Clean First or Dirty First? A Cost-Aware Self-Adaptive Buffer Replacement Policy

Yi Ou and Theo Härder

University of Kaiserslautern, Germany
{ou, haeder}@cs.uni-kl.de
Outline

- Introduction
- Related works
- CASA scheme
- Simulation
- Conclusions
Introduction (1/2)

• Nowadays, NAND flash memory has been widely used as storage medium

• The I/O cost of read and write operations is significantly asymmetric in flash memory

• To improve the performance, some flash-aware algorithms had been proposed
Introduction (2/2)

• But these algorithms ignore the R/W cost ratio of the underlying device and the update intensity of the workload

• This paper proposed a Cost-Aware Self Adaptive buffer replacement policy
Related work

- CFLRU
- LRU-WSR
- CCF-LRU
CFLRU (1/2)

- It manages pages using the LRU list

- It divides the list into a working region and a clean-first region
  - The working region contains recently referenced pages
  - The clean-first region contains older pages
CFLRU (2/2)

- When miss
  - Choose the page which is a clean page in the clean-first region

![Diagram of CFLRU showing working and clean-first regions with page transitions and labels for clean (C) and dirty (D) pages. MRU and LRU windows marked.](image)
LRU-WSR (1/2)

- It divides pages into hot dirty pages, cold dirty pages, and clean pages

- Structure
LRU-WSR (2/2)

- **When hit**
  - Move to the MRU, if it’s a clean or hot dirty page
  - Set the cold flag to 0 and move to MRU, if it’s a cold dirty page

- **When miss**
  - Evict the page, if it’s a clean or cold dirty page in LRU position
  - Set the cold flag to 1 and move to MRU, if it’s a hot dirty page in LRU position
CCF-LRU (1/5)

- Structure

<table>
<thead>
<tr>
<th>Mixed LRU List</th>
<th>Dirty Hot</th>
<th>Clean Hot</th>
<th>Dirty Cold</th>
<th>Clean Hot</th>
<th>Dirty Cold</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cold Clean LRU List</th>
<th>Clean Cold</th>
<th>Clean Cold</th>
<th>Clean Cold</th>
<th>Clean Cold</th>
<th>Clean Cold</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mixed LRU List</th>
<th>Cold Clean LRU List</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>L2</td>
</tr>
<tr>
<td>L = L1 + L2</td>
<td></td>
</tr>
</tbody>
</table>
CCF-LRU (2/5)

- When hit in Cold Clean LRU List

```
P4  P3  P2  P1
```

Cold Clean LRU

```
P2  P9  P8  P7  P6  P5
```

Mixed LRU

MRU         LRU
CCF-LRU (3/5)

- When hit in Mixed LRU List

![Diagram of CCF-LRU with elements P4, P3, P2, P1, and P7, P8, P7, P6, P5 representing Cold Clean LRU and Mixed LRU.]
CCF-LRU (4/5)

- When miss (Cold Clean LRU is not empty)

![Diagram showing P10, P4, P3, P2, P1 in Cold Clean LRU and P9, P8, P7, P6, P5 in Mixed LRU]
CCF-LRU (5/5)

- When miss (Cold Clean LRU is empty)
CASA scheme (1/5)

- It manages the buffer using two dynamic lists
  - The clean list $C$
  - The dirty list $D$
- Evicting page from $C$ or $D$ is determined by check the parameter $\tau$
  - $\tau$ is the target size of $C$
CASA scheme (2/5)

- Structure

\[ b = |C| + |D| \]

- \( b \): the total buffer size
- \(|C|\): the size of \( C \)
- \(|D|\): the size of \( D \)
- \( \tau \): the target size of \( C \)
CASA scheme (3/5)

• When hit
  ▫ Hit from $C$
    • Op = Read
    • Op = Write
  ▫ Hit from $D$
    • Op = Read
    • Op = Write
The example (when hit)

\[ \tau = \min(\tau + c_R \times (|D| \div |C|), b) \]

\[ \tau = \max(\tau - c_W \times (|C| \div |D|), b) \]

CASA scheme (4/5)
CAS A scheme (5/5)

• When miss and no free space in the buffer cache
  ▫ $|C| > \tau$
    • Evict page from the clean list
  ▫ $|C| \leq \tau$
    • Evict page from the dirty list

• Put the new page into the buffer cache
  ▫ If $op = W$, put it into the MRU of $D$
  ▫ If $op = R$, put it into the MRU of $C$
Simulation (1/7)

• The simulation run the CONCAT trace
  ▫ Four phases
    • OLTP
    • DSS
    • OLTP
    • DSS
• Virtual execution time $T_v$
  ▫ $T_v = n_R \times c_R + n_W \times c_W$
Simulation (2/7)

- Statistics of the OLTP trace and DSS trace

<table>
<thead>
<tr>
<th>Attribute</th>
<th>OLTP</th>
<th>DSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of page requests</td>
<td>1,420,613</td>
<td>3,250,972</td>
</tr>
<tr>
<td>number of distinct pages</td>
<td>59,782</td>
<td>104,308</td>
</tr>
<tr>
<td>min page number</td>
<td>0</td>
<td>220,000</td>
</tr>
<tr>
<td>max page number</td>
<td>219,040</td>
<td>325,639</td>
</tr>
<tr>
<td>number of reads</td>
<td>1,156,795</td>
<td>3,250,972</td>
</tr>
<tr>
<td>number of updates</td>
<td>263,818</td>
<td>0</td>
</tr>
<tr>
<td>update percentage</td>
<td>18.57%</td>
<td>0</td>
</tr>
<tr>
<td>locality (number of the hottest pages vs. number of requests for them)</td>
<td>11,957 vs. 1,224,613 (20% vs. 86%)</td>
<td>20,862 vs. 2,875,664 (20% vs. 88%)</td>
</tr>
</tbody>
</table>
Simulation (3/7)

- Number of physical read and write

<table>
<thead>
<tr>
<th></th>
<th>1,000 pages</th>
<th></th>
<th>10,000 pages</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n_R$</td>
<td>$n_W$</td>
<td>$n_R$</td>
<td>$n_W$</td>
</tr>
<tr>
<td>CCFLRU</td>
<td>427,012</td>
<td>71,103</td>
<td>237,518</td>
<td>35,642</td>
</tr>
<tr>
<td>CFLRU-0.25</td>
<td>393,914</td>
<td>85,818</td>
<td>175,448</td>
<td>43,096</td>
</tr>
<tr>
<td>CFLRU-0.50</td>
<td>389,653</td>
<td>79,408</td>
<td>181,223</td>
<td>39,428</td>
</tr>
<tr>
<td>CFLRU-0.75</td>
<td>388,917</td>
<td>74,688</td>
<td>195,160</td>
<td>37,188</td>
</tr>
<tr>
<td>LRU</td>
<td>403,056</td>
<td>95,849</td>
<td>177,861</td>
<td>51,157</td>
</tr>
<tr>
<td>LRUWSR</td>
<td>409,469</td>
<td>90,183</td>
<td>186,306</td>
<td>46,076</td>
</tr>
<tr>
<td>CASA 1:1</td>
<td>389,350</td>
<td>77,884</td>
<td>175,985</td>
<td>44,975</td>
</tr>
<tr>
<td>CASA 1:4</td>
<td>398,249</td>
<td>72,190</td>
<td>180,558</td>
<td>39,947</td>
</tr>
<tr>
<td>CASA 1:16</td>
<td>417,715</td>
<td>71,359</td>
<td>192,149</td>
<td>37,427</td>
</tr>
<tr>
<td>CASA 1:64</td>
<td>425,993</td>
<td>71,138</td>
<td>211,015</td>
<td>36,089</td>
</tr>
<tr>
<td>CASA 1:128</td>
<td>426,932</td>
<td>71,116</td>
<td>221,344</td>
<td>35,666</td>
</tr>
</tbody>
</table>
Simulation (4/7)

- Virtual execution time relative to LRU

![Graphs showing virtual execution time for different cost ratios](image)

(a) $T_v$ for cost ratio 1:1
(b) $T_v$ for cost ratio 1:64
Simulation (5/7)

- The size of the clean list changes with the virtual time.
Simulation (6/7)

- Virtual execution time relative to LRU with different R/W cost ratio

![Graphs showing virtual execution time relative to LRU with different R/W cost ratio for different buffer sizes and cost ratios.](image)
Simulation (7/7)

- Real execution time relative to LRU on the HDD and the SSD
Conclusions

- CASA scheme outperforms previous proposals in a variety of cost settings and under changing workloads
- CASA can work efficiently without tuning any parameter