1. Introduction

Caching is useful whenever you have a small, faster memory and a larger, slower memory. You can’t keep everything in the small, fast memory, so the goal is to keep only those objects which are likely to be needed again soon. Requested objects in the cache are hits, and misses otherwise. A replacement policy indicates which objects in the cache to discard in order to make room for a new request. Here we consider 12 replacement policies:

- 7 common policies: LRU, RAND, FIFO, LFU, LFJU and MFU
- 5 fancy recent policies: SIZE, GDS, GD*, GDSF, LFUDA

All of these policies are simple to maintain. The de facto standard is LRU.

2. Key Idea: Virtual Caches

Key to being adaptive (both online and offline) is to simulate a cache for each of the 12 baseline policies. Each policy acts on its own virtual cache.

Only meta-data (ID, size, priority) are kept in the virtual cache. Thus maintenance and storage costs are negligible.

To be fair, virtual caches are stored in the same memory as the real cache, as shown in the figure on the right.

A master policy observes each policy on its virtual cache, and bases its own replacement policy on the policies it judges to be currently best. But how to decide which is best?

3. Expert Framework

Another idea is to exploit the Expert Framework theory and algorithms for online learning. In this framework, a weight \( w_i \) is kept for each of \( N \) base policies (experts). \( w_i \) is an estimate of the performance of policy \( i \) relative to the others. After each request the master policy modifies \( w \) two ways, thereby updating its belief in each policy. A loss update quickly punishes policies with recent misses, and a share update keeps the weights of recently successful policies from becoming too low. This will help them recover weight later if they become successful again.

The weights, \( w \), lie on the probability simplex and the loss update is an exponential update which punishes policies that get a miss by multiplying their weight by a factor \( \beta \in (0,1) \):

\[
\nu_i^t = \nu_i^{t-1} \frac{\text{miss}_i}{w_i^{t-1}}, \quad \text{miss}_i \in \{0,1\} \tag{1}
\]

The share update we used (Fixed Share to Uniform Past) mixes a small fraction of the past average of weights with the current weights:

\[
w = (1-\alpha) w_i^t + \alpha w_i^{t-1} \quad v_i^t = \beta (1-\alpha) + \alpha v_i^{t-1} \tag{2}
\]

This share update is designed for the case when different experts are good in different segments of the data, and in particular when the best expert shifts between a small sub-pool of all \( N \) experts.

4. Master Policy Managing The Real Cache

Control over the real cache is given to the policy with the highest weight. However, there is a problem: The success of a policy is a essentially a function of the state of its virtual cache. Changing the governing policy without swapping virtual cache contents is sub-optimal. A technique which switches both policy and cache contents must be developed.

With this in mind, we develop the following protocol:

1. The request is processed on each virtual cache, and the loss and share updates are applied.
2. Then the request is processed on the real cache.

The goal of the master policy is the following: make the contents of the real cache match the contents of the governing policy’s virtual cache.

\- Disadvantage: the real cache lags too far behind the governing policy’s virtual cache.

\- Instantaneous Rollover: When a new policy gets high weight discard all objects not in the governing policy’s virtual cache, then refetch objects in the governing policy to capacity.

\- Disadvantage: always performs as well current governing policy.

\- Background Rollover: Assume that a limited number of refetches can be performed in the background (e.g. between requests, at idle times, while fetching objects on demand). Here, refetches are done by priority in the governing virtual cache. The following seem to be reasonable compromises:

\- Background 1: one “free” refetch of every request.

\- Background 2: one “free” refetch for every hit, five for every miss.

\- Advantages: A small amount of refetches is just what’s needed to reap the benefits of switching policies.

The figure below on the right shows how our instantaneous rollover master policy (labeled “roll”) tracks the best policy over time. The parameters \( \alpha \) and \( \beta \) are set so that governing policies switch less than 1% of the time (\( k = 75 \)).

The figure below on the right shows how each master policy measures up against our optimal offline comparators. Everybody’s favorite, LRU, is included for good measure.

Summary

Demand Rollover is slightly worse than BestFixed (SIZE). Instantaneous rollover is ~15% better than BestFixed, and only slightly worse than BestShifting (K).

The backgrond rollover policies are in the middle. Instantaneous rollover is the most “adaptive” but also the most unrealistic.