# Adaptive Software Cache Management

Gil Einziger[1], **Ohad Eytan**[2], Roy Friedman[2] and Ben Manes[3]

June 3, 2019

Middleware '18

[1]Ben Gurion University of the Negev & Nokia Bell Labs
[2]Technion - Israel Institute of Technology
[3]Independent

# The Essence of Caching

# 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

# LRU [5]

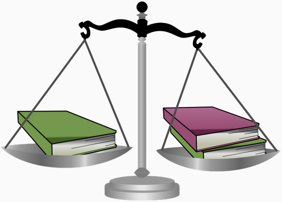
## Least Recently Used

- Idea: recently requested items probably will be requested again
- Policy: evict the oldest item from the cache
- Simple & efficient
- Easily polluted

# LFU [5]

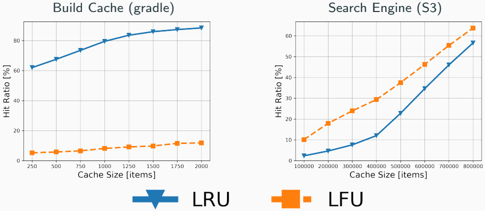
## Least Frequently Used

- 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



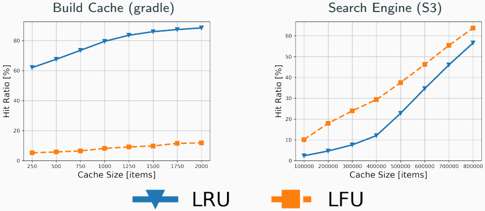
# Problem

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

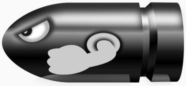


# Problem

- 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?



# Modern Cache Management Policies

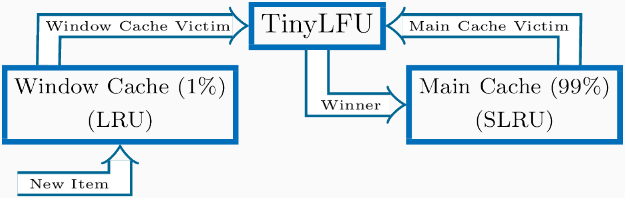
# Modern Cache Management Policies

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

# W-TinyLFU [2]

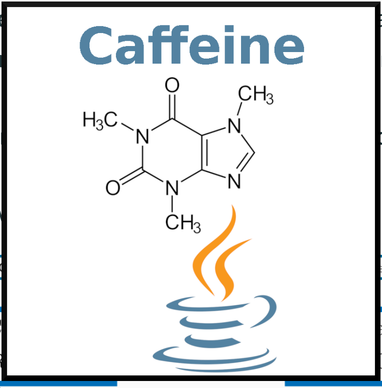
# W-TinyLFU [2]

- 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



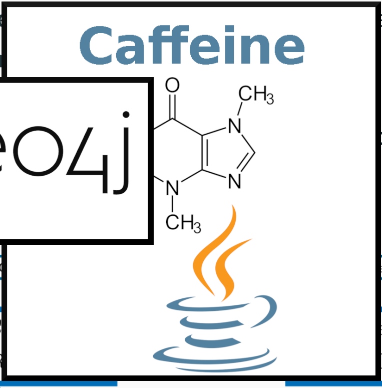
# W-TinyLFU [2]

- The cache consists of two areas:
  - ⋆ **Window Cache** which is a simple LRU cache
  - ⋆ **Main Cache** … **admission policy**
- Uses **approxim**… n statistics of items frequency (hist…
- Items evicted f… **dates** to enter the Main Cache
- Default Windo…

## W-TinyLFU [2]

- The cache consists of two areas:
  - ⋆ **Window Cache** which is a simple LRU cache
  - ⋆ **Main Cache** … **admission policy**
- Uses **approxim**… n statistics of items
- … **dates** to enter the

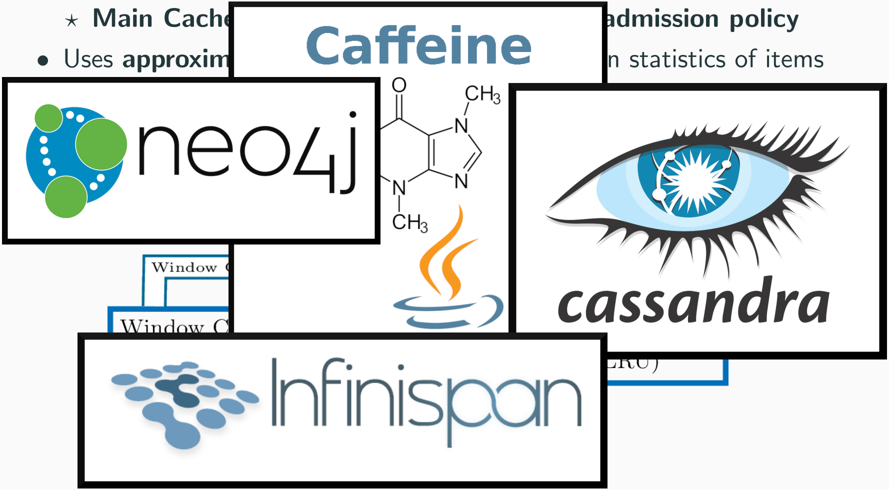



# W-TinyLFU [2]

- The cache consists of two areas:
  - ⋆ **Window Cache** which is a simple LRU cache
  - ⋆ **Main Cach**e … **admission policy**
- Uses **approxim**… n statistics of items

# W-TinyLFU [2]

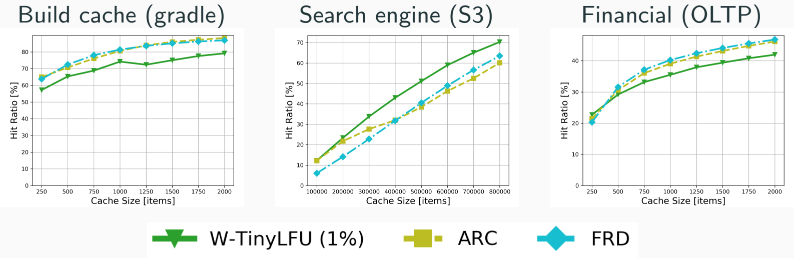
- The cache consists of two areas:
  - ⋆ **Window Cache** which is a simple LRU cache
  - ⋆ **Main Cache** … **admission policy**
- Uses **approxim**… …n statistics of items



# W-TinyLFU [2]

- The cache consists of two areas:
  - ⋆ **Window Cache** which is a simple LRU cache
  - ⋆ **Main Cache** ... **admission policy**
- Uses **approxim**... n statistics of items

# Unsatisfying



# Unsatisfying

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

# 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

# Parameters:
# Areas Ratio

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


  - ⋆ Recency biased configuration:

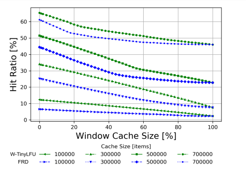
# Parameters: Areas Ratio

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

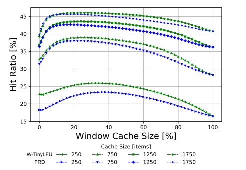

  - ⋆ Recency biased configuration:


- Very effective:

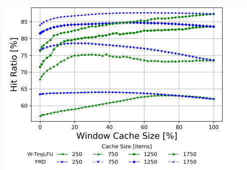
S3




OLTP

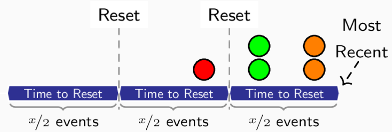



gradle

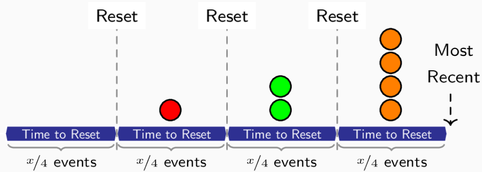


# Parameters: Sketch

- 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:

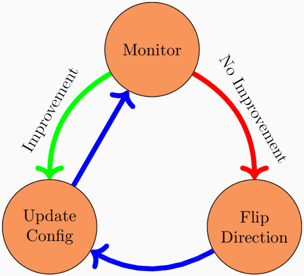



  - ⋆ Increment of 4:



# Adaptation Techniques: Hill Climbing

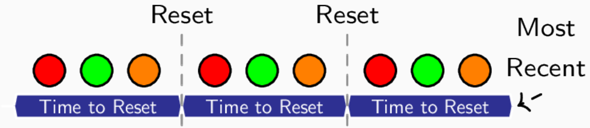
- Well known optimization technique:




- Step size: 5% or 1.
- Almost no overhead.
- Constantly changes.

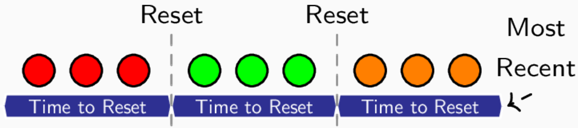
# Adaptation Techniques: Indicator

- Composed from two ingredients:
  - ⋆ **Hint** - the average of the sketch estimation for all of the accesses.



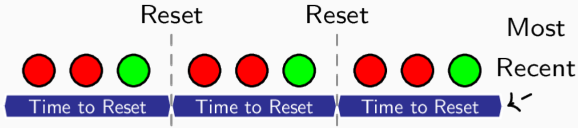
# Adaptation Techniques: Indicator

- Composed from two ingredients:
  - ⋆ **Hint** - the average of the sketch estimation for all of the accesses.



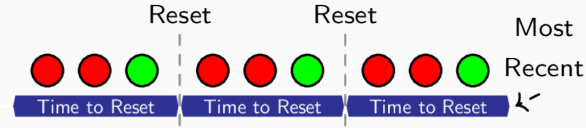
# Adaptation Techniques: Indicator

- 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:

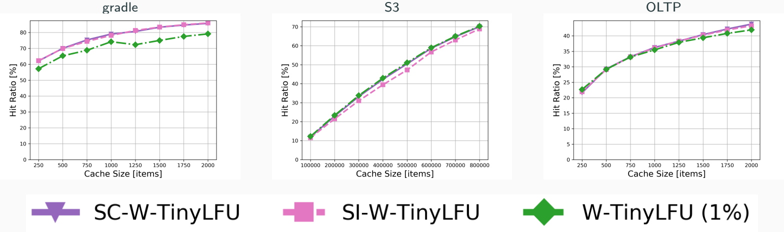
$$indicator \triangleq \frac{hint \cdot \left(1 - \min\left\{1, skew^3\right\}\right)}{maxFreq}$$

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

# Results

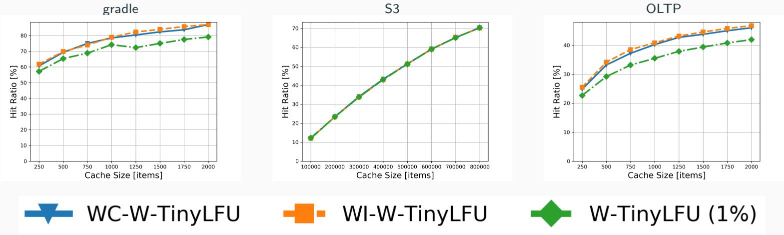
# Results

Adaptive W-TinyLFU Sketch:



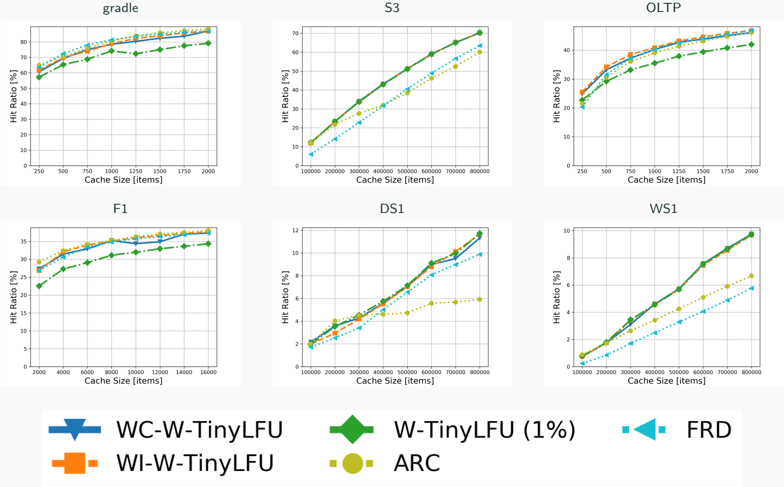
# Results

Adaptive W-TinyLFU Window:

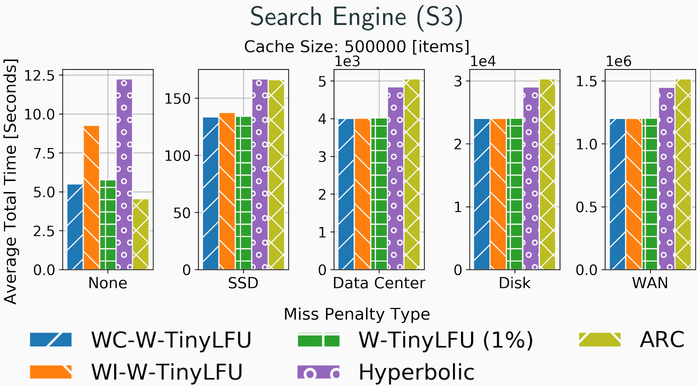


# Results

Competitive for all tested workloads: ✓



# Results: Completion Time



## Conclusions

- Adaptation works 😄
- Window adaptation better then sketch adaptation
- Indicator adapt quicker
  **But**
  Hill climber is simpler to implement and requires no extra space

# Conclusions

- Adaptation works 😄
- Window adaptation better then sketch adaptation
- Indicator adapt 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

# References i


L. A. Belady.
**A study of replacement algorithms for a virtual-storage computer.**
*IBM Systems Journal*, 5(2):78–101, 1966.

G. Einziger, R. Friedman, and B. Manes.
**Tinylfu: A highly efficient cache admission policy.**
*CoRR*, abs/1512.00727, 2015.

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*. USENIX, 2003.

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.**
Wiley, 2005.