Introduction to (Software) Caching

Disclaimer: in this lecture I will be much less precise and detailed than in the main parts of the course
The Essence of Caching

• A **fast** but relatively **small** memory

• Can temporarily store some items of the “real storage”

• Improves performance if most accesses are a **hit**
Cache Management

- When an item $x$ “arrives” and the cache is full
  - **Eviction policy:** Who should we throw out to make room in the cache? ("eviction victim")
  - **Admission policy:** Does it make sense to store $x$ in the cache instead of the eviction victim?
Looking for Hints in the Workload

- We need to decide about something that will happen in the future
  - But all we have is knowledge about the past
- Two common indicators
  - Recency
    - An item that was recently accessed is likely to be accessed again soon
  - Frequency
    - A popular item is likely to be accessed again soon
Least Recently Used (LRU)

- The eviction victim is the *least recently accessed* item in the cache.
- Often maintained as a FIFO queue, but when there is a hit, move accessed item to head of the queue.
- Builds on the *principle of locality*.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>D</th>
<th>B</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>
Least Frequently Used (LFU)

- The eviction victim is the *least frequently accessed* item in the cache
- Meta-data includes a frequency counter for each item
  - Often maintained as a heap + hash-map for fast access
Aging LFU

• Problem:
  ▫ Items’ popularity often changes over time
    • How many times have you listened to “Mesiba be-Haifa” during the last month? How about 6 months ago?

• Solutions:
  ▫ A sliding window
  ▫ Exponential decay
  ▫ Probabilistic tracking (new)
Workloads Vary

- Internet services tend to be frequency biased

- Some (local) storage workloads are recency biased while others are more frequency biased
Adaptive Replacement Cache - ARC

- An IBM patent

- $|T_1+T_2| = \text{constant (actual cache size)}$
- Intuitively, dynamically adjust between LRU-like and LFU-like according to the workload
  - If a cache miss is a hit in B1, the marker moves right
  - If a cache miss is a hit in B2, the marker moves left
  - Eviction is from the segment that is larger than the marker position
Low Inter-reference Recency Set - LIRS

- Maintains two lists (actually FIFO queues):
  1. For frequent items
     - But expressed through inter-reference count
     - All items whose IRR is below a threshold
       - Including their ghost entries if evicted until enough items are inserted there to evict them from the IRR list
     - Has a maximum size
  2. For non frequent (but reasonably recent) items
     - LRU policy
     - Only for resident items
- Evicted items are only taken from second list
LIRS Example (From Wikipedia)

- Stack (FIFO queue) S holds Low Inter-Referenced (LIR) items
- Stack (FIFO queue) Q holds high Inter-references (HIR) items

B is accessed E is accessed D is accessed C is accessed

C is accessed
W-TinyLFU

- Developed by Einziger, Friedman, Manes
  - Part of the Ph.D. thesis of Gil Einziger
- Adopted by the open source Caffeine Java 8 caching library
- Used directly or indirectly (through Caffeine) by the following products:
  - Cassandra, Apache Accumulo, RedHat’s Infinispan, VMWare’s Corfu, neo4j, Allegro, Amplitude, Spring, ratpack, finagle, druid.io, transitory, and others
• The admission policy is LFU with aging
• To be able to capture a very long history, frequency is counted using a sketch
A sliding window based frequency histogram

A new item is admitted only if it is more frequent than the victim
Exponential Decay to Eliminate the Sliding Window

- We maintain an access counter

- Each time the counter reaches $W$, we halve all entries in the histogram as well as the counter
  - This is a *Reset* operation

- **Lemma**: denote $f_i$ the frequency of item $i$ and $h_i$ the height of $i$ in the histogram
  - Just before each Reset operation, $E(h_i) = f_i \times W$
Proof of Aging Lemma

- By induction on the number of Reset operations

- For $r=1$, immediately before the 1st Reset, by definition we have that $E(h_i) = f_i \times W$

- Assume correctness for $r=j$ and prove for $r=j+1$

- By assumption, right after the $j^{th}$ Reset, $E(h_i) = f_i \times W/2$

- After an additional $W/2$ operations, the height of $i$ was increased by an additional $f_i \times W/2$ in expectation

- Since expectation is additive, the lemma holds
Should we maintain exact statistics in order to decide?

• Observation 1:
  ▫ There's no sense in being precise when you don't really know what you're talking about

• Observation 2:
  ▫ It is much cheaper to maintain an approximate statistics using a sketch
Counting with Bloom Filter

• A vector of counters (instead of bits)
• A counting Bloom filter supports the operations:
  ▫ **Increment**
    • Increment by 1 all entries that correspond to the results of the k hash functions
  ▫ **Decrement**
    • Decrement by 1 all entries that correspond to the results of the k hash functions
  ▫ **Estimate**
    • Return the minimal value of all corresponding entries

\[
\text{CBF} = \begin{array}{cccc}
0 & 1 & 0 & 1
\end{array}
\]

k=3, h_1(o1)=0, h_2(o1)=7, h_3(o1)=5, m=11, Estimate(o1)=4
Bloom Filters with Minimal Increment

- Almost identical to “count-min sketch with conservative update”
- Sacrifices the ability to decrement in favor of accuracy/space efficiency
  - During an increment operation, only update the lowest counters

SBF-MI = \[
\begin{array}{cccc}
\emptyset & & & 8 & 6 \\
\end{array}
\]

k=3, h_1(o1)=0, h_2(o1)=7, h_3(o1)=5  \quad m=11  \quad \text{Increment(o1) only adds to the first entry (3->4)}
Small Counters

- A naïve implementation would require counters of size $\log(W)$ – can we do better?
- Assume that the cache size is bounded by $C(<W)$
  - An item belongs to the cache if its access frequency is at least $1/C$
  - Hence, the counters can be capped by $W/C$ (Log($W/C$) bits)
- Example:
  - Suppose the cache can hold 2K items and the window size is 16K => $W/C=8$
  - Each counter is only 3 bits long instead of 14 bits
Summary of Meta-Data

• Can maintain approximate statistics for a sliding duration that is 10 times the cache size at the cost of ~1 byte per cache item

• No need to maintain explicit ghost entries
In principle, the ratio between Window Cache and Main Cache can be any number and the cache management of the main cache can be any scheme.
Some Performance Results
Hit Ratio Evaluation

Glimpse trace – published by authors of LIRS
Hit Ratio Evaluation

DS1 Database trace – published by authors of ARC

![Graph showing hit ratio evaluation with different cache sizes and algorithms like LRU, W-TinyLFU, Lirs, TLRU, Arc, and Optimal.](image-url)
Hit Ratio Evaluation

P8 Windows server trace – published by authors of ARC
Hit Ratio Evaluation

OLTP file system trace – published by authors of ARC
Hit Ratio Evaluation

F1 data analytics file system trace – published by UMASS
Hit Ratio Evaluation

SPC1-like synthetic trace – published by authors of ARC

![Hit Rate vs Cache Size Graph](image-url)
Hit Ratio Evaluation

Search engine (s3) trace – published by authors of ARC

![Graph showing hit ratio evaluation for different cache sizes and algorithms. The graph plots Hit Rate against Cache Size with various algorithms represented by different lines and markers. The algorithms include LRU, W-TinyLFU, LIRS, TLRU, Arc, and Optimal.]
Hit Ratio Evaluation

Search engine (WS1) trace – published by UMASS

![Graph showing hit ratio evaluation](image-url)
On Going Work

• Adaptive W-TinyLFU
  ▫ Part of the M.Sc. thesis of Ohad Eytan

• Probabilistic Frequency Tracking
  ▫ Part of the M.Sc. thesis of Dolev Adas

• Combining size and cost into the decision making
  ▫ Open issue
The End