A Goal-Oriented Self-Tuning Caching Algorithm[†]

Ganesh Santhanakrishnan Ahmed Amer Panos K. Chrysanthis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
{ganesh, amer, panos}@cs.pitt.edu

1 Introduction

A popular solution to internet performance problems is the wide-spread caching of data. Many caching algorithms have been proposed in the literature, most attempting to optimize for one criteria or another. Recent efforts have explored the automation and self-tuning of caching algorithms in response to observed workloads. We extend these efforts to consider the goal of optimizing for selectable performance criteria. In this presentation, we describe GD-GhOST (a Goal-Oriented Self-Tuning caching algorithm based on Greedy Dual-Size caching algorithms(GD)), that attempts to facilitate the specification of desirable performance metrics in addition to eliminating the need to preset any algorithm parameters.

GD-GhOST differs from prior adaptive algorithms in the sense that, at any given time, it does not select a single policy out of several ones, but combines all of them based on their weights. With our proposed algorithm, we have shown performance matching and exceeding the best performance of the known greedy dual-size algorithms for either object or byte hit ratios across different web workloads. GD-GhOST consistently outperforms the other algorithms tested, at its worst observed performance GD-GhOST exhibited equivalent miss rates to those of the best applicable GD variant [4], while achieving miss rates that were 25% lower than the worst performing variant. For byte miss rates, GD-GhOST consistently demonstrated rates lower than the best applicable GD variant. At its best, GD-GhOST offered byte miss rates 10% lower than the best variant.

2 Background and Description

Among the most successful policies for web caching, which can be done at various levels [1, 9], the GD algorithms have proven to be very effective [4]. The GD variants are well known for their abilities to maximize different performance metrics. Among the different GD variants,the specific ones used in our initial tests are: GD-Size(1), considered to be well-suited for maximizing hit ratio, GD-Size(packets), appropriate when we wish to maximize byte hit ratio and GD-Size(frequency), the best performer in terms of file access frequency [5]. Recently, research efforts have produced caching policies that, in addition to optimizing a specific performance metric, attempt to automate policy parameter tuning [8, 2]. GD-GhOST offers the automated optimization of an arbitrary performance metric for the caching of variable-sized objects.

GD-GhOST is a replacement policy based on a combination of several GD variants, that attempts to satisfy a given goal using a fully adaptive combination of these individual component algorithms. Using hit ratios and byte hit ratios as two example metrics we will now go on to describe how an arbitrary selection among these three GD algorithms can be used by GD-GhOST to determine its replacement policy. GD-GhOST combines individual component algorithms using a master-algorithm approach similar to that employed in the ACME algorithm [2], but GD-GhOST can use more meaningful numeric inputs from the different component algorithms. GD algorithms calculate an H-value for each element in the cache, indicating its relative worth for retention.

GD-GhOST combines the H values calculated by the three GD variants, and based on an on-line evaluation of each variant's performance it produces its own H value. In effect, we proportionally weight the three variants' H values based on the performance of each algorithm. The evaluation of each algorithm's performance is derived from a user-specified weighting of the importance of each metric (in this case it is either byte or object hit ratios). For cache eviction decisions, the items with the lowest combined H values (weighted by each component algorithm's credit values) are the first choices for eviction.

[†]Supported in part by the National Science Foundation awards ANI-0123929 and ANI-0325353.

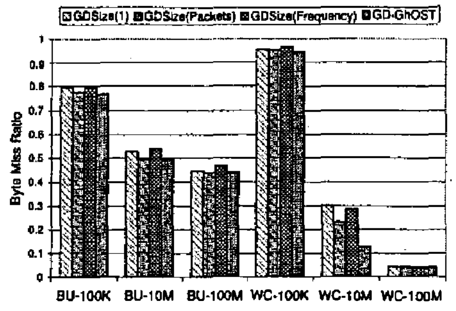

Figure 1: Average byte miss ratios.

The weighting of the policies uses a fixed number of credits that are shared among the component policies. This is similar to the weighting mechanism employed by ACME [2], as it is also based on the share update scheme [7]. This on-line update of credits ensures that at any instant in time, we are most likely to perform at least as well as the leader among the three algorithms. When the best performing algorithm degrades in performance, the redistribution of credits ensures that it does not degrade the overall performance. As a matter of fact, as we can observe in our results, we follow the best performance of the three component algorithms, and frequently exceed it.

3 Experimental Results

We conducted simulation-based experiments on real-world traces to evaluate the GD-GhOST policy and its ability to exhibit performance similar to the best policy for the selected performance metric. Specifically we tested its ability to maximize hit ratios and byte hit ratios. We will only present graphs for the byte miss rates here due to space constraints.

In Figure 1 we give the average byte miss ratios for different cache sizes of 100KB, 10MB and 100MB measured for both Boston University traces [6] and the World Cup 98 traces [3]. These represent cache sizes that are restricted, small and reasonable, respectively. We see that for the byte miss ratios for the World Cup traces, and a 10MB cache, GD-GhOST performs almost twice as well as its best component algorithm. This strongly indicates that the combined algorithm is not only capable of matching the best performing policy, but can on occasion perform significantly better than the best of its parts.

Figure 2 provides some insight as to how GD-GhOST can self-tune towards a specific selected goal and match the best component algorithm. The figure shows the variation in credits over a selected time period. This snapshot was chosen to illustrate how the best algorithm (receiving the most credit) can vary.

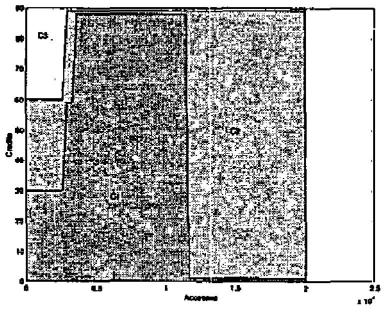

Figure 2: Credit Changes.

4 Conclusions

The contribution of this paper is a novel approach to adaptivity that combines alternatives rather than selecting one among alternatives. Using only three, homogenous, cache replacement algorithms, GD-GhOST was able to provide a cache replacement policy that requires no tuning or user-intervention beyond the initial selection of the performance criteria to be optimized. Overall, at its worst observed performance GD-GhOST was within approximately 1% of the best policy's miss ratio, and at its best, GD-GhOST reduced byte miss rates by well over 50%.

References

[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, "Caching proxies: Limitations and Potentials," in *Proc. of 4th Intl. WWW Conf.*, pp. 119–133, Dec 1995.

[2] I. Ari, A. Amer, R. Gramacy, E.L. Miller, S. Brandt, D.D.E. Long, "Adaptive Caching using Multiple Experts," in *Proc. of the WDAS*, pp. 143–157, Mar 2002.

[3] M. Arlitt and T. Jin, "1998 World Cup-WebSite Access Logs - Available at http://www.acm.org/sigcomm/ita/" Aug 1998.

[4] P. Cao and S. Irani, "Cost aware WWW proxy caching algorithms," in *Proc. of USENIX symposium on Internet Technologies and Systems*, pp. 193–206, Dec 1997.

[5] L. Cherkasova, "Improving WWW proxies performance with Greedy-Dual-Size-Frequency caching policy," HP Technical Report, Palo Alto, CA, Nov 1998.

[6] C. A. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of WWW Client Traces," *Boston Univ., Dept. of Computer Science, TR-95-010*, Apr 1995.

[7] M. Herbster and M. K. Warmuth, "Tracking the best expert," in *Proc. of the 12'th Intl. Conf. on Machine Learning*, pp. 286–94, 1995.

[8] N. Megiddo and D. S. Modha, "ARC: a self-tuning, low overhead replacement cache," in *Proc. of FAST 2003*, pp. 115–130, Mar 2003.

[9] B. Williams, "Transparent web caching solutions," in *Proc. of 3'rd Intl. WWW Caching Workshop*, Jun 1998.