ACEing the Bufferpool Management Paradigm for Modern Storage Devices

Tarikul Islam Papon         Manos Athanassoulis
Solid State Drives

- electronic device
- fast random access
- concurrent I/Os
- write latency > read latency
Concurrency

Parallelism at different levels (channel, chip, die, plane block, page)
Read/Write Asymmetry

Out-of-place updates cause invalidation

“Erase before write” approach

Garbage Collection

Larger erase granularity

All these results in higher amortized write cost
Symmetric cost for Read & Write

One I/O at a time

Read/Write Asymmetry ($\alpha$)

Concurrency ($k$)
“A Parametric I/O Model for Modern Storage Devices”

know Thy Device

exploit \( k_r \) and \( k_w \) (with care)

read concurrency

write concurrency

read and write differently

asymmetry (\( \alpha \)) controls performance

DaMoN 2021
Bufferpool is Tightly Connected to Storage
Traditional Bufferpool Manager

Page request comes

Disk

DB

Bufferpool

Main Memory

Free frame

Disk page

Dirty page
Traditional Bufferpool Manager

If page is not in BP, fetch from disk
Traditional Bufferpool Manager

Diagram showing the Traditional Bufferpool Manager with a database (DB) connected to a bufferpool. The bufferpool contains blocks, some of which are marked as disk pages and others as dirty pages.
If BP is full, one page is selected for eviction based on page replacement policy.
Tradional Bufferpool Manager

If the page is dirty, it is written back to disk
Traditional Bufferpool Manager

Requested page is fetched in its place
(exchanging one write for a read)
Traditional Bufferpool Manager

Is this Optimal?
With write asymmetry, it is **NOT** fair to exchange one write for one read.

**The Challenge**

- Do not address Asymmetry ($\alpha$)
- Do not consider Concurrency
  - ARC
  - NRU
  - FIFO
  - LRU
  - Clock
  - Sweep
  - Second
  - Chance
  - 2Q
The Challenge

- With write asymmetry, it is **NOT** fair to exchange one write for one read.

Consider Concurrency

Do not consider Concurrency

Do not address Asymmetry (α)

Address α via write-avoidance

- ARC
- NRU
- FIFO
- LRU
- Clock Sweep
- Second Chance
- 2Q

- CFLRU/C
- CFLRU/E
- DL-CFLRU/E
- LRU-WSR
- CCF-LRU
- CFLRU

Addressing α via write-avoidance

Concurrency
The Challenge

- With write asymmetry, it is **NOT** fair to exchange one write for one read.

- Do not expressly utilize the device concurrency.

With write asymmetry, it is NOT fair to exchange one write for one read.

Do not expressly utilize the device concurrency.
The Challenge

- With write asymmetry, it is **NOT** fair to exchange one write for one read.

- Do not expressly utilize the device concurrency.

```
§ With write asymmetry, it is **NOT** fair to exchange one write for one read.

§ Do not expressly utilize the device concurrency.
```
The Challenge

- device under-utilization
- poor end-to-end performance
- high deployment cost
The Solution

- **Device under-utilization**
- **Poor end-to-end performance**
- **High deployment cost**
- **Ease of integration**

**Do not consider Concurrency**

- ARC
- NRU
- FIFO
- LRU
- Clock Sweep
- Second Chance

**Addressing α via write-avoidance**

- 2Q
- CFLRU/C
- CFLRU/E
- DL-CFLRU/E
- LRU-WSR

**Addressing α via write-amortization**

- CCF-LRU
- CFLRU
- ACE

**Consider Concurrency**

- SSD Controller Optimization

- Exploiting k & Addressing α

- Exploiting k & Addressing α via write-avoidance
## Bufferpool Manager

### Eviction Policy

<table>
<thead>
<tr>
<th>Which page to evict/write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- LRU</td>
</tr>
<tr>
<td>- NRU</td>
</tr>
<tr>
<td>- Clock</td>
</tr>
<tr>
<td>- Second Chance</td>
</tr>
<tr>
<td>- CFLRU</td>
</tr>
<tr>
<td>- LRU-WSR</td>
</tr>
<tr>
<td>- CCF-LRU</td>
</tr>
</tbody>
</table>

### Read-ahead Policy

<table>
<thead>
<tr>
<th>When to prefetch?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Prefetch on miss</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Which pages?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Sequential</td>
</tr>
<tr>
<td>- History-based</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How many pages?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 1 or x pages</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How to prefetch?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Concurrently</td>
</tr>
</tbody>
</table>

Flash-friendly policies: CFLRU/C, CFLRU/E, D-L-CFLRU/E
Bufferpool Manager

Replacement Algorithm
- LRU
- NRU
- Clock
- Second Chance
- CFLRU
- LRU-WSR
- CCF-LRU
- DL-CFLRU/E

Flash-friendly policies

Eviction Policy
How many page(s) to evict?
- 1 page
- n pages

Which page(s) to evict?
- follow page replacement policy

Write-back Policy
How many pages to write?
- 1 page
- n pages (exploit $k_w$)

Which pages to write-back?
- dirty pages following replacement policy

When & how to write-back?
- background & concurrently

Write-back Policy

Read-ahead Policy
When to prefetch?
- Prefetch on miss

Which pages?
- Sequential
- History-based

How many pages?
- 1 or x pages

How to prefetch?
- Concurrently

Optional
Asymmetry/Concurrency-Aware (ACE) Bufferpool Manager
ACE Bufferpool Manager

Use device’s properties
**ACE Bufferpool Manager**

$$1 \leq n_e \leq \text{read concurrency (} k_r \text{)}$$

$$n_w = \text{device’s write concurrency (} k_w \text{)}$$

- **write** $n_w$ dirty pages concurrently
- **evict** $n_e$ pages
- **prefetch** $n_e - 1$ pages concurrently
Let’s assume: \( k_w = 3 \), LRU is the baseline replacement policy & red indicates dirty page

Write request of page 8 comes
An Example ($k_w = 3$)

write page 8

Candidate for eviction

Since candidate page is clean, we simply evict 9

After eviction:

Write request of page 1 comes
An Example \( (k_w = 3) \)

write page 1

LRU

Candidate

After eviction:

B 8 6 2 3 5 7 4

B 1
An Example ($k_w = 3$)

write page 1

<table>
<thead>
<tr>
<th>LRU</th>
<th>LRU+ACE (w/o PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>8 6 2 3 5 7 4</td>
</tr>
<tr>
<td>B</td>
<td>1 8 6 2 3 5 7</td>
</tr>
</tbody>
</table>

After eviction:
An Example \((k_w = 3)\)

write page 1

**LRU**

| B | 8 | 6 | 2 | 3 | 5 | 7 | 4 |

After eviction:

| B | 1 | 8 | 6 | 2 | 3 | 5 | 7 |

**LRU+ACE (w/o PF)**

Candidate

| 8 | 6 | 2 | 3 | 5 | 7 | 4 |

4,5,2 concurrently written
4 evicted
An Example \((k_w = 3)\)

**write page 1**

<table>
<thead>
<tr>
<th>LRU</th>
<th>LRU+ACE (w/o PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B 8 6 2 3 5 7 4</td>
<td>8 6 2 3 5 7</td>
</tr>
<tr>
<td>After eviction:</td>
<td>After eviction:</td>
</tr>
<tr>
<td>B 1 8 6 2 3 5 7</td>
<td>1 8 6 2 3 5 7</td>
</tr>
</tbody>
</table>
An Example ($k_w = 3$, $n_e = 2$)

write page 1

<table>
<thead>
<tr>
<th>LRU</th>
<th>LRU+ACE (w/o PF)</th>
<th>LRU+ACE (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>8 6 2 3 5 7 4</td>
<td>8 6 2 3 5 7 4</td>
</tr>
</tbody>
</table>

After eviction:

| B   | 1 8 6 2 3 5 7    | 1 8 6 2 3 5 7   | 1 8 6 2 3 5 7 |

Candidate

An Example ($k_w = 3$, $n_e = 2$)
An Example \((k_w = 3, n_e = 2)\)

write page 1

<table>
<thead>
<tr>
<th>LRU</th>
<th>LRU+ACE (w/o PF)</th>
<th>LRU+ACE (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B [8\ 6\ 2\ 3\ 5\ 7\ 4]</td>
<td>B [8\ 6\ 2\ 3\ 5\ 7\ 4]</td>
<td>B [8\ 6\ 2\ 3\ 5\ 7\ 4]</td>
</tr>
<tr>
<td>After eviction:</td>
<td>After eviction:</td>
<td>eviction window</td>
</tr>
<tr>
<td>B [1\ 8\ 6\ 2\ 3\ 5\ 7]</td>
<td>B [1\ 8\ 6\ 2\ 3\ 5\ 7]</td>
<td>4,5,2 concurrently written</td>
</tr>
</tbody>
</table>

4,7 evicted
An Example \((k_w = 3, \ n_e = 2)\)

write page 1

<table>
<thead>
<tr>
<th>LRU</th>
<th>LRU+ACE (w/o PF)</th>
<th>LRU+ACE (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B 8 6 2 3 5 7 4</td>
<td>8 6 2 3 5 7 4</td>
<td>8 6 2 3 5 4</td>
</tr>
<tr>
<td>After eviction:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 1 8 6 2 3 5 7</td>
<td>1 8 6 2 3 5 7</td>
<td>1 8 6 2 3 5 9</td>
</tr>
</tbody>
</table>

prefetched
# Experimental Evaluation

**PostgreSQL 11.5**

Clock Sweep

<table>
<thead>
<tr>
<th>Device</th>
<th>$\alpha$</th>
<th>$k_r$</th>
<th>$k_w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optane SSD</td>
<td>1.1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>PCIe SSD</td>
<td>2.8</td>
<td>80</td>
<td>8</td>
</tr>
<tr>
<td>SATA SSD</td>
<td>1.5</td>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td>Virtual SSD</td>
<td>2.0</td>
<td>11</td>
<td>19</td>
</tr>
</tbody>
</table>

**Workload:**

- synthesized traces
- TPC-C benchmark

LRU

CFLRU

LRU-WSR vs their ACE counterparts
ACE Improves Runtime

Device: PCIe SSD

$\alpha = 2.8$, $k_w = 8$

ACE improves runtime by 22-26%

Negligible increase in buffer miss (<0.009%)

Benefit comes at no cost

Mixed Skewed Trace

(r/w: 50/50, locality 90/10)
Higher Gain for Write-Heavy Workload

Write-intensive workloads have higher benefit (up to 32%)

Device: PCIe SSD

$\alpha = 2.8, k_w = 8$
Impact of R/W Ratio & Asymmetry

- more writes, more speedup
- higher asymmetry, higher speedup
- good benefit even for low asymmetry

**Legend**
- PCIe SSD
- Virtual SSD
- SATA SSD
- Optane SSD

**Equations**
- $\alpha = 2.8$
- $\alpha = 2.0$
- $\alpha = 1.5$
- $\alpha = 1.1$
Impact of #Concurrent I/Os

Device: PCIe SSD

\[ \alpha = 2.8, k_w = 8 \]

Highest speedup when optimal concurrency is used

Mixed Skewed Trace
(r/w: 50/50, locality 90/10)
Experimental Evaluation (TPC-C)

ACE Achieves 1.3x for mixed TPC-C
decoupled eviction and write-back mechanism can be integrated with any replacement policy
benefit comes with no penalty

Summary

**ACE Bufferpool**

<table>
<thead>
<tr>
<th>Reader</th>
<th>Evictor</th>
<th>Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq. Stream Prefetcher</td>
<td>History-based Prefetcher</td>
<td>Concurrent Device-aware Writing</td>
</tr>
</tbody>
</table>

Concurrently write back $n$ dirty pages

Parallely prefetch $n_e - 1$ pages

<table>
<thead>
<tr>
<th>Dirty page</th>
<th>Clean page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$p_9$</td>
</tr>
<tr>
<td>$p_5$</td>
<td>$p_{12}$</td>
</tr>
<tr>
<td>$p_{18}$</td>
<td>$p_{10}$</td>
</tr>
<tr>
<td>$p_{13}$</td>
<td>$p_7$</td>
</tr>
<tr>
<td>$p_{24}$</td>
<td>$p_{21}$</td>
</tr>
</tbody>
</table>

Modern Storage Devices

asymmetry concurrency
Thank You!

papon@bu.edu
## Read/Write Asymmetry - Example

<table>
<thead>
<tr>
<th>Device</th>
<th>Advertised Rand Read IOPS</th>
<th>Advertised Rand Write IOPS</th>
<th>Advertised Asymmetry</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe D5-P4320</td>
<td>427k</td>
<td>36k</td>
<td>11.9</td>
</tr>
<tr>
<td>PCIe DC-P4500</td>
<td>626k</td>
<td>51k</td>
<td>12.3</td>
</tr>
<tr>
<td>PCIe P4510</td>
<td>465k</td>
<td>145k</td>
<td>3.2</td>
</tr>
<tr>
<td>SATA D3-S4610</td>
<td>92k</td>
<td>28k</td>
<td>3.3</td>
</tr>
<tr>
<td>Optane P4800X</td>
<td>550k</td>
<td>500k</td>
<td>1.1</td>
</tr>
</tbody>
</table>
Measuring Asymmetry & Concurrency

**Device**
PCIe SSD - P4510 (1TB)

For 4K random read,
- **Asymmetry**: 2.8
- **Concurrency**: 80

**Graph**
- 4K Random Read
- 4K Random Write
- 8K Random Read
- 8K Random Write

- 2.8x
- 1.9x
Clock Sweep

CFLRU

LRU-WSR

After Eviction:

Case 1

Case 2

Since the cold flag is set, p6 is the candidate

After Eviction:
Algorithm 1: ACE

Input: $P$, $nw$, $ne$ is $pf$ _enabled_

1: // $P$ is the accessed page
2: // $nw$ is the maximum effective write concurrency ($nw = k_w$)
3: // $ne$ is the number of concurrent reads when prefetching is enabled
4: // is $pf$ _enabled_ determines if prefetching is enabled or not
5: if $P$ in bufferpool then
6:     return $P$
7: else
8:     // miss! need to bring $P$ from disk
9:     if bufferpool full then
10:        // reads $P$ and prefetches up to $ne - 1$ pages from disk
11:        // (depending on available slots)
12:        - prefetch_pages($P, ne - 1$)
13:     else
14:        - read $P$ from disk
15:     end if
16: else
17:     top_page = replacement_policy.get_one_page_to_evict()
18:     if top_page is clean then
19:        // follow classical approach if page is clean
20:        - drop top_page from bufferpool
21:     else
22:        // top_page is dirty, concurrently write $nw$ dirty pages
23:        // $P_{wb}$ is a vector containing the candidate dirty pages
24:        - $P_{wb} = \text{populate_pages_to_writeback}()$
25:        - issue $[[\text{length}(P_{wb})]]$ concurrent writes, $\forall p \in P_{wb}$
26:        - mark $[[\text{length}(P_{wb})]]$ pages as clean, $\forall p \in P_{wb}$
27:     if is $pf$ _enabled_ == true then
28:        // evict $ne$ pages
29:        // pages written and to be evicted can be different
30:        // $P_{ev}$ is a vector containing the pages to evict
31:        $P_{ev} = \text{replacement_policy.get_n_pages_to_evict}()$
32:        - drop $[[\text{length}(P_{ev})]]$ pages from bufferpool, $\forall p \in P_{ev}$
33:        // Now, prefetch
34:        - prefetch_pages($P, ne - 1$)
35:        - empty $P_{ev}$
36:     else
37:        // evict 1 page
38:        - drop top_page from bufferpool
39:        - read $P$ from disk
40:     end if
41:     - empty $P_{wb}$
42: end if
43: end if

1: Procedure populate_pages_to_writeback()
2: // follow the underlying page replacement policy to generate $P_{wb}$
3: - select next $nw$ dirty pages based on the underlying page replacement policy
4: - return this vector

1: Procedure prefetch_pages(page $P$, int $x$)
2: if $P$ in Sequential Table then
3:     // start of a sequential stream!
4:     // read $P$ and the next $x$ pages concurrently
5:     - prefetch Sequential ($P$)
6: else
7:     // use the history based prefetcher
8:     // read $P$ and $x$ pages (selected by prefetcher) concurrently
9:     - prefetch History ($P$)
10: end if
11: /* note that $P$ should be placed in the most recently used position in the bufferpool whereas other pages should be placed in the least recently used positions */
12: - place these $x + 1$ pages into bufferpool
Initial State (Before `write p7`)  

<table>
<thead>
<tr>
<th>mru</th>
<th>CFLRU</th>
<th>lru</th>
<th>ACE-CFLRU (w/o PF)</th>
<th>ACE-CFLRU (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>CD</td>
<td>D</td>
<td>CD</td>
<td>CD</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>p2</td>
<td>p3</td>
<td>p4</td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>p6</td>
<td>clean-first region</td>
<td>clean-first region</td>
</tr>
</tbody>
</table>

Intermediate State  

<table>
<thead>
<tr>
<th>CFLRU</th>
<th>ACE-CFLRU (w/o PF)</th>
<th>ACE-CFLRU (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>CD</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>p2</td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>p6</td>
</tr>
</tbody>
</table>

Final State (After `write p7`)  

<table>
<thead>
<tr>
<th>CFLRU</th>
<th>ACE-CFLRU (w/o PF)</th>
<th>ACE-CFLRU (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>p1</td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>p5</td>
</tr>
</tbody>
</table>

prefetched
### Initial State (Before `write p7´)

<table>
<thead>
<tr>
<th>mru</th>
<th>LRU-WSR</th>
<th>lru</th>
<th>ACE-LRUW (w/o PF)</th>
<th>ACE-LRUW (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Cold</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>p2</td>
<td>p3</td>
<td>p4</td>
</tr>
</tbody>
</table>

### Intermediate State

<table>
<thead>
<tr>
<th>LRU-WSR</th>
<th>ACE-LRUW (w/o PF)</th>
<th>ACE-LRUW (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Cold</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>p2</td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>p4</td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>p6</td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>p1</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>p4</td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>p6</td>
</tr>
</tbody>
</table>

### Final State (After `write p7´)

<table>
<thead>
<tr>
<th>LRU-WSR</th>
<th>ACE-LRUW (w/o PF)</th>
<th>ACE-LRUW (w/ PF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Cold</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>p1</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>p3</td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>p5</td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>p3</td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>p2</td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>p5</td>
</tr>
</tbody>
</table>

- **Cold**

- **Prefetched**

48
Experimental Evaluation

ACE performs well under memory pressure.
Impact on #writes

# Writes (in Millions)

- SOA-LW
- ACE-LW
- SOA-PW
- ACE-PW

<table>
<thead>
<tr>
<th></th>
<th>MS</th>
<th>RIS</th>
<th>WIS</th>
<th>MU</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA-LW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACE-LW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOA-PW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACE-PW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Writes (in Millions)