Time-Decayed Caching: A Novel Rule Benchmarked Against Established Policies

Cătălin Pângăleanu

Abstract:

AMS Mathematical Subject Classification: 68M20, 68P05

ACM Computing Reviews Categories and Subject Descriptors: Theory of computation, Design and analysis of algorithms, Online algorithms, Caching and paging algorithms

Contents

Introduction

We propose a time-decayed policy based on the nuclear physics concept of half-life. We introduce a one-parameter eviction rule that maintains a per-key decayed score using a half-life (H) constant. The policy blends recency and frequency and evicts the item with the lowest current decayed score.

Motivation

1. Summarize classical policies such as LRU, LFU, SLRU
2. Explain the need for a new policy that combines both recency and frequency

Contributions

1. Go into more detail for half-life policy. In short, H = the horizon where an old hit is worth half a new hit. If H = 100 requests, then a hit that happened 100 requests ago counts ½; 200 requests ago counts ¼, etc.
2. Explain automatic adjustment of H-constant. H = c * R, where c is a predetermined constant and R = exponential weighted moving average (R[i + 1] = (1 - η) * R[i] + η * gap, where gap = now - last_hit[key] in requests)

Research Questions

1. On different workloads and fixed capacity, does HL-Cache improve hit rate and tail latency over LRU/SLRU/LFU?
2. Does the new policy offer CPU/memory savings?
3. How sensitive is the policy to half-life mis-specification, and does online auto-tuning (formula in section above) keep performance better?

Background and Related Work

Terminology and Metrics

1. Explain cache-related terminology (hit etc.)
2. Explain what we would count as success for the proposed policy (possible metric is hit-rate gain compared to other classical policies etc.)

Classical Policies

1. Go into detail on LRU, LFU etc.

Other Related Work

1. Go over some of the more recent policies and optimizations (from the bibliography)
2. Explain general use of caching policies (in OS, databases etc.)

The Proposed Policy: Time-Decayed Caching

Formulation

1. Present half-life policy from the ground up in detail
2. Present recurrence relations and all mathematical formulas associated with the new policy

Complexity

1. Explain how we still keep the O(1) complexity associated with LRU/LFU

Implementation

1. Provide implementation details
2. Show pieces of code with explanations

Evaluation

Methodology

1. Implement classical policies (LRU, LFU etc.)
2. Get metrics such as hit-rate, CPU/memory usage, tail latency etc.
3. Benchmark current policy to classical ones based on metrics above. Use at least 2 different generated/pre-existing datasets
4. Determine sensitivity of half-life policy to changing constants (see c, η in formulas above)

Results

1. Summarize results and provide information visually (graphs etc.)

Future Work

Conclusion

Bibliography

[1] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, C.-S. Kim, “LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies,” IEEE Transactions on Computers, 1352–1361, 2001.

[2] N. Megiddo, D. S. Modha, “ARC: A Self-Tuning, Low Overhead Replacement Cache,” USENIX FAST, 116–130, 2003.

[3] T. Johnson, D. Shasha, “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm,” VLDB, 439–450, 1994.

[4] G. Einziger, R. Friedman, B. Manes, “TinyLFU: A Highly Efficient Cache Admission Policy,” arXiv, 2015.

[5] A. Blankstein, S. Sen, M. J. Freedman, “Hyperbolic Caching: Flexible Caching for Web Applications,” USENIX ATC, 499–511, 2017.

[6] O’Neil, Elizabeth J.; O’Neil, Patrick E.; Weikum, Gerhard. The LRU-K Page Replacement Algorithm for Database Disk Buffering. Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD ’93), Washington, DC, USA, 297–306, 1993.

[7] Jiang, Song; Zhang, Xiaodong. LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance. Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS ’02), 31–42, 2002.

[8] Bansal, Sorav; Modha, Dharmendra S. CAR: Clock with Adaptive Replacement. Proc. USENIX Conference on File and Storage Technologies (FAST ’04), 187–200, 2004.

[9] Jiang, Song; Chen, Feng; Zhang, Xiaodong. CLOCK-Pro: An Effective Improvement of the CLOCK Replacement. Proc. USENIX Annual Technical Conference (USENIX ATC ’05), 323–336, 2005.

[10] Zhou, Shuhui; Philbin, John; Li, Kai. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. Proc. USENIX Annual Technical Conference (USENIX ATC ’01), 91–104, 2001.

[11] Smaragdakis, Yannis; Kaplan, Scott F.; Wilson, Paul R. EELRU: Simple and Effective Adaptive Page Replacement. Proc. ACM SIGMETRICS ’99, 122–133, 1999.

[12] Robinson, John T.; Devarakonda, Murthy V. Data Cache Management Using Frequency-Based Replacement. Proc. ACM SIGMETRICS ’90, 134–142, 1990.

[13] Cao, Pei; Irani, Sandy. Cost-Aware WWW Proxy Caching Algorithms (GreedyDual-Size). Proc. USITS ’97, 193–206, 1997.

[14] Smith, Alan Jay. Disk Cache—Miss Ratio Analysis and Design Considerations. ACM Transactions on Computer Systems (TOCS) 3(3), 161–203, 1985.

[15] Ruemmler, Chris; Wilkes, John. UNIX Disk Access Patterns. Proc. USENIX Winter ’93, 405–420, 1993.