Efficient MRC Construction with SHARDS

Carl Waldspurger       Nohhyun Park
Alexander Garthwaite   Irfan Ahmad

CloudPhysics, Inc.

USENIX Conference on File and Storage Technologies
February 17, 2015
Motivation

• Cache performance highly non-linear
• Benefit varies widely by workload
• Opportunity: dynamic cache management
  – Efficient sizing, allocation, and scheduling
  – Improve performance, isolation, QoS
• Problem: online modeling expensive
  – Too resource-intensive to be broadly practical
  – Exacerbated by increasing cache sizes
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
MRC Algorithm Research

Mattson Stack Algorithm
single pass
\(O(M), O(NM)\)

Bennett & Kruskal
balanced tree
\(O(N), O(N \log N)\)

Kessler, Hill & Wood
set, time sampling
\(O(M), O(N \log M)\)

Olken
tree of unique refs
\(O(M), O(N \log M)\)

UMON-DSS
hw set sampling

PARDA
parallelism

SHARDS
spatial hashing
\(O(1), O(N)\)

Bryan & Conte
cluster sampling

RapidMRC
on-off periods
\(O(\log M), O(N \log M)\)

Space, Time Complexity
\(N = \text{total refs}, M = \text{unique refs}\)
Key Idea

• Track only a *small subset* of blocks
  – Filter input to existing algorithm
  – Run *full* algorithm, using only sampled blocks
  – Cheap/accurate enough for practical online MRCs?

• SHARDS approximation algorithm
  – Randomized spatial sampling
  – Uses hashing to capture all reuses of same block
  – High performance in tiny constant footprint
  – Surprisingly accurate MRCs
Spa0ally	
  Hashed	
  Sampling

USENIX	
  FAST	
  '15

L_i → hash(L_i) mod P → T_i

randomize

sample?

< T → process

no → skip

yes

sampling rate R = T / P

subset inclusion property maintained as R is lowered

CloudPhysics, Inc.

USENIX FAST '15

6
Basic SHARDS

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

randomize  sample?  compute distance  scale up

$L_i \rightarrow \text{hash}(L_i) \mod P \rightarrow T_i < T \rightarrow \text{yes} \rightarrow \text{Standard Reuse Distance Algorithm} \rightarrow \div R$

$\rightarrow \text{no} \rightarrow \text{skip}$

CloudPhysics, Inc.
USENIX FAST '15
**SHARDS in Constant Space**

1. **randomize**
   - $L_i \rightarrow T_i$
   - hash($L_i$) mod $P$

2. **sample?**
   - $< T$
   - Standard Reuse Distance Algorithm
   - yes -> scale up
   - sample set

3. **compute distance**
   - $\div R$

4. **scale up**

---

**evict samples to bound set size**

- lower threshold $T = T_{\text{max}}$
- reduces rate $R = T / P$

---

CloudPhysics, Inc.  USENIX FAST '15  8
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\textsubscript{adj} adjustment
Dynamic Rate Adaptation

- Adjust sampling rate
  - Start with $R = 0.1$
  - Lower $R$ as $M$ increases
  - Shape depends on trace

- Rescale histogram counts
  - Discount evicted samples
  - Correct relative weighting
  - Scale by $R_{new} / R_{old}$
Experimental Evaluation

- Data collection
  - SaaS caching analytics
  - Remotely stream VMware vscsiStats

- 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

CloudPhysics, Inc.

USENIX FAST '15
Error Analysis

- Mean Absolute Error (MAE)
  - $|\text{exact} - \text{approx}|$
  - Average over all cache sizes
- Full set of 124 traces
- Error $\propto 1 / \sqrt{s_{\text{max}}}$
- 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 \( \approx R \times \text{baseline} \) 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 $41$–$1029\times$
- Fixed-size SHARDS
  - New space-efficient code
  - Overhead for evictions
  - $S_{\text{max}} = 8K$ speedup $6$–$204\times$
# Counter Stacks Comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory (MB)</th>
<th>Throughput (Mrefs/sec)</th>
<th>Error (MAE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter Stacks</td>
<td>80.0</td>
<td>2.3</td>
<td>0.0025</td>
</tr>
<tr>
<td>SHARDS $S_{\text{max}}=32K$</td>
<td>2.0</td>
<td>16.9</td>
<td>0.0026</td>
</tr>
<tr>
<td>SHARDS $S_{\text{max}}=8K$</td>
<td>1.3</td>
<td>17.6</td>
<td>0.0061</td>
</tr>
</tbody>
</table>

- **Quantitative**
  - Same merged MSR “master” trace
  - Counter Stacks roughly 7× slower, 40–62× bigger

- **Qualitative**
  - Counter Stacks checkpoints support splicing/merging
  - SHARDS maintains block ids, generalizes to non-LRU
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
Questions?

• {carl,nohhyun,alex,irfan}@cloudphysics.com
• Visit our poster
• BoF 9-10pm tonight in Bayshore West
• Potential academic and industry collaboration
• Application areas include capacity planning, dynamic partitioning, tuning, policies, ...
• We’re also hiring!