Kinetic Modeling of Data Eviction in Cache

Xiameng Hu, Xiaolin Wang, Lan Zhou, Yingwei Luo
Peking University
Chen Ding
University of Rochester
Zhenlin Wang
Michigan Technological University
Background

• Miss Ratio Curve (MRC) is a powerful metric for cache optimization:
  • Allocation, Partition, Scheduling, QoS managing...

• Online MRC profiling techniques have been developed for decades.

• Ultimate goals:
  • Less space consumption.
  • Lower time complexity.
Background

• A brief history of MRC techniques.
Our Model: Average Eviction Time

• Linear time
• Constant space
• Composability
Eviction Time

<table>
<thead>
<tr>
<th>time</th>
<th>data</th>
<th>MRU</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>a-b-c-e</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>d</td>
<td>d-a-b-c</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>a-d-b-c</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>d-a-b-c</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>a-d-b-c</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>c-a-d-b</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>b</td>
<td>b-c-a-d</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>e</td>
<td>e-b-c-a</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>d</td>
<td>d-e-b-c</td>
<td></td>
</tr>
</tbody>
</table>

1st access

Eviction time

2nd access

Residence time

Eviction

3rd access
Eviction Time

• The *eviction time* is the time between the last access and the eviction.

• Property of eviction time:
  • If the *reuse time* of an access is larger than its *eviction time*, it’s a miss.
  • *Reuse time*: the time between an access and its next reuse. The reuse time of cold miss is defined as infinite.
Back to the example

<table>
<thead>
<tr>
<th>time</th>
<th>data</th>
<th>MRU</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>d</td>
<td>d</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>d</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>7</td>
<td>b</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>8</td>
<td>e</td>
<td>e</td>
<td>b</td>
</tr>
<tr>
<td>9</td>
<td>d</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Reuse time = \( \infty \)  
Miss!

Reuse time = 2  
Hit!

Eviction time = 4

Reuse time = 5  
Miss!
Average Eviction Time

- *Average Eviction Time* (AET) is the mean eviction time of all data evictions in a fully associative LRU cache.
- We can assume all data references with a reuse time larger than AET are misses.
How to model AET?

- Move condition #1:
  - Cache hit inserts the *lower priority position* data to the LRU stack top.

```
   MRU   LRU
     d    a    b    c
   access: a
     a    d    b    c
```

- Move condition #2:
  - Cache miss inserts a *missed* data to the LRU stack top.

```
   MRU   LRU
     d    a    b    c
   access: e
     e    d    a    b
```
How to model AET?

- Stay condition:
  - Cache hit inserts the *higher priority position* data to the LRU stack top.

```
MRU     LRU
a       d       c
b       d       

access: b
```

```
  b     a
  d     c
```
How to model AET?

- We define the *arrival time* $T_m$ as the expected time it takes for an evicting data to reach the $m$-th position (from its last access).
- A data block at position $m$ move one step down whenever the reuse time of current access is greater than the $T_m$.
- $P(t)$ is the probability for an access with a reuse time greater than $t$.
- The movement condition is now a probability. Every access, a data block at stack position $m$ moves by $P(T_m)$. 
Kinetic Model

- Data travels in one direction with changing speed:

\[ V(t) = P(t) \]

- In general, if the time that evicting data already traveled is \( t \), its’ current evicting speed is \( P(t) \).
Average Eviction Time

- **Physics**: the integration of speed over time is travel distance.
- The length of LRU list is the travel distance of every eviction. Which is the cache size $c$.

\[
\int_0^{AET(c)} P(t)dt = c
\]

- With $P$, we calculate AETs of different cache sizes in linear time.
- $P$ can be acquired online by monitoring the *reuse time histogram*. 
From AET to MRC

• The miss ratio $mr(c)$ at cache size $c$ is the probability that a reuse time is greater than the average eviction time $AET(c)$:

$$mr(c) = P(AET(c))$$
AET Design Overview

- Program Monitoring
- Access Trace
- Reuse Time Histogram
- AET
- Miss Ratio Curve
Random Sampling

- Randomly pick current accessed data to monitor its reuse time.
- The distance between two sampled is a random value.
- Constrain the random value range to control sampling rate.
- A hash table is required. It maintains current monitored data.
- The space consumption is linear but limited.
Reservoir Sampling

• To bound the space cost to constant. $O(1)$
• When the $i$-th sampled data arrives, reservoir sampling keeps the new data in monitoring set with probability $\min(1, k/i)$ and randomly discards an old data when the set is full.
• It ensures the equal probability for every sampled reuse to update reuse time histogram.
• While the number of samples be recorded is bounded.
AET in Shared Cache

- Composability: co-run behavior can be computed from the metric of solo-runs.
- When $n$ programs share the cache of size $c$, all $n + 1$ co-run $AET$s, $AET_i(c)$ for each program $i$ and $AET(c)$ for the group, are the same:

$$AET_1(c) = AET_2(c) = \cdots = AET_n(c) = AET(c)$$

- Detailed modeling is described in paper.
Evaluation

• AET vs Counter Stacks (OSDI’14)
• AET vs SHARDS (FAST’15)
• Shared Cache AET
AET vs Counter Stacks

• Counter Stacks:
  • Only requires extremely small space while maintaining an acceptable accuracy.
  • HyperLogLog counter to track reuse distance.
  • Balance accuracy and space by limiting the number of counters.

• Benchmarks:
  • Microsoft Research Cambridge (MSR) storage traces.
  • Configured with only read requests of 4KB cache blocks.
AET vs Counter Stacks
# AET vs Counter Stacks

<table>
<thead>
<tr>
<th></th>
<th>AET Random Sampling ($1 \times 10^{-5}$)</th>
<th>AET Reservoir Sampling 8k entries</th>
<th>Counter Stacks High fidelity ($d = 1M, s = 60, \delta = 0.02$)</th>
<th>Counter Stacks Low fidelity ($d = 1M, s = 3600, \delta = 0.1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Absolute Error</td>
<td>0.96%</td>
<td>1.12%</td>
<td>0.77%</td>
<td>1.26%</td>
</tr>
<tr>
<td>Average Space Cost</td>
<td>452KB</td>
<td>384KB</td>
<td>7363KB</td>
<td>1292KB</td>
</tr>
<tr>
<td>Average Throughput</td>
<td>63.99M reqs/sec</td>
<td>61.99M reqs/sec</td>
<td>1.73M reqs/sec</td>
<td>5.86M reqs/sec</td>
</tr>
</tbody>
</table>
AET vs SHARDS

- **SHARDS:**
  - hash-based spatial sampling
  - a splay tree to track the reuse distances of the sampled data.
  - Limits the space overhead to a constant by adaptively lowering the sampling rate.

- **Benchmarks:**
  - “master” MSR, which is a 2.4 billion-access trace combining all 13 MSR traces by ranking the time stamps of all accesses.
AET vs SHARDS

![Graph showing miss ratio vs cache size for Real, SHARDS, and AET models.](image)
## AET vs SHARDS

<table>
<thead>
<tr>
<th></th>
<th>AET Random Sampling ((1 \times 10^{-5}))</th>
<th>AET Reservoir Sampling 8k samples</th>
<th>SHARDS 8k samples</th>
<th>Counter Stacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Absolute Error</td>
<td>1%</td>
<td>1%</td>
<td>0.6%</td>
<td>0.3%</td>
</tr>
<tr>
<td>Average Space Cost</td>
<td>1.7MB</td>
<td>1.4MB</td>
<td>2.3MB</td>
<td>80MB</td>
</tr>
<tr>
<td>Average Throughput</td>
<td>79M reqs/sec</td>
<td>66.6M reqs/sec</td>
<td>81.4M reqs/sec</td>
<td>3.2M reqs/sec</td>
</tr>
</tbody>
</table>
Shared Cache AET

• We choose Four MSR storage traces \{prn, src2, web, stg\} as a co-run group.
• Generate a combined trace from the four traces under equal speed assumption.
• We compare MRC composed by individual AET modeling of each trace, as well as the real MRC of the combined trace.
Shared Cache AET

![Graph showing the relationship between cache size (GB) and miss ratio. The graph includes lines for Shared AET, Real, prn, web, src2, and stg, indicating the performance of these categories across different cache sizes.](image)

6/29/16

Usenix ATC'16
Summary

• A new model to characterize cache behavior.
  • Enable fast MRC profiling with O(1) space and O(n) time.
  • Predict shared cache MRC without co-run testing.
  • Perfect for online deployment with limited overhead.

<table>
<thead>
<tr>
<th>Model</th>
<th>Time complexity</th>
<th>Space complexity</th>
<th>Memory</th>
<th>Runtime</th>
<th>Composability</th>
<th>Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Processing</td>
<td>$O(NM)$</td>
<td>$O(N)$</td>
<td>10GB</td>
<td>&gt; 1 day</td>
<td>No</td>
<td>accurate</td>
</tr>
<tr>
<td>Search Tree</td>
<td>$O(N \log M)$</td>
<td>$O(M)$</td>
<td>21GB</td>
<td>482 secs</td>
<td>No</td>
<td>accurate</td>
</tr>
<tr>
<td>Scale Tree</td>
<td>$O(N \log \log M)$</td>
<td>$O(M)$</td>
<td>17GB</td>
<td>333 secs</td>
<td>No</td>
<td>bounded err</td>
</tr>
<tr>
<td>Footprint</td>
<td>$O(N)$</td>
<td>$O(M)$</td>
<td>17GB</td>
<td>50 secs</td>
<td>Yes</td>
<td>conditional</td>
</tr>
<tr>
<td>Counter Stacks</td>
<td>$O(N \log M)$</td>
<td>$O(\log M)$</td>
<td>80MB</td>
<td>1034 secs</td>
<td>No</td>
<td>bounded err</td>
</tr>
<tr>
<td>SHARDS</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>2.3MB</td>
<td>29.6 secs</td>
<td>No</td>
<td>conditional</td>
</tr>
<tr>
<td>AET model</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>1.7MB</td>
<td>30.5 secs</td>
<td>Yes</td>
<td>conditional</td>
</tr>
</tbody>
</table>
Thank you for your attention!

Q&A

Email: hxm@pku.edu.cn
AET vs StatStack

• StatStack:
  • Designed for CPU workloads.
  • It samples cache blocks and measures their reuse time using performance counters and watchpoints.
  • Reuse time histogram -> Reuse distance histogram.

• Benchmarks:
  • SPEC CPU2006, 30 benchmarks.
  • For each benchmark, we intercept 1 billion references from their execution using the instrumentation tool Pin.
  • We measure the cumulative distribution function (CDF) of absolute error of full-trace StatStack, full-trace AET, sampling AET.
AET vs StatStack

The diagram shows a comparison between AET and StatStack, with the x-axis representing the prediction error and the y-axis representing the percentage. The graph includes lines for AET, StatStack, 1%, and 0.01% prediction error rates, demonstrating the performance of each method across different error levels.