SIEVE is Simpler than LRU: An Efficient Turn-Key Eviction Algorithm for Web Caches

Yazhuo Zhang, Juncheng Yang, Yao Yue, Ymir Vigfusson, K.V. Rashmi

Emory University, Carnegie Mellon University, Pelikan Foundation
Caching System is Important

Page Cache

Limited Space!

Web Caches

Core: Eviction Algorithm
Cache Metrics

- Efficiency
- Scalability

- Cache Miss Ratio
- Reqs/Second
Throughput Measured in Cachelib

Twitter workload

Number of Threads

Throughput (Mops)

LRU
Throughput Measured in Cachelib

Twitter workload
Cache Metrics

- Efficiency
- Scalability
- Simplicity

Cache Miss Ratio

Reqs/Second
A Rich Literature of Eviction Algorithms

- FIFO
- LRU
- CLOCK
- LRU-k
- TwoQ
- SLRU
- GDSF
- ARC
- CAR
- MQ
- LHD
- Tiny-LFU
- LHFU
- LRD
- LRB
- HALP
- GL-Cache
- CACHE
- US


complexity
The Trouble with Complexity

- Difficult to debug and maintain
- Difficult to tune the parameters

“Predicting which pages will be accessed in the near future is tricky, and the kernel has evolved many mechanisms to improve its chances of guessing right. But the kernel not only often gets it wrong, but also spends a lot of CPU time to make the incorrect choice.”

-- Linux kernel developer
SIEVE: a Simple and Efficient Cache Eviction Algorithm

*Measured by lines of code
†Measured by average object miss ratio reduction from FIFO
SIEVE Design
The Secret to Designing Efficient Eviction Algorithms

Lazy promotion

Quick demotion

[HotOS'23] FIFO queues can be better than LRU: the Power of Lazy Promotion and Quick Demotion
[SOSP'23] FIFO Queues are all You Need for Cache Eviction
The Secret to Designing Efficient Eviction Algorithms

Lazy promotion

Retain popular objects with minimal effort
- Improve throughput due to less computation
- Improve efficiency due to more information at eviction

[HotOS’23] FIFO queues can be better than LRU: the Power of Lazy Promotion and Quick Demotion
[SOSP’23] FIFO Queues are all You Need for Cache Eviction
The Secret to Designing Efficient Eviction Algorithms

Quick demotion

Quickly remove most new objects, such as one-hit-wonders (no request after insertion)

[HotOS’23] FIFO queues can be better than LRU: the Power of Lazy Promotion and Quick Demotion
[SOSP’23] FIFO Queues are all You Need for Cache Eviction
The Secret to Designing Efficient Eviction Algorithms

Lazy promotion
Retain popular objects with minimal effort

Quick demotion
Remove unpopular objects fast, such as one-hit-wonders

[HotOS’23] FIFO queues can be better than LRU: the Power of Lazy Promotion and Quick Demotion
[SOSP’23] FIFO Queues are all You Need for Cache Eviction
**LRU**
- Insert: LRU
- Evict: LRU
- Efficiency: Eager promotion, No quick demotion
- Scalability: 3 stars

**FIFO**
- Insert: FIFO
- Evict: FIFO
- Efficiency: No promotion, No quick demotion
- Scalability: 5 stars
**FIFO**

- **Efficiency**
  - Eager promotion
  - No quick demotion

- **Scalability**
  - 3 stars

**LRU**

- **Efficiency**
  - No promotion
  - No quick demotion

- **Scalability**
  - 3 stars

**FIFO-Reinsertion**

- **Efficiency**
  - Lazy promotion
  - No quick demotion

- **Scalability**
  - 5 stars
FIFO-Reinsertion
cache hit on D

FIFO-Reinsertion

visited not visited
cache miss

1. FIFO-Reinsertion
2. FIFO-Reinsertion
3. FIFO-Reinsertion

reinsert & reset visited bit

1. insert
2. evict

visited
not visited
- **LRU**
  - Efficiency: Eager promotion, No quick demotion
  - Scalability: 3 stars

- **FIFO**
  - Efficiency: No promotion, No quick demotion
  - Scalability: 5 stars

- **FIFO-Reinsertion**
  - Efficiency: Lazy promotion, No quick demotion
  - Scalability: 5 stars
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Efficiency</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>Eager promotion</td>
<td>3 stars</td>
</tr>
<tr>
<td></td>
<td>No quick demotion</td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td>No promotion</td>
<td>1 star</td>
</tr>
<tr>
<td></td>
<td>No quick demotion</td>
<td></td>
</tr>
<tr>
<td>FIFO-Reinsertion</td>
<td>Lazy promotion</td>
<td>4 stars</td>
</tr>
<tr>
<td></td>
<td>No quick demotion</td>
<td></td>
</tr>
<tr>
<td>SIEVE</td>
<td>No promotion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No quick demotion</td>
<td></td>
</tr>
</tbody>
</table>
cache hit on D

visited  not visited
cache miss

reset visited bit & move hand

1

SIEVE

2

SIEVE

3

SIEVE

visited

not visited
Quickly remove new objects

insert

SIEVE

evict
Separate new and old objects

CLOCK
- Insert
- Evict
- Reinsert

SIEVE
- Insert
- Evict

“Survived” object
Newly inserted object

[NSDI’21] SegCache: memory-efficient and high-throughput DRAM cache for small objects
### FIFO-Reinsertion
- **Insert**: FIFO-Reinsertion
- **Evict**: Reinsert
- **Efficiency**: Eager promotion, No quick demotion
- **Scalability**: No quick demotion

### LRU
- **Insert**: LRU
- **Evict**: None
- **Efficiency**: Eager promotion, No quick demotion
- **Scalability**: No quick demotion

### FIFO
- **Insert**: FIFO
- **Evict**: None
- **Efficiency**: No promotion, No quick demotion
- **Scalability**: No quick demotion

### SIEVE
- **Insert**: SIEVE
- **Evict**: None
- **Efficiency**: Lazy promotion, Quick demotion
- **Scalability**: No quick demotion
SIEVE Evaluation
Web Cache Workloads

- Simulator: libCacheSim
- Prototype: Cachelib
- Testbed: Cloudlab

<table>
<thead>
<tr>
<th>trace collection</th>
<th>collection time</th>
<th>#traces</th>
<th>cache type</th>
<th># request (million)</th>
<th># object (million)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDN1</td>
<td>2021</td>
<td>1273</td>
<td>object</td>
<td>37,460</td>
<td>2,652</td>
</tr>
<tr>
<td>CDN2</td>
<td>2018</td>
<td>219</td>
<td>object</td>
<td>3,728</td>
<td>298</td>
</tr>
<tr>
<td>Tencent Photo</td>
<td>2018</td>
<td>2</td>
<td>object</td>
<td>5,650</td>
<td>1,038</td>
</tr>
<tr>
<td>Wiki CDN</td>
<td>2019</td>
<td>3</td>
<td>object</td>
<td>2,863</td>
<td>56</td>
</tr>
<tr>
<td>Twitter KV</td>
<td>2020</td>
<td>54</td>
<td>KV</td>
<td>195,441</td>
<td>10,560</td>
</tr>
<tr>
<td>Meta KV</td>
<td>2022</td>
<td>5</td>
<td>KV</td>
<td>1,644</td>
<td>82</td>
</tr>
<tr>
<td>Meta CDN</td>
<td>2023</td>
<td>3</td>
<td>object</td>
<td>231</td>
<td>76</td>
</tr>
</tbody>
</table>

1559 traces | 247,017 million requests | 14,852 million objects
SIEVE: Efficiency

SIEVE reduces FIFO’s miss ratio by more than 42% on 10% of the traces (top whisker) with a mean of 21%

CDN1, 1273 traces (37,460 million requests)
SIEVE: Efficiency

SIEVE achieves the best efficiency on the well-studied Zipfian workloads

CDN1, 1273 traces (37,460 million requests)
SIEVE: Throughput

![Graph showing throughput comparison between LRU (optimized), LRU, and SIEVE. The graph illustrates that SIEVE has a 6x improvement in throughput compared to LRU (optimized).]
## SIEVE: Simplicity

<table>
<thead>
<tr>
<th>Cache library</th>
<th>Language</th>
<th>Lines of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupcache</td>
<td>Golang</td>
<td>21</td>
</tr>
<tr>
<td>mnemonic</td>
<td>Javascript</td>
<td>12</td>
</tr>
<tr>
<td>Iru-rs</td>
<td>Rust</td>
<td>16</td>
</tr>
<tr>
<td>Iru-dict</td>
<td>Python + C</td>
<td>21</td>
</tr>
</tbody>
</table>
SIEVE: Simplicity

<table>
<thead>
<tr>
<th>Cache library</th>
<th>Language</th>
<th>Lines of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupcache</td>
<td>Golang</td>
<td>21</td>
</tr>
<tr>
<td>mnemonist</td>
<td>Javascript</td>
<td>12</td>
</tr>
<tr>
<td>Iru-rs</td>
<td>Rust</td>
<td>16</td>
</tr>
<tr>
<td>Iru-dict</td>
<td>Python + C</td>
<td>21</td>
</tr>
</tbody>
</table>

Adoption

Large systems: 🦁 Pelikan 🦁 Nyrkö 🦄 SkiftOS 🦄 DragonFly 🦄 DNSCrypt-proxy 🦄 encrypted-dns-resolver

Cache libraries: 🦁 golang-fifo 🦁 js-sieve 🦄 rust-sieve-cache 🦄 go-sieve

- sieve_cache (Ruby)
- zig-sieve (Zig)
- sieve (Swift)
- sieve (JavaScript)
- sieve (Elixir)
- sieve (Nim)
- sieve-cache (Java)
- sieve (Python)
- sieve-cache-in-rust
- sieve-cache (JavaScript)
- gosieve, sieve (typescript)
SIEVE: Primitive

LeCaR: LRU + LFU + ML

TwoQ: LRU + FIFO

ARC: LRU + LRU + 2 ghost queues
SIEVE: Primitive

LeCaR: LRU + LFU + ML
TwoQ: LRU + FIFO
ARC: LRU + LRU + 2 ghost queues

Replace LRU with SIEVE
More in the paper

- Why SIEVE is effective
- Byte miss ratio
- When SIEVE is not effective
- Comparison to ML algorithms
SIEVE Adoption

• SIEVE is available in over 20 cache libraries with 10+ programming languages

• Production systems start integrating SIEVE: Pelican, SkiftOS, DragonFly, and etc

Turns out Ristretto cache is "async"... I switched WarpStream's footer cache from Ristretto to golang-fifo (Sieve algo) and got a 33x reduction in cache misses and 16% CPU savings...

Cache Loads

9:35 PM - Jan 20, 2024 - 17.3K Views
Takeaway

- Lazy promotion and quick demotion are key to efficient eviction algorithm
- SIEVE uses a moving hand to 1) retain popular objects in place, and 2) remove unpopular objects quickly
- The simplest algorithm with state-of-the-art efficiency and scalability

https://sievecache.com