Efficient Modeling of Random Sampling-Based LRU

Junyao Yang, Yuchen Wang, Zhenlin Wang
{junyaoy, yuchenwa, zlwang}@mtu.edu

Michigan Technological University
Background - Caching

Caching Everywhere!

redis

DB

MEMCACHED

CPU

Cache

Main Memory
Background – Cache Performance Modeling

**Cache miss ratio is a big deal!**
A slight decrease -> huge end-to-end improvement

- **Miss Ratio Curve** relates miss ratio to cache size under a given replacement policy
  - A very powerful tool for dynamic cache memory management
Background
Random Sampling Based LRU (K-LRU)

Random sampling-based replacement uses random sampling for eviction
ex. Hyperbolic Caching(ATC’17), LHD(NSDI’18)

In memory cache like Redis uses K-LRU and K-LFU

**K-LRU:**
The cache randomly selects K objects,
then evicts the least recently used object among the K objects
Motivation K-LRU MRCs

Different sampling sizes exhibit large impact on the corresponding miss ratio curve.

Existing work only captures the LRU curve.

Goal: K-LRU MRCs
Background-MRC History

Mattson’s stack algorithm computes MRC in only one pass of the workload. However, the complexity is too high for actual use.

SHARDS and AET enable online LRU MRC construction in just linear time and constant space overhead.
Overview

Mattson’s general stack model

KRR: a probabilistic stack algorithm, approximates K-LRU

KRR + Spatially Hashed Random Sampling

Online K-LRU MRC Generation

acdefgh....acde...
aefh....ae...
Original Stack Algorithm

- Proposed by Mattson et al (1970)
- Simulates miss ratio curve for policies that satisfy inclusion* in just one pass
  - LRU, MRU, LFU, OPT ...
- The cost per stack update is $O(M)$

*Inclusion Property:
larger caches contain contents of smaller caches.
Miss Ratio Curve

- Stack Distance:
  - On reference to an object, the object’s position $\varphi$ is its **stack distance**

- Miss Ratio Curve:
  - Equivalent to stack distance distribution
Challenge

- K-LRU does not satisfy inclusion property
  - Larger caches not always contain contents of smaller cache

Solution:
- Define a new replacement policy (KRR)
  - which is a stack algorithm
  - approximates K-LRU well
**K-LRU vs. KRR**

Eviction probability of $i^{th}$ item = \( \frac{i^K-(i-1)^K}{C^K} \)

**K-LRU** : the item has $i^{th}$ recency ranking  

**KRR** : the item appears on the $i^{th}$ position of the stack
KRR Stack Update

If $s_{t-1}(i) \neq s_t(i)$, then the position $i$ is a swap position.

Probability of pushing $s_{t-1}(4)$ down the stack is $\frac{4^K - 3^K}{4^K}$. 

KRR stack update optimization

- Under KRR stack algorithm:
  - There are only $O(\log(\varphi))$ swap positions per update
- Instead of linearly scanning through entire stack
  - Just perform one way shifts on these swap positions
- **Problem:** How to identify those swap positions?
KRR: Top-Down Stack Update

- Asymptotic complexity per update:
  - $O(\log^2(\varphi))$
KRR: Backward Stack Update

- Based of eviction probability: 
  \[ \frac{i^K - (i-1)^K}{C^K}, \]  
  where \( i \) is the item's stack position, \( C \) is cache size.

- Then from the CDF of the eviction probability, swap positions can be computed backward from the hit position to stack top.
  - On each iteration, yields exactly one swap position.

- Asymptotic Complexity per update:
  - \( O(\log(\varphi)) \)
Lightweight Online KRR: Spatially Hashed Sampling

- Proposed by Waldspurger et al
  - SHARDS (FAST’15), Miniature Simulation (ATC’17)

*Flowchart*

- Reference L
- Hash(L) mod P < T
- Yes: KRR Stack
- No: Skip
- Scale up stack distance: SD*P/T

ICPP'2021 7/29/2021
Evaluation

- MSR – 13 Storage Traces, 1 merged master trace
- YCSB – 6 Traces
- Twitter – 4 traces

*See paper for detail description
K-LRU MRCs – K Sensitive Traces

[Graphs showing miss ratio vs cache size for different workloads and K values.]
K-LRU MRCs – K Insensitive Traces

msr_usr

ycsb_workload_C alpha=0.99

tw_cluster45.0

Miss Ratio

Cache Size (# of objects)

1e7

1e7

1e7

K = 1
K = 2
K = 4
K = 8
K = 16
K = 32
exact LRU
Accuracy

YCSB Workload E, alpha = 1.5

MSR src1

The average MAE across all

<table>
<thead>
<tr>
<th>Method</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRR</td>
<td>~0.001</td>
</tr>
<tr>
<td>KRR+Spatial</td>
<td>~0.003</td>
</tr>
</tbody>
</table>
Time & Space Overhead

<table>
<thead>
<tr>
<th>Combined MSR Trace w/ same sampling rate 0.001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
</tr>
<tr>
<td>Top Down+Spatial</td>
</tr>
<tr>
<td>Backward+Spatial</td>
</tr>
<tr>
<td>SHARDS</td>
</tr>
</tbody>
</table>

- When attached to Redis:
  - Time Overhead: ~0.1% total execution time.
  - Space Overhead: less than ~1MB
Conclusions

• KRR: A new probabilistic stack algorithm.
• KRR can construct K-LRU MRC for cache configured with arbitrary K in just one pass.
• Practical online K-LRU MRC generation with spatial sampling