Adaptive Software Cache Management

Gil Einziger\textsuperscript{1}, Ohad Eytan\textsuperscript{2}, Roy Friedman\textsuperscript{2} and Ben Manes\textsuperscript{3}

June 3, 2019

Middleware ’18

\textsuperscript{1}Ben Gurion University of the Negev & Nokia Bell Labs
\textsuperscript{2}Technion - Israel Institute of Technology
\textsuperscript{3}Independent
The Essence of Caching
The Essence of Caching

• A fast but relatively small memory
• Can temporarily store some items of the "real storage"
• Improves performance if hit-ratio is high
The Essence of Caching

- A fast but relatively small memory
- Can temporarily store some items of the "real storage"
- Improves performance if hit-ratio is high
Least Recently Used

- Idea: recently requested items probably will be requested again
- Policy: evict the oldest item from the cache
- Simple & efficient
- Easily polluted
LFU [5]
Least Frequently Used

- Idea: most popular items probably will be requested again
- Policy: evict the item with the lowest access count from cache
- Complex to implement efficiently
- No freshness mechanism
Problem

- Different workloads have different access patterns:
  - Some are recency biased
  - Some are frequency biased.
  - In fact, most are mixed.
Problem

- Different workloads have different access patterns:
  - Some are recency biased
  - Some are frequency biased.
  - In fact, most are mixed.

- Can we develop a silver bullet policy?

![Graphs showing hit ratio with cache size for Build Cache (gradle) and Search Engine (S3) with LRU and LFU policies.](image)
Modern Cache Management Policies
Modern Cache Management Policies

- ARC (2002)
- Hyperbolic (2017)
- Mini-Sim (2017)
- FRD (2017)
- W-TinyLFU (2017)
W-TinyLFU [2]

The cache consists of two areas:

⋆ Window Cache, which is a simple LRU cache
⋆ Main Cache, which is a SLRU cache with an admission policy

It uses an approximate counting scheme to maintain statistics of item frequency (histogram) with periodic aging.

Items evicted from the Window Cache are candidates to enter the Main Cache.

The default Window Cache size is 1% of the cache size.
W-TinyLFU [2]

- The cache consists of two areas:
  - **Window Cache** which is a simple LRU cache
  - **Main Cache** which is a SLRU cache with an **admission policy**
- Uses **approximate counting scheme** to maintain statistics of items frequency (histogram) with **periodic aging**
- Items evicted from the Window Cache are **candidates** to enter the Main Cache
- Default Window Cache is **1%** of the cache
The cache consists of two areas:

- **Window Cache** which is a simple LRU cache
- **Main Cache** which is a SLRU cache with an admission policy

- Uses approximate counting scheme to maintain statistics of items frequency (histogram) with periodic aging
- Items evicted from the Window Cache are candidates to enter the Main Cache
- Default Window Cache is 1% of the cache

Caffeine [Image]
The cache consists of two areas:

- **Window Cache** which is a simple LRU cache
- **Main Cache** which is a SLRU cache with an admission policy

- Uses approximate counting scheme to maintain statistics of items frequency (histogram) with periodic aging
- Items evicted from the Window Cache are candidates to enter the Main Cache
- Default Window Cache is 1% of the cache
The cache consists of two areas:

- **Window Cache** which is a simple LRU cache
- **Main Cache** which is a SLRU cache with an admission policy

Uses an approximate counting scheme to maintain statistics of items.

Items evicted from the Window Cache are candidates to enter the Main Cache.

Default Window Cache is 1% of the cache.
The cache consists of two areas:

- **Window Cache** which is a simple LRU cache
- **Main Cache**, a SLRU cache with an admission policy

It uses an approximate counting scheme to maintain statistics of item frequency with periodic aging.

Items evicted from the Window Cache are candidates to enter the Main Cache.

The Default Window Cache is 1% of the cache.
The cache consists of two areas:

- **Window Cache** which is a simple LRU cache
- **Main Cache** which is a SLRU cache with an admission policy

Uses approximate counting scheme to maintain statistics of items

Items evicted from the Window Cache are candidates to enter the Main Cache

Default Window Cache is 10% of the cache
Unsatisfying

Build cache (gradle)  Search engine (S3)  Financial (OLTP)

W-TinyLFU (1%)  ARC  FRD
Unsatisfying

- Build cache (gradle)
- Search engine (S3)
- Financial (OLTP)

Graphs showing hit ratio (%)

- ben-manes / caffeine
- cache performing worse than LRU #106
  - phraktle opened this issue on Aug 7, 2016 · 38 comments
Our Adaptive Caching
Basic Idea

- Dynamically adjust a selected tuning parameter
Basic Idea

• Dynamically adjust a selected tuning parameter

• Suggested tuning **parameters**:
  ★ For W-TinyLFU: change the ratio between the cache areas
  ★ For W-TinyLFU: change the sketch increment parameter

• Suggested **adaptation approaches**:
  ★ Hill climbing: try and see what happens
  ★ Indicator: track statistics and decide directly

• We end up with **4** suggested policies
Parameters:
Areas Ratio

The partition between the cache areas implies a trade-off between recency and frequency:

- Frequency biased configuration:
- Recency biased configuration:

Very effective: S3 OLTP gradle
Parameters: Areas Ratio

- The partition between the cache areas implies a trade-off between recency and frequency:
  - Frequency biased configuration:
    - Very effective: S3 OLTP gradle
  - Recency biased configuration:
Parameters: Areas Ratio

- The partition between the cache areas implies a trade-off between recency and frequency:
  - Frequency biased configuration:
  - Recency biased configuration:

- Very effective:

![Graphs showing hit ratio vs. window cache size for different configurations and workloads.](image-url)
Parameters:
Sketch

- TinyLFU sketch:
  - **Aging** mechanism divides all the counters by 2 each $S$ steps.
  - The counters are **bounded** by 16.
- Enlarging the **counters increment** on each item’s access from 1 to a larger value **favors recency**:
  - Increment of 2:
  - Increment of 4:
Adaptation Techniques:
Hill Climbing

• Well known optimization technique:

• Step size: 5% or 1.
• Almost no overhead.
• Constantly changes.
Adaptation Techniques:
Indicator

• Composed from two ingredients:
  ⚫ **Hint** - the average of the sketch estimation for all of the accesses.
Adaptation Techniques: Indicator

- Composed from two ingredients:
  - **Hint** - the average of the sketch estimation for all of the accesses.
  - **Skew** - an estimation of the skewness of the items.

We define:

\[
\text{Indicator} \equiv \text{Hint} \cdot \left(1 - \min\{1, \text{Skew}^3\}\right)
\]

Which gives us a value in \([0, 1]\).
Adaptation Techniques:
Indicator

- Composed from two ingredients:
  - **Hint** - the average of the sketch estimation for all of the accesses.
  - **Skew** - an estimation of the skewness of the items.

We define:

\[
\text{Indicator} \approx \text{Hint} \cdot (1 - \min\{1, \text{Skew}^3\})
\]

Which gives us a value in \([0, 1]\).
Adaptation Techniques: Indicator

- Composed from two ingredients:
  - **Hint** - the average of the sketch estimation for all of the accesses.
  - **Skew** - an estimation of the skewness of the items.

- We define:

\[
indicator \triangleq \frac{hint \cdot (1 - \min \{1, skew^3\})}{maxFreq}
\]

Which gives us a value in $[0, 1]$. 

\[
\begin{align*}
\text{Reset} & \quad \text{Time to Reset} \\
\text{Reset} & \quad \text{Time to Reset} \\
\text{Most Recent} & \quad \text{Time to Reset}
\end{align*}
\]
Results
Results

Adaptive W-TinyLFU Sketch:

- gradle
- S3
- OLTP

Graphs showing hit ratio vs cache size for various workloads.
Results

Adaptive W-TinyLFU Window:

WC-W-TinyLFU  WI-W-TinyLFU  W-TinyLFU (1%)
Results

Competitive for all tested workloads: ✓

- gradle
- S3
- OLTP
- F1
- DS1
- WS1

Legend:
- WC-W-TinyLFU
- WI-W-TinyLFU
- W-TinyLFU (1%)
- ARC
- FRD
Results: Completion Time

Search Engine (S3)
Cache Size: 500000 [items]
Conclusions

- Adaptation works 😊
- Window adaptation better than sketch adaptation
- Indicator adapts quicker

But
Hill climber is simpler to implement and requires no extra space
Conclusions

- Adaptation works 😊
- Window adaptation better than sketch adaptation
- Indicator adaptation quicker

**But**

Hill climber is simpler to implement and requires no extra space
Thank You

Questions/Ideas?

P.S. If you could share a trace with variable item sizes for further research, please contact me at ohadey@cs.technion.ac.il
L. A. Belady.  
**A study of replacement algorithms for a virtual-storage computer.**  

G. Einziger, R. Friedman, and B. Manes.  
_Tinylfu: A highly efficient cache admission policy._  

N. Megiddo and D. S. Modha.  
_ARC: A self-tuning, low overhead replacement cache._  
In J. Chase, editor, _Proceedings of the FAST ’03 Conference on File and Storage Technologies, March 31 - April 2, 2003, Cathedral Hill Hotel, San Francisco, California, USA_.  

S. Park and C. Park.  
_Frd: A filtering based buffer cache algorithm that considers both frequency and reuse distance._  
2017.

A. Silberschatz, P. B. Galvin, and G. Gagne.  
_Operating system concepts, 7th Edition._  