FRD: A Filtering based Buffer Cache Algorithm that Considers both Frequency and Reuse Distance

Sejin Park* and Chanik Park**

2017. 05. 18

*SK telecom Corporate R&D Center | Network IT Convergence R&D Center, New Computing Lab

**POSTECH Department of Computer Science and Engineering, System Software Lab
Table of Contents

• Motivation
• Workload Analysis and Observations
• Design
• Evaluation
• Summary
Motivation

- Buffer cache management algorithm is one of the oldest topic in computer science area
- Existing buffer cache algorithm concentrates on how to maintain meaningful blocks?
  - LRU, LFU, OPT, ...
  - LIRS (ACM SIGMETRICS 2002, S. Jiang. et. al.)
    - Two LRU Stacks (LIRS, HIRS)
    - Reuse distance ordering
  - ARC (USENIX FAST 03, Megiddo. et. al.)
    - Two LRU Stacks (Recency-T1, Frequency-T2)
    - Adaptive resizing
- In this study, we concentrate on how to exclude the cache-unfriendly blocks
  - We analyzed real-world workload and found characteristics of cache-unfriendly blocks
Example: LRU

- Depending on their eviction policy, blocks that can make cache pollution could be maintained in cache space.
- LRU believes that recently used blocks will make more cache hit.
  - If the recently used blocks are infrequently accessed and rarely used, it causes cache pollution!
Example: ARC

- Recency buffer T1 and Frequency buffer T2 in ARC works as LRU cache
- If a block is reused, it moves into T2 even if it is infrequently accessed block
  - This can cause cache pollution for T2
**Workload Description**

- Real-world workloads downloaded from SNIA.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLTP</td>
<td>Application</td>
<td>Online transaction processing</td>
</tr>
<tr>
<td>Web12</td>
<td>Web server</td>
<td>A typical retail shop</td>
</tr>
<tr>
<td>Web07</td>
<td>Web server</td>
<td>A typical retail shop</td>
</tr>
<tr>
<td>prxy_0</td>
<td>Data center</td>
<td>Firewall/web proxy</td>
</tr>
<tr>
<td>wdev_0</td>
<td>Data center</td>
<td>Test web server</td>
</tr>
<tr>
<td>hm_0</td>
<td>Data center</td>
<td>Hardware monitoring</td>
</tr>
<tr>
<td>proj_0</td>
<td>Data center</td>
<td>Project directories</td>
</tr>
<tr>
<td>proj_3</td>
<td>Data center</td>
<td>Project directories</td>
</tr>
<tr>
<td>src1_2</td>
<td>Data center</td>
<td>Source control</td>
</tr>
</tbody>
</table>
Workload Analysis

- Reuse Distance Distribution
  - Reuse Distance: # of unique blocks between the same blocks request
Workload Analysis

- CDF of Number of accessed count for each block

(a) OLTP  (b) Web12  (c) Web07  (d) prxy_0  (e) wdev_0

(f) hm_0  (g) proj_0  (h) proj_3  (i) src1_2

X axis: Number of accessed count for each block

CDF (Percentage)
Workload Analysis

- Observation #1: Most blocks (about 50 – 90%) are infrequently accessed in the real-world workload.

![CDF plots for different workloads](image-url)
Workload Analysis

• CDF of reuse distance distribution for the infrequently accessed blocks (represented by percentage of cache size)

(a) OLTP  (b) Web12  (c) Web07  (d) prxy_0  (e) wdev_0

(f) hm_0  (g) proj_0  (h) proj_3  (i) src1_2
Workload Analysis

- Observation #2: Reuse distance for the infrequently accessed blocks is extremely long or extremely short
  - In terms of cache size: under 10% and over 100% of cache size are dominant

(a) OLTP  (b) Web12  (c) Web07  (d) prxy_0  (e) wdev_0

(f) hm_0  (g) proj_0  (h) proj_3  (i) src1_2

CDF (Percentage)

X axis: Reuse distance (represented with percentage of given cache size)
Observations

• Observation #1: Most blocks are infrequently accessed in the real-world workload  
  – These blocks are cache-unfriendly blocks that cause cache pollution

• Observation #2: Reuse distance for the infrequently accessed blocks is extremely long or extremely short  
  – The cache-unfriendly blocks have distinct characteristics

• Therefore,  
  – “Frequency” and “Reuse distance” are the key metrics to filter out the cache-unfriendly blocks
Design

- **Block Classification**

<table>
<thead>
<tr>
<th>Class</th>
<th>Accessing Frequency</th>
<th>Reuse Distance</th>
<th>Cache-Hit Target</th>
<th>Cache Pollution (Filtering target)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1 (FS)</td>
<td>Frequent</td>
<td>Short</td>
<td>V</td>
<td>-</td>
</tr>
<tr>
<td>Class 2 (FL)</td>
<td>Frequent</td>
<td>Long</td>
<td>V</td>
<td>-</td>
</tr>
<tr>
<td>Class 3 (IS)</td>
<td>Infrequent</td>
<td>Short</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>Class 4 (IL)</td>
<td>Infrequent</td>
<td>Long</td>
<td>-</td>
<td>V</td>
</tr>
</tbody>
</table>

- **Design Goal**
  - Maintains Class 1 and 2 blocks in cache
  - Maintains Class 3 blocks but preventing it from polluting cache
  - Filters out Class 4 blocks from cache
**FRD Algorithm**
- A Filtering based Buffer Cache Algorithm that Considers both Frequency and Reuse Distance

Parameter = FilterStack (%)  
(Default = 10%)

1. New Entry insertion
2. Resident Block Insertion
3. History Block Insertion
4. Cache Hit
5. Cache Miss
6. Cache Hit

* If RD stack is not full  
  New entry is inserted to RD stack.

**Filter Stack**

**Reuse distance Stack**

- Resident Block
- History Block
Analysis of FRD Algorithm

* If RD stack is not full, new entry is inserted to RD stack.

Parameter = FilterStack (%) (Default = 10%)
Evaluation

- Environment
  - Simulation based evaluation
  - Compared with OPT, LRU, ARC, LIRS
Hitratio Result

- Case of LIRS’ unstable hitratio result
Hitratio Result

<Legend>
- FRD is highest
- LIRS is highest
- LIRS is unstable
- ARC is highest
- ARC is unstable

![Graphs showing hitratio results for different cache sizes and algorithms.](image-url)
**Hitratio Result**

- Case of ARC’s unstable hitratio result

<Legend>
- FRD is highest
- LIRS is highest
- LIRS is unstable
- ARC is highest
- ARC is unstable
### Evaluation

- **Overall Average Result (1.0 is OPT’s hitratio)**

<table>
<thead>
<tr>
<th>Workload</th>
<th>LRU</th>
<th>ARC</th>
<th>LIRS</th>
<th>FRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLTP</td>
<td>0.674</td>
<td>0.746</td>
<td>0.691</td>
<td>0.753</td>
</tr>
<tr>
<td>Web12</td>
<td>0.829</td>
<td>0.852</td>
<td>0.827</td>
<td>0.857</td>
</tr>
<tr>
<td>Web07</td>
<td>0.800</td>
<td>0.839</td>
<td>0.812</td>
<td>0.847</td>
</tr>
<tr>
<td>prxy_0</td>
<td>0.844</td>
<td>0.870</td>
<td>0.870</td>
<td>0.898</td>
</tr>
<tr>
<td>wdev_0</td>
<td>0.647</td>
<td>0.723</td>
<td>0.728</td>
<td>0.745</td>
</tr>
<tr>
<td>hm_0</td>
<td>0.598</td>
<td>0.700</td>
<td>0.723</td>
<td>0.724</td>
</tr>
<tr>
<td>proj_0</td>
<td>0.612</td>
<td>0.722</td>
<td>0.740</td>
<td>0.780</td>
</tr>
<tr>
<td>proj_3</td>
<td>0.172</td>
<td>0.241</td>
<td>0.516</td>
<td>0.478</td>
</tr>
<tr>
<td>src1_2</td>
<td>0.620</td>
<td>0.697</td>
<td>0.799</td>
<td>0.813</td>
</tr>
</tbody>
</table>
Parameter Sensitivity (Size of the Filter stack)

- Variation of filter stack size from 1% to 25% of cache size.
- 10% shows the best performance on average but the difference is negligible.
Summary

• FRD: A Filtering based Buffer Cache Algorithm that Considers both Frequency and Reuse Distance

  – A new buffer cache algorithm that filters out cache-unfriendly blocks

  – Careful analysis on real-world workload gives characteristics of cache-unfriendly blocks

  – The experimental result shows that it outperforms state-of-the-art cache algorithms like ARC or LIRS.
Backup slides
**Hitratio Analysis**

- Filter stack performance
Revisiting LIRS and ARC

**ARC** (Initial: $T1= T2= B1 = B2 = 0, p = 0$)

$T1 + T2 + B1 + B2 <= 2C$

New Entry

Replace($p$)

<table>
<thead>
<tr>
<th>$T1$</th>
<th>$B1$</th>
</tr>
</thead>
</table>

$T2$

| $B2$ |

$p = \min\{c, p + \max\{|B2|/|B1|,1\}\}$

Replace($p$)

$p = \max\{0, p - \max\{|B1|/|B2|,1\}\}$

Replace($p$)

Subroutine **Replace($p$)**

If $(|T1| \geq 1)$ and $(x \in B2$ and $|T1| = p)$ or $(|T1| > p)$ then move the LRU page of $T1$ to the top of $B1$ and remove it from the cache.

Else move the LRU page in $T2$ to the top of $B2$ and remove it from the cache.

**LIRS** ($HIRstack + LIRstack = c, 1:99$)

New Entry

| $HIRstack$ |

| $LIRStack$ |

$\text{Metadata (Non-resident)}$

Keep Non-Resident till $\text{RMAX}$

$\text{Resident}$

$\text{Non-Resident}$

Exploiting $\text{LIRS}$ and $\text{ARC}$

$p = \min\{c, p + \max\{|B2|/|B1|,1\}\}$

Replace($p$)

$p = \max\{0, p - \max\{|B1|/|B2|,1\}\}$

Replace($p$)
## Design comparison with ARC and LIRS

<table>
<thead>
<tr>
<th></th>
<th>ARC</th>
<th>LIRS</th>
<th>FRD</th>
</tr>
</thead>
<tbody>
<tr>
<td># LRU stack</td>
<td>Two</td>
<td>Two</td>
<td>Two</td>
</tr>
<tr>
<td>Adaptive Resizing</td>
<td>O</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Eviction Point</td>
<td>Two</td>
<td>One</td>
<td>Two</td>
</tr>
<tr>
<td></td>
<td>(Two LRU stacks are</td>
<td>(Two LRU stacks are</td>
<td>(Two LRU stacks are</td>
</tr>
<tr>
<td></td>
<td>isolated)</td>
<td>not isolated)</td>
<td>isolated)</td>
</tr>
<tr>
<td>History size</td>
<td>Cache size x 2</td>
<td>Max resident block</td>
<td>Max resident block</td>
</tr>
</tbody>
</table>