Who Is More Adaptive?
ACME: Adaptive Caching using Multiple Experts

Ismail Ari   Ahmed Amer   Ethan Miller   Scott Brandt   Darrell Long

{*ari,amer4,elm,scott,darrell* }*@cs.ucsc.edu*
Storage Systems Research Group

University of California Santa Cruz

Abstract

The trend in cache design research is towards finding the single optimum replacement policy that performs better than any other proposed policy by using all the useful criteria at once. However, due to the variety of workloads and system topologies it is daunting, if not impossible, to summarize all this information into one magical *value* using any static formula. We propose a workload and topology adaptive cache management algorithm to address this problem. Based on proven machine learning techniques, this algorithm uses a weighted voting mechanism guided by a pool of cache replacement policies. Objects that collect the highest total vote from all policies stay in the cache. The policies that predict the workload well are rewarded by an increase in their weight and the policies that lead to wrong decisions are punished by a decrease in their weight. Weight adjustments of the replacement policies, or *caching experts*, are managed by extremely powerful but computationally simple machine learning algorithms. Our scheme is different from hybrid criteria schemes and partitioned cache management policies because it is adaptive, and because it favors objects that are rated highly by many policies rather than simply favoring objects with high weight by a single, possibly complex policy.

1 Introduction

The number of users connected to the Internet via slow links at the edges of the networks is increasing exponentially. Satisfying so many users with fast response times while transparently saving network bandwidth demands efficient proxy caching techniques. The cache sizes that are technically and economically possible are dwarfed when compared to the infinite unique document space accessed by so many users. Only the most “valuable” objects are to be kept in the cache, where the value of an object is *a dynamic parameter that leads to a global improvement in both client response times and network bandwidth usage, when equally applied to all objects being accessed*. This value is the output of policies making use of one or more criteria.

Static caching policies cannot adapt to changes in workload and network topology. These policies perform poorly as the characteristics of the workload shift over time. Moreover, they are prone to filtering effects [1] and perform poorly when used at intermediate nodes in the hierarchy [4, 21]. Continuous monitoring and manual tuning are tedious and daunting tasks, if they are possible at all. Solutions that try to adapt by exchanging messages to learn who caches what [10] or inquiring to a central database to locate cached copies [15] have limited scalability. Autonomous caching schemes that don't exchange messages, but summarize useful criteria such as time, frequency and size into one magical value based on a static formula discard information by collapsing multiple possible dimensions or viewpoints into one. Different viewpoints could lead to different decisions, which may be more valuable than a single static viewpoint in different circumstances.

To address this problem, we propose an adaptive weighted voting scheme that employs a pool of cache replacement policies. In our design, we separate the caching criteria from policies, providing flexibility in policy choice both in time and location. The ideas proposed in this paper can be applied to any caching system, including hierarchical web proxies, distributed file systems with cooperative caching and storage embedded network clusters.

There are five significant motivations for our research. The first is the unprecedented need to eliminate the daunting tasks of continuous monitoring, tracing and manual parameter tuning for performance improvements. The second is the proven success of using heterogeneous policies [4] within caching clusters to achieve exclusive caching. Although far from being adaptive, this novel idea implies that if the right combinations of policies can be found, the global power of a cache cluster scales linearly with the number of nodes due to the caching of different objects by different policies—heterogeneous caching eliminates the need for extra messaging. The third motivation is the need for flexibility to support new criteria. Many successful caching strategies that use uncommon criteria such as object IDs and QoS priority information are emerging. We predict that, in the future, more criteria are to come and therefore there is need for flexibility to support all. The fourth motivation is the remarkable success of computationally simple machine learning algorithms [13, 3], and their proven success in addressing non-trivial operating systems problems [11]. The fifth and final motivation is the abundance of processing power as an enabling technology for the slightly more complex cache management scheme that we are proposing.

2 Related Work

Table 1 lists some very popular and recently proposed criteria and the policies that use these criteria to make local replacement decisions. Random, First-In-First-Out (FIFO) and Last-In-First-Out (LIFO) do not require any information about the objects. Time, frequency and object size are the most commonly used criteria for local replacement decisions. Least Recently Used (LRU) uses recency of access as the sole criteria for replacement, while Least Frequently Used (LFU) uses frequency of access. SIZE replaces the largest object and Greedy-Dual-Size (GDS) [14] replaces the smallest key $K_i = C_i/S_i + L$, where C_i is the retrieval cost, S_i is the size and L is a running age factor. GDS with Frequency (GDSF) adds the frequency of access, F_i, into the same equation. Lowest Relative Value (LRV) [18] replacement makes a cost benefit analysis using the access time, access frequency and size information about objects.

Hashing or more complex Bloom filters on object IDs are often preferred for local decisions in the building blocks of a global system of caches. If the ID hash implies that the neighboring node should be caching that object then it could be replaced quickly. Hop-counts provide another set of criteria that can passively provide an indication of the logical location of a cache without resorting to full location-awareness. Up-stream hop counts are a loose measure of how far a cache is from the closest data source, while down-stream hop counts indicate logical distance from clients. Recent research [22] points to the benefits of keeping a record of access latency information, providing yet another potential caching criterion (e.g., it s wise not to discard items from the cache that are very costly to retrieve). Stor-serv [7] proposes Quality of Service (QoS) ideas used in networking to be applied to storage systems for giving differentiated services.

criteria	**algorithm**
–	Random, FIFO, LIFO
time	LRU, MRU, GDS, GDSF, LRV
freq	LFU, MFU, GDSF, LRV
size	SIZE, GDS, GDSF, LRV
retrieval cost	GDS, GDSF, LRV
ID	hash, Bloom-Filter
hop-count	–
QoS priority	stor-serv

Table 1: An extended taxonomy of existing and proposed cache replacement policies.

Table 1 does not intend to cover all the proposed algorithms; rather, our goal is to show two things. First, the possible criteria and the ways to use them are endless. Second, the trend in cache replacement algorithms is towards finding the functions that unite all the criteria in a single key or value. Other taxonomies of time, frequency and size based policies are presented in prior work [14, 6].

Virtual cache management [2] divides the cache into static partitions and lets a few successful policies work in separate partitions. Objects evicted from one partition go to the next until they are moved out of the cache. This scheme is not as homogeneous as our voting scheme and the performance is bound by the performance of the best partition [2].

Adaptive web caching [17] proposes that nearby caches self-configure themselves into a mesh of overlapping multicast groups and exchange messages to locate the nearby copies of requested data and to find out about topology changes. Although scalability was a major goal in this design, like any other system that needs global bookkeeping and exchange of recorded information [19, 8], the scalability and performance will be limited due to the vast amount of objects floating on the Internet. There also some other deployment problems with multicast [9].

AutoRAID [20], which automates the RAID level selection process in disk arrays, is an excellent example for the benefits of switching from manual to adaptive systems. Hybrid Adaptive Caching (HAC) [5] proposed within the context of system memory combines the virtues of page and object caching, while avoiding their disadvantages.

3 Design of ACME

Figure 1 shows our design for an adaptive (self-tuning or autonomous) cache using multiple experts. In our design, cache management criteria and policies are separated. The *criteria pool* is filled with general descriptions of information regarded as useful by one or more of the cache replacement policies. Most popular criteria such as recency of access, frequency of access and size are initially registered to the pool. A new policy registering into the *policy pool* has to register its criteria in case they are not already registered. ACME ensures that the registered criteria are recorded and updated on a per object basis. The policies themselves are just descriptions of how to use criteria to order the objects: e.g. $LFU = [list, key =< pageid, frequency >]$. These may include any proposed cache management policy — the machine learning algorithms will ensure that the best ones for the current workload receive the highest weight and therefore have the largest effect on the cache management decisions.

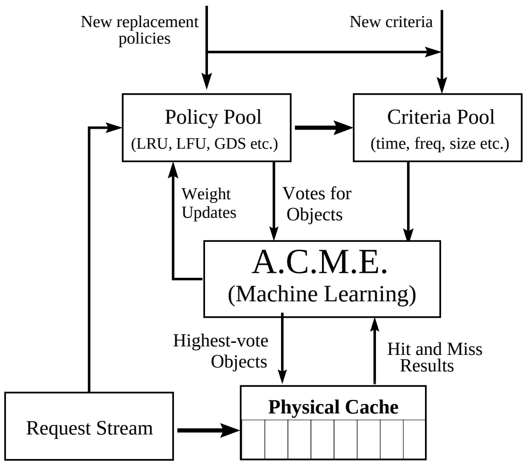


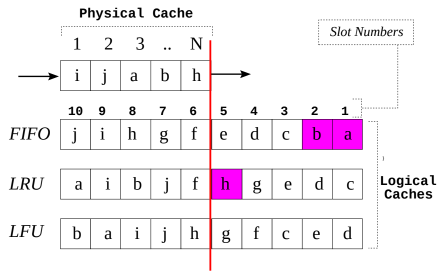
Figure 1: Design of Adaptive Caching using Multiple Experts (ACME).

ACME orders the *physical cache* depending on the votes coming from the policies. The hit and miss events occurring as a result of the request stream and cache placements are fed back to ACME. The policies that predict the workload well and vote well are rewarded by an increase in their weight and the policies that lead to wrong decisions are punished by a decrease in their weight. Weight adjustments of the replacement policies or *caching experts* are managed by the machine learning algorithms. These learning algorithms include *weighted majority* [16] and the *share* algorithm [12], both developed by members of our local machine learning group. In machine learning, expert-based algorithms provide solutions that represent a combination of such experts that can dynamically adapt to changes in experts' performance.

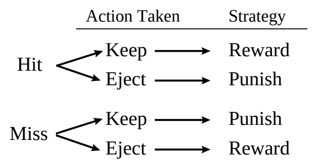
Figure 2(a) gives a simple illustration of how multiple cache replacement algorithms may be used to decide what objects to cache. By using each simple algorithm to generate a relative ordering of potentially cached objects, each algorithm can be considered an “expert” providing its own opinion about what data should be cached. Algorithms do not work on the physical cache, but a larger logical cache, whose size is related to the physical cache size. Information in the logical cache is stored in the policy pool; objects themselves are stored in the physical cache.

Each policy needs only to maintain an ordering of object identifiers, allowing the identification of cache misses that could have been avoided, or cache hits that were ranked appropriately. Some cache replacement policies can provide a numerical ranking for items in the cache, *e.g.*, space-time algorithms, whereas others only offer relative orderings. For the latter case we simply assume the rankings to be the “slot numbers” (see Figure 2(a)). Such rankings allow the simple caching policies, our experts, to provide a consistent representation of their opinions. The effective ranking of items in the cache, used by our automated cache for replacement decisions, is based on the weighted combination of the different experts’ rankings.

Many questions we aim to answer in this project concern the best choices for expert evaluation. Figure 2(b) presents the main choices when reevaluating the weight assigned to a particular policy. Should a cache miss occur, we can “punish” algorithms by decreasing the weight for the policies that did not rank the requested item highly, “reward” algorithms by increasing the weights of the policies that ranked it



(a) *Multiple Cache “Experts”*



(b) *Evaluation Paths*

Figure 2: Adaptive Caching

highly, or combine reward and punishment for algorithms that were ambivalent about the item. Similar questions are posed when a cache hit occurs: how are algorithms rewarded and punished. Other questions include finding the best rate of weight adjustment, *i.e.*, how often and how drastically do we modify the weights and the effects that modifying the combination of caching algorithms has on the cache’s ability to perform well while quickly selecting the best combined policy.

4 Conclusions

It is generally true that the multiple-criteria policies are more successful than single-criteria policies at deciding which objects to cache. However, a single policy that represents a single view cannot be guaranteed to work equally well under all types of workloads and topologies. Our adaptive caching scheme based on voting experts gives a chance to all views and evaluates performance based on hits and misses using machine learning algorithms, providing improved cache performance by adaptively choosing the best policies, and thus criteria, to apply to a particular cache. Because our system is adaptive, nodes can vary their policies over time, producing better performance over a wide range of both workloads and cache topologies.

5 Acknowledgements

We are grateful to Manfred Warmuth and the Machine Learning Group for their valuable contributions to the machine learning algorithms in our on-going work.

References

[1] A. Amer and D. D. E. Long. Adverse filtering effects and the resilience of aggregating caches. In *Proceedings of the Workshop on Caching, Coherence and Consistency (WC3 '01)*, Sorrento, Italy, June 2001. ACM.

[2] M. Arlitt, L. Cherkasova, J. Dilley, and R. Friedrich. Evaluating content management-techniques for web proxy caches. In *Proceedings of the 2nd Workshop on Internet Server Performance WISP'99*, Atlanta, Georgia, May 1999.

[3] O. Bousquet and M. K. Warmuth. Tracking a small set of experts by mixing past posteriors. In D. Helmbold and B. Williamson, editors, *Proceedings of the 14th Annual Conference on Computational Learning Theory and 5th European Conference on Computational Learning Theory*, pages 31–47. Springer, July 2001.

[4] M. Busari and C. Williamson. Simulation evaluation of heterogeneous web proxy caching hierarchies. In *Proceedings of the Ninth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2001)*, pages 379–388, Cincinnati, OH, Aug. 2001. IEEE.

[5] M. Castro, A. Adya, B. Liskov, and A. C. Myers. HAC: Hybrid adaptive caching for distributed storage systems. In *Symposium on Operating Systems Principles*, pages 102–115, 1997.

[6] K. Cheng and Y. Kambayashi. Adavanced replacement policies for www caching. In *Web-Age Information Management, First International Conference, WAIM*, pages 21–23, Shanghai, China, June 2000.

[7] J. Chuang and M. Sirbu. stor-serv: Adding quality-of-service to network storage. In *Workshop on Internet Service Quality Economics*, Cambridge MA, Dec. 1999.

[8] M. Cieslak, D. Forster, G. Tiwana, and R. Wilson. Web Cache Coordination Protocol (WCCP) V2.0, Internet-Draft , draft-wilson-wrec-wccp-v2-00.txt, July 2000.

[9] C. Diot, B. Levine, J. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14:10–20, 2000.

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary Cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[11] D. P. Helmbold, D. D. E. Long, T. L. Sconyers, and B. Sherrod. Adaptive disk spin-down for mobile computers. *ACM/Baltzer Mobile Networks and Applications (MONET)*, pages 285–297, 2000.

[12] M. Herbster and M. K. Warmuth. Tracking the best expert. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 286–94, Tahoe City, CA, 1995. Morgan Kaufmann.

[13] M. Herbster and M. K. Warmuth. Tracking the best linear predictor. *Journal of Machine Learning Research*, 1:281–309, September 2001.

[14] S. Jin and A. Bestavros. Greedydual* web caching algorithm: Exploiting the two sources of temporal locality in web request stream. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.

[15] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao.

Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[16] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–61, 1994.

[17] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web caching: towards a new global caching architecture. *Computer Networks and ISDN Systems*, 30(22–23):2169–2177, 1998.

[18] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking*, 8(2):158–170, 2000.

[19] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. Design considerations for distributed caching on the Internet. In *International Conference on Distributed Computing Systems*, pages 273–284, 1999.

[20] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 96–108, Copper Mountain, CO, 1995. ACM Press.

[21] T. M. Wong, G. R. Ganger, and J. Wilkes. My cache or yours? Making storage more exclusive. Technical report, CMU-CS-00-157 Carnegie Mellon University, Nov. 2000.

[22] R. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *In Proceedings of the Sixth International World Wide Web Conference*, pages 325–334, Santa Clara, CA, Apr. 1997.