IMPLEMENTATION OF OPTIMIZED CACHE REPLENISHMENT ALGORITHMS IN A SOFT CACHING SYSTEM

J. Kangasharju, Y. Kwon, A. Ortega
Integrated Media Systems Center
University of Southern California
Los Angeles, CA

X. Yang*, K. Ramchandran
University of Illinois
Urbana-Champaign, IL

Abstract - In this paper we address practical issues which arise in the implementation of optimized cache replenishment algorithms within a "soft" caching framework.

We study the algorithms that have been proposed for optimized soft caching and simulate them using actual proxy traces. Our objective is to determine what compromises have to be made in order to approximate the desired optimal performance while maintaining a complexity level sufficiently low to enable a real-time implementation.

INTRODUCTION

With the popularity of the Internet and various web browsing applications, an increasing amount of image data is transmitted over the network. Proxy caches, or intermediate network nodes in which frequently used data are stored, emerge as an efficient solution to reduce network traffic. In a typical server-proxy-client scenario, the client makes requests through a proxy which retrieves the image from the server, delivers it to the client and caches a copy.

A question naturally arises as to how to efficiently allocate the proxy cache memory to achieve a minimum expected delay time. This so-called web caching problem has recently attracted a lot of research [4, 8].

The key difference between image-specific caching and conventional caching is as follows. Conventional cache management are "hard" because objects are either present in the cache or not, but cannot be partially available. Soft caching, however, is more flexible, in that variable amounts of cache memory can be assigned to each image. Therefore assuming that a progressive image format is used, a low resolution version of the original image can be kept in the cache. The main advantage of this approach is that, in many cases, the user will not need to download the full resolution image from the server.

In this work we study the implementation issues that arise in a soft caching scenario [1–3, 7, 9]. In [7], two soft caching scenarios are distinguished, namely,

*Currently with HP Labs, Palo Alto, CA

(i) hard access, in which the user always requires the entire image, and (ii) soft access, in which she can possibly choose to view the image at a certain resolution level. Obviously, soft-access-soft-caching represents the most realistic scenario, and is the focus of the current research.

In [9] efficient low-complexity algorithms to optimize the management of a soft cache are proposed. It is proved that the soft-access problem can be converted into a hard-access problem by breaking image resolution levels into separate subimages. Such a conversion is shown to maintain resolution-consistency in cache loading, i.e., the lower resolution version will always be loaded prior to the enhancement layers for higher resolutions.

In this paper we review the optimization algorithm of [9] and study the issues involved in implementing it in a real time system.

MAIN DERIVATION

Hard Access

Let us start with the simpler case of hard access. We consider a set of N images, and assume a known access probability $\{P_i\}_{i=1,N}$ to them. Image i has size R_i and is initially stored at a server of bandwidth B_{S_i}. A proxy cache of size C and transmission rate B_C is used as intermediate storage. Usually $B_C >> B_{S_i}$. The problem is how to assign cache memory to the images so that the average access delay is minimized. The following reviews the algorithm of [9] and the reader is referred to [9] for details.

Suppose image i is assigned r_{C_i} bits, for which $0 \leq r_{C_i} \leq R_i$. Hard access will always request the entire image of R_i bits. The total delay for accessing image i will then be

$$\delta_i(r_{C_i}) = B_C^{-1}.r_{C_i} + B_{S_i}^{-1}.(R_i - r_{C_i}) \tag{1}$$

The expected delay time to retrieve all the images is therefore $\sum_{i=1}^{N} P_i\delta_i(r_{C_i})$.

To minimize this, we imagine that the cache is filled at infinitesimal steps. If we allocate a very small amount of ϵ bits to image i, we achieve a delay reduction of $P_i(B_{S_i}^{-1} - B_C^{-1})\epsilon$. Obviously, to maximize this reduction we should pick the image that has the largest $P_i(B_{S_i}^{-1} - B_C^{-1})$. Therefore, the solution to the hard access problem is simply *to compute a priority index* $\beta_i = P_i(B_{S_i}^{-1} - B_C^{-1})$ *for each image, sort the images from large to small on* β_i. Then, when it is necessary to remove images from the cache this can be done following the order determined by the priority index.

Soft Access

Now, each image has a finite number of resolution levels. Resolution j of image i has r_{ij} bits ($0 < r_{ij} < r_{i,j+1} \leq R_i$), and access probability P_{ij}, with

$\sum_{i,j} P_{ij} = 1$. Suppose the cache holds r_{C_i} bits for image i, its total delay in this case is,

$$\delta_i(r_{C_i}) = \sum_{j:r_{ij} \le r_{C_i}} P_{ij} r_{ij} B_C^{-1} + \sum_{j:r_{ij} > r_{C_i}} P_{ij}(r_{C_i} B_C^{-1} + (r_{ij} - r_{C_i}) B_{S_i}^{-1}) \quad (2)$$

$\delta_i(r_{C_i})$ is piecewise linear and convex [7].

Consider the situation where each image has already been assigned r_i bits, in the range of resolution j, i.e., $r_{ij} \le r_i < r_{i,j+1}$, and the cache has some vacancy. If we allocate ϵ bits to image i, the reduction in average delay is:

$$\delta_i(r_i) - \delta_i(r_i + \epsilon) = (\sum_{l \ge j} P_{il})(B_{S_i}^{-1} - B_C^{-1})\epsilon \quad (3)$$

The first term in (3) is the probability that these ϵ bits in image i will get accessed. Obviously, we should give these ϵ bits to the image with the largest $(\sum_{l \ge j} P_{il})(B_{S_i}^{-1} - B_C^{-1})$.

The above analysis reveals a close tie between soft access and hard access: they are both solved using some priority indices. In fact, if we conceptually break image i into several subimages, let a subimage I_{ij} represent the difference between resolution levels j and $j-1$ of image i, and update the accessing probabilities accordingly, then the soft access problem is converted into a hard access problem.

In [9], it is shown that resolution consistency is maintained in cache allocation. In other words, subimage I_{ij} is always loaded into the cache prior to $I_{i,j+1}$.

DESIGN AND IMPLEMENTATION

Under the above conversion rule, every soft access problem is in fact a hard access problem with a larger set of images. In this section we explain how the access mechanism in our testbed can be formalized and a specific priority index assigned to each image.

Access type

Even though the soft access scenario is more appealing, all modern browsers implement only hard access. In order to have an access mechanism closer to soft access, the semantics of hard access need to be relaxed.

In our experiments we have the following access mechanism: the client requests an image within a page and receives the image available in the cache (which could be at a lower resolution) thus requiring time $B_C^{-1} \cdot r_{C_i}$. If the user is not satisfied with the image resolution a full reload is requested so that the image is downloaded *in its entirety* from the server. Thus the time required when such a "miss" occurs is $B_{S_i}^{-1} \cdot R_i$ where R_i is the size of the i-th image.

The optimized algorithm requires the access probabilities for each of the subimages. Under hard access these are not available and they will need to be approximated. Our approximation technique will be detailed later on.

Simple LRU-based algorithm

Cache replacement strategies have been extensively studied in the context of normal (i.e. hard) web caching [4, 8], with LRU [8] being one of the most popular approaches. In a LRU environment the basic idea is to remove the image (regardless of size, server bandwidth and other considerations) which was accessed the least recently.

While LRU is an attractive strategy given its simplicity, it is not always easy to implement an LRU strategy in a soft cache. In a soft cache, the image with the lowest priority will be *recoded, instead of being removed,*. Now, consider the image with oldest access time. This image will be then recoded, but if the same algorithm (i.e. finding the LRU image) is used again, it is likely that this image *will remain the LRU*. Thus, if the recoding level is not considered as part of the removal decision the result will be that images are recoded repeatedly until the lowest resolution is reached and they are removed.

Therefore the access time of the recoded images needs to be adjusted as a function of the recoding level. For this we use the following formula in which the new access time is

$$new = old + \frac{remaining\_recoding\_levels}{max\_recoding\_level}(current\_time - old) \qquad (4)$$

The new access time for an image that has been recoded only a few times ($remaining\_recoding\_levels$ is large) is close to the current time, which increases its priority in the LRU algorithm. A heavily recoded image is probably not very useful and its new access time is set closer to its old access time. This does not increase its priority very much and therefore it will be removed (or recoded again) sooner.

Optimized sorting based algorithm

Let us now consider a practical implementation based on the optimal replacement criterion. The priority indices are defined as

$$\beta_{ij} = P_{ij}(B_{S_i}^{-1} - B_C^{-1}) \qquad (5)$$

Because exact values for the three parameters, P_{ij}, B_{S_i} and B_C are not available, they will be estimated in the following ways.

All the clients are assumed to be connected over similar links. Therefore the client bandwidth, B_C, is constant and does not need to be estimated.

The bandwidth to server i, B_{S_i}, is estimated from the size of image i and the time it took to retrieve it from the server, averaged over subsequent

retrievals of image i. This is an imperfect estimator, since using all images from server i would yield a better estimate, albeit still not necessarily correct. This, however, would require a dedicated data structure for each server and would be far costlier in terms of memory. The Squid proxy used as the basis for our testbed [6] expresses the image size in bytes and the duration of the request in milliseconds giving as the natural unit of bandwidth bytes per millisecond.

The main problem is estimating the values of P_{ij}. These can be estimated by the number of times the object has been referenced, but because of hard access, we do not have accesses to anything but the last resolution. Therefore all the lower resolutions share this same reference count. This would reduce the replacement algorithm into a hard replacement algorithm. Also because the reference count is reset to 0 every time the object is released from the cache, the estimate is somewhat inaccurate. For a better estimate, it would be necessary to keep the reference counts of every object ever seen by the cache, but this is completely infeasible in practice.

When an image is sent to the recoder, we simulate soft accesses by increasing the reference count of the next highest level. The average bandwidths in the experiments were around 4.5 bytes per millisecond and the soft accesses were simulated by adding 0.1 to the reference count. The effect from this is that the lower resolutions have a higher priority index and do not get recoded right away. It should be noted that for normal objects, only a single priority index is calculated, but such objects are not recoded, instead they are removed from the cache.

SIMULATIONS

We conducted simulations using a trace from our own Squid-proxy at our lab. This trace contained almost 150000 URLs and 1.4 gigabytes of data. Recoding was performed only on JPEG-images which represented 16 % of the bytes. The different replacement algorithms were run over different cache sizes.

Figure 1 shows the average download time of an object when using one of four different replacement policies. LRU means a standard LRU, LRU-SOFT refers to LRU with the access time modification (4), OPT-HARD is the optimal replacement policy without the simulated soft access, i.e. working as a hard replacement policy, and OPT-SOFT is the same policy with simulated soft access, i.e. recoding images and increasing their reference counts.

The results show that even with the sub-optimal estimators, the optimal replacement policy has the shortest average download time. Also the simple modification to LRU yields a shorter download time than stock LRU. As cache size increases, any cache can hold all of the cacheable traffic, thereby the gains disappear.

More results are available on our web site [5] and more will be made available as they are generated. This web site also contains a patch for the Squid

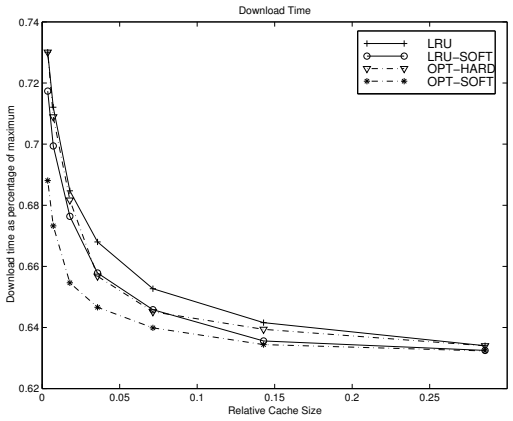

Figure 1: Relative gain in download time

proxy to make it operate as a soft cache.

References

[1] J. Kangasharju, Y. Kwon, and A. Ortega. Design and implementation of a soft caching proxy. In *3rd WWW Caching Workshop*, Manchester, UK, June 1998.

[2] A. Ortega, F. Carignano, S. Ayer, and M. Vetterli. Soft caching: Web cache management for images. In *IEEE Signal Processing Society Workshop on Multimedia*, Princeton, NJ, June. 1997.

[3] A. Ortega, Z. Zhang, and M. Vetterli. A framework for the optimization of a multiresolution remote image retrieval system. In *Proc. of Infocom*, Toronto, June. 1994.

[4] J. Shim P. Scheuermann and R. Vingrale. A case for delay-conscious caching of web documents. In *Proc. Intl. WWW Conf*, Santa Clara, CA, Apr. 1997.

[5] Soft Caching Project Page. http://sipi.usc.edu/~ortega/softcaching/.

[6] Squid Home Page. http://squid.nlanr.net/squid/.

[7] C. Weidmann, M. Vetterli, A. Ortega, and F. Carignano. Soft caching: Image caching in a rate-distortion framework. In *Proc. of ICIP*, Santa Babara,CA, Oct. 1997.

[8] R. P. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *Proc. Intl. WWW Conf*, Santa Clara, CA, Apr. 1997.

[9] X. Yang and K. Ramchandran. An optimal and efficient soft caching algorithm for network image retrieval. In *Proc. of ICIP*, Chicago, IL, Oct. 1998.