Erasing Belady’s Limitations: In Search of Flash Cache Offline Optimality

Yue Cheng1,2  Fred Douglis2  Philip Shilane2  Michael Trachtman2
Grant Wallace2  Peter Desnoyers3  Kai Li4

1Virginia Tech  2EMC  3Northeastern University  4Princeton University
Background

• Unlike magnetic HDDs, flash SSDs erase data in large blocks
• Limited number of erasures before errors occur

... Want to make effective use of SSDs while being cognizant of erasure penalty
Flash Caching

- Using SSDs for an HDD cache
  - SSD cache as a tier btw. DRAM and HDD
    - Goal: to balance performance against endurance
  - Nitro [USENIX ATC ‘14], CacheDedup [USENIX FAST ‘16]
  - RIPQ [USENIX FAST ‘15]
  - Pannier [ACM Middleware ’15]

- How do we know if we’re doing well?
  - Comparison against an offline optimal "best case"
  - But what is the offline optimal for flash caches?
Offline Optimality

• Belady’s MIN: A simple offline caching algorithm when the next access is known
  – Inserts a new entry into the cache, removing the entry that will be used furthest in the future
  – Yields the optimal read hit ratio (RHR)

• MIN is not able to provide the optimal erasures in the context of flash caching
  – MIN inserts data that *won’t actually be read*
Flash Cache Offline Optimality

• Objectives:
  – O1: Minimize erasures s.t. maximal RHR
    • Never insert items if it does not increase RHR
  – O2: Maximize RHR s.t. an endurance limit
  – O3: Maximize combination of RHR and erasures

• True optimal
  – How to compute the offline optimal?
  – What is the complexity?

• Heuristics
  – How can we approximate?

The focus of our work
MIN

• Blocks
  – Minimum unit of access to the cache (e.g., 4KB)

• MIN priority queue (ranked based on next ref. timestamp)
  1. Don’t insert blocks w/ furthest next ref.
MIN

- **Blocks**
  - Minimum unit of access to the cache (e.g., 4KB)

- **MIN priority queue**
  (ranked based on next ref. timestamp)
  1. Don’t insert blocks w/ furthest next ref.
  2. A new block is inserted
Key Components

- **MIN priority queue**
  - In-RAM structure tracking runtime status at block granularity

- **Containers**
  - Unit of insertion & eviction for flash cache (e.g., 2MB)

- **Write buffer**
  - Packs blocks into in-RAM write buffer (e.g., 8MB)

- **Container priority queue**
  - In-RAM structure tracking flash containers
Container-optimized Heuristic

- Insert a block
  - Into the write buffer (sorted by *eviction timestamp*)
  - Into the MIN queue (sorted by *next ref. timestamp*)
Container-optimized Heuristic

- Insert a block
  - Into the write buffer (sorted by *eviction timestamp*)
  - Into the MIN queue (sorted by *next ref. timestamp*)
Container-optimized Heuristic

- Write buffer is dispersed into containers when it is **FULL**
- The containers are written to the flash cache
Container-optimized Heuristic

- Update block status on invalidation or when MIN would evict block as furthest in future (“evict-pending”)
  - Remains in container until that is GC’d
- Rank container queue by # valid blocks
- Evict the tail container to make room for new data
- Copy forward valid blocks to the write buffer
Optimizations

• **R₁**: Only insert blocks read at least once before eviction

• **TRIM**: Skips *dead* blocks during GC
  – Dead: overwritten or never reaccessed

• **Copy-forwarding reduction**: Eliminates wasted CF blocks

• **E**: Segregates blocks by eviction timestamp
Experimental Methodology

• Storage traces (34): limit to large enough datasets
  – EMC VMAX: 25
  – MSR Cambridge: 3
  – MS Production Servers: 6

• Implementation
  – Full-system flash cache simulator
  – Vary cache size as function of unique data accessed in trace: 1-10%

• Metrics
  – Performance: Read hit ratio (RHR)
  – Endurance: Erasures per block per day (EPBPD)
  – Function of RHR and EPBPD: Weighted flash usage effectiveness (WFUE)
Comparing Algorithms

All 3 offline algorithms achieve the same (optimal) RHR

(O=Offline, C=Container-optimized)

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
<th>O</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>Least recently used</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>RIPQ+</td>
<td>Static web content</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Pannier</td>
<td>Handles divergent containers</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>MIN</td>
<td>Don’t insert data w/ furthest next ref</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>MIN+</td>
<td>Don’t insert data evicted w/o read</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>Our container-optimized heuristic</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Higher is better!

Read hit ratio (%)
Comparing Algorithms

**Read Hit Ratio (%)**
- **Online 2.5% cache size**
- **Offline**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Online</th>
<th>Offline</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>34.6</td>
<td>43.6</td>
</tr>
<tr>
<td>RIPQ+</td>
<td>37.7</td>
<td></td>
</tr>
<tr>
<td>Pannier</td>
<td></td>
<td>53.6</td>
</tr>
<tr>
<td>MIN</td>
<td>53.6</td>
<td>53.6</td>
</tr>
<tr>
<td>MIN+</td>
<td>53.6</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>53.6</td>
</tr>
</tbody>
</table>

- Higher is better!
- Lower is better!

**EPBPD**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Online</th>
<th>Offline</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>484</td>
<td></td>
</tr>
<tr>
<td>RIPQ+</td>
<td>181</td>
<td>171</td>
</tr>
<tr>
<td>Pannier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>MIN+</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- C has significantly fewer EPBPD w/ identical RHR!
Evaluating Heuristic Techniques

- **$R_1$:** Omit insertion with no reread
- **TRIM:** Notify GC to omit dead blocks
- **CFR:** Avoid wasted CF blocks
- **E:** Segregate blocks by evict timestamp

![Bar chart showing EPBPD values for different techniques at 2.5% cache size.](chart.png)

- Base MIN: 30
- $+R_1$: 22 (27% lower)
- $+TRIM$: 15 (32% lower)
- $+CFR$: 12 (20% lower)
- $+E$: 10 (67% lower)
Evaluating Heuristic Techniques

- **R**: Omit insertion with no reread
- **TRIM**: Notify GC to omit dead blocks
- **CFR**: Avoid wasted CF blocks
- **E**: Segregate blocks by evict timestamp

<table>
<thead>
<tr>
<th>Technique</th>
<th>Lower is better!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base MIN</td>
<td>6.3</td>
</tr>
<tr>
<td>+R₁</td>
<td>5.8</td>
</tr>
<tr>
<td>+TRIM</td>
<td>3.0</td>
</tr>
<tr>
<td>+CFR</td>
<td>2.9</td>
</tr>
<tr>
<td>+E</td>
<td>2.8</td>
</tr>
</tbody>
</table>

10% cache size

Virginia Tech

EMC

Northeastern University

18
Conclusion

- Important to have a baseline for the offline optimal considering both RHR and endurance

- Our container-optimized heuristic **maintains the optimal RHR** while reducing erasures by **up to 67%**

- Additional optimizations may be possible to move this heuristic to the **true** optimal
Erasing Belady’s Limitations: In Search of Flash Cache Offline Optimality

Thank you! Q & A

Acknowledgments:
Dan Arnon, Cheng Li, Stephen Manley, Darren Sawyer, Dan Tsafrir, Kevin Xu, and Sanjeev Arora and his students from Princeton University