Recently-Evicted-First Buffer Replacement Policy for Flash Storage Devices

Dongyoung Seo
Samsung Electronics, Suwon, Korea

and

Dongkun Shin
Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea

Outline

• Introduction
• Related Works
• Proposed Scheme
• Simulation
• Conclusion
Introduction (1/3)

• The advantages of **NAND Flash memory**’s advantages over hard disk drive are **low-power consumption**, **small size**, and **high shock resistance**.

• Recent applications for flash memory are complex and diverse.

• **Flash memory**’s feature
  – **high read** performance
  – **low write** performance due to its “**erase-before-write**” constraint
Introduction (2/3)

• The **log buffer-based FTL** separated the physical flash memory into log blocks and data blocks.

• To enhance the write performance, most existing log-buffer-based FTL schemes aim to **reduce the number of block merge**.

• Most of current FTL techniques show poor performance for **random writes**, which incur frequent log block merges.
Introduction (3/3)

• The **buffer cache** can reduce the number of write requests sent to the flash memory.
  – issue: the buffer replacement policy of buffer cache
    • determines the flash memory write pattern.

• The proposed policy is called **REF (Recently-Evicted-First)**, which improves the performance of flash memory by 20% - 30% compared with the LRU policy for benchmarks.
Related Works (1/8)

• Log Buffer-Based FTL
  – two schemes
    • 1:1 log block mapping (BAST)
    • 1:N log block mapping (FAST)

• Flash-Aware Buffer Schemes
  – FAB
  – BPLRU
1:1 log block mapping (BAST) (2/8)

update requests on the pages “p0” and “p4” come ...

<table>
<thead>
<tr>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>p4</td>
<td>p8</td>
<td>p12</td>
<td>p16</td>
</tr>
<tr>
<td>p1</td>
<td>p5</td>
<td>p9</td>
<td>p13</td>
<td>p17</td>
</tr>
<tr>
<td>p2</td>
<td>p6</td>
<td>p10</td>
<td>p14</td>
<td>p18</td>
</tr>
<tr>
<td>p3</td>
<td>p7</td>
<td>p11</td>
<td>p15</td>
<td>p19</td>
</tr>
</tbody>
</table>

Invalid page

data block

log block

write sequence: “p8, p12, p1, p5, p9, p13”
write request “p8” comes ...

write sequence: “p8, p12, p1, p5, p9, p13”
write request “p12” comes ...

write sequence: “p8, p12, p1, p5, p9, p13”
1:1 log block mapping (BAST) (3/8)

write sequence: “p8, p12, p1, p5, p9, p13”

• Result
  1. every write request incur a block merge
  2. low space utilization

• Problem: log block thrashing
1:N log block mapping (FAST) (4/8)

write sequence: “p8, p12, p1, p5, p9, p13”

- **Result**
  1. no block merge
  2. high block associativity

- **Problem:** requires a large cost per block merge
Related Works (5/8)

• Flash-aware buffer techniques
  – FAB (Flash-Aware Buffer management)
    • the block which has the \textbf{largest number of pages} in the buffer cache is the victim block.
    • evicts all the pages of a block at a time, it reduces the block merge cost.
  – BPLRU (Block Padding Least Recently Used)
    • discuss later

<table>
<thead>
<tr>
<th>evicts all the pages of a block</th>
<th>do nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>log block</td>
<td>log block</td>
</tr>
<tr>
<td>B1’s page</td>
<td>B1’s page</td>
</tr>
<tr>
<td>B1’s page</td>
<td>B2’s page</td>
</tr>
<tr>
<td>B1’s page</td>
<td>B3’s page</td>
</tr>
<tr>
<td>B4’s page</td>
<td>B4’s page</td>
</tr>
<tr>
<td>Block associativity: 2</td>
<td>Block associativity: 4</td>
</tr>
</tbody>
</table>
Related Works (6/8)

There are still 4 block merge.

And, it also possibly evict the recent used pages, which may make the page eviction useless.
Related Works (7/8)

• Flash-aware buffer techniques
  – FAB (Flash-Aware Buffer management)
  – BPLRU (Block Padding Least Recently Used)

• Difference:
  – determines the victim block based on the block-level LRU value
  – invokes only switch merges because of block padding

when the write request “p2” comes ...

• Problem: waste buffer cache space
Related Works (8/8)

- Three kinds of block merges
  - **full merge**
    - requires many page copies and block erases
  - **switch merge** and **partial merge**
    - invoke low page migration costs

![Diagram showing three kinds of block merges: full merge, switch merge, and partial merge.](image)
Proposed Scheme (1/7)

• Log Buffer-Aware Buffer Management
  – REF Page Eviction
  – Victim Block Selection

• Block Padding REF
  – an extended version of REF
Proposed Scheme (2/7)

• REF Page Eviction
  – three main features
    • Block-level page eviction
      – evict only the pages of the victim blocks from buffer cache
    • Log buffer-aware victim selection
      – as far as possible to make the victim block are the same to the data blocks associated with the log blocks
    • Page-level recency consideration
      – victim pages are selected among the not-recently-used pages
Proposed Scheme (3/7)

- REF Page Eviction

<table>
<thead>
<tr>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>p4</td>
<td>p8</td>
<td>p12</td>
<td>p16</td>
</tr>
<tr>
<td>p1</td>
<td>p5</td>
<td>p9</td>
<td>p13</td>
<td>p17</td>
</tr>
<tr>
<td>p2</td>
<td>p6</td>
<td>p10</td>
<td>p14</td>
<td>p18</td>
</tr>
<tr>
<td>p3</td>
<td>p7</td>
<td>p11</td>
<td>p15</td>
<td>p19</td>
</tr>
</tbody>
</table>

LRU eviction: p0, p4, p9, p13, p1, p5 => 6 block merges
REF eviction: p9, p13, p0, p4, p1, p5 => 2 block merges

Fig. 3. LRU eviction vs. REF eviction in BAST scheme.
## Proposed Scheme (4/7)

<table>
<thead>
<tr>
<th>MRU</th>
<th>write buffer</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initial log block

B2

B3

L0's block associativity: 4

L1's block associativity: 4

After LRU eviction: B0, p4, p9, p13, p1, p5

L0's block associativity: 2

L1's block associativity: 2

After REF eviction: B9, p13, p0, p4, p1, p5

VB = {B2,B3}

VB = {B0,B1}

**Fig. 4.** LRU eviction vs. REF eviction in FAST scheme.
Proposed Scheme (5/7)

• Victim Block Selection
  – The number of VB should be smaller than the number of log blocks
  – REF selects the blocks to be included into VB using the **victim window (VW)** to prevent the recently-used pages from being evicted
    • from the experiments of benchmark, the proper value of VW is about 75%

---

**Fig. 5.** Victim block selection in REF.
### Proposed Scheme (6/7)

<table>
<thead>
<tr>
<th>I/O Request</th>
<th>LRU Cache</th>
<th>Log Block</th>
<th>FAB Cache</th>
<th>Log Block</th>
<th>BPLRU Cache</th>
<th>Log Block</th>
<th>REF Cache</th>
<th>Log Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write [p0] (B0)</td>
<td>I [p0]</td>
<td></td>
<td>I [p0]</td>
<td></td>
<td>I [p0]</td>
<td></td>
<td>I [p0]</td>
<td></td>
</tr>
<tr>
<td>Write [p5] (B1)</td>
<td>E [p0]</td>
<td>L.0 [p0]</td>
<td>E [p0]</td>
<td>L.0 [p0]</td>
<td>E [p0]</td>
<td>L.0 [p0]</td>
<td>E [p0]</td>
<td>L.0 [p0]</td>
</tr>
</tbody>
</table>

I : Insertion  E : Eviction  M : Block Merge

**Fig. 6.** Comparison among LRU, FAB, BPLRU and REF. (2 log blocks, 4 pages per a block, 3 page-sized buffer cache, VW = 100%, |VB| = 2)
Proposed Scheme (7/7)

- Block Padding REF

![Diagram of Proposed Scheme](image)

Flash Memory

<table>
<thead>
<tr>
<th></th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>p8</td>
<td>p16</td>
<td>p24</td>
<td>p32</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>p9</td>
<td>p17</td>
<td>p25</td>
<td>p33</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>p10</td>
<td>p18</td>
<td>p26</td>
<td>p34</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>p11</td>
<td>p19</td>
<td>p27</td>
<td>p35</td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>p12</td>
<td>p20</td>
<td>p28</td>
<td>p36</td>
<td></td>
</tr>
<tr>
<td>p5</td>
<td>p13</td>
<td>p21</td>
<td>p29</td>
<td>p37</td>
<td></td>
</tr>
<tr>
<td>p6</td>
<td>p14</td>
<td>p22</td>
<td>p30</td>
<td>p38</td>
<td></td>
</tr>
<tr>
<td>p7</td>
<td>p15</td>
<td>p23</td>
<td>p31</td>
<td>p39</td>
<td></td>
</tr>
</tbody>
</table>

Log Block

<table>
<thead>
<tr>
<th></th>
<th>L0</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p8</td>
<td>p16</td>
</tr>
<tr>
<td></td>
<td>p9</td>
<td>p17</td>
</tr>
<tr>
<td></td>
<td>p10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p14</td>
<td></td>
</tr>
</tbody>
</table>

Buffer Cache

- (Total 10 pages)

- VB = \{B1, B2\}

- page within VW
- page outside VW

Fig. 7. Block padding in BP-REF.
Simulations (1/5)

• Three benchmark programs:
  – Internet Explorer
  – MS-Office install application
  – JPEG file copy application

<table>
<thead>
<tr>
<th>Structural parameter</th>
<th>Page size</th>
<th>Block size</th>
<th>Log buffer</th>
<th>Buffer cache</th>
<th>2 KB</th>
<th>128 KB (64 pages)</th>
<th>8 blocks</th>
<th>16 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing Parameters (flash memory)</td>
<td>Page read</td>
<td>Page write</td>
<td>Block erase</td>
<td></td>
<td>0.01 ms</td>
<td>0.2 ms</td>
<td>2 ms</td>
<td></td>
</tr>
</tbody>
</table>
Simulations (2/5)

Fig. 8. Comparisons of normalized execution times.
Simulations (3/5)

Fig. 9. Comparisons of merge count (Internet Explorer).
Simulations (4/5)

Fig. 10. Comparisons of the number of NAND operations (Internet Explorer).
Fig. 11. Normalized execution times under BP-REF varying the block padding threshold (Internet Explorer).
Conclusions

• The proposed REF enforces the buffer cache to evict only the pages of the victim block to reduce the block thrashing and the block associativity.

• The proposed BP-REF does the block padding selectively to reduce the block padding overhead.

• The REF scheme can be further improved by dynamic adaptation technique.