Replacement policies for a proxy cache

Paolo Lorenzetti Luigi Rizzo
Lorenzo Vicisano
Dipartimento di Ingegneria dell'Informazione
Università di Pisa – via Diotisalvi 2 – 56126 PISA
email: `l.rizzo, l.vicisano@iet.unipi.it`
Draft – please take the updated version from
`http://www.iet.unipi.it/~luigi/caching.ps.gz`

Abstract

In this paper we analyze access traces to a Web proxy, looking at statistical parameters to be used in the design of a replacement policy for documents held in the cache. We first present a number of interesting properties on the lifetime and statistics of access to documents, which show why LRU works reasonably well but can be improved. We then propose an alternative replacement policy, called LRV, which selects for replacement the document with the Lowest Relative Value among those in cache. The value of a document is computed basing on information readily available to the proxy server, and the computations associated with the replacement policy requires only a small constant time. We show how LRV outperforms LRU and other policies, and can significantly improve the performance of the cache, especially for small ones.
Keywords: communication networks, Web, caching, replacement algorithms

1 Introduction

The caching of Web documents is widely used to reduce both latency and network traffic in accessing data. A first-level cache is usually built into the browser, which allocates a small amount of memory and disk space to store frequently accessed documents. The browser's cache has a limited size and it is only used by a single client, thus being mainly useful to store images (backgrounds, icons, etc.) which occur frequently in a set of related documents. A second-level cache is provided by caching proxy servers (*proxies*), which retrieve documents from the original site (or another proxy) on behalf of the client. Proxies serve a large set of clients, so they can use the history of requests for a document as hints on its popularity and decide if it is worthwhile storing it in the local cache. The large storage area available on the proxy can give large improvements in the access to remote documents. Several proxy caches have been developed in recent years. Among them, `cern_httpd` [6], `harvest` [3], and its successor `squid` [8] are widespread programs which are available in source format. The latter two use a number of ingenious solutions to improve performance, and introduce the concept of cooperating proxies to make the system perform even better.

There are however some limitations in the gains deriving from the use of a proxy. Some Web documents are intrinsically uncacheable, usually because they are the output of a program's execution, or they are marked as uncacheable by the originating server. We will not consider uncacheable documents in this paper, as a proxy has no way of optimizing accesses to them. Once uncacheable documents have been discarded, it turns out that a relatively large fraction of documents is accessed only once; storing them on the proxy is a waste of space, but they cannot be easily distinguished from more popular documents at their first access. In our logs, roughly 2/3 of documents is accessed only once, although these accesses account for only 1/3 of the total. Since the proxy has finite storage, some strategy must be devised to periodically purge documents which are not interesting, in favor of more popular ones. The problem is essentially one of guessing, among the documents currently in cache, which one is best to remove in order to make room for new documents.

The problem of finding a suitable replacement policy for a proxy cache has received limited attention this far, for a number of reasons: the apparent similarity of this problem with other caching problems (e.g. in processor architecture or operating systems); the relatively recent introduction of the Web; and, especially, the fact that the most common technique to manage caches – LRU – has proved to work reasonably well in the context of proxy caches. Related work is discussed further in Section 4.

In this paper we develop a cost/benefit model to determine the relative value of each document in the cache, so as to allow the replacement algorithm to select the document with the Lowest Relative Value (LRV algorithm). The value of each document is computed on the basis of statistical parameters collected by the server. We will show that most of such parameters can be computed 'on-line' by the cache itself at a negligible cost.

The rest of the paper is organized as follows. In Section 2 we introduce the problem formally, and define the basic cost/benefit metric. Section 3 analyzes the probabilistic parameters which influence the value of each document, discussing how they affect already-known replacement algorithms. A discussion on related work is presented in Section 4. In Section 5 we present the LRV algorithm together with a discussion of its implementation. The performance of LRV, compared to other algorithms, is then presented basing on real traces and for a number of different cache sizes.

2 Problem's definition

A proxy aims to minimize two parameters: the response time in serving documents to a client, and the network traffic to the outside. The *hit rate* (*HR*) and *byte hit rate* (*BHR*) are two parameters which describe reasonably the effectiveness of the proxy. They indicate the fraction of documents and bytes, respectively, which are served from the cache instead of requesting them from the network. *BHR* is a direct measure of the savings in network traffic. In computing these two parameters, uncacheable documents are usually discarded. The response time in serving documents also depends strongly on both *HR* and *BHR*.

We call *history* the history of accesses to a proxy. The history holds information on the URL, size, type, requestor, transfer time for a document. The history is usually available through the access logs recorded by the proxy server. We call *future history* the history of the accesses issued after a given time t_0; the remaining part of the history is the *past history*.

With respect to a given time t_0, the *accessed* documents are all documents whose URLs appear in the past history. We call *dead* documents the subset of accessed documents which do not appear in the future history. Document die because they are deleted at the source, change their content (thus becoming new documents), or simply because nobody accesses them anymore. *Live* documents are the subset of accessed documents which also appear in the future history.

Let D_i be the set of documents accessed at least i times. A parameter which we will often use is $P_i = \frac{\|D_{i+1}\|}{\|D_i\|}$, corresponding to the probability that a document is accessed again after the $i-th$ access. P_1 is a direct indication of the percentage of documents for which caching is useful. The D_i's and P_i's can be computed on a subset of documents, in which case we add a suffix to the names to indicate the selection criterion.

In order to increase the hit rate, a proxy might try to prefetch documents, anticipating clients' requests, so that even the first request for a given document is served by the cache. In principle, this approach might have no influence on the total amount of network traffic, bring the hit rate to 100%, and reduce the latency experienced by clients in accessing documents. In practice, anticipating clients' requests is non trivial, prone to errors, and it usually has high costs in terms of additional network traffic and storage use. In [7] Padmanabhan and Mogul propose a prefetching scheme in which clients and servers cooperate, achieveing a substantial reduction in latency at the cost of an increase in the network traffic. However their algorithm requires modifications both on the server and on the client and some minor additions to the HTTP protocol. In this paper we will not consider prefetching proxies.

Both our traces and other data available in the literature show that the maximum hit rate – both in documents (HR_{max}) and in bytes (BHR_{max}) – that we are generally dealing with is much lower than unity. Given the history of accesses to a proxy, we can compute HR_{max} and BHR_{max}. Such hit rates will be easily achieved by a cache with sufficient storage to hold accessed documents (in principle, it would suffice to hold all live documents, if they were distinguishable from others). With caches of finite size, we generally need to purge documents from the cache to make room for new ones. Given sufficient disk space, an optimal replacement policy would purge dead documents and keep all live documents. Clearly, such a decision can only be taken if the future history of accesses is known. Such a caching policy is able to achieve both BHR_{max} and HR_{max} if there is enough room to store the set of all live documents.

If the cache is not large enough to store all live documents, optimal policies can still be defined as those maximizing *HR* or *BHR*, and can be used as a reference in evaluating the goodness of a cache replacement algorithm. Note that even if future history were known – which is not – an optimal policy is computationally too hard to implement.

A consistency problem always arises when using caches. Consistency in Web caching is faced using some support provided by the HTTP protocol [2] and some heuristics implemented in the caches [6, 3, 8]. An analysis of consistency issues is given in [5]. In the following we do not deal with the consistency problem as it is ortogonal with respect to replacement policies: we view the Web as a read-only system and consider modified objects as brand new ones.

2.1 A cost/benefit model

Purging a documents from the cache has a benefit and a cost. The benefit B essentially comes from the amount of space freed, which is roughly proportional to the size of the document plus its metadata, possibly rounded to a multiple of the file-system's block-size.

The cost can be expressed as the cost, C, of fetching the document from the original site, multiplied by P_r, the probability that the document is accessed again in the future. Several different metrics are commonly used to compute C:

connections Each retrieval of a document has the same cost, so we can assume $C = 1$. This is only appropriate when the cost of setting up a communication with the server dominates over other costs.

traffic In this case C =size(document)+overhead, i.e. the amount of traffic needed to establish a communication, request the document and get the response. This metric is appropriate when the communication speed is independent from the source of a document, or communication costs are based on the number of bytes transferred.

time In this case the cost of a retrieval is proportional to the time needed to fetch the document, so $C = t_{fetch}$. This number corresponds roughly to product of the size of the document and the inverse of the bandwidth towards the server. This metric is useful when the bandwidth to different servers has large variations, or we want to minimize the delay in serving documents to clients.

Given the cost and benefit in *purging* a document, we want to determine how *valuable* is the document for the proxy. To this purpose, we can define the value, V, of a document (relative to other documents in the cache) as

$$V = \frac{C}{B} P_r$$

The computation of C and B is relatively easy, as it only depends on the size of the document and possibly the bandwidth towards the server; these values are readily available to the proxy. Using metrics based on traffic or time, C/B is approximately constant and independent of the size of the document. A metric based on connections tends to overestimate small documents; since a cache is usually sized in bytes, not in number of documents, this metric improves the HR, but certainly not the BHR (which corresponds to saving communications, something of interest to those who run the proxy), or the response time to clients (which is of interest to clients).

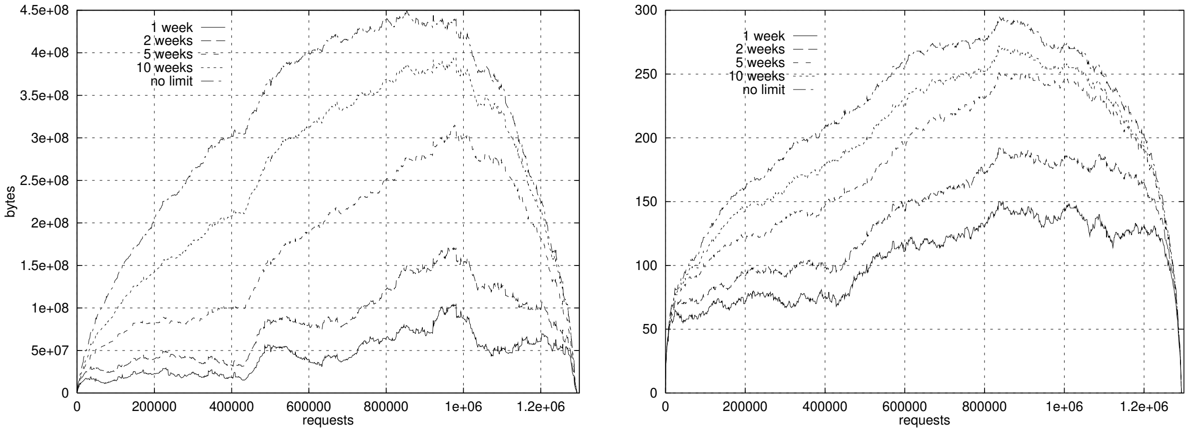

Figure 1: Left: total size of live documents vs. requests; right: active host clients vs. requests.

The computation of P_r is more complex. P_r is in general different for each document, and is time-dependent. What we try to do in the next sections is to determine if it can be estimated by looking at the document itself (size, type, server etc.) and at the history of previous accesses to the document, as seen by the proxy.

3 Trace evaluation

We have based our analysis on the history of cacheable accesses recorded by our departmental proxy over a period of five months. The trace – about 1,300,000 accesses – includes about 1,000 clients, 20,000 servers, 450,000 different URLs. We are well aware of the existence, on the Internet, of some large sites running proxies with 10..1000 times our traffic. We do not expect all of our results to hold for these very large proxies (although some of the properties might well be the same). Nevertheless, our traces are significant because they are representative of a probably much larger number of installations where a proxy serves a relatively small community of local users.

The set of all accessed documents in our data set amounts to about 7GB of data. Only a small fraction of the documents is accessed more than once ($P_1 = 0.325$), but $HR_{max} = 0.66$, meaning that documents accessed more than once feature a large number of accesses. $BHR_{max} < HR_{max}$ means that short documents are reaccessed more frequently.

3.1 Live documents

It is unreasonable to think that a cache can store all of the accessed documents, no matter how big are its disks: in fact, the number (and total size) of accessed documents grows with time and with the number of clients. However, it is only necessary that the cache retains live documents to achieve the maximum hit rate. Live documents are a small fraction of the total, and they reach – in our dataset – a maximum occupation of 450MB (see Figure 1, left), when computed on the whole length of our traces. It is probably more appropriate to consider dead those documents which have not been requested for more than some reasonably large time (e.g.1-2 weeks). For reference, Figure 1 also shows the size of live documents when limiting the lifetime of a document to 1..10 weeks. The figure shows that reducing the lifetime of documents to 2 weeks reduces the size of live documents to 1/3; it is noticeable that HR_{max} and BHR_{max} computed with such a lifetime only decrease by about 10%.

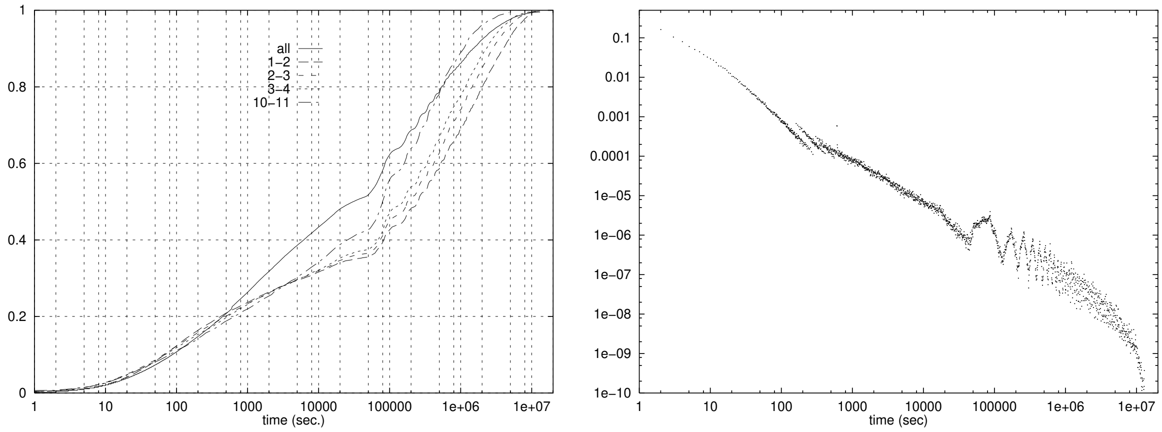

Figure 2: Distribution $D(t)$ (left) and probability density function $d(t)$ (right) of interaccess times. $D(t)$ is computed both for i/(i+1)th access and all accesses to the same document.

The number of active clients over time is shown by Figure 1, right (by considering inactive a client which does not issue requests for longer than some time $t = 1..10$ weeks). The curves representing the size of live documents and number of clients have a similar shape; this suggests that the size of live documents is only a function of the number of clients; as such, it should remain relatively constant over time for a fixed number of clients. The relation between the size of live documents, clients and hit rates is still under investigation.

Another noticeable effect is that, over time, new documents are generated and old ones die with roughly the same, constant, rate. The actual rate (approximately 1/3 of all requests in our case) should depend on the type and number of clients. A low birth/death rate means either large overlaps in requests coming from different clients, or the ineffectiveness of the first-level cache implemented in clients. Our data suggest that the latter is true, since further trace analysis has shown relatively little overlap among the interests of our 1000 clients.

Having characterized the information contained in our log files, we can start now the analysis of the parameters which influence the probability P_r of a new access to the same document. In the next section, we compute the conditioned probabilities of a document being accessed again, depending on various known information such as the time from the previous access, the number of previous accesses, the server of the document, the client originating the first request, etc. Our goal is to find one or more parameters which allow us to decide which documents is more convenient to keep in cache.

3.2 Interaccess times

Figure 2 shows the distribution $D(t)$ and the probability density function $d(t)$ of times between consecutive requests to the same document. The time axis is logarithmic to ease the reading of the graph. We can identify at least two distinct regions: the first one covers up to one day from the previous access, while the second one covers the remaining time scale.

Globally, about 60% of accesses occur within one day, with a marked peak around 24 hours, and a relatively flat area between 8 and 18 hours. This is not surprising because our users reside within the same timezone, and most of accesses occur during office hours. About 20% of accesses occur in the first 15 minutes, and 10% in the first minute. This is an indication of frequent reloads, and possibly a consequence of a feature of `harvest` and `squid`, which can retrieve documents in the background: some users often do a first pass touching interesting documents, then request them again at a second time when the transfer is presumably complete.

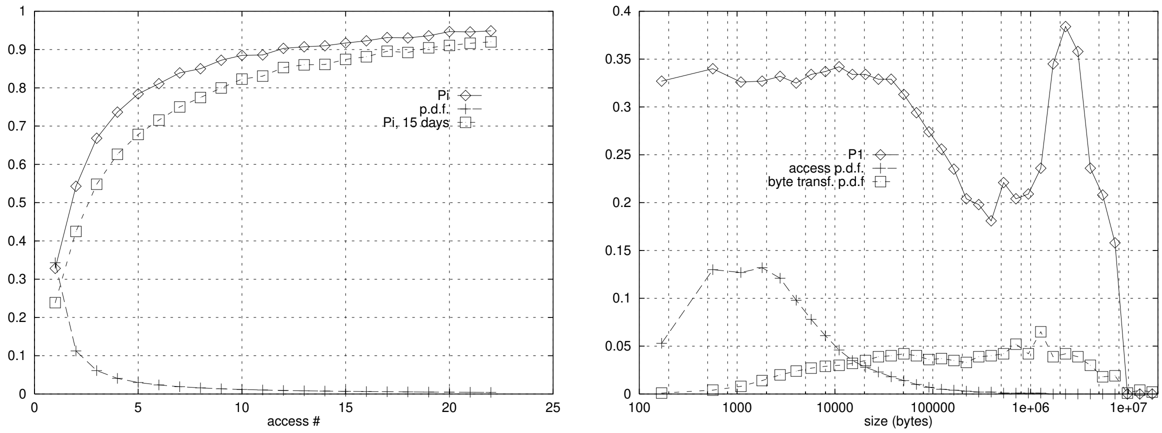

Figure 3: Left: The probability of more accesses after i previous ones (P_i). The line on the bottom shows D_i/D_1, the percentage of accesses in each set D_i. The third curve shows P_i, computed without considering reaccesses issued after 15 days. Right: Percentage of documents with at least 2 accesses (P1) vs. document size. Also access and bytes transferred p.d.f. vs. size.

In the second area (from one day to the end) there is an approximately exponential decay of interaccesses. The daily peaks are still visible (especially in $d(t)$) but have a decreasing amplitude.

Figure 2 also shows the distribution of interaccess times for the i-th/$i+1$-th pair. Despite some unavoidable differences (documents with more accesses have lower interacces times), the curves retain approximately the same shape. The dependency of P_r on t, the time from the last access, can be expressed as $P_r(t) = 1 - D(t)$. Being $D(t)$ a distribution function, $P_r(t)$ always decreases with time, independently from the shape of $D(t)$; thus, a policy such as LRU is always the best one if we only consider the time from the last access in our replacement policy.

However, we are interested in the actual shape of $D(t)$ because we want to use it together with other parameters in the computation of P_r. To have a suggestion on how to approximate $D(t)$ we can look at its derivative $d(t)$ (the p.d.f. of interacces times). It turns out that $d(t)$ can be reasonably approximated with k/t in the first part ($t < 1$ day), while in the last part it is better approximated by $e^{-\alpha t}$. Integrating the two functions yields the following approximation for $D(t)$:

$$D(t) = .035\log(t+1) + .45(1 - e^{-\frac{t}{2E6}})$$

where the constants have been determined experimentally on our data set. Intuitively, the logarithm has greater weight for $t < 1day$, while the exponential dominates the behavior in the remaining area. It should be pointed out that the approximated value for $D(t)$ is not formally correct because it can become larger than 1, but this has no practical consequences, as the function remains monotonically increasing with t. Besides, very long interaccess times are probably not meaningful and might be truncated anyways.

3.3 Number of previous accesses

If we look at the various P_i, the conditioned probability of a document being reaccessed when it has been accessed i times, we see (Figure 3, left) that their values grow significantly with i, up to 0.9 and more for $i > 12$. Hence, the number of previous accesses appears to be a good indicator of the probability of a new access. However, using this parameter alone to determine P_r is a bad idea, because this does not permit the aging of documents, something that is absolutely necessary in order to avoid pollution of the cache. From Figure 2 we see that the dependency on time still remains in $i/i+1-$th access pairs, so P_r can be computed as $P_r(i,t) = P_i \cdot (1 - D_i(t))$; $D_i(t)$ can be further approximated with $D(t)$. As a result, the number of previous accesses only modifies the LRU ordering for documents with a comparable time from the last access. In some cases, a document with many past accesses can have a higher probability of being reaccessed than a much more recent document with just one or two accesses. The use of P_i does not change the LRU ordering within classes of documents with the same number of accesses.

As the largest set of documents – those with one access – belongs to the same class, we would like to make further distinction among them basing on some other parameter. Documents with only a single access carry the following information: size, document's URL (which can be decomposed in protocol, server, document type), and the client originating the request.

3.4 Document size

Figure 3, right shows the dependency of P_1 on the size of a document. It is well known that short documents tend to be preferred over long ones, especially by clients working over slow connections. This explains why a large fraction of requests refers to documents with a size of 5KB or less (Figure 3, access p.d.f), for which the delays in transferring the document over a slow modem line are still bearable. Below this threshold, differences in size do not affect significantly the behavior of clients. Large documents, taking much longer to download, might discourage clients from accessing them, but in some cases they are still popular (e.g. in the case of software distributed over the net). This also influences the value of P_1: as shown by the graph, P_1 is slightly larger for sizes $< 30KB$, while documents in the 30KB-1.5MB range are definitely less popular. The peak around 2MB corresponds to large software packages distributed through the Web.

The dependency on the size is an useful parameters to make a selection among documents with one access for at least two reasons. First, small documents (those with a relatively higher P_1) take less space in the cache, so that we have more room for documents with two or more accesses, for which the probability of reaccess is generally higher. Second, this parameter partitions documents in groups of comparable sizes with different P_1, so that the use of this parameter has more influence on the performance of a replacement algorithm.

3.5 Other parameters

Document's source

The dependency of P_r on the source of the document has been evaluated using the following technique. For each server, we keep the value $P_{1S} = \frac{D_2}{D_1}$, computed on documents coming from the same server. Documents are then partitioned in 10 groups, depending on the P_{1S} of their server *at the time of the first access to the document*. Finally, the P_1 is computed for each group. Figure 4, left, shows the dependency of P_1 on the P_{1S}. If documents were allocated to a group after the second access to the document, the graph would be a straight line between 0 and 1. Since the value of P_{1S} for a given document is computed at the time of the first access, the graph deviates from the linear behavior. In general, a high P_{1S} for a server means a high chance that a document coming from that server is accessed again. Note that we have used P_{1S} rather than the hit rate to classify servers: with the latter, even a single, very popular, document could bring the parameter arbitrarily close to 1.

While there appears to be some correlation between P_1 and P_{1S}, the problem with this parameter is that a large number of documents comes from servers with too little information to get reliable statistics (Figure 4, "access p.d.f." shows the percentage of documents belonging to each class).

Client requesting the document

The dependency of P_r on the client which first requests a document has been evaluated with a technique similar to the one used for servers. Figure 4, right shows the dependence of P_1 on the P_{1C} computed for documents requested by the same client. As in the case of P_{1S}, the graph has been computed by only considering clients with a sufficiently large number of requests.

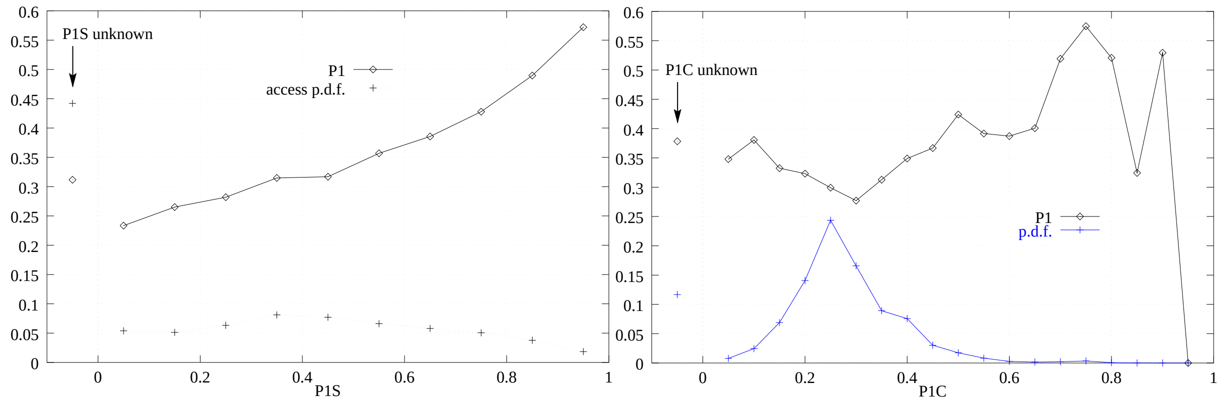

Figure 4: Left: Percentage of documents with at least 2 accesses (P_1) computed on a per-server-class basis. Documents are grouped basing on the average P_1 computed on the set of documents coming from the same server (P_{1S}). Below, the percentage of documents in each class. Right: Percentage of documents with at least 2 accesses (P_1) computed on a per-client-class basis. Documents are grouped basing on the average P_1 computed on the set of documents requested by the same client (P_{1C}). Below, the percentage of documents in each class.

Here, a low P_{1C} means a properly working cache on the client (which requests most documents only once), while a large value is a clear indication that the cache on the client is ineffective. Documents requested by clients with a low P_{1C} appear to have a marginally higher P_1, but the difference is too small to be useful. On the other hand, clients with a large P_{1C} are simply not reliable because what they try to do is to use the proxy as their first level cache, an approach which is not scalable and thus should not be used to influence the behavior of the proxy.

Type of document

We have computer the values of P_1 per type of document. Almost 40% of the requested documents are `gif` files, and for them $P_1 = 0.36$, slightly larger than the average. Overall, the most popular documents (graphics, home pages, html files) account for the vast majority of files and all have a similar P_1. These results are consistent with those found in [1] and [9]. This parameter is not useful in building a cache replacement policy since the only documents which have a significantly different (and lower) P_1 are archive files, which have a large average size and are already acted upon by size-based decisions.

3.6 Summary

At this point it is worth summarizing what we believe are general enough features of accesses to a proxy server.

- there appears to be little overlap among requests coming from different clients.
- despite this, proxy hit rates are relatively high because clients tend to use the proxy as an extension of their first level cache.
- the distribution of interaccess times for the same document decays rapidly with time. A significant fraction of reaccesses is concentrated in a few minutes' interval. Also, some periodicity appears to exist in access patterns.
- the probability of reaccess appears to depend heavily on the number of previous accesses, and more weakly on the size of the document;

- some parameters, such as document's source, or the client requesting the document, do not appear to influence significantly the probability of reaccess;
- the probability of reaccess depends on the file type, but this parameter gives approximately the same information as the file size.

4 Related work

Very few studies exists on the caching algorithms for the Word Wide Web, mostly because of its recent introduction. Some useful hints can be derived from similar works issued in the field of file-system caching. Commonly used policies include Least Recently Used (LRU), Least Frequently Used (LFU or NREF), policies based on the SIZE of objects, or combinations of them.

In [4] Karedla et. al. pointed out one of the main weaknesses of LRU, namely the fact that the cache can be flooded by documents that are referenced only once, flushing out documents with higher probability of being reused. This situation is very likely to occur in Web caches, where, as we observed, references to object accessed only once make up 2/3 of the total references. As we did in the case of Web caching, the authors of [4] observed that the probability of an object being referenced again quickly grows after the second reference. That accounts for a good behavior of NREF policy. Nevertheless NREF prevents 'dead' document, with large reference counts from being purged. This causes the so called *cache pollution* phenomenon and yields a reduction of the effective cache size. An aging policy is often used to cope with this problem. [4] also presents a number of frequency-based variants of LRU, devised to cope with LRU weakness. These are hybrid LRU-LFU policies that allows to exploit better than LRU the object access history.

The problem of determining an efficient replacement algorithm for a proxy cache has been studied in [9]. The authors present a taxonomy of removal policies, basing on the observation of traces corresponding to five different workloads. They analyse the use of hierarchical keys to keep documents sorted in the cache, and conclude that document size outperforms all other possible keys. However, they use relatively short periods of observation, a limited number of different clients, and the cache size (10% of the total size of accessed documents) used in the experiments presumably exceeds by far the set of live documents. In policies based on the size of a document, there is little or no chance that small documents can be discarded; as a consequence, in the long term, the cache fills up with small, old documents, and its performance decays. The phenomenon is only visible when such garbage brings the useful space in the cache well below the size of live documents, while the parameters used in [9] cannot evidence this phenomenon. As a consequence, the results presented in [9] need validation with more significant experiments. As also noted by the authors, a problem with hierarchical keys is that it is difficult to sort the keys by importance, and a key can be effectively used only if all the previous ones give rise to frequent ties. As a consequence, often only the first key is effective.

5 The LRV algorithm

In Section 2.1 we have introduced the relative value of a document (V) as a function of P_r, the probability that a document is accessed again. In Section 3 we have discussed how various parameters can be used to compute P_r.

In this section we present the LRV algorithm, based on the relative value (V) of a document, and we show how to obtain an approximation of V suitable for our purposes. The LRV algorithm simply selects the document with the Lowest Relative Value as the most suitable candidate for the replacement. As already seen, the relative value, V, of a document is proportional to P_r, thus, in the end, the issue is evaluating this probability.

Section 3 has evidenced some of the features of the accesses to a proxy server. It has been shown how P_r is strongly influenced by the time from the previous access, but also other parameters have some impact on this probability. A class of documents which is particularly difficult to handle is the one containing documents which have been accessed only one: it comprises many documents which generally have a low probability of being accessed again.

To simplify the problem of computing P_r, we have made the simplifying assumption that the variables defining the document state are independent, this way we can express P_r as the product of the various probabilities. Moreover, in order to keep the algorithm simple, we have tried to use as little parameters as possible and we have neglected the dependence of P_r on some of the chosen parameters in certain circumstances. We have only used those parameters which, after extensive simulations, have shown a significant improvement in the performance of the algorithm. Finally, when possible, we have tried to make the algorithm fast, and to be able to compute its parameters adaptively, rather than using precomputed probabilities which might not suit all situations.

The parameters that we have selected are the following:

- *time from the last access, t*, for its large influence on the probability of a new access. As shown in Section 3.2, the probability of a new access conditioned to the time from the last access can be expressed as $1 - D(t)$. For the difficulty in computing $D(t)$ at run time, here we did not succeed in our goal of avoiding precomputed values. The approximation presented in Section 3.2 is the one which, in the simulations, has given the best results. The choice of the function and the the values of the constants have been motivated in Section 3.2. We believe that this approximation is sufficiently general, since it tries to model the bimodal nature of $D(t)$, and the boundary between the two regions, at $t = 1$ day, is related to users' behaviours.
- *number of previous accesses, i*. From Figure 4 we see that this parameter allows the proxy to select a relatively small number of documents with a much higher probability of being accessed again. The computation of P_i can be done very easily by the server, and only a few classes are necessary as P_i's tend to saturate for small values of i. Simulations have shown that the use of this parameter has a great benefit on the performance of the cache.
- *document size, s*. This seems to be the most effective parameter to make a selection among documents with only one access. Thus we use it only for these documents. As the policy based on document size tends to discard large documents, this also allows us to employ less storage for document with one access, saving space for the other documents (which generally have a higher probability of being reaccessed). As in the case of the P_i's, $P_1(s)$ can be computed adaptively by the proxy at little cost.

We compute $P_r(i,t,s)$ as follows:

$$P_r(i,t,s) = \begin{cases} P_1(s)(1 - D(t)) & \text{if } i = 1 \\ P_i(1 - D(t)) & \text{otherwise} \end{cases}$$

Note that we neglect the dependence of $P_r(i,t,s)$ on s when $i > 1$. For a proper implementation of LRV, especially for the computation of P_i, it is required that metadata for documents are retained longer than the associated documents. Since metadata take on average about 1-2% of the space taken by documents, this does not constitute a problem.

5.1 Implementation details

The implementation of LRV can be highly efficient. In fact, $(1 - D(t))$ is a monotonic function of t, P_i's only assume a small number of different values, and $P_1(s)$ can be similarly quantised (on s) and represented with a small number of different values. As a consequence, the algorithm can be implemented by holding documents in a small number of queues (one for each value of $P_i/P_1(s)$). Updates are only necessary when a document is accessed, and require one extraction and one head insertion, both constant-time operations. A noticeable property of this organization is that P_i's and $P_1(s)$'s can be dynamically computed by the proxy, rather than requiring hardwired values which might not suit all settings.

The selection of which document to replace when a memory shortage occurs is similarly simple: it suffices to make a single pass over all tail elements to determine which one has the lowest value. Since a

BHR	100M	250M	500M	750M	1G
LRU	0.320	0.374	0.413	0.434	0.446
FIFO	0.308	0.359	0.397	0.418	0.431
SIZE	0.212	0.277	0.331	0.365	0.392
NREF	0.313	0.368	0.402	0.419	0.434
LRV	0.347	0.393	0.429	0.446	0.458

HR	100M	250M	500M	750M	1G
LRU	0.406	0.483	0.538	0.566	0.584
FIFO	0.381	0.454	0.508	0.537	0.558
SIZE	0.419	0.525	0.592	0.620	0.635
NREF	0.456	0.518	0.558	0.576	0.592
LRV	0.482	0.539	0.577	0.597	0.610

Table 1: The values of BHR and HR for different policies and cache sizes

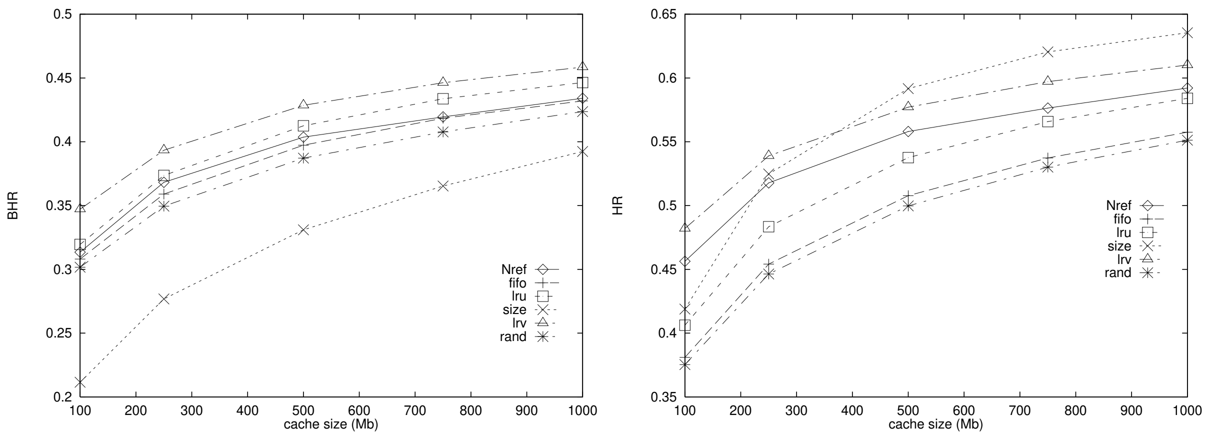

Figure 5: The values of BHR and HR for different policies vs. the cache size.

constant number of queues exists, this is also a quick, constant-time operation, which allows storage to be reclaimed on demand rather than setting thresholds which trigger the execution of expensive garbage collection phases, and reduce the effectiveness of use of memory.

5.2 Performance

The performance of LRV, compared to other algorithms, such as LRU, size, FIFO and others, has been simulated basing on traces of accesses to our proxy. Initial evaluations have been done using a general purpose simulator for networks of cooperative proxies [10]. Subsequent experiments have been done with specialized C programs, much more efficient although less flexible. One of these programs contains a full implementation of *LRV* and all the other policies that have been evaluated. The various programs process sanitized traces where strings (e.g. URL, client names etc.) are replaced by unique numbers, thus allowing a very fast processing.

The performance, for different cache sizes, is shown in Table 1. It must be kept in mind that the set of documents accessed at least once amounts to 7Gb of data, while the set of live documents is always lower than 450MB (see Figure 1), hence we have used comparable cache sizes (100MB..1GB). For simplicity, we have run garbage collection between two thresholds set at 90% and 100% of the cache size.

Table 1 and Figure 5 show the values of HR and BHR for different policies and cache sizes. As it can be seen, LRV features a consistently higher BHR than other policies in all conditions. The same happens for the HR, except in the case of the SIZE policy with caches $\geq$ 500MB (but this is not accompanied by a comparable BHR). Reducing the cache size causes the SIZE policy to worsen because of the pollution of the cache with small documents which are never replaced. This phenomenon does not appear with

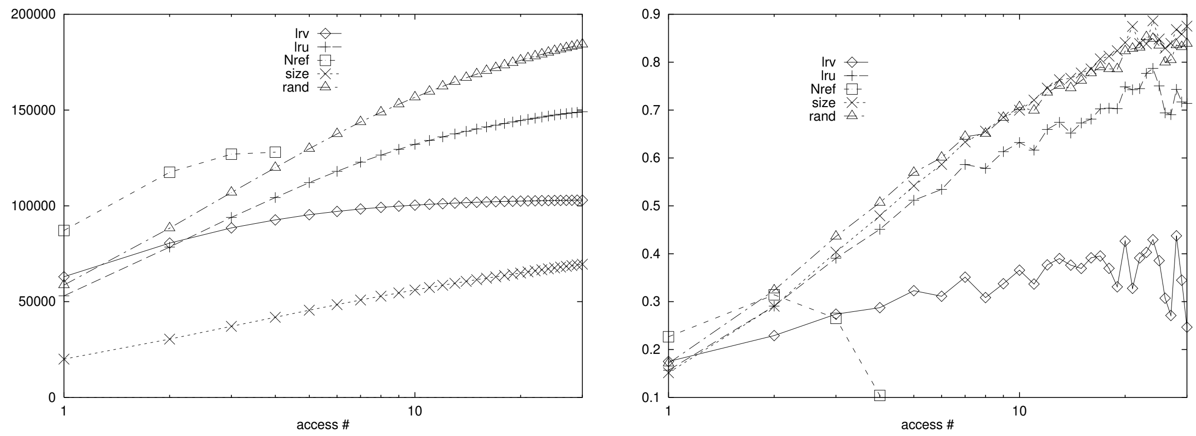

Figure 6: The percentage (right) and the cumulative number (left) of wrong choices in discarding documents vs. the number of accesses issued to the document at the moment of the choice; the cache size is 500MB.

larger cache sizes just because our traces are not long enough.

It is interesting to evaluate the number of errors made by the various replacement policy in discarding documents which will be accessed again. Figure 6 shows these values for documents with different number of accesses, obtained using a cache size of 500MB. On the left, we show the cumulative number of errors. The NREF policy does no errors for documents with a large number of accesses, but does many of them for documents with fewer accesses. The SIZE policy does fewer errors than other policies, yielding to a higher HR. However, it discards, on average, larger documents than other policies, hence its lower BHR. The graph on the right shows the percentage of errors for each class of documents with the same number of accesses, i. Here we see that policies which do not consider the number of previous accesses have a comparable behavior, with an increasing percentage of errors with i. This happens because these documents have a larger P_i, making it easier to take the wrong decision when purging one of them. Both NREF and LRV work instead much better because they give the correct importance to i.

6 Conclusions and future work

We have shown how the probability of a document being reaccessed depends upon a number of parameters, and derived an easy-to-compute formula to determine the relative value of each document. Basing on these dependences, and on a cost/benefit model for Web documents, we have designed a replacement policy called *Lowest Relative Value* (*LRV*) to achieve a better selection of documents to purge. We have shown how LRV overcomes some limitations of LRU and other policies, and that it can outperform them in all cases. LRV proves to be particularly useful in presence of small caches.

The implementation of LRV is trivial, and it can be easily inserted in publicly-available proxy servers such as `squid`. With a proper organization of data, keeping documents sorted by value requires constant-time operations on data held in main memory, thus allowing replacement on demand rather than requiring periodic garbage collections.

This work is by no means final. The characterization of proxy accesses presented in Section 3 is reasonably general, yet these results need to be validated on a larger number of traces. Also, the statistical properties of live documents, dead documents and of errors made by the replacement policy need to be studied in more detail. These results are extremely important in the design and evaluation of algorithms for proxy cooperation.

References

[1] M.F.Arlitt, C.L.Williamson, "Web Server Workload Characterization: The Search for Invariants", Proc. of SIGMETRICS 96, May 1996, Philadelphia, PA, USA

[2] T.Berners-Lee, R.Fielding and H.Frystyk, "Hypertext Transfer Protocol – HTTP/1.0", *RFC 1945*, May 1996.

[3] A.Chankhunthod, P.Danzig, C.Neerdaels, M.Schwartz and K.Worrell, "A Hierarchical Internet Object Cache", Proc. of the 1996 USENIX Technical Conference, January 1996, San Diego, CA, USA.

[4] R.Karedla, J.S.Love, B.G.Wherry, "Caching Strategies to Improve Disk System Performance", *IEEE Computer*, pp.38-46, v.27, n.3, March 1994.

[5] J.Gwertzman, M.Seltzer, "World Wide Web Cache Consistency ", Proc. of the 1996 USENIX Technical Conference, January 1996, San Diego, CA, USA.

[6] A.Luotonen, H.Frystyk, T.Berners-Lee, "W3C httpd," http://www.w3.org/hypertext/WWW/Daemon/Status.html.

[7] V.N.Padmanabhan, J.C.Mogul, "Using Predictive Prefetching to improve World Wide Web Latency", *ACM Computer Communication Review*, pp.22-36, v.27, n3, July 1996.

[8] "Squid Internet Object Cache", http://www.nlanr.net/Squid/.

[9] S.Williams, M.Abrams, C.R.Standridge, G.Abdulla, and E.A.Fox, "Removal Policies in Network Caches for World-Wide Web Documents", Proc. of ACM Sigcomm96, August 1996, Stanford University, CA, USA.

[10] P.Lorenzetti "Simulatore per cluster di proxy", Degree Thesis, University of Pisa, 1996 (In Italian)