A New Buffer Cache Design
Exploiting both Temporal and Content Localities

Jin Ren and Qing Yang
Dept of Electrical, Computer, and Biomedical Engineering
University of Rhode Island, Kingston, RI
Email: {rjin, qyang}@ele.uri.edu

2010 International Conference on Distributed Computing Systems
Outline

• Introduction
• LPU Design
• Implementation
• Experiments and Evaluations
• Conclusions
Introduction (1/2)

• Buffer cache plays a critical role between disk drive and main memory.

• Content locality has not been one of the major considerations to traditional cache designs.

• Content locality refers to the fact that many data blocks in disk storage share similar or even same content.
The objective of this paper is to exploit the ever increasing content locality in buffer cache design to minimize disk I/O operations.

The idea is to dynamically identify the most popular data blocks.

A new cache replacement algorithm, Least Popularly Used (LPU).
Overview of LPU Design

• The key to LPU is how to find blocks that are both accessed frequently and resembled by as many other blocks as possible.

• In order to allow the buffer cache to be managed based on popularity.
Popularity

• Each cache block is divided into $S$ sub-blocks.
• A sub-signature is calculated for each of the $S$ sub-blocks.
• Heatmap is maintained in our LPU buffer cache design.
  – It has $S$ rows and $Vs$ columns.
  – $Vs$ is the total number of possible signature values for a sub-block.
Popularity

If sub-signature is 8bits.

- $S = 8, V_s = 256$
- the size of a cache block is fixed at 4 KB.

A block :

```
| 0 | 1 | 2 | ... | 16 | ... | 32 | ... | 64 | 65 | ... | 510 | 511 |
```

1st sub-signature :

```
0 + 16 + 32 + 64 = 55
```
Popularity

- Each data block is divided into 8 sub-blocks. \( S = 8 \)
- If the sub-signature is 8 bits, \( V_s = 256 \)
• Each blocks has 2 sub-blocks ($S = 2$)
• Each sub-signature has only four possible values ($Vs = 4$)
• Assume that all possible contents of sub-blocks are $A, B, C, and D$ and their corresponding signatures are $a, b, c, and d$, respectively.

<table>
<thead>
<tr>
<th>I/O sequence</th>
<th>Content</th>
<th>Signature</th>
<th>Heatmap[0] a b c d</th>
<th>Heatmap[1] a b c d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Initialized</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>LBA1</td>
<td>AB</td>
<td>ab</td>
<td>1 0 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>LBA2</td>
<td>CD</td>
<td>cd</td>
<td>1 0 1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>LBA3</td>
<td>AD</td>
<td>ad</td>
<td>2 0 1 0</td>
<td>0 1 0 2</td>
</tr>
<tr>
<td>LBA4</td>
<td>BD</td>
<td>bd</td>
<td>2 1 1 0</td>
<td>0 1 0 3</td>
</tr>
<tr>
<td>LBAs</td>
<td>Block</td>
<td>Popularity</td>
<td>LRU</td>
<td>Reference</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>------------</td>
<td>-----</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AB</td>
<td>CD</td>
</tr>
<tr>
<td>LBA1</td>
<td>AB</td>
<td>2+1 = 3</td>
<td>AB</td>
<td>AB</td>
</tr>
<tr>
<td>LBA2</td>
<td>CD</td>
<td>1+3 = 4</td>
<td>CD</td>
<td>CD</td>
</tr>
<tr>
<td>LBA3</td>
<td>AD</td>
<td>2+3 = 5</td>
<td>AD</td>
<td>_D</td>
</tr>
<tr>
<td>LBA4</td>
<td>BD</td>
<td>1+3 = 4</td>
<td>BD</td>
<td>BD</td>
</tr>
<tr>
<td></td>
<td>Cache space</td>
<td>4</td>
<td>3.5</td>
<td>3</td>
</tr>
</tbody>
</table>

- Our objective is to select a **reference block** in such a way to maximize the number of remaining blocks that have small differences from the reference block.
Cache Management

• LPU divides cache into 3 parts as virtual block list, data blocks, and delta blocks.
Cache Management

• A virtual block can be one of three different types:
  – reference block
  – associate block
  – independent block
Implementation

- Our LPU is designed as a cache layer within KVM.

> LPU uses mapping file to build a shard buffer for all VMs.
Implementation

- The I/O requests captured by LPU are identified as virtual blocks.
- The LPU queue is scanned after every 50,000 I/O requests to select reference blocks based on their popularities.
- LPU picks up 20,000 warm blocks in the LPU queue and groups them to ensure reference blocks are sufficiently different from each other so that we can have as many associate blocks as possible.
The signature of the 1st block: 00 00 00 00 00 00 00 00

The signature of the 2nd block: 80 00 00 00 00 00 00 00
Experiments and Evaluations

• Two Severs:
  – Host VM Sever: Dell PowerEdge T410 with 1.8GHz Xeon CPU, 2GB RAM, and 160G Dell SATA drive.
  – Guest VM Sever: A Dell Precision 690 with 1.6GHz Xeon CPU, 2GB RAM, and 400G Seagate SATA drive.

• They are connected using a gigabit Ethernet switch.

• The host OS and virtual guest OS are both Ubuntu 8.10 server edition.
Experiments and Evaluations

- Benchmarks:

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RU</td>
<td>RUBiS</td>
<td>e-Commerce web server workload</td>
</tr>
<tr>
<td>TP</td>
<td>TPCC-UVA</td>
<td>Database server workload</td>
</tr>
<tr>
<td>SM</td>
<td>SPECmail2009</td>
<td>Mail server workload</td>
</tr>
<tr>
<td>SB</td>
<td>SPECwebBank</td>
<td>Online banking</td>
</tr>
<tr>
<td>SE</td>
<td>SPECwebEcommerce</td>
<td>Online store selling computers</td>
</tr>
<tr>
<td>SS</td>
<td>SPECwebSupport</td>
<td>Vendor support website</td>
</tr>
<tr>
<td>MX</td>
<td>Mixed</td>
<td>Heterogeneous workload</td>
</tr>
</tbody>
</table>

- LRU and data de-duplication are also implemented for the purpose of performance comparison with LPU.
Performance Evaluations and Comparisons

Figure 4. I/O rate of RUBiS.

Figure 5. I/O rate of TPCC-UVA.

Figure 6. I/O rate of SPECmail.

Figure 7. I/O rate of SPECwebBank.

Figure 8. SPECwebEcommerce.

Figure 9. SPECwebSupport.
Figure 10. I/O rate of Mixed workloads.

Figure 11. Application performance and CPU utilization of RUBiS.

Figure 12. Application performance and CPU utilization of TPCC-UVA.
Conclusions

• The paper has presented a novel buffer cache design that exploits both temporal locality and content locality of disk operations.