PMEM

**DRAM method:** use clients’ BW as indicator; higher BW -> more interference

**Problems:** PMEM Bandwidth is not a good indicator of interference

- Problem 1: write interference > read interference
Problems: Interference Analysis on PMEM

DRAM method: use clients’ BW as indicator; higher BW -> more interference

Problems: PMEM Bandwidth is not a good indicator of interference
- Problem 1: write interference > read interference

Victim P99 Latency
(256B read, us)

Co-located Traffic
(GB/s)
Problems: Interference Analysis on PMEM

**DRAM method:** use clients’ BW as indicator; higher BW -> more interference

**Problems:** PMEM Bandwidth is not a good indicator of interference
- Problem 1: write interference > read interference

---

**Victim P99 Latency**
(256B read, us)
Problems: Interference Analysis on PMEM

**DRAM method:** use clients’ BW as indicator; higher BW -> more interference

**Problems:** PMEM Bandwidth is not a good indicator of interference

- Problem 1: write interference > read interference

![Graph showing Victim P99 Latency and Co-located Traffic](image)
Problems: Interference Analysis on PMEM

**DRAM method:** use clients’ BW as indicator; higher BW -> more interference

**Problems:** PMEM Bandwidth is not a good indicator of interference

- Problem 1: write interference > read interference
- Problem 2: small accesses (<256B) interference > large access, with the same BW
e.g., 1GB/s 64B writes cause 2x the interference as 1GB/s 256B writes
Problems: Interference Analysis on PMEM

**DRAM method:** use clients’ BW as indicator; higher BW -> more interference

**Problems:** PMEM Bandwidth is not a good indicator of interference

- Problem 1: write interference > read interference
- Problem 2: small accesses (<256B) interference > large access, with the same BW
  
  e.g., 1GB/s 64B writes cause 2x the interference as 1GB/s 256B writes

*We need new high-fidelity interference analysis for PMEM sharing*
Solutions: NyxCache – Interference Analysis
Solutions: NyxCache – Interference Analysis

**Goal:** Answer who is interfering the most with a given client

- No special hardware – software solution
- Minimal device assumptions – treat devices as black box
Solutions: NyxCache – Interference Analysis

**Goal:** Answer who is interfering the most with a given client

**Solution:** runtime micro-, controlled-experiments
Solutions: NyxCache – Interference Analysis

**Goal:** Answer who is interfering the most with a given client

**Solution:** runtime micro-, controlled-experiments

- **Setup:** cache A, B, C; who is interfering A the most?
Solutions: NyxCache – Interference Analysis

**Goal:** Answer who is interfering the most with a given client

**Solution:** runtime micro-, controlled-experiments

- **Setup:** cache A, B, C; who is interfering A the most?

![Current State Diagram]

A Performance: L
Solutions: NyxCache – Interference Analysis

Goal: Answer who is interfering the most with a given client
Solution: runtime micro-, controlled-experiments
● Setup: cache A, B, C; who is interfering A the most?

Current State
A Performance: L

Exp 1: B - Δ
A Performance: L + Δ_{L1}
Solutions: NyxCache – Interference Analysis

**Goal:** Answer who is interfering the most with a given client

**Solution:** runtime micro-, controlled-experiments

- **Setup:** cache A, B, C; who is interfering A the most?

Exp 1: B - $\Delta$

A Performance : L + $\Delta_{L1}$

Exp 2: C - $\Delta$

A Performance : L + $\Delta_{L2}$

Current State

A Performance: L
Solutions: NyxCache – Interference Analysis

Goal: who is interfering the most with a given client -> who yields the largest $\Delta_L$

Solution: runtime micro-, controlled-experiments

- Setup: cache A, B, C; who is interfering A the most?

Exp 1: B - $\Delta$
A Performance : L + $\Delta_{L1}$

Current State
A Performance: L

Exp 2: C - $\Delta$
A Performance : L + $\Delta_{L2}$
Evaluation: NyxCache – QoS

What’s the benefit of NyxCache interference analysis mechanism?
● **Setup**: cache A, B, C
  - **Cache A**: latency-critical cache (fixed)
  - **Cache B**: read-dominant best-effort cache (fixed)
  - **Cache C**: write-dominant best-effort cache (dynamic)
NyxCache Ensures QoS and High Utilization

Latency-critical cache
P99 latency (us)

Best-effort (BE) cache
throughput (GB/s)

Target

Cache B (reads)
Cache C (writes)
NyxCache Ensures QoS and High Utilization

Best-effort cache C burst writes

Latency-critical cache target violation

Latency-critical cache
P99 latency (us)

Best-effort (BE) cache
throughput (GB/s)

Cache B (reads)
Cache C (writes)
NyxCache Ensures QoS and High Utilization

**DRAM solution:** throttle caches with higher bandwidth

---

**Latency-critical** cache  
P99 latency (us)

**Best-effort (BE)** cache  
throughput (GB/s)

---

0  15  30  45  
Time(s)

---

Target

---

Cache B (reads)

---

Cache C (writes)
NyxCache Ensures QoS and High Utilization

DRAM solution: throttle caches with higher bandwidth

Latency-critical cache
P99 latency (us)

Best-effort (BE) cache
throughput (GB/s)

Cache C’s interference caused Cache B throttled to 0
NyxCache Ensures QoS and High Utilization

**DRAM solution:** throttle caches with higher bandwidth

**NyxCache:** throttle caches causing larger interferences

---

**Latency-critical** cache  
P99 latency (us)

**Best-effort (BE) cache**  
throughput (GB/s)

---

**NyxCache throttles the right interference source**

---

- **Cache B (reads)**
- **Cache C (writes)**
NyxCache Ensures QoS and High Utilization

**Latency-critical** cache
P99 latency (us)

**Best-effort (BE)** cache
throughput (GB/s)

**DRAM solution:** throttle caches with higher **bandwidth**

**NyxCache:** throttle caches causing larger **interferences**

NyxCache throttles the right interference source

6x higher Cache B throughput

— Cache B (reads)
— Cache C (writes)
## NyxCache Summary

PMEM sharing necessitates evolving software/hardware stack. Our contributions:

- **Define** what are important sharing mechanisms (the substrate)
- **Analyze** problems with existing mechanisms on PMEM
- **NyxCache** – design **new** software PMEM sharing **mechanisms**
- **NyxCache** – **revise policy** implementations based on new mechanisms

### Future Directions

- **Hardware Redesigns and Hardware/Software Codeigns for PMEM Sharing**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6x system utilization</td>
<td>5x better perf. isolation</td>
<td>2x better fairness</td>
<td>Interference-aware idle resource donation</td>
</tr>
</tbody>
</table>

**Contact:** kanwu@cs.wisc.edu  
**Code:** cs.wisc.edu/~kanwu