ICPP’2021

Efficient Modeling of Random Sampling-Based LRU

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

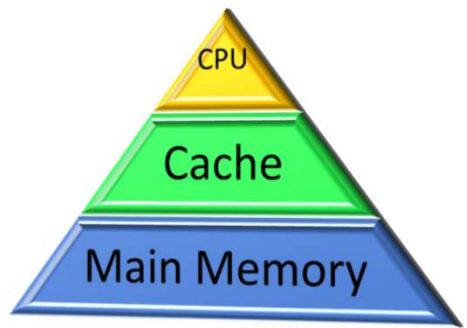
Michigan Technological University

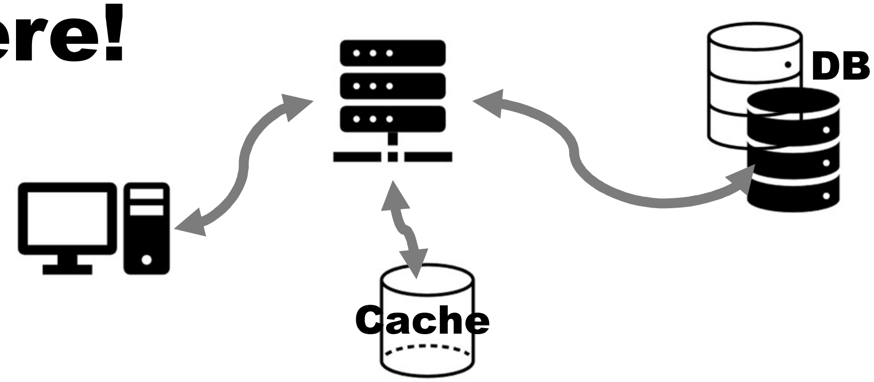
Background - Caching

redis

MEMCACHED

Caching Everywhere!

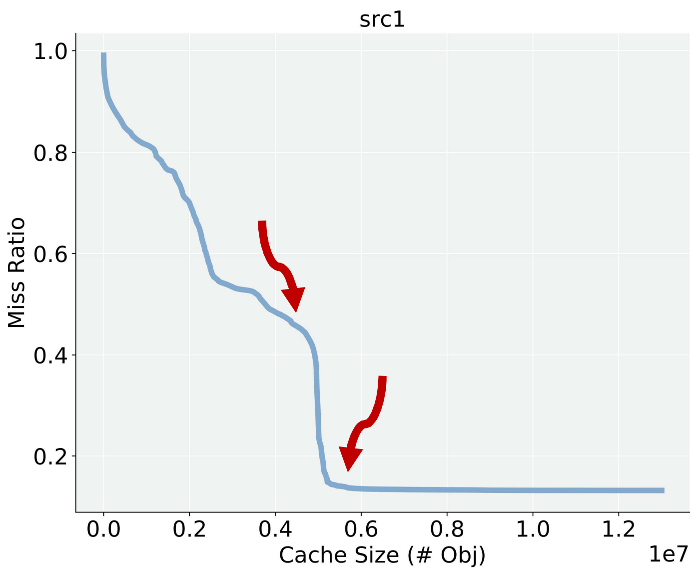




Background – Cache Performance Modeling

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

redis

K-LRU:

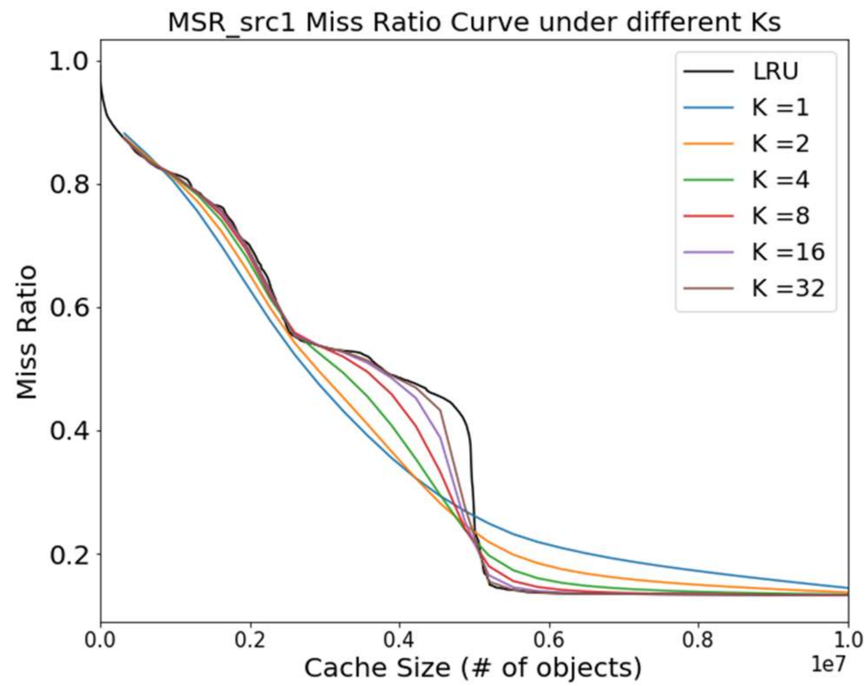
The cache randomly selects K objects,
then evicts the least recently used object among the K objects

Motivation K-LRU MRCs

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

Existing work only captures the LRU curve.

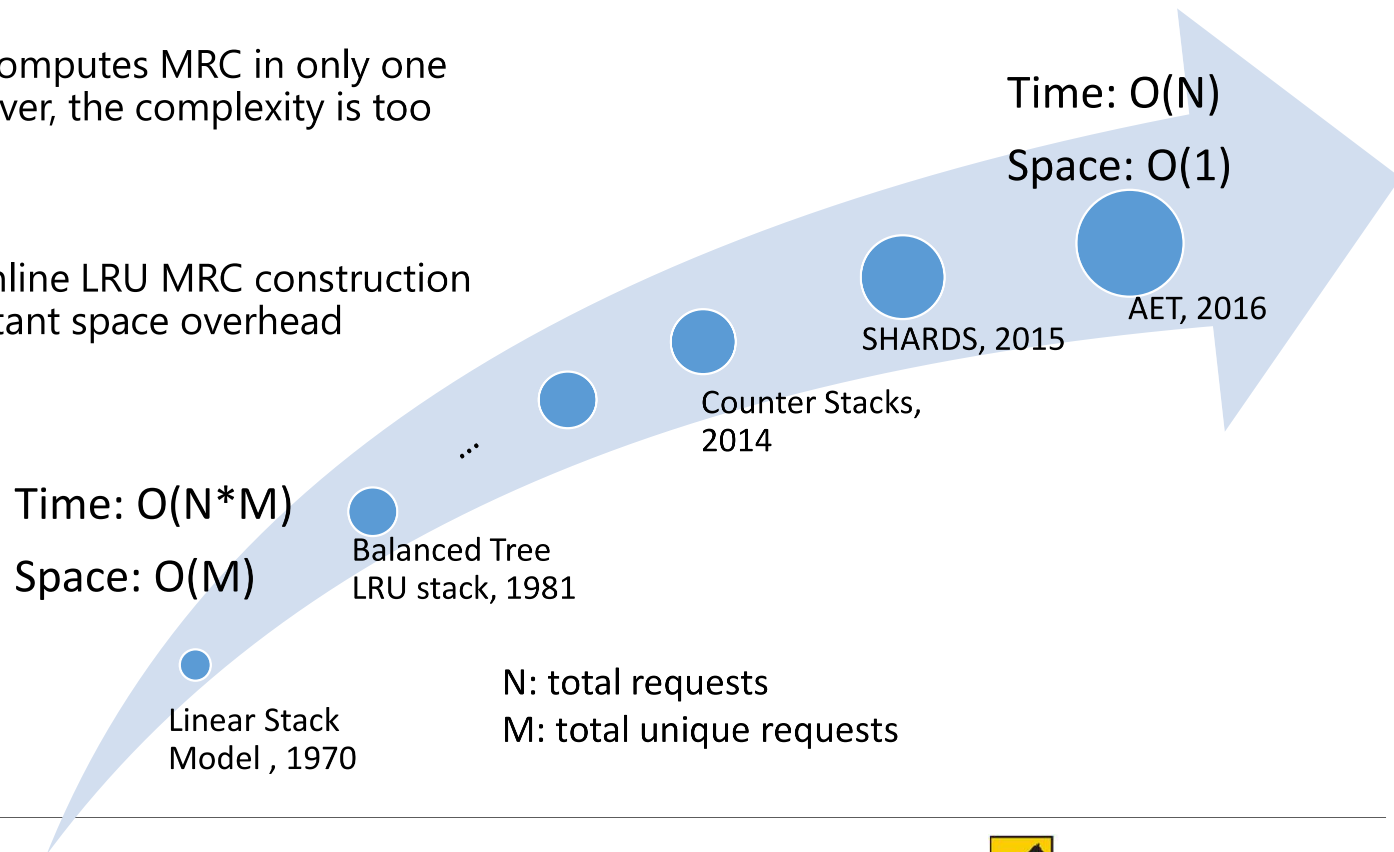
Goal: K-LRU MRCs



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



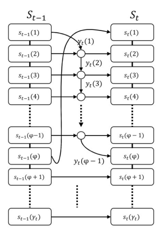
Overview

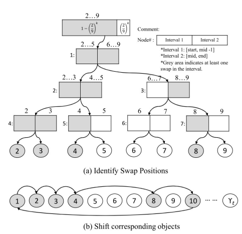
Mattson's general stack model

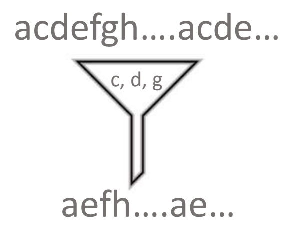
KRR: a probabilistic stack algorithm, approximates K-LRU

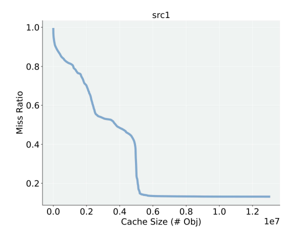
KRR + Spatially Hashed Random Sampling

Online K-LRU MRC Generation



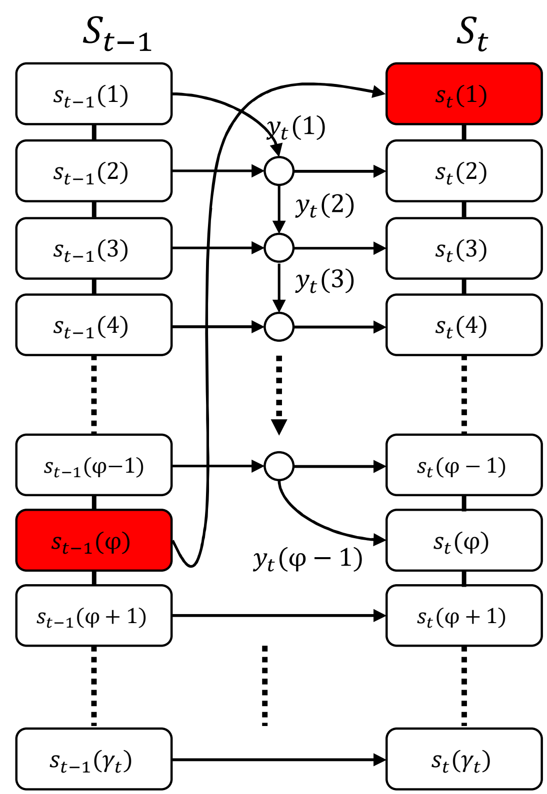






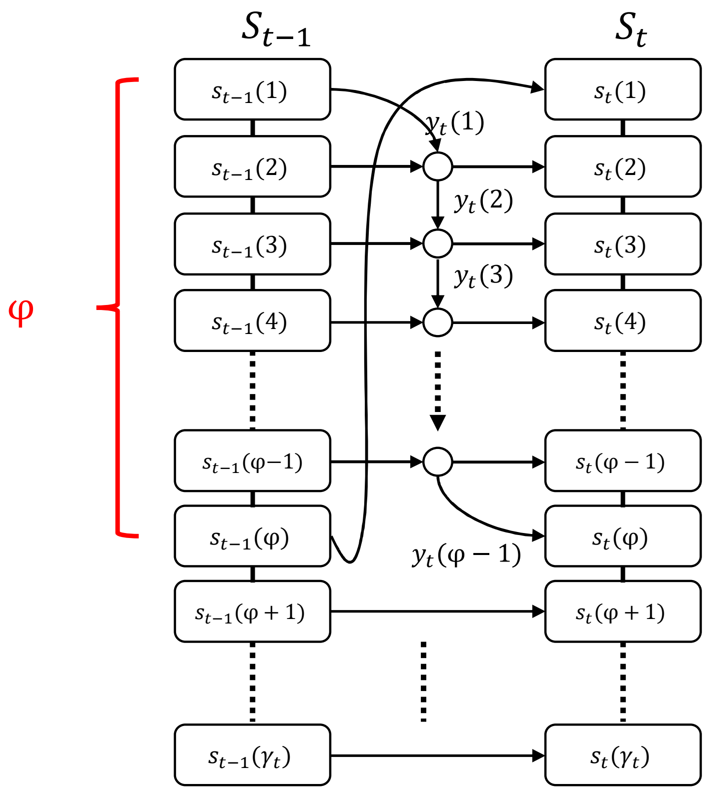
Original Stack Algorithm

- 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



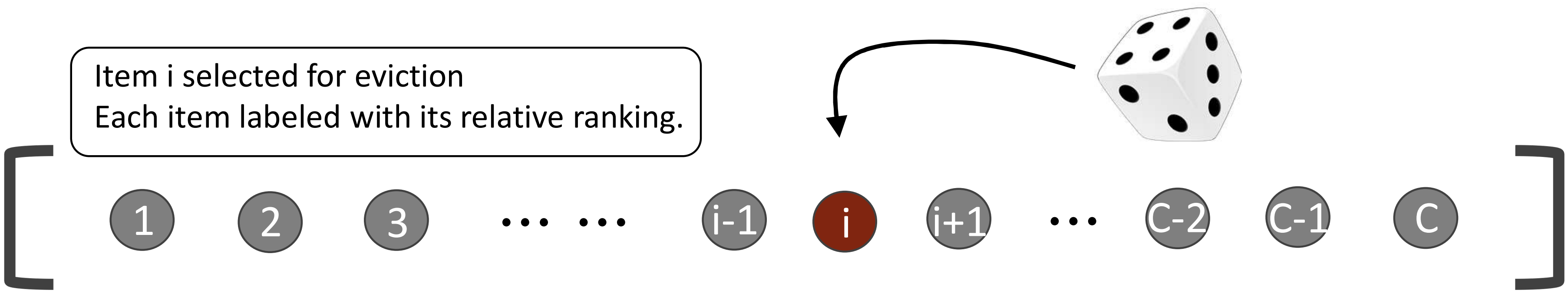
- Stack Distance:
  - On reference to an object, the object's position **φ** is its **stack distance**
- Miss Ratio Curve:
  - Equivalent to stack distance distribution

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

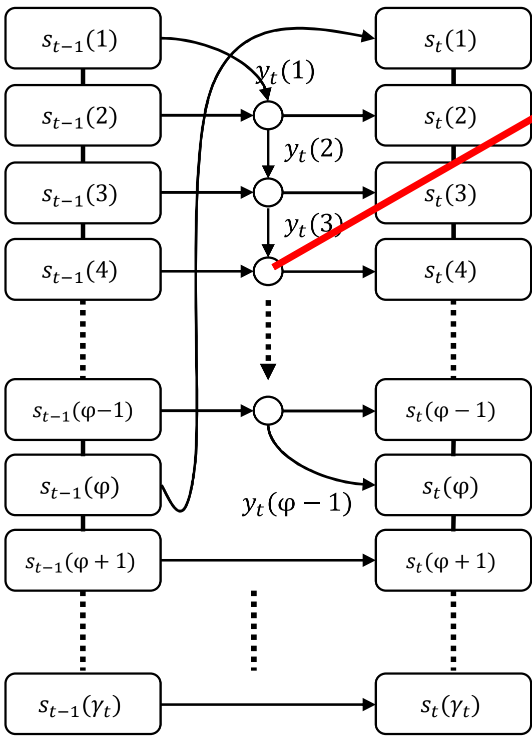


Eviction probability of i^{th} item = $\frac{i^K-(i-1)^K}{C^K}$

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

KRR : the item appears on the i^{th} position of the stack

KRR Stack Update



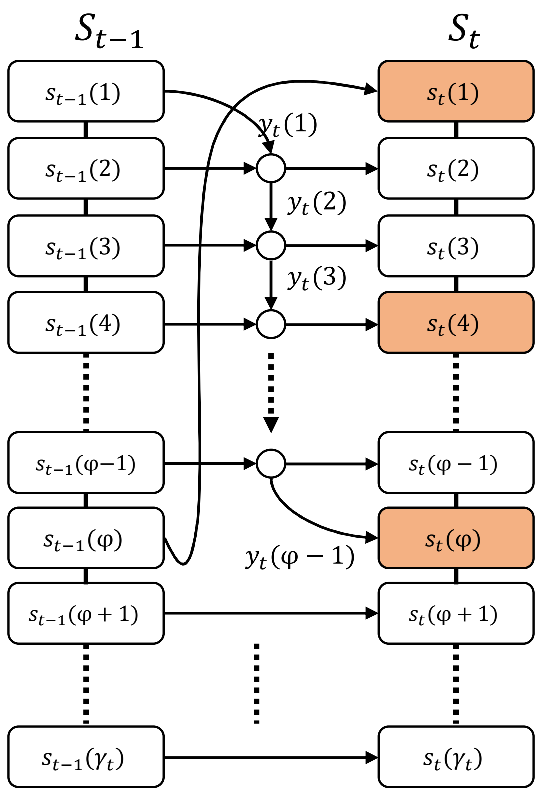
Probability of pushing $s_{t-1}(4)$

down the stack is $\dfrac{4^K - 3^K}{4^K}$

If $s_{t-1}(i) \neq s_t(i)$,

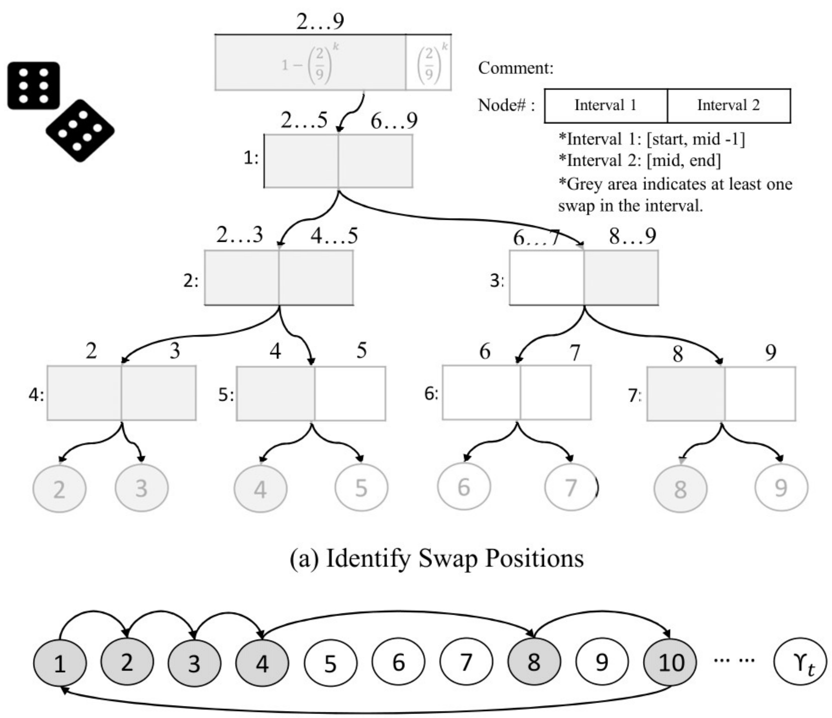
then the position $\boldsymbol{i}$ is a **swap position.**

KRR stack update optimization



- Under KRR stack algorithm:
  - There are only **O(Log(φ))** 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



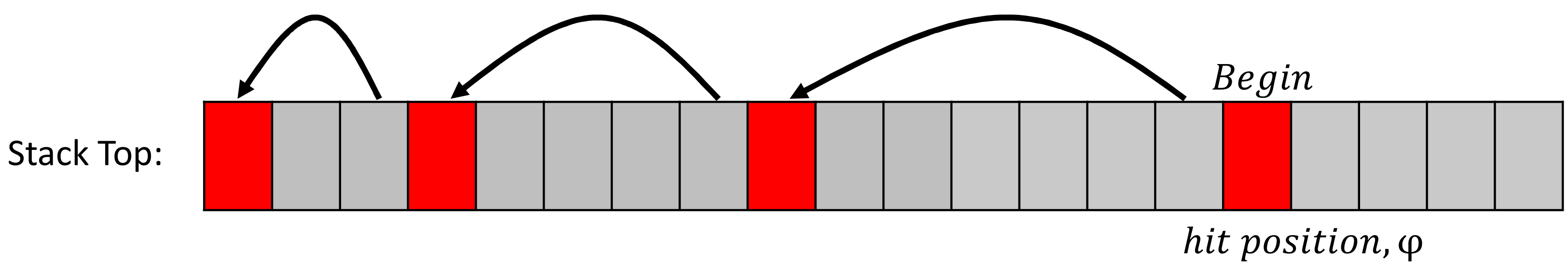
(a) Identify Swap Positions

(b) Shift corresponding objects

- Asymptotic complexity per update:
  - **O(Log^2(φ))**

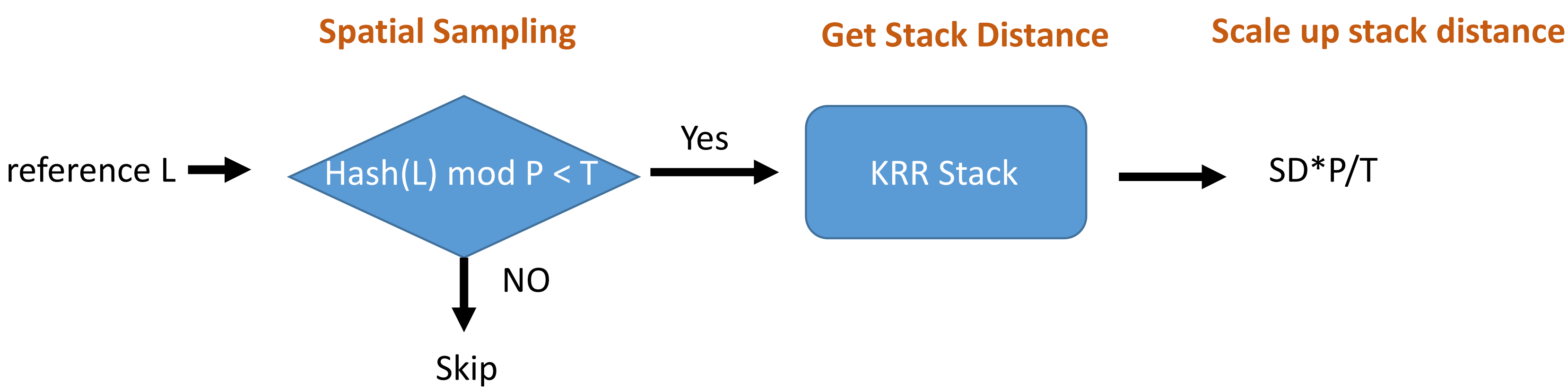
KRR: Backward Stack Update

- 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(φ))**



Lightweight Online KRR: Spatially Hashed Sampling

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

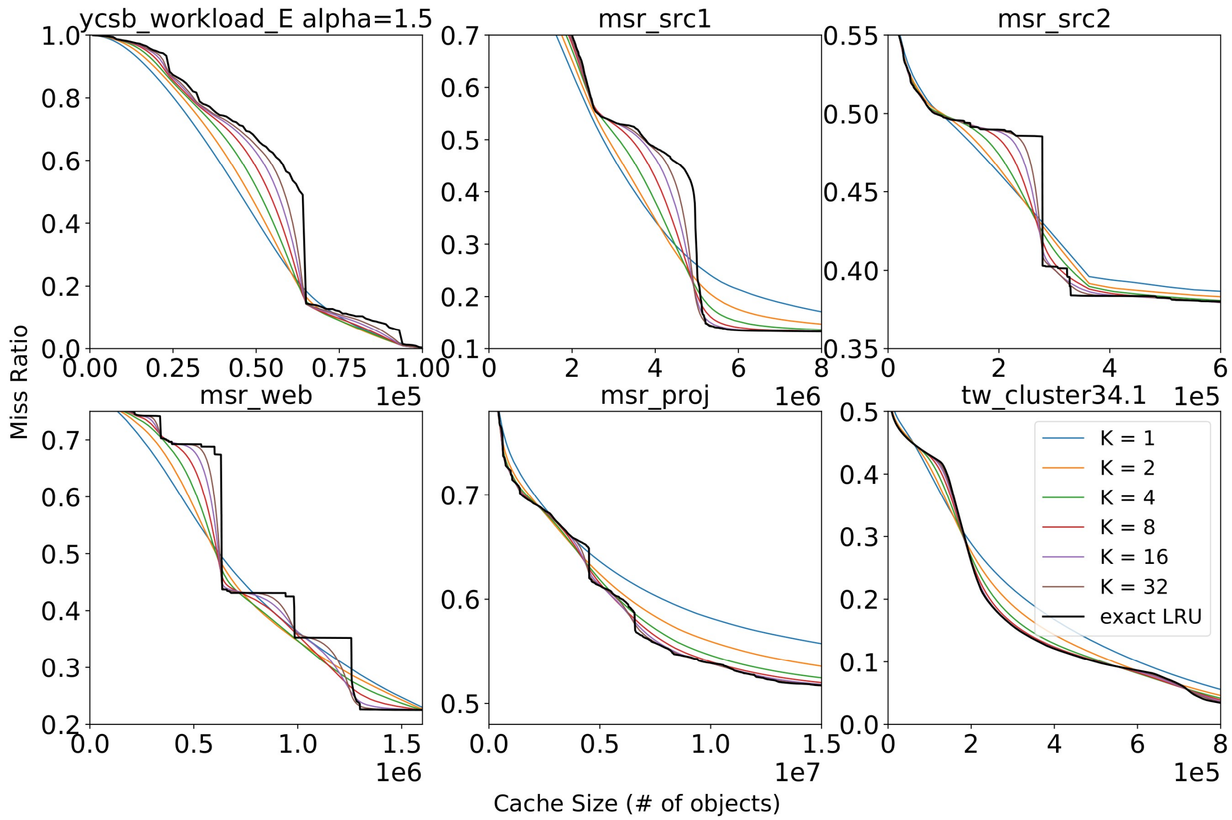


Evaluation

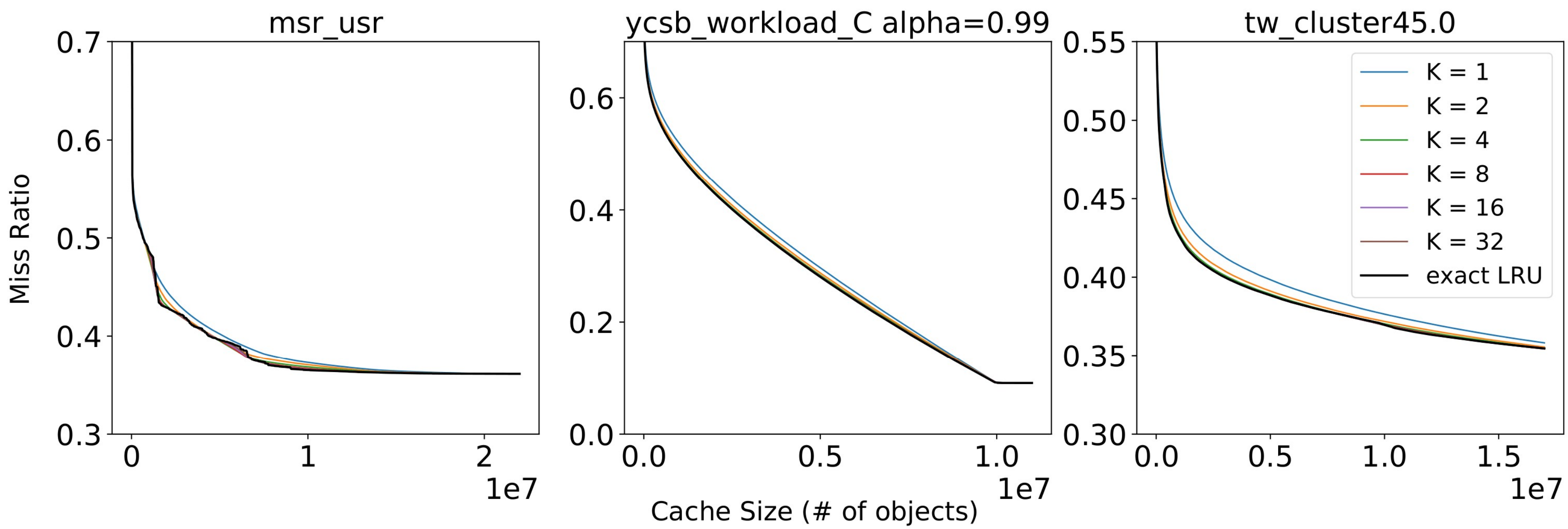
- MSR – 13 Storage Traces, 1 merged master trace
- YCSB – 6 Traces
- Twitter – 4 traces

*See paper for detail description

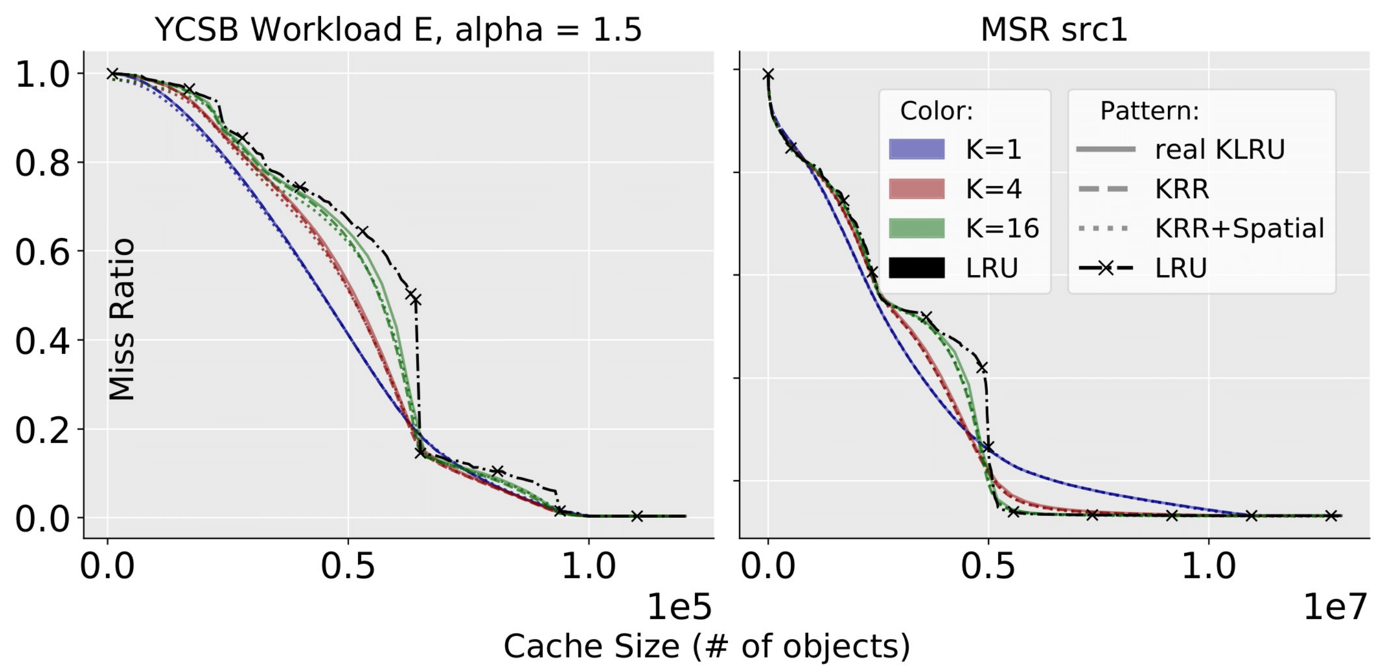
K-LRU MRCs – K Sensitive Traces



K-LRU MRCs – K Insensitive Traces



Accuracy



The average MAE across all

	MAE
KRR	~0.001
KRR+Spatial	~0.003

Time & Space Overhead

Combined MSR Trace w/ same sampling rate 0.001	
Method	Time overhead (sec)
Top Down+Spatial	39.1
Backward+Spatial	22.4
SHARDS	19.7

- 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