CFDC – A Flash-aware Replacement Policy for Database Buffer Management

Yi Ou
University of Kaiserslautern
Germany

Theo Härder
University of Kaiserslautern
Germany

Peiquan Jin
University of Science and Technology of China
P.R. China
Outline

• Introduction
• Related Works
• Proposed Scheme
• Experiments
• Conclusion
Introduction

• traditionally, the goal of buffer replacement policy is the minimization of the buffer fault ratio

• As flash is involved, several criteria need to be considered
  – flash’s read-write asymmetry
  – spatial locality
  – sequentiality of access patterns

• **CFDC**: Clean First, Dirty Clustered
Related Works (1/6)

• The buffer manager decides \textbf{when} and \textbf{how} to write
  – reduce the number of writes
    • CFLRU, LRU-WSR
  – read/write entire flash block
    • FAB, BPLRU
  – make use of spatial locality
    • REF
Related Works (2/6)

- **CFLRU (Clean-First LRU)**
  - evict the clean page in the window first

---

CFLRU List

<table>
<thead>
<tr>
<th>CFLRU List</th>
<th>Working region</th>
<th>Clean-first region</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRU</td>
<td>P1 (C) → P2 (D) → P3 (C)</td>
<td>P4 (D) → P5 (C) → P6 (D)</td>
</tr>
</tbody>
</table>

- **C**: Clean page
- **D**: Dirty page

Window, \( w \)

victim
Related Works (3/6)

- **LRU-WSR (Write Sequence Reordering)**
  - assign each page with a bit flag called “cold-flag”
  - dirty pages that are marked as cold will be treated like clean pages

- **Hit**
  - move to MRU, if clean or hot dirty
  - set $cf = 0$ and then move to MRU, if cold dirty

- **Miss**
  - evict LRU page, if clean or cold dirty
  - set $cf = 1$ and then move to MRU, if hot dirty
Related Work (4/6)

- **FAB**
  - the block which has the **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.

![Diagram of buffer cache and eviction order]

- eviction order: B4, B3, B1, B0

When garbage collection occurs, B4 needs 3 page copy, B4’ only needs 1 page copy
Related Works (5/6)

• BPLRU (Block Padding Least Recently Used)

1. Flush a block-level buffer that has two sectors

2. Read sectors 13, 14 for page padding

3. Write four sectors sequentially

4. Replace old data block with sequentially written log block (Switch Merge)
• **REF (Recently-Evicted-First)**
  
  – Victim Block Selection

  • REF selects the blocks to be included into VB using the **victim window (VW)** to prevent the recently-used pages from being evicted.
Proposed Scheme (1/7)

- CFLRU
  - Search cost
  - Utilization of dirty pages
  - Hot pages are pushed away by sequentially accessed pages

Figure 1: CFLRU replacement policy
• The two-region scheme
  – Working region
    • Keep hot pages that are frequently accessed
  – Priority region
    • Optimizing replacement costs by assigning varying priorities to pages
    • two queues, one for clean pages and one for dirty pages
      – Search costs for clean pages are eliminated
    • priority window
      – Determines the size ratio of priority region to the total buffer

![Diagram of two-region scheme](image)

Figure 2: Generalized two-region scheme
Proposed Scheme (3/7)

• Page clustering
  – Maintains a priority queue of page clusters
    • cluster: a list of pages located in proximity

• example
  – Update 6, 3, 2, 5, 4, 1
  – Two pages per block
  – One log block

<table>
<thead>
<tr>
<th>Req:</th>
<th>data block</th>
<th>log block without clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 5 3</td>
<td>4 Write: 14</td>
</tr>
<tr>
<td></td>
<td>2 6 4</td>
<td>1 Erase: 6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Req: (6,5), (3,4), (2,1)</th>
<th>data block</th>
<th>log block</th>
<th>page clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 3 5</td>
<td></td>
<td>Write: 0</td>
</tr>
<tr>
<td></td>
<td>2 4 6</td>
<td></td>
<td>Erase: 0</td>
</tr>
</tbody>
</table>
Proposed Scheme (4/7)

- example (Cont.)
  - Update 6, 3, 2, 5, 4, 1
  - Two pages per block
  - One log block

<table>
<thead>
<tr>
<th>Req:</th>
<th>data block</th>
<th>log block without clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 5 3</td>
<td>4 14</td>
</tr>
<tr>
<td></td>
<td>2 6 4</td>
<td>1 6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Req:</th>
<th>data block</th>
<th>log block</th>
<th>page clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 3 5</td>
<td>2 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 4 6</td>
<td>1 3</td>
<td></td>
</tr>
</tbody>
</table>

switch merge
Proposed Scheme (5/7)

• priority function
  – For a cluster \( c \) with \( n \) pages, its priority \( P(c) \) is

\[
P(c) = \frac{\sum_{i=1}^{n-1} |p_i - p_{i-1}|}{n^2 \times (\text{globaltime} - \text{timestamp}(c))}
\]

• \( p_0, \ldots, p_{n-1} \) are the page numbers ordered by their time of entering the cluster

  – tends to assign large cluster a lower priority

• flash disks are efficient in writing such clustered pages due to their spatial locality

• the pages in a large cluster have a higher probability to suffer from sequential accesses
• priority function (Cont.)

– For a cluster $c$ with $n$ pages, its priority $P(c)$ is

$$P(c) = \frac{\sum_{i=1}^{n-1} |p_i - p_{i-1}|}{n^2 \times (\text{globaltime} - \text{timestamp}(c))}$$  \hspace{1cm} (1)

– to distinguish between randomly accessed clusters and sequentially accessed clusters

• example

  – a cluster with pages \{0, 1, 2, 3\} has a dividend of 3
  – a cluster with pages \{7, 5, 4, 6\} has a dividend of 5

– to prevent randomly, but rarely accessed small clusters from staying in the buffer forever

• $\text{timestamp}(c)$ is the value of global time at the time of its creation
• each time a dirty page is evicted from the working region, global time is incremented by 1
Proposed Scheme (7/7)

• priority function (Cont.)
  – example
    • global time = 10
    • from left to right, the cluster priorities are
      – \(\frac{(7+5)}{3^2}(10-4) = \frac{2}{9}\)
      – \(\frac{1}{12}(10-2) = \frac{1}{8}\)
      – \(\frac{2}{22}(10-3) = \frac{1}{14}\)
      – \(\frac{(1+1)}{32}(10-6) = \frac{1}{18}\) - min

\[
P(c) = \frac{\sum_{i=1}^{n-1} |p_i - p_{i-1}|}{n^2 \times (\text{globaltime} - \text{timestamp}(c))}
\]  

Figure 3: Prioritized clusters
Experiments (1/4)

• Database engine
  – XTC (XML Transaction Coordinator)

• test machine
  – AMD Athlon Dual Core processor with 512MB RAM
  – Ubuntu Linux with kernel version 2.6.24-19
  – equipped with a magnetic disk and a flash disk
  – test data resides on the 32GB flash disk

• Workload
  – insert one million equal-length records into a B* tree
  – 50% read & 50% write
Experiments (2/4)

- **Fig4**
  - CFDC vs. CFLRU
    - 14% to 41%
  - CFLRU vs. LRU
    - 6%

- **Fig5**
  - write count close to CFLRU
Experiments (3/4)

• Fig 6
  – clustered writes are efficient

• Fig 7
  – When high write intensity, CFDC performs best, CFLRU degenerates to LRU
Fig 8
- performance of CFDC declines with an increasing sequentiality of workload

Figure 8: Influence of increasing scan fractions

Fig 9
- Both policy benefit from a larger priority window
- Future Work: dynamic window size adjustment

Figure 9: Impact of priority window size
Conclusion

• for a flash disk,
  – the number of writes should be minimized, and
  – spatial locality of access pattern should be exploited

• Future work
  – use other algorithms for the working region
  – evaluate further write techniques such as page padding