CSE 421

Greedy Algorithms / Caching Problem

Yin Tat Lee
Optimal Caching/Paging

Memory systems

- Many levels of storage with different access times
- Smaller storage has shorter access time
- To access an item it must be brought to the lowest level of the memory system

<table>
<thead>
<tr>
<th>Type</th>
<th>Latency</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>0.25 ns</td>
<td>36 KB</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>1 ns</td>
<td>192 KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>3 ns</td>
<td>1.5 MB</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>14 ns</td>
<td>15 MB</td>
</tr>
<tr>
<td>DRAM</td>
<td>66 ns</td>
<td>32 GB</td>
</tr>
<tr>
<td>SDD</td>
<td>0.15 ms</td>
<td>480 GB</td>
</tr>
<tr>
<td>Internet</td>
<td>7 ms</td>
<td></td>
</tr>
</tbody>
</table>

My home computer
Optimal Caching/Paging

Memory systems

- Many levels of storage with different access times
- Smaller storage has shorter access time
- To access an item it must be brought to the lowest level of the memory system

Consider the problem between 2 levels

- Main memory with $n$ data items
- Cache can hold $k < n$ items
- Assume no restrictions about where items can be
- Suppose cache is full initially
  - Holds $k$ data items to start with
Optimal Offline Caching

Caching
- Cache with capacity to store $k$ items.
- Sequence of $m$ item requests $d_1, d_2, \ldots, d_m$.
- Cache hit: item already in cache when requested.
- Cache miss: item not already in cache when requested: must bring requested item into cache, and evict some existing item, if full.

Goal
- Eviction schedule that minimizes number of evictions.

Example: $k = 2$, initial cache $= a, b$, requests: $a, b, c, b, c, a, a, b$.

Optimal eviction schedule: 2 cache misses.

Why 2 is optimal?
Optimal Offline Caching: Farthest-In-Future

Farthest-in-future

- Evict item in the cache that is not requested until farthest in the future.

  current cache: a b c d e f

  future queries: g a b c e d a b b a c d e a f a d e f g h ...

  cache miss  eject this one

Theorem

- [Bellady, 1960s] FIF is an optimal eviction schedule.

Exchange Argument

- We can swap choices to convert other schedules to Farthest-In-Future without losing quality.
Why Exchange Argument?

Greedy cannot handle problems with many local minimum.

Exchange argument basically proving there is no local min.
Warm up \( n = k + 1 \)

Farthest-in-future

- Evict item in the cache that is not requested until farthest in the future.

  current cache: \[ a \quad b \quad c \quad d \quad e \quad f \]

  future queries: \[ g \quad a \quad b \quad c \quad e \quad d \quad a \quad b \quad b \quad a \quad c \quad d \quad e \quad a \quad f \]

  \[ \uparrow \quad \text{cache miss} \quad \uparrow \quad \text{eject this one} \]

When \( n = k + 1 \),
between the cache miss and the farthest-item in the future,
“\[ g \quad a \quad b \quad c \quad e \quad d \quad a \quad b \quad b \quad a \quad c \quad d \quad e \quad a \quad f \]”
contains all the item.
Hence, any algorithm must miss once.
Problem: What’s the problem of this proof for \( n > k + 1 \).
Reduced Eviction Schedules

Definition
• A reduced schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

Intuition
• Can transform an unreduced schedule into a reduced one with no more cache misses.

![Unreduced Schedule](image1)

![Reduced Schedule](image2)

an unreduced schedule

a reduced schedule
Reduced Eviction Schedules

Claim

• Given any unreduced schedule $S$, can transform it into a reduced schedule $S'$ with no more cache misses.

Proof (by induction on number of unreduced items)

• Suppose $S$ brings $d$ into the cache at time $t$, without a request.
• Let $c$ be the item $S$ evicts when it brings $d$ into the cache.

Case 1: $d$ evicted at time $t'$, before next request for $d$.
Case 2: $d$ requested at time $t'$ before $d$ is evicted. □
Farthest-In-Future: Analysis

Theorem
• FIF is optimal eviction algorithm.

Proof. (by induction on number or requests $j$)

**Invariant:** There exists an optimal reduced schedule $S$ that makes the same eviction schedule as $S_{FIF}$ through the first $j + 1$ requests.

Let $S$ be reduced schedule that satisfies invariant through $j$ requests. We produce $S'$ that satisfies invariant after $j + 1$ requests.

• Consider $(j + 1)^{\text{st}}$ request $d = d_{j+1}$.
• Since $S$ and $S_{FIF}$ have agreed up until now, they have the same cache contents before request $j + 1$.

**Case 1:** ($d$ is already in the cache).

$S' = S$ satisfies invariant. (used $S$ is reduced here)

**Case 2:** ($d$ is not in the cache and $S$ and $S_{FIF}$ evict the same element).

$S' = S$ satisfies invariant.
Farthest-In-Future: Analysis

Proof. (continued)

Case 3: \((d \) is not in the cache; \(S_{\text{FIF}}\) evicts \(e\); \(S\) evicts \(f \neq e\)).

- begin construction of \(S'\) from \(S\) by evicting \(e\) instead of \(f\)

<table>
<thead>
<tr>
<th>j</th>
<th>S</th>
<th>S'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>same</td>
<td>e</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>same</td>
<td>e</td>
</tr>
<tr>
<td>evicted by S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>j + 1</th>
<th>S</th>
<th>S'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>same</td>
<td>e</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>same</td>
<td>d</td>
</tr>
<tr>
<td>evicted by (S_{\text{FIF}})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- now \(S'\) agrees with \(S_{\text{FIF}}\) on first \(j + 1\) requests; we show that having element \(f\) in cache is no worse than having element \(e\)
  - Continue building \(S'\) to be the same as \(S\) until forced to be different
Farthest-In-Future: Analysis

Proof. (continued)

Let \( j' \) be the first time after \( j + 1 \) that \( S \) and \( S' \) must take a different action, and let \( g \) be item requested at time \( j' \).

\[
\begin{array}{c|c|c}
\text{j'} & \text{same} & e \\
S & & \\
\end{array} \quad \begin{array}{c|c|c}
\text{same} & f \\
S' & & \\
\end{array}
\]

Case 3a: \( g = e \).

Can't happen: \( e \) was evicted by Farthest-In-Future so there must be a request for \( f \) before \( e \).

Case 3b: \( g = f \).

Element \( f \) can't be in cache of \( S \), so let \( e' \) be the element that \( S \) evicts.

- if \( e' = e \), \( S' \) accesses \( f \) from cache; now \( S \) and \( S' \) have same cache
- if \( e' \neq e \), \( S' \) evicts \( e' \) and brings \( e \) into the cache; now \( S \) and \( S' \) have the same cache

\[ \text{Note: } S' \text{ is no longer reduced, but can be transformed into a reduced schedule that agrees with } S_	ext{FIF} \text{ through step } j + 1 \]
Farthest-In-Future: Analysis

Proof. (continued)
Let $j'$ be the first time after $j + 1$ that $S$ and $S'$ must take a different action, and let $g$ be item requested at time $j'$.

Case 3c: $g \neq e$ and $g \neq f$.
$S$ must evict $e$.
Make $S'$ evict $f$; now $S$ and $S'$ have the same cache.

In each case can now extend $S'$ using rest of $S$ at no extra cost. $S'$ is optimal, reduced, and agrees with $S_{FIF}$ for $j + 1$ steps. Optimality of $S_{FIF}$ follows by induction.
Online Caching

- Online vs. offline algorithms.
  - Offline: full sequence of requests is known a priori.
  - Online (reality): requests are not known in advance.
  - Caching is among most fundamental online problems in CS.

- LIFO. Evict page brought in most recently.
- LRU. Evict page whose most recent access was earliest.

- Theorem. FIF is optimal offline eviction algorithm.
  - Provides basis for understanding and analyzing online algorithms.
  - LRU is k-competitive. [Section 13.8]
  - LIFO is arbitrarily bad.
$k$-server problem

- There are $k$ fire trucks.
- When fire happens, we need to move a truck there.
- Define $ALG$ is the total movement of all fire trucks.
- Define $OPT$ is the total movement of the optimal plan if we know where the fire happens in advance.
- Define the competitive ratio is $ALG/OPT$. 
Greedy does not work

\[ ALG = +\infty. \]
\[ OPT = O(1). \]

So, the competitive ratio is \( \frac{ALG}{OPT} = +\infty. \)

For long time, the best competitive ratio is \( O(k). \)

It was conjectured that one can get \( \log^{O(1)}(k). \)

This man did it.

Fusible HSTs and the randomized k-server conjecture

James R. Lee

(Submitted on 6 Nov 2017 (v1), last revised 21 Feb 2018 (this version, v2))