Dealing with One-Timer-Documents in Web Caching

A. Belloum and L.O. Hertzberger
Computer Architecture and Parallel Systems Group
Dept. of Computer Science
Universiteit van Amsterdam
Kruislaan 403 NL 1098 SJ Amsterdam
{adam,bob}@wins.uva.nl

Abstract

A proper initialization requires starting the process in a state close to the expected steady-state. In web caching, the initialization problem is faced each time a new document enters the cache, independently of the method used to sort the documents into the cache, the newly referenced document is inserted in a so called "removal-list", from which documents are removed when storage space is needed. Often, undesirable documents are being assigned a high priority, consequently these documents remain for quite a long time in the cache, leading to a decrease in cache server performances. In this paper, we shall investigate one category of undesirable documents, which passe the filters commonly used to control the cache processing.

1. Introduction

Web caching has been introduced to solve the problem of the rapid increase of the Internet traffic. Basically, web caching consists of providing throughout the Internet web sites that keep copies of documents requested by the users. The expectation is the multiple accesses to the same documents are serviced to the user without having to go through the international busy connection to the origin server. To be efficient, the web cache servers have to be spread out over the Internet so that the end-user can redirect its request to the nearest one. This issue outlines the problem of the geographical distribution of the cache servers. Besides that, web caching mechanisms should keep in their local storage only the most frequently requested documents, those documents that generate the major part of the Internet traffic.

Web traffic analysis showed that access patterns for each document over the Internet is not uniform, the dynamics of Web traffic seem to be difficult to characterize and there are several differences between the Web and other network traffics [2, 10, 1] . This means that some sites, and more precisely, some documents at some particular sites are requested more frequently than others, and these sites and documents are spread non uniformly over the Internet [8]. Among all these requests only a small fraction accounts for most of the Internet accesses. Besides, these studies showed that the most popular files are the less frequently updated [7]. Another important characteristic of the so called popular documents is their relatively small size (less than 10 kB [4]). Therefore, it becomes clear that this category of documents is well suited for caching approaches. A large number of these documents could be cached and remain up-to-date for relatively long periods of time (usually a few days), which will lead to a dramatic reduction in both of the Internet traffic and the user perceived time[1].

Web caching would be the ideal solution to the increase of the Internet traffic if the documents would be cached for a long period of time. Unfortunately, due to the lack of storage space, the cache manager has to remove a number of cached document in order to make room for a newly referenced one.

The documents replacement strategy is composed of two phases. First, the documents are sorted in the cache in order to determine the removal ordering. This task is performed at each new request. Second, one or more documents are removed from the head of the removal list. Several strategies have been proposed such as the least frequently used (LFU), first in first out (FIFO) and many others. Only few of them have been implemented and used in real cache servers. Most of the time, documents are sorted in the cache according to the least recently used (LRU) strategy.

Our main interest in this paper is the removal list used in almost all of the document removal policies. The cached documents are stored within this list according to the different removal keys described previously. Unfortunately most of the removal policies do not have a mechanism to identify the so called *one-timer* documents. Those docu-

[1] The perceived time is the elapse time between sending the request and receiving the document

ments which are requested only once or a small number of times. Often, document replacement strategies assign high priorities to *one-timer* documents allowing them to remain in the cache for quite a long period of time without being referenced again. This could considerably affect the cache performance especially when the number of *one-timer* documents is large. The web traffic analysis presented in [2] shows that about one-third of the incoming requests belongs to this category of documents. It is thus of great importance to introduce policies that could better discriminate these *one-timer* documents. One solution, proposed by the VT-NRG Group and Arlitt et al, is to keep a list of all documents that have been accessed and only retains documents in the cache when they have been requested a second time. Only the documents that are accessed twice would be cached, this approach is known as the *Ignore First Hit*. This method introduce a extra latency on the second request to the document and computational overhead due to maintaining the list of *one-timer* documents.

The rest of this paper is organized as follows: in Section 2, a number of removal policies are presented and the impact of the *the one-timer documents* on these policies is discussed. In Section 3, a new method is proposed to deal with the problem of *one-timer documents*. In Section 4, a set of experiments are presented identifying the impact the proposed method to deal with *one-timer documents*. Finally Section 5 concludes the paper.

2. Replacement strategies

2.1. The LFU strategy

At first sight, the LFU (Least Frequently Used) strategy fits better to the web cache problem, which states that the most frequently requested documents are responsible for the major part of the web traffic. By maintaining a reference count for each cached document, the LFU strategy could estimate the frequency of references for each document. Two main problems arise with such an approach. First, once the cache reach its steady state and documents start being replaced frequently, the newly referenced documents are always removed first since they have the lowest reference count. The new documents have no time to get their count increasing in order to be cached for a longer period of time. Second, documents that build up an extremely high reference counts are rarely (if ever) replaced, even if they are no more requested.

The LFU strategy only focuses on one parameter in sorting the documents in the cache, it is clear that the two problems described here result form the fact that the time is ignored by the LFU strategy. To overcome these problems new versions of the LFU strategy have been proposed, the LFU_Aging and the LFU*. These studies were proposed by Arlitt in [2] to deal with respectively the high reference count and the *one-timer* documents.

The LFU_Aging avoid the building up of a high reference count by limiting and aging reference counts. To combat the high reference count, the *LFU_Aging* fixes the maximal number of reference per document and records the age of each reference count.

The LFU* differs from the LFU policy in that on a cache miss, the newly requested document is not always added to the cache. More specifically, only documents with reference count equal to one are candidate for replacement. The one timer documents are removed first but the building up of high reference count is always possible.

As stated above the LFU strategy implicitly solves the problem of *one-timer documents*. However, it could also lead to a stale cache, where the previously popular documents that have a very high reference count, will block the access to the cache for new documents getting more and more popularity.

2.2. Other replacement strategies

Except for the LFU strategy, the rest of the web caching replacement policies have no real mechanism to identify the *one-timer* documents.

Ignore the first hit is the only way to deal with *one-timer* documents. On the first request the document is not cached, only its header is kept in the cache. On the next request, the document is identified as not a *one-timer*. This technique introduce a substantial latency due to a real data transfer form the origin server on each document second hit.

Since it is not possible to know a priori if the new reference is a *one-timer* document or not. Thus, it is important that each new reference to be kept for a period of time, to make sure it will not receive other references in the nearest future. If documents identified at current time as *one-timer* are not removed first, a newly requested document is more likely to see it reference count increasing. The LRU strategy use such a mechanism, the documents are sorted according to their last reference, therefore, the least recently used document is removed first. The LRU strategy insert each new reference at the bottom of the removal list, giving it the opportunity to stay in the cache for a longer period of time. When real *one-timer* enters the cache, the LRU strategy often removes a number of multi-referenced documents, which reduces the server performances. The document replacement strategies not considering neither the time

nor the number of references, such as the SIZE based one, are the most sensitive to the *one-timer* documents. Once a *one-timer* document enters the cache it can remain for a long time (if ever).

In the previous subsections, two factors have been identified as having a direct impact on the *one-timer* document identification process: the number of references and the time of the last reference. In the following section, a method that considers both of these two factors is proposed to deal with *one-timer* document identification.

2.3. Sorting process initialization

We propose a proper initialization of the document sorting process involved in the replacement policies to deal with problem of *one-timer* documents. Assigning an initial value to each new reference in the cache will prevent from considering these documents as highly potential to be kept in the cache. The cached documents are assigned a different value only if they are requested more than once. The *one-timer* documents remain with their initial value which push them to the head of the removal list. The advantage of the sorting process initialization over the Ignore First Hit approach is for the documents requested more than once, we do not have to request again the document from the original server. Besides that, the computational overhead requires to maintain and search within the *one-timer* document list is avoided.

2.4. Sorting one-timer documents

According to the sorting process initialization method the *one-timer* documents are assigned the same initial value to distinguish them from the rest of the documents (This initial value is referenced as the *unknown priority*). Such a method of assigning priority does not allow the identification of documents within the *one-timer* category. It is therefore necessary to use a second sorting key to perform this selection process, any one of the commonly used replacement strategy could be used, however it is important to note that each *one-timer* document is likely to get a higher priority at each new request. Each *one-timer* document should remain in the cache for a certain period of time to get enough time to see whether or not its priority changed. The parameter time plays an important role in this process. Therefore the LRU based methods will fit better to this problem. Combining the initialization approach with the LRU strategy provide a simpler solution to the cache "purging" rather than the method used in cache Harvest ver1.4pl3 [2], which consists of a complicate mechanism using an upper and lower threshold to control the cache disk usage and which leads to suboptimal performance since the cache is kept below 100% utilization [10].

[2]Harvest is a proxy cache server http://harvest.cs.colorado.edu

2.5. Cache partitioning

To illustrate the initialization of the document sorting process, the cache has been partitioned into two parts: one for the *one-timer* documents (partition A) and one for the rest (partition B). In a mono-partition cache, the number of *one-timer* documents will decrease with the number of requests, since a newly referenced document will be removed first. After a while their number has become so small that any newly referenced document is likely to be removed on the next request. In such a situation, no more documents will have the chance to get higher priority, this will result in a reduction of cache performance since its content will become out-of-date after a certain time. Splitting the cache in two partitions will keep a fixed number of *one-timer* documents cached so that the updating process of the cache is always fair. The partitioning of the cache will be performed according to the total number of documents in the cache and not according to the data size. The *one-timer* documents partition is always a fixed percentage of the current number of documents. To support such a partitioning process a mechanism converts the document with the lowest priority within the partition B into a *one-timer* document as shown in Figure 1. When a document passes from partition B to partition A its priority is assigned an unknown value, which make it a *one-timer* document, since the sorting process in partition A is based on the time, this document will be pushed at the top of the removal list.

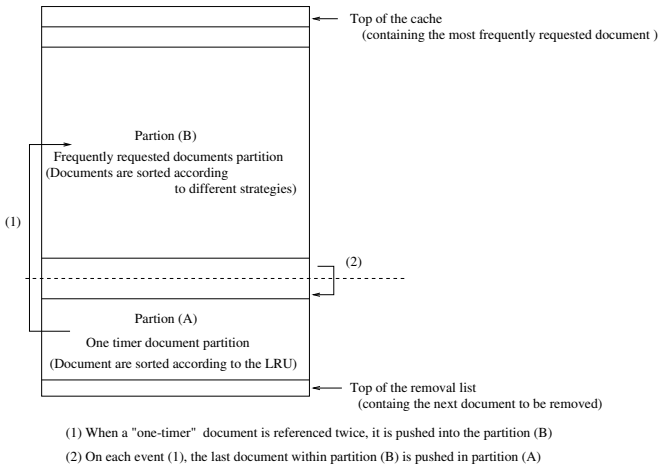

Figure 1. Cache partitioning process

3. The experiments

In the following experiments we outlined the impact of the initialization of the document sorting process on the web

cache performance by just applying this concept to the replacement strategies discussed in the previous section. The performance metrics used in this web cache analysis mainly focus on the document hit rate and the byte hit rate. The reason for this choice is the fact that byte hit rates give more information of the network bandwidth while the document hit rate focuses only on the number of requests satisfied using the cached copies. In web caching documents have different sizes, recording then only the document hit rate will not give any ideas on how the method impacts the network bandwidth. Such an information is very important for a web cache performance analysis, since the current caching methods seem to give advantage to small size document. Besides this, since we are investigating a two level cache server topology, we have redefined both the document hit rate and the byte hit rate in order to record hits on the two cache levels. The Two metrics are:

- The document hit rate "SHR": This metric records the global hit rate obtained using a two level cache configuration.
- The byte hit rate "SBHR": This metric records the global byte hit rate obtained using a two level cache configuration.

3.1. Workload traces

The workload used in this study are part of access log-files provided by the web server of the Computer Science Department of the University of Amsterdam WINS (wins.uva.nl) and the proxy server NLANR (ircache.nlanr.net). These workload traces represent two types on web cache server, more information on these workloads are provided in table 1. The WINS workload contains external requests to documents provided by the wins.uva.nl server, this type of workload trace exhibit a strong document reference locality. The NLANR workload involves requests coming form different web severs that use NLANR as a proxy server, usually this kind of servers is much more busy than the normal cache server. They receive requests for documents belonging to other web servers because usually proxy cache servers keep a copy of the documents serviced in the past. Most of the time these access log-files are large and it is very hard to use more than few days of workload duration. In the following experiments, we use these two workloads to outline the impact of the workload characteristics on the mechanism we are introducing in this paper.

3.2. The LFU strategy

In this experiment, the document replacement policy examined is the LFU. The simulation results presented in Figures 2 and 3 show a slight improvement for both of the cache

workload	Duration	Transfered data	Number of requests
WINS	1 month	4.2 GB	737750
NLANR	1 day	2.5 GB	261135

Document size	$<$ 10 KB	$<$ 1MB	$>$ 1 MB
WINS	70% (1.04 GB)	9% (3.10 GB)	0.01% (0.15 GB)
NLANR	56.5% (0.47 GB)	18.4% (0.42 GB)	0.09% (0.58 GB)

Table 1. *Workloads characteristics*

metrics (SHR and SBHR) when introducing the sorting process initialization to the LFU strategy (LFU_init). The small increase in the cache performances is not only due to the small amount of *one-timer* documents involved in the WINS workload, but it is also the result of the fact that implicitly, the LFU replacement strategy has solved the problem on the *one-timer* documents by assigning a reference count to each document. Since the sorting process initialization has been introduced to deal with this problem, its impact on the cache performance have been thus considerably reduced.

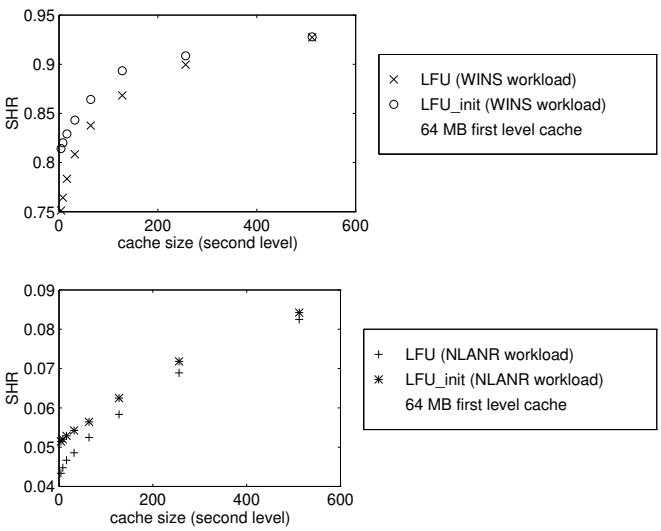

Figure 2. The document hit rates

Using the NLANR workload, where there is plenty of *one-timer* documents, does not impact dramatically the web cache performances (Figures 2 and 3). This confirms the fact that LFU strategy is less sensitive to the sorting process initialization. The *one-timer* documents are removed first in both the LFU and the LFU_init. However, the sorting process initialization has allowed the *one-timer* to remain in the cache for a longer period of time.

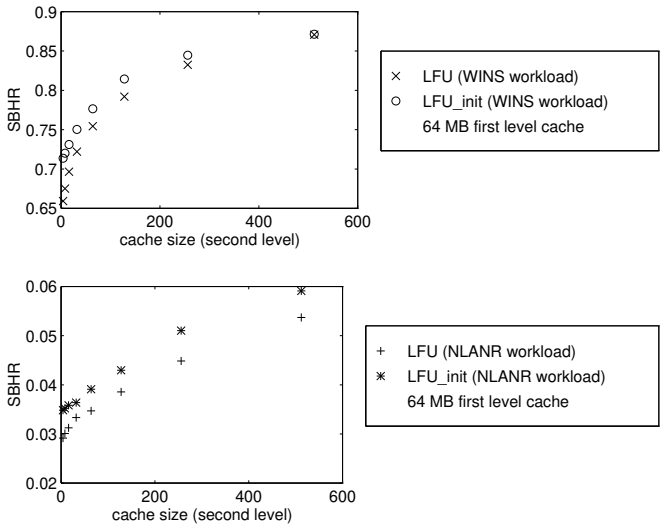

Figure 3. The byte hit rates

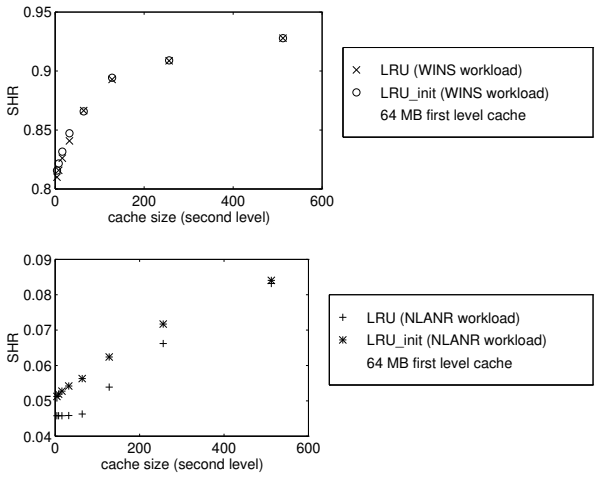

Figure 4. The document hit rates

3.3. The LRU strategy

In this experiment, we have compared the different hit rates obtained when using the LRU strategy with their equivalent rates when we introduce the sorting process initialization (LRU_init). The combination of the LRU strategy and the sorting process initialization, which is also sorting *one-timer* documents according to their entry time, has lead to equivalent global cache performances. Figure 4 shows almost no improvements in both the SHR and SBHR when using the WINS workload. This is probably due to the strong locality of reference, which reduces the number of *one-timer* documents.

When using the NLANR workload which present a low locality of reference, we have first noticed that the second level cache does not have any impact on both the SHR and the SBHR unless it is larger than the first level cache (more details are given in [3]). This phenomenon is the result of the large number of misses recorded in the first level cache, leading to an equivalent number of forwarding documents to the second level cache. The document removal process start being used almost at the same time in the two cache levels, since the same removal policy is used in both of the two cache levels, the probability of a hit in the second level cache is very low. This equivalence between the two cache levels disappears as soon as the second level cache becomes larger, which increases the number of hit in the second level cache.

Combining the LRU strategy with the sorting initialization process, when a large number of *one-timer* documents exist, has increased the web cache performances (SHR and SBHR). However, this combination failed to provide any improvement if the first cache level is relatively small (less than 64 MB). For these cache configurations, the LRU_init did not reduce the high traffic between the two cache levels, which has lead to an intensive use of the document removal process, this has reduce the document storage time in the second level cache. Since the workload do not exhibit a strong reference locality, the probability of a hit in the second level cache is considerably reduced [3].

comparing to the results of LFU strategy presented in the previous section, the impact of the sorting initialization process on both the document hit rate and the byte hit rate is less important when the number of one-timer documents is small, as it is the case of the WINS workload. It seems that the LRU strategy deals better with these documents, it allows them to remain for a longer period of time in the cache, but since their number is reduced they do not disturb the cache performances. The necessary time requires to judge, if a newly requested document is a one-timer document, is satisfied which allows the LRU to distinguish between the two categories of documents. When the number of one-timer documents increases the impact of the initialization process becomes more important, which implies that the transition time of these documents in the cache is large enough to reduces the performance of the system. In this case the combination of the sorting initialization process and the LRU has reduced this transition time which improves the web cache performances.

3.4. The size based strategy

The following experiment shows the impact of the sorting process initialization when using the size of the document as primary sorting key. As it was stated by Williams