Optimization of Proxy Caches using Proxy Filters

Fabian Weber, Marcel Daneck, Christoph Reich
Hochschule Furtwangen University
78120 Furtwangen, Germany
{fabian.weber, marcel.daneck, christoph.reich}@hs-furtwangen.de

Abstract— **Web proxy caches are a widely used tool to reduce the network load by storing often needed web pages. Increasing amounts of data transferred require new intelligent caching strategies with smarter replacement algorithms to predict, which web documents will be requested most likely in the future. However, these algorithms are relatively complex and require a lot of computing power. This paper describes an approach to design more intelligent and efficient web caches by adding a document filter that decides whether a document should be cached or whether it should be ignored in order to save disk space. The filter uses server connection information for its decision. The evaluation shows a reduction of required cache space of almost 90% compared to a traditional proxy cache.**

Keywords - Proxy; Cache; Filter; Replacement-Algorithm; LRU; LRU-threshold; LRU-MIN; SIZE; Log2(SIZE); Hybrid; MIX; GreedyDual-Size

I. INTRODUCTION

Whenever multiple users have to share a single internet connection, bandwidth bottlenecks are imminent. This problem occurs in the private as well as business sector. In the latter case, bottlenecks often harm productivity, as the employees have to wait for websites, documents etc. to be loaded. To circumvent this problem, a common practice is to use a web proxy cache server. Proxy caches store web documents that are frequently requested by web users to avoid repeated downloads of the same information from the originating web server and therefore reduce bandwidth utilization. Typically, this server is located in the local network and avoids WAN (Wide Area Network) bottleneck at the edge server.

However, the utilization of a proxy cache introduces a few difficulties when working with large amounts of data transferred [1]. The main problem is that the cache can only store a limited amount of documents, because of the limited disk space available. To cope with this problem, more intelligent cache replacement algorithms are needed to increase the efficiency of the cache. These algorithms are very complex and require a high amount of computational power, effectively limiting the efficiency of a proxy cache not only by disk space, but also by CPU power.

However, some of the documents downloaded need caching more than others. For example, files that are served with a speed almost as fast as the proxy server's connection do not benefit as much from caching as files transferred from a very slow originating server. A filter can ensure the files that benefit mostly are more likely to be cached and to remain in the cache.

This paper describes the basis to proxy caches in Section II, outlines the state of the art and technology in Section III and shows a new way to increase the efficiency of proxy caches without having to use complex replacement algorithms by filtering the web documents in Section IV. In Section V, the new proxy filter is evaluated and a conclusion can be found in Section VI.

II. TRADITIONAL PROXY CACHES

Caching is divided into three main areas: *Client-Caching* [2] is performed at the user's own system by the browser. *Server-Caching* (or *Reverse-Proxy-Caching*) [2] on the other hand is accomplished by a remote server. This technique is normally used to reduce the work load of a web server. The proxy server is therefore located in the network of the originating web server. The performance improvements are most notably with web servers that handle complex dynamic websites. The last area is *Proxy-Caching*. Here, the proxy server is located at the client's subnet, ensuring high bandwidth and low latency for this connection. Every document request of the client is sent through the proxy server. This enables the server to cache frequently accessed documents and deliver them to multiple users through the local network infrastructure, rather than through the relatively slow internet connection [3].

The utilization of proxy caching has a wide range of advantages: a) Internet users will benefit from faster page load times through a higher bandwidth and therefore have a better web experience. b) The internet-infrastructure can be improved by decreasing the outbound network traffic, effectively increasing the total network performance. Since large enterprises often buy their internet connection volume-based, reduction of the internet traffic can also save expenses. c) Web servers can benefit from proxy caching, because the work load on these servers is reduced [3]. This leads to higher performance without the need of hardware upgrades.

There are some important issues to be considered when using a proxy server:

- Even though mass storage is not very expensive, proxy cache servers do have space limitations. When the available disk space is filled, a replacement algorithm is utilized to decide which documents can be evicted in order to make room for new documents. Therefore, an optimal algorithm must be chosen depending on how clients are using the web. Several of these algorithms will be discussed in the related work section of this paper.

 ISBN: 978-1-61208-239-4

- Another requirement for a cache server is to ensure the consistency of the stored documents. This can be realized using one of two methods: Either the cache server asks the web server, whether the cached document has been changed in the meantime, or the web server can inform the cache server about a change of the document. The last method is not very popular, since there are no well-defined standards and it is much harder to implement.
- A last possible feature of a proxy cache is to predict the user's behavior and to pre-load documents the user is likely to request next.

This paper focuses on a method to improve the efficiency of proxy caches by using the concept of content filtering for proxy caches (proxy filters) explained in Section IV.

III. RELATED WORK

Until now, optimization efforts in the field of proxy caching are mostly aimed to improve the underlying replacement algorithm. Such an algorithm is applied whenever a new document has to be stored while the cache storage is already full. Next, some of these replacement algorithms will be shown and analyzed.

One of the most well-known replacement algorithm is *LRU* (least recently used) [4]. As the name suggests, this algorithm always evicts the least recently used document from the cache. Therefore, a simple list is used. Upon request, a document is moved to the top, while the document to be deleted is taken from the bottom of the list. This procedure also explains the very low complexity of this algorithm at $O(1)$ [5]. The major downside of this algorithm is its simplicity of predicting which document will be requested in the future and therefore the hit rate of cached documents is rather low. Another downside is the weakness to calculate the cost of caching a requested document. This means, a large downloaded and cached document will overwrite many small and maybe more frequently used websites. Many important websites are replaced by one rather useless document.

To take countermeasures against these problems, some LRU derivatives were developed. One of these derivatives is *LRU-threshold* [6], which supports the definition of a maximum document size. Documents that exceed the given size threshold are never cached (not even if storage space is left). Apart from that, LRU-threshold acts like LRU.

SIZE [7] and *Log2(SIZE)* [7] represent two algorithms that use the document size as their primary caching decision. While SIZE always evicts the largest document first, Log2(SIZE) groups the documents by a logarithmic value of their size. Within a group, the least frequently used document is evicted (using LRU). Both of these algorithms have a complexity of $O(\log(n))$ [5].

Another popular LRU derivative is *LRU-MIN* [6]. This algorithm replaces larger documents earlier than smaller documents. Therefore, whenever a new document of size S has to be cached, all documents with size greater or equal to S are grouped. Within this group, the document is selected with LRU. If no documents remain with size $\geq S$, $S/2$ is used as selector (then $S/4$ and so on). The disadvantage of the algorithm is the inability to consider the cost of a document. On caches with very large storage spaces, a high amount of computing power will be required to utilize this algorithm as it has a very high complexity of $O(n)$ [5].

All the aforementioned algorithms assume that large documents (like file downloads) are less frequently requested than small documents (i.e., websites) and therefore less important to cache. This assumption may have been true a few years ago, but a study in [8] suggests an oncoming change in user behavior. With Web 2.0 and media services like YouTube even large documents (i.e., videos) will be requested frequently. Eventually, these files are also eligible for caching.

In [9], Wooster and Abrams introduce the *Hybrid* algorithm, designed to reduce the document access delay. Therefore, the algorithm considers the round trip time (RTT), the bandwidth between proxy server and originating server as well as the quantity of requests since a specific document has been stored into the cache. Using these parameters, a utility value is calculated for each document in the cache. The document with the lowest utility value finally gets replaced. This algorithm is the basis of the *MIX* algorithm, developed by Niclausse, Liu and Nain [10]. In MIX, the time since the last access of a document in the cache is added to the formula, thus introducing a possibility to remove obsolete documents like LRU does. Different to LRU, however, is that all characteristic parameters of the Hybrid algorithm are considered, too. Both methods provide benefits when documents from very slow servers are fetched. These web documents produce a very high utility value, courtesy of the low server bandwidth and therefore stay in the cache for a relatively long time. Because these files would normally be served very slowly, the performance gain is very high. On the other hand, documents that are on fast servers will be saved as well (even if the remote server speed is almost as high as the request speed from the proxy cache). These documents will ultimately be evicted in the near future, but at first they will get cached and replace other, more important documents. Additionally, both algorithms have a relatively high complexity of $O(\log(n))$ [5].

Cao and Irani introduced the *GreedyDual Size* (GDS) algorithm in [11], which is an improvement of Young's *GreedyDual* [12]. The GDS algorithm calculates the cost to cache for each document. Key parameters are the connection time, access time, transfer time and document size. Whenever a document has to be evicted, the file with the lowest value is deleted, like with Hybrid and MIX. GDS additionally incorporates a LRU-like behavior by subtracting the value of an evicted document from the values of all remaining documents. If a document is requested again while it is still remaining in the cache, its value has to be restored. This way, less frequently requested documents gradually lose their value and get evicted. The major benefit of GDS is clearly the consideration of caching costs. Therefore, large documents originating from fast servers get a relatively low value and are deleted shortly. If the connection speed of the

remote server is slow, the documents are rated with a high value – even though they are relatively large – and stay in the cache for a longer time. Additionally, small documents are also treated the same way: if they are downloaded from a very fast remote server, the utility value of these documents is low (because the performance improvement of caching these documents is very low). Ultimately, these documents will be evicted soon and do not have to be kept for an unnecessary amount of time. The disadvantage of this algorithm on the other hand is the high complexity of $O(\log(n))$ [5].

In conclusion, the utilization of a complex replacement algorithm like Hybrid, MIX or GreedyDual Size results in a high hit/miss-ratio and a reasonable good selection of documents to be evicted, but with the need of high computing power (especially with large caches) caused by the high complexity of these algorithms. Furthermore, none of these algorithms takes into consideration whether the caching of a specific document entails a performance improvement in the first place.

In Table 1, an overview of the aforementioned algorithms is shown.

TABLE 1: OVERVIEW OF SELECTED REPLACEMENT ALGORITHMS [5]

Replacement algorithm	Relevant keys	Complexity
LRU	Time since last access	$O(1)$
LRU-threshold	File size Time since last access	$O(1)$
LRU-MIN	File size Time since last access	$O(n)$
SIZE	File size	$O(\log(n))$
Log2(SIZE)	File size	$O(\log(n))$
Hybrid	File size Round-Trip-Time Bandwith between proxy and server Number of hits	$O(\log(n))$
MIX	File size Round-Trip-Time Bandwith between proxy and server Number of hits Time since last access	$O(\log(n))$
GreedyDual Size	File size Connection time Access time Transfer time	$O(\log(n))$

IV. PROXY FILTER

Current proxy caches generally only limit the maximum document size. Documents that exceed this size are never cached while documents smaller than the maximum size are always cached and replace other documents if the storage is full. Admittedly, as shown in the last chapter, a wide range of more or less smart replacement algorithms can be used to select documents for eviction, but none of these considers whether it is reasonable to store a web document in the cache or not. The assumption is that it might be better not to cache a document at all, because the performance improvement of storing a document in the cache is not worth it.

For example, dynamic web pages that take a long time to generate can be cached, while images or style sheets embedded in the site are not cached because they are static files located on the originating web server and provide very low access latency. In this specific case, transfer time is a less important criterion than access time.

A. Concept

The decision whether a document should be cached or not is done by a proxy filter module. Several factors influence the decision and have to be taken into account by the proxy filter:

File size: The file size is an important factor, because the cache can store a lot more small files than large files. This basically means that a large file occupies the space that otherwise very many small files would use and is therefore considered less valuable. Contrary to existing algorithms, the file size is not considered an absolute limit, but it is relativized with the other factors.

Request time: This reflects the time needed to connect to the remote server and send the request. The request time is particularly interesting because servers that operate under high load and reach their connection limit cannot react to the request in a decent time frame.

Access time: High access times are mostly a result of dynamic content, which has to be computed by the server. This is, for example, the case with Web-Content-Management systems, forums or other Web 2.0 pages. The access time is measured by the timespan between sending the request and receiving the first byte of the response.

Transfer time: Another interesting factor is the transfer time. If it is an unusual high value, caching the document may result in high performance improvements despite the eventually large file size. Documents that are fetched with a low bandwidth generally achieve a higher performance gain than documents transferred through a fast connection.

A few samples of latency distributions for web requests are shown in Figure 1. The first bar represents the download of a small static style sheet file from a heavily used server (request time is relatively high). The second element represents accessing a dynamic website. Here the request time is very low, while the access time is extremely high. This document will benefit from a great performance gain upon caching. The last bar visualizes the download of a large file. The most notable factor is the transfer time. Caching benefits for this file have to be evaluated by consideration of the server's bandwidth (the file size in relation to the transfer time).

 ISBN: 978-1-61208-239-4

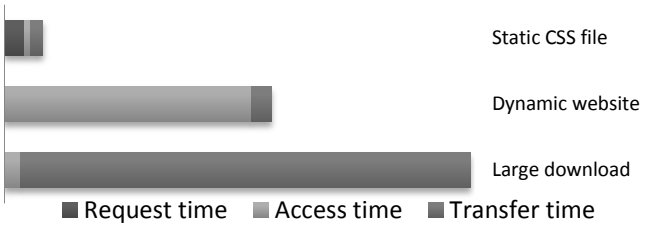

Figure 1: Exemplary latency distribution for web requests

To be able to determine, which documents should be cached, the factors mentioned above have to be weighted and summed up. Therefore, in a first step, the weighting of each factor has to be configured. The algorithm then multiplies each factor with the associated weighting and sums the (weighted) factors up. More precisely, the filter uses the formula:

$$r * w_r + a * w_a + t * w_t + s * w_s$$

where r stands for the request time, a is the access time, t the transfer time and s is the file size. w_r, w_a, w_t and w_s are the weightings for each factor. If the result is greater than or equal to zero, the document is considered relevant for caching. Documents with ratings less than zero will not be cached, because these documents would not get enough performance gain when loaded from the cache as opposed to being loaded from the internet.

Furthermore, this filter can be designed to act intelligent by setting the weightings dynamically. This way, the system would be capable of adjusting itself to changed conditions like an increase in available storage space. To implement such intelligent behavior, a background task could be set that runs at the end of the day and analyzes the hit/miss ratio, byte-hit/miss ratio etc. of the proxy cache and adjust the parameters accordingly. At the next run, this optimization job can compare the last results with the new test results and adjust the weightings again.

The proxy filter does not take the place of cache replacement algorithms. These algorithms still have to be used whenever a new document has to be saved to the already filled storage space. However, these algorithms will be used less often since documents that would get evicted again after a short timespan will never be cached in the first place.

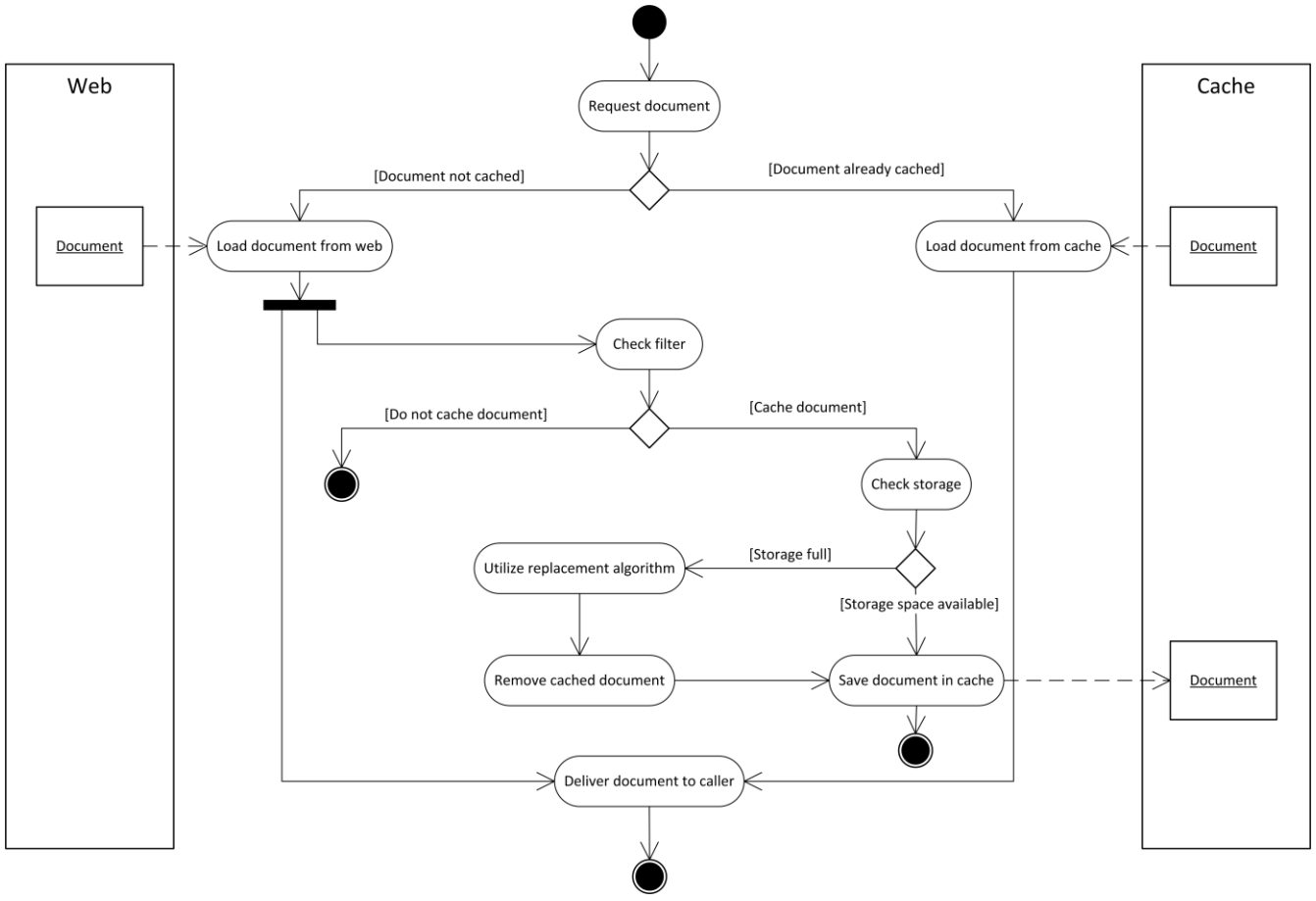

Figure 2: Activity diagram of a proxy cache with enabled filter

 ISBN: 978-1-61208-239-4

The proxy filter – which is placed before the actual caching logic (see Figure 2) – improves the performance of the total proxy cache. The hit/miss ratio of the cache will not be greatly improved by utilization of a proxy filter, but the cache will work much more efficiently. One reason is that important documents are not replaced by less valuable files. Another reason is due to the lightweight algorithm for filtering, which requires much less computing power. As shown in Table 1, most of the replacement algorithms have a complexity of at least $O(\log(n))$. For big caches with many stored documents, a lot of documents have to be analyzed for replacement decision. Some of the algorithms analyze every document in the cache and therefore need a lot of compute power. The filter algorithm, however, just has to analyze the current document. The resulting complexity is $O(1)$.

Figure 2 shows how the proxy filter is integrated into the proxy cache. If a document is requested and already stored in the cache it is directly delivered. If a document is requested and not stored in the cache it is downloaded from the addressed web server logging the file size, request time, access time and transfer time. According to these parameters a decision whether this document needs to be cached or not can be reached.

V. EVALUATION

To be able to evaluate this concept, a first step was to collect proper test data. Therefore, various websites were visited, files downloaded and media streamed. Meanwhile, all web requests were logged, causing a total of 6644 data sets. Each of these data sets reflects one downloaded document of types like websites, embedded pictures, style sheets and JavaScript files, as well as video files, etc.

In total, these 6644 files take around 505 MB of space and the download time of these files (including connection time, access time and transfer time) was about 1 hour and 35 minutes.

TABLE 2: WEIGHTING OF FACTORS

Factor	Weighting
File size	-1
Request time	50
Access time	100
Transfer time	250

To evaluate the filter algorithm, for each data set the performance improvement has to be determined when it is stored in the cache. Using the filter we can decide if it is worth to store a document or if it is better not to store it and save the cache space for other documents. As mentioned above, the factors file size, request time, access time and transfer time needed to be weighted. Therefore, the weights were configured as shown in Table 2 (these values were selected experimentally).

A sample calculation in Table 3 shows, how the document rating was concluded from the factors and weights of three exemplary documents:

TABLE 3: RATINGS OF EXEMPLARY DOCUMENTS

File size [Bytes]	Request time [ms]	Access time [ms]	Transfer time [ms]	Rating
142,694	< 1	1,514	4,352	1,096,706
91,989	< 1	31	203	-38,139
10,121,411	140	250	133,693	23,333,839

The first document has a high access time as well as a high transfer time, meaning a low server bandwidth. The document rating is therefore positive and the document gets cached.

The size of the second document is even smaller than the first document, but because of its low access time and high bandwidth, the rating is negative. This document does not get cached, because caching would not bring a high performance gain (the originating server is almost as fast as the proxy cache).

Even though the third document is with almost 10 MB rather large, it is cached because of the very high transfer time indicating a server with a slow internet connection. This way, long waiting times when downloading this file are circumvented.

After applying the algorithm with the weightings from Table 2, it indicates that of the 6644 requested documents, 5682 (that is 85.5%) would have been cached. These 85.5% of files take 63 MB storage space, meaning a reduction of disk space of more than 87%.

To estimate the performance improvements achieved by caching these documents, the request time at the proxy server was set at 10 ms and access time was set to be 30 ms. In practice, these values are most likely even lower. Since the proxy cache is typically located in the physical network of the users, the bandwidth was assessed at 100 Mbit/s.

To calculate the performance gain, the total response time (request time, access time and transfer time) for each document not in the cache was summed up. For each document that was cached, a request time of 10 ms, access time of 30 ms and the transfer time according to the file size through a 100 Mbit/s connection were summed up. In total, the time required to load all documents would be barely 9 minutes. Compared to the 1.5 hours needed for the initial download this is a performance improvement of over 90%.

By changing the weightings of the factors, the ratio of storage space and performance gain can be varied. A short outlook of these variations is given in Figure 3. The chart shows clearly that the system works most efficiently while between 10% and 15% of the traffic is cached. Caching the other documents would require a huge amount of storage space while the performance improvements would be at a minimum.

 ISBN: 978-1-61208-239-4

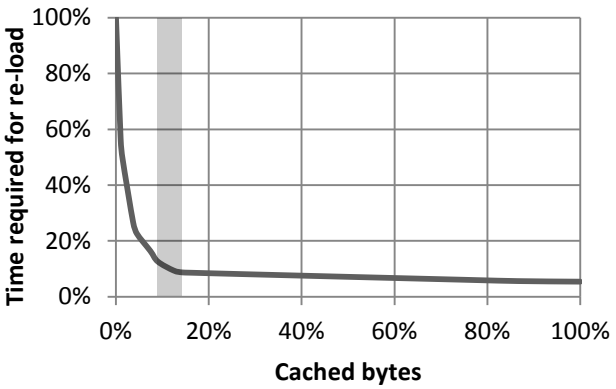

Figure 3: Performance improvement in relation to cached bytes

This chart also shows the difference between a proxy cache that uses the filter and one that does not. The proxy filter has the ability to choose the 12% of the documents that cause 90% of performance improvement by caching. A proxy cache without this filter is not able to process this information, since every document is stored, regardless of the performance gain.

VI. CONCLUSION

Proxy caches can utilize a wide range of cache replacement algorithms. Depending on the selected algorithm, different factors are then used to select a document for eviction whenever storage space is needed to cache a new document. However, because these algorithms only take action when a document needs to be deleted, none of them can predict whether the caching of a specific document makes sense in terms of performance improvements.

The newly introduced proxy filter fills this gap by trying to estimate the performance gain of each document upon request. Only documents that promise high performance improvements will be cached. This method highly aids the selected replacement algorithm – which can still be used without modifications – because the filter uses less computational power to execute.

As shown in the evaluation, using the proxy filter only 12% of the transferred amount of data is cached, resulting in a performance increase of over 90%. Because of this data reduction the cache has a lot less swap and the replacement algorithm is utilized less often. This improves the system responsiveness and saves resources since the replacement algorithm has to analyze every document cached to select one to evict while the filter only analyzes the current document.

REFERENCES

[1] Elias Balafoutis, Antonis Panagakis, Nikolaos Laoutaris and Ioannis Stavrakakis: *Study of the Impact of Replacement Granularity and Associated Strategies on Video Caching* from: Cluster Computing, Vol. 8, No. 1, pp. 89-100, 2005

[2] Zeeshan Naseh and Haroon Khan: *Designing Content Switching Solutions*, Cisco Press, March 14, 2006

[3] Daniel Zeng, Fei-Yue Wang, and Mingkuan Liu: *Efficient Web Content Delivery Using Proxy Caching Techniques* from: IEEE Transactions on Systems, Man, and Cybernetics - TSMC , Vol. 34, No. 3, pp. 270-280, 2004

[4] Andrew S. Tanenbaum: *Modern Operating Systems*, Prentice Hall, 2nd Edition: December 6, 2001

[5] Abdullah Balamash and Marwan Krunz: *An Overview of Web Caching Replacement Algorithms* from: IEEE Communications Surveys and Tutorials - COMSUR, Vol. 6, No. 1-4, pp. 44-56, 2004

[6] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox: *Caching Proxies: Limitations and Potentials* from: 4th International World-wide Web Conference, Dec. 1995, retrieved from: http://www.w3.org/Conferences/WWW4/Papers/155/

[7] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox: *Removal Policies in Network Caches for World-Wide Web Documents* from: Computer Communication Review - CCR, Vol. 26, No. 4, pp. 293-305, 1996

[8] Geetika Tewari and Kim Hazelwood: *Adaptive Web Proxy Caching Algorithms*, Computer Science Group Harvard University Cambridge, Massachusetts, 2004

[9] Roland P. Wooster and Marc Abrams: *Proxy Caching That Estimates Page Load Delays* from: Computer Networks and Isdn Systems - CN, Vol. 29, No. 8-13, pp. 977-986, 1997

[10] Nicolas Niclausse, Zhen Liu, and Philippe Nain: *A New Efficient Caching Policy for the World Wide Web* from: Workshop on Internet Server Performance, June 1998

[11] Pei Cao and Sandy Irani: *Cost-Aware WWW Proxy Caching Algorithms* from: USENIX Symposium on Internet Technologies and Systems - USITS, pp. 193-206, 1997

[12] Neal Young: *The K-Server Dual and Loose Competitiveness for Paging* from: Algorithmica, Vol. 11, No. 6, pp. 525-541, June 1994

 ISBN: 978-1-61208-239-4