SHARDS & Talus: Online MRC estimation and optimization for very large caches

Nohhyun Park

CloudPhysics, Inc.
Introduction

• *Efficient MRC Construction with SHARDS* – FAST’15 Waldspurger at al.
• *Talus: A simple way to remove cliffs in cache performance* – HPCA’15 Beckmann and Sanchez

• Two complementary techniques that improves cache performance
• Both techniques rely on same finding.
SHARDS

Efficient MRC Construction with SHARDS
Modeling Cache Performance

- Miss Ratio Curve (MRC)
  - Performance as $f(size)$
  - Working set knees
  - Inform allocation policy

- Reuse distance
  - Unique intervening blocks between use and reuse
  - LRU, stack algorithms
Motivation

• Cache partitioning.

• Simulation of various cache parameters.
  • Cache block size, write handling, shadow partition

• Workload partitioning.
  • By IO meta information (IO size, filesystem info, etc.)

• Problem: requires online modeling expensive
  • Too resource-intensive to be broadly practical
  • Exacerbated by increasing cache sizes
MRC Algorithm Research

- **Bennett & Kruskal balanced tree**: $O(N)$, $O(N \log N)$
- **Mattson Stack Algorithm single pass**: $O(M)$, $O(NM)$
- **Olken tree of unique refs**: $O(M)$, $O(N \log M)$
- **Kessler, Hill & Wood set, time sampling**
- **UMON-DSS hw set sampling**
- **PARDA parallelism**
- **SHARDS spatial hashing**: $O(1)$, $O(N)$
- **Bryan & Conte cluster sampling**
- **Counter Stacks probabilistic counters**: $O(\log M)$, $O(N \log M)$
- **RapidMRC on-off periods**

**Space, Time Complexity**

$N =$ total refs, $M =$ unique refs
Key Idea

• Random spatial sampling results in a similar MRC scaled by the sampling rate.
Spatially Hashed Sampling

\[ \text{hash}(L_i) \mod P \]

\[ L_i \rightarrow T \]

randomize

sample?

\[ < T \]

process

\[ \text{yes} \]

\[ \text{no} \]

skip

sampling rate \( R = \frac{T}{P} \)

subset inclusion property maintained as \( R \) is lowered

adjutable threshold

sampled

unsampled
Each sample statistically represents $1/R$ blocks
Scale up reuse distances by same factor

$\text{hash}(L_i) \mod P \rightarrow \begin{array}{c} \text{randomize} \\ \text{sample?} \end{array} \rightarrow \begin{array}{c} \text{compute distance} \\ \text{scale up} \end{array}$

$T_i < T \quad \text{yes} \rightarrow \text{Standard Reuse Distance Algorithm} \rightarrow \div R$

$\text{no} \rightarrow \text{skip}$
SHARDS in Constant Space

randomize → sample? → compute distance → scale up

$L_i \rightarrow T_i \rightarrow \text{hash}(L_i) \mod P$

$\text{hash}(L_i) \mod P < T \rightarrow \text{yes}$

Standard Reuse Distance Algorithm

Evict samples to bound set size

$T_{\text{max}}$

Lower threshold $T = T_{\text{max}}$

Reduces rate $R = T / P$
Example SHARDS MRCs

- Block I/O trace *t04*
  - Production VM disk
  - 69.5M refs, 5.2M unique

- Sample size $s_{max}$
  - Vary from 128 to 32K
  - $s_{max} \geq 2K$ very accurate

- Small constant footprint
- SHARDS$_{adj}$ adjustment
Experimental Evaluation

- Data collection
  - SaaS caching analytics
  - Remotely stream VMware vsckiStats
- 124 trace files
  - 106 week-long traces CloudPhysics customers
  - 12 MSR and 6 FIU traces SNIA IOTTA
- LRU, 16 KB block size
Exact MRCs vs. SHARDS
Error Analysis

- Mean Absolute Error (MAE)
  - \(|\text{exact} - \text{approx}|\)
  - Average over all cache sizes
- Full set of 124 traces
- Error \( \propto \frac{1}{\sqrt{s_{\text{max}}}} \)
- MAE for \( s_{\text{max}} = 8K \)
  - 0.0027 median
  - 0.0171 worst-case
Memory Footprint

- Full set of 124 traces
- Sequential PARDA
- Basic SHARDS
  - Modified PARDA
  - Memory $\approx R \times$ baseline for larger traces
- Fixed-size SHARDS
  - New space-efficient code
  - Constant 1 MB footprint
Processing Time

- Full set of 124 traces
- Sequential PARDA
- Basic SHARDS
  - Modified PARDA
  - R=0.001 speedup 41–1029×
- Fixed-size SHARDS
  - New space-efficient code
  - Overhead for evictions
  - $S_{\text{max}} = 8K$ speedup 6–204×
Generalizing to Non-LRU Policies

• Many sophisticated replacement policies
  • ARC, LIRS, CAR, CLOCK-Pro, ...
  • Adaptive, frequency and recency
  • No known single-pass MRC methods!

• Solution: efficient scaled-down simulation
  • Filter using spatially hashed sampling
  • Scale down simulated cache size by sampling rate
  • Run full simulation at each cache size

• Surprisingly accurate results
Scaled-Down Simulation Examples

**ARC — MSR-Web Trace**

**CLOCK-Pro — Trace t04**
Conclusions

• New SHARDS algorithm
  • Approximate MRC in O(1) space, O(N) time
  • Excellent accuracy in 1 MB footprint

• Practical online MRCs
  • Even for memory-constrained drivers, firmware
  • So lightweight, can run multiple instances

• Scaled-down simulation of non-LRU policies
Talus

A simple way to remove cliffs in cache performance
Key Idea

• Random spatial sampling results in a similar MRC scaled by the sampling rate.
Shards and Talus

• One way to think about SHARDS is that it simulates N size cache using N/r size cache with sampling rate of r.

• If we use N/r size cache with sampling rate of r’ where r’ < r, than the effective cache size increases. If r’ > r than the effective cache size decreases.

• If a knee in the MRC curve does not fit the cache size, we can fit it by increasing the effective cache size.
Talus
Talus Property

• Can make ANY MRC curve to follow the convex hull of the original MRC.

• With SHARDS, the overhead is fairly small.

• All resulting MRC is convex.
Talus insight.

- 0 hits until the cache size is big enough to fit the entire workload.
Talus insight.

- 0 hits until the cache size is big enough to fit the entire workload.
- We can reduce the miss rate by 50% by feeding only the 50% of the addresses to the cache.
Talus insight.

- 0 hits until the cache size is big enough to fit the entire workload.
- We can reduce the miss rate by 50% by feeding only the 50% of the addresses to the cache.
- By repeating the experiment for all cache sizes, we can verify that it forms the convex hull of the MRC.
Talus results
SHARDS + Talus

• Use 1MB for the MRC prediction for stack algorithms like LRU.

• Use 32MB for the MRC prediction for other caching algorithms.
  • With 32 SHARDS.

• Calculate Convex hull.

• Apply Talus.

• Less than 0.01% overhead.
Benefits of SHARDS + Talus

- Removes the cliffs.
- Resulting MRC is convex – partitioning problem is now greedy.
- Very low cost.
  - SHARDS capacity also serves actual cache request.
- Seems to work with any caching algorithm.
- Convex hull is fairly stable over time.
Conclusion

• Online generation of multiple MRCs for very large caches is possible.
  • Using fixed memory cost.
  • Low CPU cost.
  • Using different parameters.

• MRC driven QoS.
  • Control average latency via miss rate control

• Larger effective cache size via Talus.
  • Comes for almost free with SHARDS.
Q & A
Carl Waldspurger: carl@cloudphysics.com
Alex Garthwaite: alex@cloudphysics.com
Irfan Ahmed: irfan@cloudphysics.com
Nohhyun Park: nohhyun@cloudphysics.com

Thank you!