Advanced Replacement Policies for WWW Caching *

Kai Cheng and Yahiko Kambayashi

Graduate School of Informatics, Kyoto University
Sakyo, Kyoto 606-8501, Japan
{chengk, yahiko}@isse.kuis.kyoto-u.ac.jp

Abstract. WWW caching necessitates advanced replacement policies that include sophisticated control logic and efficient contents management. This paper presents a constructive approach for the design and analysis of such advanced policies. Based on this approach, we develop a new caching policy, namely, PSS-W. Trace-driven simulations show PSS-W outperforms most contemporary policies in both hit rates and byte hit rates.

1 Introduction

Employing caches in the World Wide Web has been proven to be an effective approach to alleviating performance bottlenecks of web access [1]. Key to the effectiveness is how to decide and manage a suitable subset of requested data so as to maximize the hit ratio and other performance metrics. Strategies for this purpose, known as *cache replacement policies*, have been a focus of research for decades in paging scenarios[5].

However, due to the protocol restriction, WWW caching (web caching) has to deal with whole documents instead of data blocks. Web documents vary in sizes, costs, file types and sources. To cope with such complications, it is necessary to employ sophisticated control logic and special performance metrics. Furthermore, web caching is featured by large storage space, which, on the one hand, implies documents can be ”probated” in cache for some long time, but on the other hand, it also means high overhead in maintaining large number of documents. The management of cache contents plays an important role in replacement policies of web caching.

In this paper, we propose a constructive approach for design and analysis of advanced replacement policies. According to this approach, an advanced replacement policy is constructed from several simple ones. The resulted policy is parameter-less, low overhead, able to deal with comprehensive factors and implement sophisticated control logic. Based on this framework, we develop a new replacement policy, namely Pyramidal Selection Scheme with aWard (PSS-W). Trace-driven simulations show that PSS-W outperforms most other algorithms in both hit rate and byte hit rate.

2 Related Work

Replacement policies for WWW caching have been extensively investigated in recent years. Charu Aggarwal et al in a recent paper [2] have surveyed and suggested classify-

* In *Proc. 1st International Conference on Web-Age Information Management(WAIM'2000)*. June 2000. LNCS 1846 pages 239-244

ing current replacement schemes for the Web into three categories: direct extensions of traditional policies, key-based policies and function-based policies.

However, the key-based policies are too simple and the priority between sub-keys is not always ideal [8]. On the contrary, the function-based policies are too complicated and often suffer from heavy parameterization and high overhead. In practice, most well-known caching schemes turn out to be some hybrid ones where simple policies are organized to manage a segmented cache space. *Size-Adjusted LRU* is constructed from LRU and SIZE [2]; *Least Relative Value* (LRV) is an integration of SIZE, LFU and FIFO [7]; and the algorithm given by Pitkow and Recker is constructed from SIZE and LRU where the time since last access is rounded to days [6]. However, the constructions of these policies are mostly based on personal experiences. It is hard to tell what are the substantial differences between them.

3 Constructed Replacement Policies

A cache can be characterized as a software agent that consists of a finite storage space (*cache space*), a set of objects (*object space*), a replacement policy and a set of *constraints*. Replacement policy determines the contents and how to manage the contents of the cache. Constraints are additional conditions that a cache has to satisfy. We distinguish three classes of constraints: *admission constraints*, *consistency constraints*, and *miscellaneous constraints* such as comfortable level of cache space.

3.1 Framework of Constructed Replacement Policies

As described previously, the diversity of web documents and large scale of cache space complicate the design of new replacement policies. Thus, we propose a constructive approach to design and analysis of advanced replacement policies. As depicted in Fig. 1, an advanced policy is constructed from several simple policies with (1) a set of *classification rules*; (2) few *unit caches* and (3) a *central cache*

Classification rules

Classification rules are based to allocate objects and space among all unit caches. An object can belong to only one unit cache at a specific time. The classification rule may be a simple function, or a sophisticated set of logic rules. For example

1. *If object X has a specific size, then cache X in unit* $\lfloor log_2(size) \rfloor$
2. *If object* X *is from Japan and content is about baseball, and its type is picture and size is bigger than 24KB, then keep* X *in unit 1*

Unit caches

A unit cache is an independent cache that has its own object space, cache space, caching policy and constraints. Which unit an object belongs to is determined by classification rules. Thus, unit cache can use simpler caching policy. For example, if classification rule is rule 1 as in previous example, then objects in the same unit cache have the similar sizes. As a result, the unit cache need not take size into account. For this reason, simple policies such as LRU, FIFO, and SIZE [8] are often used in this case.

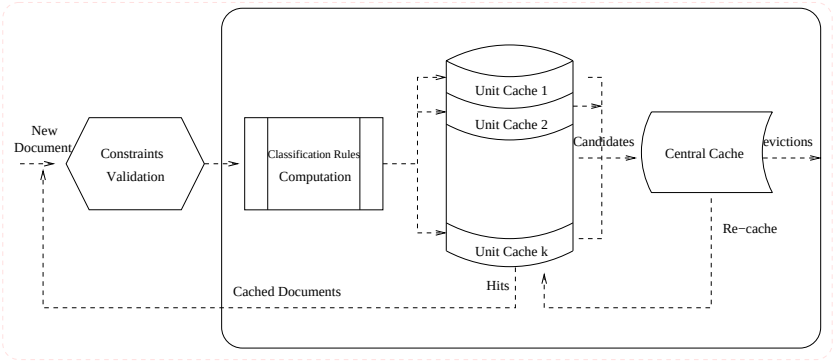

Fig. 1. Framework of constructed replacement policies

Table 1. Comparison of Various Constructive Policies

Algorithm	Rules	U	Unit Caches	Central Cache	Factors
Segmented LRU	$nref$	2	LRU	Re-cache in D_{i-1}	atime, nref*
Size-Adjusted LRU	$\lfloor log_2(size) \rfloor$	24	LRU	$\max(size * atime)$	atime, size*
Least Relative Value	$nref$	10	SIZE, FIFO	$\max(lrv)$	atime, nref*, size
Pitkow/Recker	$day(atime)$	7	SIZE, LRU	$\max(day(atime))$	atime*, size
PSS-W	$\lfloor log_2(\frac{size}{nref}) \rfloor$	24	LRU	$\max(\frac{size*atime}{nref})$	atime, nref*, size*

Central cache

A central cache manages the candidate evictions of unit caches and makes final decision. Central cache may be without its own cache space and since there are only a limited number of eviction candidates, the central caching policy can be very elaborate, with many factors in consideration. The caching decisions have several types: *eviction*, *re-cache*, and *probation*.

An eviction object will be purged at once. However, if an object is selected to re-cache or to probate, it will remain in cache. The distinction between re-cache and probation is: an object to be re-cached will be cached at once in a certain *unit cache*, while a probation object will be held by central cache in its own cache space and the final decision is expected to make in the next turn.

3.2 Analysis of Existing Caching Policies

From the constructive point of view, we analyze the caching policies surveyed in Section 2. Results are listed in Table 1, where $nref$ represents the reference frequency of an object, $size$ is the object size and $atime$ is the elapsed time since the object's last access.

Segmented LRU

Segmented LRU [4] use a classification rule based on reference frequency ($nref$). This is because objects with at least two accesses are for more popular than those with only one access. Cache space is partitioned to two segments: *probationary segment* and *the protected segment*. Objects with at least two accesses are kept in the protected segment, while new objects (with only one access) are first faulted into the probationary segment. When a probationary object gets one more reference, it will change to the protected segment.

Both unit caches are managed as LRU queues. When the whole cache space becomes full, the least recently used object in the probationary segment will first be replaced. The protected segment is finite in size and when it gets full, the overflowed will be *re-cached* in probationary segment.

Size-Adjusted LRU [2]

The Size-Adjusted LRU chooses a victim by sorting all objects in cache in terms of the cost-to-size ratio, $1/(size \cdot atime)$. It then greedily discards those with least cost-to-size ratios from the cache. Size-Adjusted LRU uses a pyramidal selection scheme (PSS) to manage cache space and object space.

The classification rules is based on $\lfloor log_2(size) \rfloor$, that is to say, objects within a same group are similar in sizes. Each group is maintained using a LRU mechanism. Though a hit will make the object move to the most recently used end, but an object can not move to another group. The computation of $1/(size \cdot atime)$ is done only to a limited set of least recently used objects from all nonempty groups and the object with largest $(size \cdot atime)$ will be purged from the cache.

4 Pyramidal Selection Scheme with award (PSS-W)

Object size has been considered as one of the most important features of web caching [3, 7]. Reference frequency is a strong indicator to the overtime popularity of web objects [7]. However, Segmented LRU takes advantage of the popularity information but fail to distinguish object sizes ($size$); whereas Size-Adjusted LRU generalizes LRU to handle variable sizes but fails to utilize reference frequency ($nref$).

We extend the cost-to-size ratio in Size-Adjusted LRU by incorporating frequency ($nref$). Sine each hit will reasonably increase the cost savings, we use ratio of $(nref/atime)$ to size and choose the object with minimum value of $(nref/(size \cdot atime))$. Based on this benefit-to-cost ratio, objects with more references are given larger benefit-to-cost ratio and can stay more time before aged out.

Now we can construct this new replacement policy. First, to simplify unit cache, the classification rule is chosen to be $\lfloor \log_2(size/nref) \rfloor$. Since, $nref$ changes with each hit, when $\lfloor \log_2(size/nref) \rfloor$ changed, an object may move to another unit cache. Compared to Size-Adjusted LRU, the $size/nref$ results in a small value in adjusting LRU choice and long stay in cache for an object with larger $nref$. For this reason, we call the new policy *Pyramidal Selection Scheme with aWard* (PSS-W).

Each unit cache is managed using a LRU policy. The $(nref/(size \cdot atime))$ is computed for the eviction candidates from all nonempty groups, purging the object with least $(nref/(size \cdot atime))$. The number of units is 24, because the objects larger than 16MB(2^{24}B) are very rare.

5 Performance Evaluation

Through trace-driven simulations, we evaluate the replacement polices listed in Table 1. The dataset used in our experiments is a one-week top level caching proxy traces publicly available (ftp://ircache.nlanr.net/Traces/). This dataset contains 1,848,319 requests with total 21.0 GB Web data, where unique data is 15.9 GB with a maximum hit rate 0.228 and byte hit rate 0.245.

The candidate replacement policies to be evaluated are LRV (Least Relative Value), SLRU (Segmented LRU), Pitkow(Pitkow/Recker algorithm) together with PSS-W. The results shown in Fig. 2. Plots in the left side depict the hit rates. The hit rate of PSS-W is much better than the rest. Plots in the right size are byte hit rates. PSS-W can achieve quite high byte hit rate.

Furthermore, the time complexity of PSS-W in servicing each request is a small constant in maintaining LRU queues and computing and comparing cost-to-size ratios. Thus, PSS-W is an ideal replacement policy for web caching.

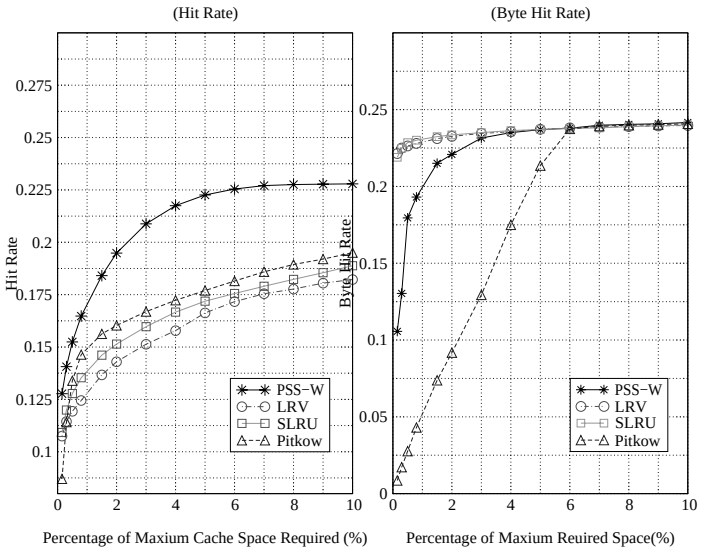

Fig. 2. Hit Rates and Byte Hit rates of various policies

6 Conclusion and Future Work

To cope with the complications in design and analysis of advanced replacement policies for web caching, we have proposed a constructive framework. This framework has been proven helpful in analysis of various current policies, and useful in making the design of PSS-W, a new efficient caching policy for Web caching. However, this framework has several aspects to be completed or improved. One of our future works is to study advanced classification rules since classification rules play a key role in construction of an advanced policies.

References

[1] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Wililams, and Edward A. Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of the Fourth International WWW Conference*, 1995.

[2] Charu Aggarwal, Joel L. Wolf, and Philip S. Yu. Caching on the World Wide Web. *IEEE transactions on knowledge and data engineering*, 11(1), 1999.

[3] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pages 193–206, December 1997. http://www.cs.wisc.edu/cao/publications.html.

[4] Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wheery. Caching Strategies to Improve Disk System Performance. *IEEE Computer*, 27(3):38–46, March 1994.

[5] T. H. Merrett and Yahiko Kambayashi. Join scheduling in a paging environment using the consecutive retrieval property. In *Proceedings of International Conference on Foundations of Data Organization and Algorithms (FODO)*, pages 323–347, 1981.

[6] James E. Pitkow and Margret M. Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Petterns. Technical Report VU-GIT-94-39, GVU Technical Report, 1994.

[7] Luigi Rizzo and Lorenzo Vicisano. Replacement Policies for a Proxy Cache. Technical report rn/98/13, University College London, Department of Computer Science, Gower Street, London WC1E 6BT, UK, 1998. http://www.iet.unipi.it/ luigi/ caching.ps.gz.

[8] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. In *Proceedings of ACM SIGCOMM'97*, pages 293–305, Stanford, CA, August 1997.