A Low-Overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References

Jong Min Kim  Jongmoo Choi  Jesung Kim
Sam H.Noh  Sang Lyul Min  Yookun Cho
Chong Sang Kim

School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea

Department of Computer Engineering
Hong-Ik University
Seoul 121-791, Korea
Outline

- Introduction
- Related work
- The Unified Buffer Management Scheme
- Simulations
- Implementation
- Conclusions
Efficient management of the buffer cache is important.

The LRU scheme is still widely used due to its simplicity.

A main drawback of the LRU scheme, is that it cannot exploit regularities in block accesses such as sequential and looping references.
It devises a new buffer management scheme called the Unified Buffer Management (UBM) scheme.

The UBM scheme exploits regularities in reference patterns such as sequential and looping references.
Related Work

- Page/block replacement schemes:
  - It’s based on frequency and/or recency factors. ex: LRU, LFU, LRFU.
  - It’s based on user-level hints.
  - It’s making use of regularities of references such as sequential references and looping references. ex: 2Q, SEQ, EELRU.
Related Work

SQE:

It detects **long sequences** of page faults and applies the MRU scheme to those pages.
- **2Q**
  - **A1** in queue (FIFO)
  - **A1** out queue (LRU)

- **EELRU**
  - (early eviction LRU)

---

If
- time 0, access 6 pages (page: a~f)
- time 2, access 1 page (page: z)
- time 3, access 6 pages (page: g~l)
- time 4, access 2 pages (page: x,y)
- time 5, access 10 pages (page: n~w)

Then
Evicted block order: (n~w), (g~l), (a~f), (z), (x,y)
The UBM Management Scheme

- It is composed of following three main modules:
  - Detection
  - Replacement
  - Allocation
Part 1: Detection

- **Sequential references** that are consecutive block references occurring only once.
- **Looping references** that are sequential references occurring repeatedly with a regular interval.
- **Other references** that are detected neither as sequential nor as looping references.
Figure 2: Classification process of the UBM scheme.
Part 1: Detection

- For example:
Part 2 : Block Replacement Schemes

- The buffer cache is divided into three partitions to accommodate the three different types of references.
  - **Sequential reference** :
    MRU replacement policy is used.
  - **Loop reference** :
    It’s based on their periods.
  - **Other reference** :
    It’s based on LRU replacement scheme.
Part 3: Buffer Allocation Based on MG

- Marginal gain
  - Which has frequently been used in resource allocation strategies in various computer systems areas.
  - It’s defined as $\text{MG}(n) \approx \text{Hit}(n) - \text{Hit}(n-1)$

- The expected number of buffer hits per unit time of sequential references when using $n$ buffers is $\text{Hit}_\text{seq}(n) = 0$, $\therefore \text{MG}_\text{seq}(n) = 0$. 
Part 3: Buffer Allocation Based on MG

- For a looping reference: loop$_i$, with loop length $l_i$ and loop period $p_i$
  - $Hit_{loop_i}(n) = \min[l_i, n]/p_i$
  - If $n < l_i$, $MG_{loop_i}(n) = n/p_i - (n - 1)/p_i = 1/p_i$
  - If $n > l_i$, $MG_{loop_i}(n) = l_i/p_i - l_i/p_i = 0$
Part 3: Buffer Allocation Based on MG

- For other references:
  - Belady’s lifetime function
    $$\text{hit}_{\text{other}}(n) = h_1 + h_2 + h_3 + \ldots + h_n \approx 1 - c \times n^{-k}$$
  - $$\text{Hit}_{\text{other}}(n) = \text{hit}_{\text{other}}(n) \times \left[ \frac{n_{\text{other}}}{n_{\text{total}}} \right]$$
  - $$\text{MG}_{\text{other}}(n) = \text{Hit}_{\text{other}}(n) - \text{Hit}_{\text{other}}(n-1).$$
Overall structure of the UBM scheme

1. Update the table and detect sequential and looping references.
2. Request new buffer space.
3. Send a replacement request to this partition based on marginal gains.
4. Select a victim block using an appropriate replacement scheme.
5. Deallocation buffer space of the victim block.
6. Allocate new buffer space.
7. Fetch the missed block from the disk.

Detector

Other reference
Sequential reference
Looping reference

Block references

Other references
Sequential reference
Looping references

Allocator
Characteristics of the traces used.

<table>
<thead>
<tr>
<th>Trace</th>
<th>Applications executed concurrently</th>
<th># of references</th>
<th># of unique blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi1</td>
<td>cscope, cpp</td>
<td>15858</td>
<td>2606</td>
</tr>
<tr>
<td>Multi2</td>
<td>cscope, cpp, postgres</td>
<td>26311</td>
<td>5684</td>
</tr>
<tr>
<td>Multi3</td>
<td>cpp, gnuplot, glimpse, postgres</td>
<td>30241</td>
<td>7453</td>
</tr>
</tbody>
</table>

We built simulators for the LRU, 2Q, SEQ, EELRU, and OPT schemes as well as the UBM scheme.
Simulations

Multi1 (cscope+cpp) trace

Multi2 (cscope+cpp+postgres) trace
Simulations
Implementation

- **Settings**
  - In the FreeBSD operation system
  - On a 133MHz Intel Pentium PC with 128MB RAM and a 1.6GB Quantum Fireball hard disk.
  - Cache size: 8MB, 12MB and 16MB with block size set to 8KB
Implementation

cpp+postgres+cscope

Elapsed Time (seconds)

LRU  SEQ  UBM  LRU  SEQ  UBM  LRU  SEQ  UBM

8MB Cache       12MB Cache       16MB Cache

600

400

200

0

glimpse+cpp+postgres+cscope

Elapsed Time (seconds)

LRU  SEQ  UBM  LRU  SEQ  UBM  LRU  SEQ  UBM

8MB Cache       12MB Cache       10MB Cache

600

400

200

0
Implementation

![Graph showing performance comparison for different cache sizes and schemes.

- 8MB Cache
- 12MB Cache
- 16MB Cache

- LRU Scheme
- SEQ Scheme
- UBM

Performance measured in elapsed time (seconds): 0 to 600 seconds.]

 CPP + Postgres + Cscope + Mpeg_player
Conclusions

- The UBM scheme shows substantial performance improvements increasing the buffer hit ratio and the elapsed time by compared to the LRU scheme.

- We are attempting to apply to other references the Least Recently/Frequently Used (LRFU) scheme based on both recency and frequency factors rather than the LRU scheme.