Le Mans

Enoch Huang
Sudarshan Seshadri

Design Methodology

This predictor is based on the LeCaR [1] cache replacement policy. LeCaR utilizes reinforcement learning to select between two cache replacement policies. In the original leCaR paper, LRU and LFU were the replacement policies; given that SRRIP also replaces cache based on recency of usage, it was used in place of LFU as an implementation of it already existed in the competition codebase. The LeCaR algorithm works as follows: on each cache eviction, the evicted cache is placed in a history buffer the size of the cache along with a tag indicating the eviction policy used (LRU or SRRIP). Weights are adjusted if the address associated with a cache miss is found in history. For example, if a cache miss due to LRU is identified in the history, the weight of SRRIP is increased (and vice versa). The weights are initialized as 0.5 for each policy and updated as below, where wLFU is the weight for the SRRIP replacement policy and wLRU is the weight for the LRU replacement policy:

Input: page q, learning rate λ, discount rate d
$t :=$ time spent by page q in History
$r := d^t$
if q is in $H_{\mathbf{LRU}}$ **then**
  $w_{\mathbf{LFU}} := w_{\mathbf{LFU}} * e^{\lambda * r}$ // increase $w_{\mathbf{LFU}}$
else if q is in $H_{\mathbf{LFU}}$ **then**
  $w_{\mathbf{LRU}} := w_{\mathbf{LRU}} * e^{\lambda * r}$ // increase $w_{\mathbf{LRU}}$
$w_{\mathbf{LRU}} := w_{\mathbf{LRU}} / (w_{\mathbf{LRU}} + w_{\mathbf{LFU}})$ // normalize
$w_{\mathbf{LFU}} := 1 - w_{\mathbf{LRU}}$

Algorithm 1: weight replacement policy

The learning rate was 0.45, and the discount rate was 0.005**(1/CACHE_SIZE), as identified by the original LeCaR paper.

Results

The cache replacement policy achieved a miss rate of 0.5061 in competition results using config1.

Tradeoffs

This design consumes a high amount of storage due to the fact that two predictors are used and a history queue the size of the entire cache set must be kept as well. Due to the fact that history must be saved, one cannot simply use a portion of the sets as a basis of storage reduction. However the high storage pays off in a low miss rate.

Storage Overhead

This predictor consumes 3x more storage compared to a simple LRU predictor. There are two arrays the size of the cache, keeping track of which sets to replace under each replacement policy (LRU and SRRIP) as well as a history queue the size of the cache.

References

[1] Giuseppe Vietri, Liana V. Rodriguez, Wendy A. Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. 2018. Driving cache replacement with ML-based LeCaR. In Proceedings of the 10th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage'18). USENIX Association, USA, 3.