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

Yi Ou$^1$  Theo Härder$^1$  Peiquan Jin$^2$  Karsten Schmidt$^1$

$^1$University of Kaiserslautern
Germany

$^2$University of Sci. & Techn. of China
China

The Fifth International Workshop on Data Management on New Hardware
Outline

Flash Disk
Outline

Flash Disk

Existing Approaches
Outline

Flash Disk

Existing Approaches

The CFDC Algorithm
Outline

Flash Disk

Existing Approaches

The CFDC Algorithm

Experiments
Outline

Flash Disk

Existing Approaches

The CFDC Algorithm

Experiments

Conclusion & Outlook
Flash Memory

<table>
<thead>
<tr>
<th>Media</th>
<th>Read (512 B)</th>
<th>Write (512 B)</th>
<th>Erase (16 KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>2.56 µs</td>
<td>2.56 µs</td>
<td></td>
</tr>
<tr>
<td>Flash Memory</td>
<td>35.9 µs</td>
<td>226 µs</td>
<td>2 ms</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>12.4 ms</td>
<td>12.4 ms</td>
<td></td>
</tr>
</tbody>
</table>

- asymmetric read/write latencies
- update-in-place problematic
- erase at the block level
Flash-Disk Performance

- asymmetric latencies visible to applications
- more efficient at larger transfer units
- e.g., 0.5 MB/s at 4KB vs. 12.8 MB/s at 128 KB
Flash-Disk Anatomy

- logical-to-physical
- block level vs. page level
- FTL partly addresses the write/erase problem
- maintains a pool of log blocks, mapped at the page level
- performance sensitive to spatial locality of update requests
Flash-aware Buffer Existing Approaches

the buffer manager decides *when* and *how* to write
the buffer manager decides *when* and *how* to write

**Basic Idea 1**
reduce the number of writes (CFLRU, LRU-WSR)
Flash-aware Buffer

Existing Approaches

the buffer manager decides \textit{when} and \textit{how} to write

\textbf{Basic Idea 1}
reduce the number of writes (CFLRU, LRU-WSR)

\textbf{Basic Idea 2}
read/write entire flash blocks (FAB, BPLRU)
Flash-aware Buffer Existing Approaches

the buffer manager decides **when** and **how** to write

**Basic Idea 1**
reduce the number of writes (CFLRU, LRU-WSR)

**Basic Idea 2**
read/write entire flash blocks (FAB, BPLRU)

**Basic Idea 3**
make use of spatial locality (REF, DULO)
Flash-aware Buffer Existing Approaches

the buffer manager decides when and how to write

**Basic Idea 1**
reduce the number of writes (CFLRU, LRU-WSR)

**Basic Idea 2**
read/write entire flash blocks (FAB, BPLRU)

**Basic Idea 3**
make use of spatial locality (REF, DULO)

- all based on LRU
- CFDC: clean-first, dirty-clustered
CFLRU

- search cost
- utilization of dirty pages
CFDC

- generalized two-region scheme, independent of LRU
- parameter *priority window*: size of the priority region
- separation of clean and dirty pages
- dirty pages are grouped in clusters
- clusters are ordered by priority
Priority Function

Definition
For a cluster $c$ with $n$ pages, its priority $P(c)$ is computed according to Formula 1:

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

(1)

where $p_0, ..., p_{n-1}$ are the page numbers ordered by their time of entering the cluster.

Ideas behind the formular

- larger clusters
- sequential clusters
- small but rarely accessed clusters
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6

\[
\begin{array}{ccc}
1 & 3 & 5 \\
2 & 4 & 6 \\
\end{array}
\]

\(w=0\)
\(e=0\)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6

\[
\begin{array}{cccc}
1 & 3 & 5 & 6 \\
2 & 4 & 6 & \\
\end{array}
\]

\[w=1, \quad e=0\]
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>5</th>
<th></th>
<th></th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

w=1, e=0
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

w=4
e=0
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6

1, 4, 5, 2

1
2
3
4
5
6

w=0
e=3
Clustered Write

Example
- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6

```
1, 4, 5
1  2  3  4  5  6
1  2  3  4  5  6
w=1
e=0
```
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6

![Diagram showing clustered write example with numbers and boxes representing pages and updates.](attachment:diagram.png)
Clustered Write

Example
- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6

| 1, 4 |
| 1 | 5 | 3 |
| 2 | 6 | 4 |

w=0

E=3
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random

1, 4, 5, 2, 3, 6
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random

1, 4, 5, 2, 3, 6
total w=14, e=6
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total $w=14$, $e=6$

Clustered
(1,2), (4,3), (5,6)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>5</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

w=1  
e=0
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total w=14, e=6

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total w=14, e=6

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total w=14, e=6

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total w=14, e=6

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total w=14, e=6

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total \( w=14 \), \( e=6 \)

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total $w=14$, $e=6$

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total w=14, e=6

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total \( w=14, \ e=6 \)

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total w=14, e=6

Clustered
(1,2), (4,3), (5,6)
Clustered Write

Example

- update 1, 4, 5, 2, 3, 6
- two pages per block
- one log block

Random
1, 4, 5, 2, 3, 6
total w=14, e=6

Clustered
(1,2), (4,3), (5,6)
total w=8, e=3
Database Engine & Workload

database engine

- XTC (XML Transaction Coordinator), five-layer reference architecture
- three layers used: file manager, buffer manager, index manager

workload

- one million equal-length records stored in a $B^*$ tree
- random access with 80-20 locality
- 50% read & 50% write
Result 1

- CFDC vs. CFLRU: 41%
- CFLRU vs. LRU: max 6%
Result 1

- CFDC vs. CFLRU: 41%
- CFLRU vs. LRU: max 6%

Clustered writes are efficient
Result 1

- **CFDC vs. CFLRU**: 41%
- **CFLRU vs. LRU**: max 6%
  
  write count close to CFLRU
Result 2

- for read-only workload, all winners
- for update-only workload, CFLRU \(\equiv\) LRU
Result 3

- clean-first *always* a good idea?
- online algorithm to adjust the priority window dynamically
Conclusion

Principle 1
reduce number of writes

Principle 2
make use of spatial locality

Principle 3
hit ratio is still important
Outlook

- use other algorithms for the working region
- evaluate further write techniques such as page padding
- traces from real database applications
- raw disk access to eliminate file system and operating system influences
Acknowledgements

Thanks to...
Questions and Suggestions?