An Adaptive Neural Network-Based Method for WWW Proxy Caches

Rachid el Abdouni Khayari * Mohammad S. Obaidat † Sami Celik ‡

Abstract— **Developing adaptive methods to deal with the fast changes in communication systems workload request patterns has become a big challenge in the last years. Self-regulating mechanisms should be able to permanently analyze the system state and to modify its mode of operation if necessary. This modification can be achieved, e.g. by restructuring of the whole system or by setting adequate parameters of the used algorithms and methods.**

In this paper, we discuss the use of neural networks to support the adaptivity of the Class-based Least Recently Used (C-LRU) caching algorithm. The C-LRU caching algorithm has been shown to deliver good results for the cache performance, as measured by the hit rate and the byte hit rate. Furthermore the C-LRU allows for an adaptive caching strategy, since a change in the request patterns results in a change in the parameter setting. For our purpose, trace driven simulations are used. Our experiments show that neural networks are in fact able to achieve this aim.

Keywords: Caching, Self-regulating, Proxy server, Class-Based Least Recently Used, Neural Networks

1 Introduction

The increasing use of WWW-based services has not lead to high frequented web servers but also to heavily-used components of the Internet [25]. Fortunately, it is well known that there are popular and frequently requested sites, so that caching can be used to reduce Internet bandwidth consumption caused by the tremendous growth of the World-Wide Web [9]. Some of the main reasons for which a user would go for Web caching include the following [24]:

- to increase the bandwidth availability by curbing the transmission of redundant data,
- for reducing network congestion,
- for improving response times and
- for achieving savings in terms of cost (for e.g. cost of bandwidth)

Web caching is one of the most used technique to decrease the end-to-end delays and to reduce the Internet traffic [20]. Web caching works in a similar way to classical memory system caching; it stores web documents in anticipation of future requests [19].

Significant differences between memory system and web caching however exist; they result primarily from the following three properties of web caching: the non-uniformity of web object sizes, retrieval costs, and cacheability. For more details, we refer to [10, 19, 20]. These specific properties of web caching have been studied in details in many studies, e.g. in [10, 15, 20, 26]. It has been found that these specific characteristics are essential for the web caching and make the developing of performant web caching algorithms very difficult [20, 26]. Furthermore in [10, 20], three possible locations of the cache (namely at the client, at the Proxy Server and at the Primary Web server) and their pros and cons have been studied in detail.

In the last years, many caching strategies have been developed and studied [3, 4, 6, 8, 13, 21, 22, 31, 32]. The goodness of such caching strategies has been evaluated by the delivered hit rate (defined as the percentage of requests that could be served from the cache) and the byte hit rate (defined as the percentage of bytes that could be served from the cache).

The gist of all the approaches is the following question: which object has to be replaced when a new object has to be stored and the cache is already completely filled? To determine the document popularity, three properties of WWW requests, namely frequency of reference, recency of reference and the referenced object size, are usually used and exploited [20]. A compact overview over the

*University of the Armed Forces Munich, Department of Computer Science, Germany

†Monmouth University, Department of Computer Science, West Long Branch, USA

‡RWTH Aachen, Department of Computer Science, Germany

well-known caching strategies, as well as their evaluation are given in [20].

Most of these caching algorithms have been developed for specific contexts (e.g., for memory, web, disk or database) and therefore rate some object characteristics more important that others [15, 20]. In [4, 6], it has been shown that an algorithm that is optimal for one context may fail to provide good results in another one.

In most cases the validation of these caching algorithms have been performed and tested by considering only one trace, which reflects the user behavior for only a limited time period. The changes in the workload requests pattern have never been studied. In this context, and due to fast changes in the workload request pattern in communication systems (Proxy server cache in our case), most of these methods have to be verified. To cope with this challenge, future communication systems should integrate self-reconfiguration mechanisms.

In the last years, some studies have been conducted to develop self-reconfigurable methods and systems, e.g. [11, 12, 14, 27, 29, 30]. The aim of these studies was to develop appropriate mechanisms to enable computing software and hardware systems (also including communication systems) to deal with observed changes in the workload, in the environment or in components states.

In this context, the C-LRU can be seen as potentially adaptive to the considered workload, since it is parameterized using information on the requested object-size distribution (see Appendix A). In this paper, the adaptability of C-LRU will be studied and validated in detail (see bellow).

The rest of this paper is organized as follows. In Section 2, we will first discuss some methods for the realization of the adaption of C-LRU based on periodical application and on neural networks. The results of the application and validation for the adaptive variant of C-LRU are shown in Section 3. Finally, the paper is concluded in Section 4.

2 Adaptive methods for C-LRU

The Class-based Least Recently Used (C-LRU) caching strategy has been implemented with the aim to get a balance between large and small documents in the cache [10, 19, 20]. Consequently, it has been shown in earlier studies that this method provides good results for the hit rate as well as for the byte hit rate (for more detail see [10, 19, 20]).

The C-LRU is based on the use of the EM algorithm to compute a hyper-exponential distribution function from the observed and collected measurements data. In that way, he C-LRU method allows for an adaptive caching strategy, since it sets its parameters depending on the observed workload. This property is very mandatory for web caching algorithms due to the observed fast changes in the web traffic. A compact introduction in the C-LRU is given in appendix A.

The C-LRU approach exploits characteristics of the requested objects, so it is important to have an accurate description of the requested objects [20]. Therefore, we have to determine how often one has to adapt the characterization. As it has been shown in [10, 20], three possibilities for parameter computation of the CLRU algorithm can be deployed:

Static use (once-only determination): A sample of collected request log for a long Periode of time is chosen. The appropriate parameters are determined/computed once-only; it is assumed that the workload changes in the future are not very significant.

Periodical application: After a predetermined period of time, e.g., every 24 hour, one week or one year, one reanalyzes the object-size characteristics. If significant changes are recognized, the parameters of the caching strategy have to be recomputed and adapted accordingly.

Application on-demand: By observing the performance values, e. g., the hit rate, the byte hit rate or the response time, over the time, a decrease in performance might be observed, which can lead to an updating of the caching parameters by using some predefined thresholds mechanism.

(a few minutes for traces covering several months and hundreds of millions of requests) an important benefit of C-LRU over other approaches is that it is adaptive. The investigation of the two above adaptation strategies, the periodical application and the application on demand, and their performance implications on C-LRU will be studied bellow. In this paper, we compare the static use (the once-only determination) of the C-LRU, the periodical application and an use by demand. We check the adaptability for the C-LRU and examine the possibility for determining the distribution parameters in an efficient way.

2.1 Peridiocal application:

In this first variant, the C-LRU caching algorithm has to run after a pre-defined period of time, e. g. after 24 hours. Another possibility is to make the new computation depending on a certain number of observed requests. For our implementation, we will choose the second variant, e. g. after $1,000$, $10,000$, $100,000$ and $500,000$ detected new requests. As soon as a difference between the old and new distribution functions of the document sizes has been observed, a reconfiguration of the caching parameters has to be done. This reconfiguration should be done automatically in that new class boundaries r_i and class fractions p_i have to be adapted/recomputed, with the aim to get optimal results for the hit rate and/or byte hit rate.

2.2 Application on demand

In this second variant, a neural network has been implemented. There are many specific forms of neural networks; in this work we focus on the multi-layer perceptron (see Figure 1). The Multi Layer Perceptron (MLP) is a specific class of neural networks, it consists of multiple layers of computational units, usually interconnected in a feedforward way. This means that each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function ($f(x) = \frac{1}{1+e^{-x}}$) as an activation function. For more details see [1, 28].

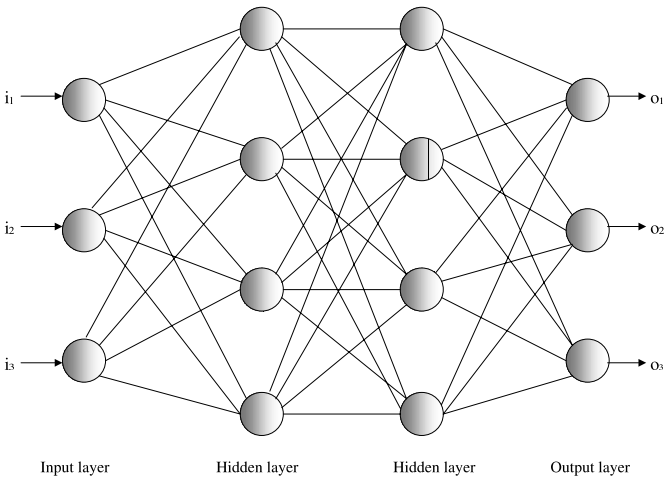

Figure 1: MLP with two hidden layers

Our objective hereby is to determine/recompute the optimale distribution parameters by the use of neural networks. For that, the number of the used neurons has been varied between 10 and 100. Furthermore two different neural networks have been implemented, the first one for the class boundaries r_i and the second one for the class fractions p_i. The definition and meaning of these both parameters (r_i and p_i) are given in appendix A.

3 Application and validation

For our analysis, we used three traces from the access log files of Proxy servers:

- The first and the second trace are stemming from the Technical University of Aachen, Germany (RWTH). To study the adapt-ability of our method, we use two access logs; one collected in early 2000 (RWTH2000) and the second in September 2002 (RWTH2002).
- The third used trace is the well-known Pittsburgh-Pennsylvania trace. It is a part of the *NLANR Cache Locations* [2]. This trace has been collected in 2002 and contains the description of requests during a time period of 43 days.

The traces are stemming from different areas, the users have also different behavior. The two traces RWTH2000 and RWTH2002 have a scientific nature and the PB2002 mirrors user behavior with different interests.

3.1 Statistical analysis of the traces

The important statistical information about the three traces are given in Table 1. A typical property of the web traffic can be seen, namely the heavy-tailed distributed document sizes for all these traces: high squared coefficients of variation and very small medians (compared to the means). This property has also been found in many other studies, e. g. in [5, 20, 23]. Furthermore, we can do the following remarks related to the changes of the web traffic over the time:

- The average of the object sizes becomes larger, and
- the fraction of dynamic objects becomes higher.

3.2 Periodical application

We simulated the C-LRU with three different periods depending on the number of the requested objects, namely after $1,000$, $10,000$, $100,000$ and $500,000$ requests for the three traces, the RWTH2000, the RWTH2002 and the PB2002. We compared the results of the periodical

Table 1: Statistics of the traces RWTH2000, RWTH2002 and PB2002

	RWTH 2000	RWTH 2002	PB 2002
total requests	32,341,063	22,875,618	24,386,018
static requests	31,139,386	21,428,326	21,906,745
dynamic requests	1,201,677	1,447,292	2,479,273
fraction static	96.28%	93.67%	89.83%
fraction dynamic	3.71%	6,32%	10.16%
total Byte	329.006 GB	318.279 GB	410.792 GB
Byte static	314.564 GB	301.793 GB	389.585 GB
Byte dynamic	14.441 GB	16.48 GB	21.207
fraction dynamic (B)	4.38%	5.17%	5.16%
smallest object	17 Byte	17 Byte	17 Byte
largest object	218 MB	858 MB	408 MB
average object size	10,846 Byte	15,122 Byte	19,095 Byte
median	2,240 Byte	2,400 Byte	2,980 Byte
squared coef. of var.	408.33	2,285,77	509.87

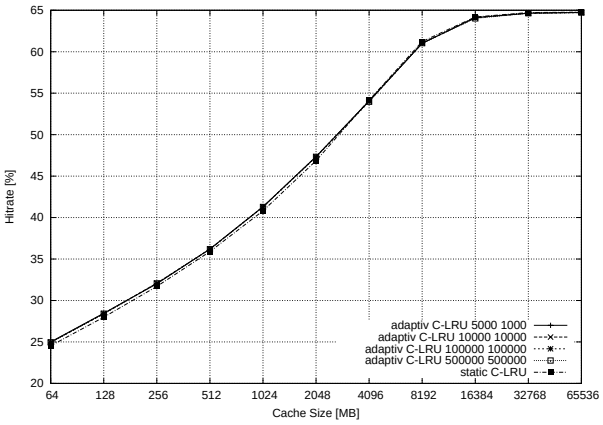

Figure 2: Hit rate: Comparison between static and adaptiv C-LRU for the RWTH2002 trace.

application with those obtained by the static application of C-LRU. The results for the RWTH2002 and PB2002 are plotted in Figure 2 for the hit rate and in Figure 3 for the byte hit rate, resp. in Figure 4 and Figure 5. Similar results have also been found for the periodical application for the RWTH2000 trace (see also [7]).

As it can be seen, the difference between the two variants is minimal, and can be neglected. The cause of this phenomenon can be explained as follows; the results for the static C-LRU are based on the EM application over a long period of time. During this long period, the algorithm sets its parameters which mirror the behavior of many users over this long period of time. For this reason, the parameters are also convenient for a transient allocation of the cache in the Web Proxy server.

This transient behavior can be found over many recurring

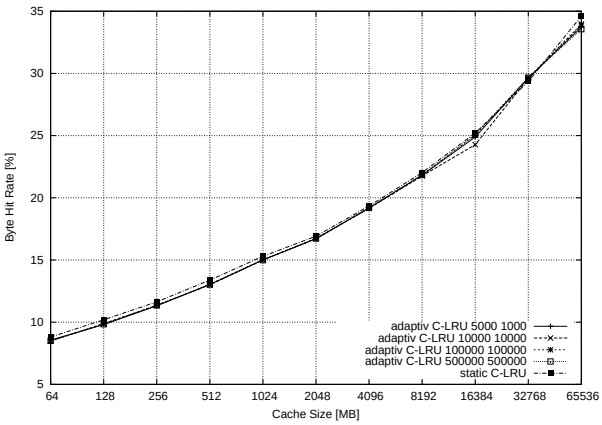

Figure 3: Byte hit rate: Comparison between static and adaptiv C-LRU for the RWTH2002 trace.

time intervals. Related to this, the efficiency of determining the C-LRU caching parameters can be seen here as an advantage of a periodical use of the C-LRU. Since for a new computation only a relative small number of measurements data is used for the EM-Algorithm. The EM-algortihm has a time complexity of $O(N \cdot I)$, where N the number of used measurement data and I is the number of phases in a hyper-exponential distribution [17, 18]. So the periodical use has in fact a lower run time, and delivers good results anyway.

3.3 Adaptivity by the use of neural networks

The idea behind the use of neural network is to learn the progression/evolution of the document sizes distribution over the time from the requests stream of the Proxy server and to adapt the cache configuration if necessary. This procedure corresponds to the execution of an EM algorithm run, where the hyper exponential distribution parameters are determined for the document sizes. To apply the neural network method on the data sizes distribution, it is necessary to transform it in a reduced vector description (feature analysis) and to retransfer the results to the real form/description thereafter.

Furthermore it is necessary to have an automated mechanism for detecting changes in document sizes distribution to signalize a new computation of the cache parameters. This is done by considering the hit rate and the Byte hit rate. These two parameters are monitored during the caching mode. By a significant decrease under a certain threshold, a distribution change will be assumed.

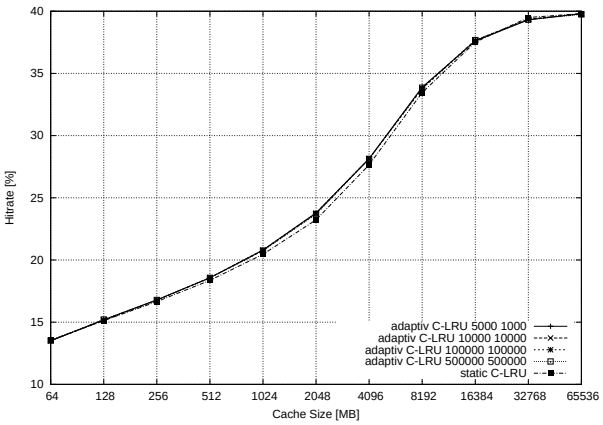

Figure 4: Hit rate: Comparison between static and adaptiv C-LRU for the PB2002 trace.

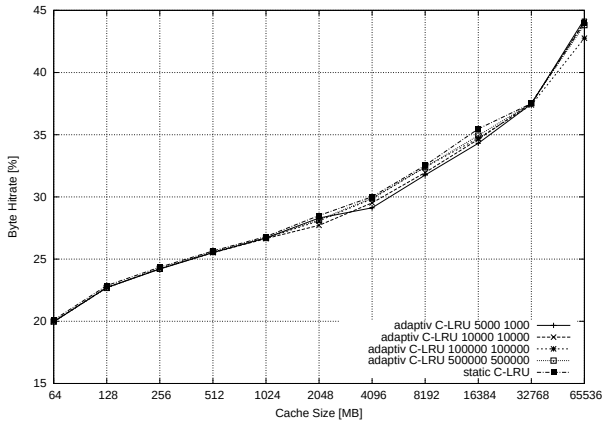

Figure 5: Byte hit rate: Comparison between static and adaptiv C-LRU for the PB2002 trace.

For the use of neural networks, two phases are needed; a training phase and a test/estimation phase. The objective of the 'training/learning phase' is to 'learn' from the training data. The collected knowledge in this first phase should be used in the test phase to show how far the neural networks are able to deliver the assumed correct output. This step is equivalent to classify the input vector adequately (optimizing a so called 'target function' [1, 28]). The neural network has to be trained to deliver the same results as the EM algorithm. A training data has the form: $(pattern, result)$. During the training phase many such tuples are generated; hereby the neural network tries to find the appropriate configuration of its intern configuration (layer weights). During the test phase the goodness of the training phase is validated using an error computation (e.g. MSE see bellow).

For a better representation, the amount of the data (here document sizes) considered is of a huge interest; the used data in the experiments should cover all possible range of values. By a given large trace T and a predetermined number of training data t; where each data has a documents, a window with size a will be shifted t times over the trace T with a chosen pre defined step size. By each step a feature vector will be generated and an EM algorithm run will be executed.

In this section, we will present some results obtained by the 'training' method for the RWTH2002 trace. First, we generated different training and test data:

- Distribution with $1,000,000$ sizes:
  - 200 training data with $25,000$ step sizes and,
  - 100 test data with $10,000$ step sizes.
- Distribution with $10,000$ sizes:
  - 400 training data with $12,500$ step sizes and,
  - 100 test data with $10,000$ step sizes.
- Distribution with $1,000$ sizes:
  - 500 training data with $10,000$ step sizes and,
  - 100 test data with $10,000$ step sizes.

The used neural network for the computing of the class boundaries r_i and class fractions p_i contains one input layer, one output layer and one hidden layer. The neural network has been analyzed for different input dimensions (that are the dimensions of the feature vectors at the input) and for a different number of neurons in the hidden layer.

The goal is to find an appropriate network structure/configuration in the hidden layer. The abort condition is defined by the SMSE value. When it becomes smaller than as a pre-defined bound, the training will end, and the results are then presented. To better evaluate the results, the average of the collected MSE by the Experiments has been used.

The goodness of the method can be estimated, e. g. by using the standard deviation. For our purpose, we use the scaled mean squared error (SMSE) as parameter for the goodness of the estimation. The MSE is defined as $MSE(\hat{\mu}, \mu) = E[(\hat{\mu} - \mu)^2]$, where $E[X]$ describes the mean function. Thereby, the SMSE is the average of the last 20 collected MSEs. The results for the class boundaries and for the class fractions can be shown in Table 2, resp. Table 3. For $1,000,000$ buffer size and only 10 neurons in the hidden layer, good results have been found for

Table 2: Neural networks: results for the class boundaries r_i

buffer size	neurons	abort SMSE	#iter.	Mean MSE
1,000,000	10	0.001	13,000	0.005936
1,000,000	20	0.001	18,000	0.006909
1,000,000	30	0.001	16,000	0.006263
1,000,000	40	0.001	18,000	0.008148
1,000,000	50	0.001	16,000	0.007057
10,000	10	0.01	19,000	0.189287
10,000	20	0.01	13,000	0.187664
10,000	30	0.01	18,000	0.181471
10,000	40	0.01	12,000	0.127080
10,000	50	0.01	10,000	0.182708
1,000	10	0.01	11,000	0.089712
1,000	20	0.01	15,000	0.082128
1,000	30	0.01	9,000	0.066266
1,000	40	0.01	16,000	0.094396
1,000	50	0.01	8,000	0.077687

Table 3: Neural networks: results for the class fractions p_i

buffer size	neurons	abort SMSE	#iter.	MSE
1,000,000	10	0.00001	6,000	0.0000210
1,000,000	20	0.00001	7,000	0.0000299
1,000,000	30	0.00001	8,000	0.0000396
1,000,000	40	0.00001	10,000	0.0000383
1,000,000	50	0.00001	9,000	0.0006751
10,000	10	0.00001	8,000	0.0000183
1,000	20	0.00001	8,000	0.0000293
10,000	30	0.00001	9,000	0.0000396
10,000	40	0.00001	11,000	0.0000378
10,000	50	0.00001	12,000	0.0006510
1,000	10	0.00001	7,000	0.0000184
1,000	20	0.00001	11,000	0.0000298
1,000	30	0.00001	9,000	0.0000388
1,000	40	0.00001	10,000	0.0000378
1,000	50	0.0001	9,000	0.0006712

the class boundaries as well as for the class fractions (see Table 2 and Table 3). The results delivered for a buffer size of $10,000$ and $1,000$ are only partially adequate. This can be shown by considering the stop criterion and the average of the MSEs; the neural network is not able to 'learn' if the MSE has reached a certain value regardless of the length of the training phase [7].

The results for the class boundaries and fractions for the optimal neural configurations are presented also in Table 4, resp. Table 5 showing the difference between the real input and the computed output results. Hereby the parameter *dim* describes the length of the input vector describing the distribution function.

4 Conclusions and outlook

The Class-based Least Recently Used (C-LRU) caching strategy has been implemented with the aim to get a balance between large and small document in the cache.

Table 4: Neural networks: optimal results for the class boundaries r_i

dim/ buffer size	class	min.dev.%	max.dev.%	mean dev.%
30 / 1,000,000	1	0.012	1.56	0.326
	2	0.356	2.63	0.863
	3	0.051	5.71	2.00
	4	0.076	20.58	11.31

Table 5: Neural networks: optimal results for the class boundaries p_i

dim/ buffer size	class	min.dev %	max.dev.%	mean dev.%
30 / 1,000,000	1	0.001	1.90	0.913
	2	0.003	3.19	0.968
	3	0.064	9.41	3.99
	4	0.379	42.3	14.91

Consequently, it has been shown in earlier studies that this method provides good results for the hit rate as well as for the byte hit rate. The C-LRU is based on the use of the EM algorithm to compute a hyper-exponential distribution function from the observed and collected measurements data. In that way, he C-LRU method allows for an adaptive caching strategy, since it sets its parameters depending on the observed workload. This property is very mandatory for web caching algorithms due to the observed fast changes in the web traffic.

In this study, we compared the static use of the C-LRU, a periodical application and an use by demand. We check the adaptability for the C-LRU and examine the possibility for determining the distribution parameters in an efficient way.

For the periodical use, the C-LRU has to be executed after a certain period of time which is equivalent to an execution after a certain number of observed/collected requests. We found that the advantage of this periodical application of the C-LRU lay primarily the efficiency of the computation.

For an application on demand, we used neural networks. Due to the self-similarity in the web traffic, which can be explained by the heavy-taildeness property of the involved distributions, e.g. for the document sizes, we excepted that these changes can be 'learned' with neural networks. The objective here is to 'learn' the distribution parameters, which are well approximated by the EM algorithm. We found, that neural networks are able to achieve this aim by adequately dimensioning the used paradigm.

In a new conducted study, we have also found that a periodical application of the scheduling algorithm WFQ (Weighted Fair Queueing) can be used for optimizing WWW server performance. We will report about our results in a separate paper [16].

A Appendix C-LRU

In this section, we will give a compact introduction of the Class-based Least Recently Used caching Algorithm (C-LRU). For more details, we refer to [10, 19, 20].

A.1 C-LRU: introduction

By the use of C-LRU, the cache is divided into portions reserved for objects of a specific size. Each portion (class or cluster) is reserved for documents of as specific size. The algorithm works as follows [20] (see Figure 6):

- The available memory for the cache is divided into I partitions where each partition i (for $i = 1, \ldots, I$) takes a specific fraction p_i of the cache ($0 < p_i < 1, \sum_i p_i = 1$).
- Partition i caches objects belonging to class i, where class i is defined to include all objects of size s with $r_{i-1} \leq s < r_i$ ($0 = r_0 < r_1 < \ldots < r_{I-1} < r_I = \infty$).
- Each partition in itself is managed with the LRU strategy.

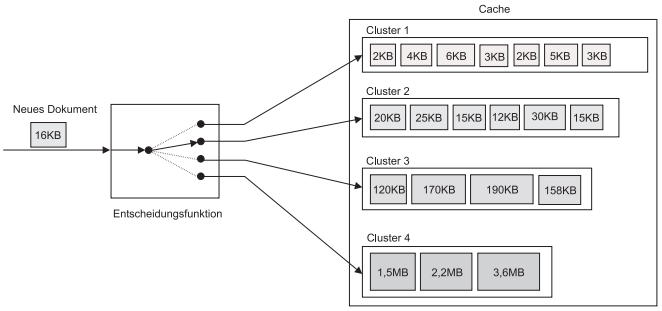

Figure 6: Principle of Class-LRU (adopted from [7])

A.2 C-LRU: Cache fractions p_i and boundaries r_i

The object sizes distribution is described by a hyper-exponential distribution using the EM-algorithm [17, 18, 20]. The object sizes density $f(x)$ takes the form of a probabilistic mixture of exponential terms:

$$f(x) = \sum_{i=1}^{I} c_i \lambda_i e^{-\lambda_i x}, \text{ with } \sum_{i=1}^{I} c_i = 1, and \quad 0 \leq c_i \leq 1, \quad \text{for } i = 1, \ldots, I. \tag{1}$$

Interpretation:
the weights c_i indicate the frequency of occurrence for objects of class i and the average size of objects in class i is given by $1/\lambda_i$.

Cache fractions p_i:
For the fraction p_i, two possible actions are proposed:

(a) $p_i = c_i$ for optimizing the hit rate

(b) $p_i = \frac{c_i/\lambda_i}{\sum_{j=1}^{I} c_j/\lambda_j}$ for optimizing the byte hit rate.

Cache boundaries r_i:
The range boundaries r_i are computed using Bayesian decision (see [10, 19, 20]);the appropriate class $C(s)$ for a given object is taken such that the decision error is minimized:

$$C(s) = \underset{i}{\text{argmax}} \ \{p(i) \cdot p(s|i)\}. \tag{2}$$

$$C(s) = \underset{i}{\text{argmax}} \ (c_i \lambda_i e^{-\lambda_i s}), \qquad i = 1, \ldots, I. \tag{3}$$

$$r_i = \frac{\ln(c_i \lambda_i) - \ln(c_{i+1} \lambda_{i+1})}{\lambda_i - \lambda_{i+1}}, \qquad i = 1, \ldots, I-1 \tag{4}$$

Table 6: Complexity for various cache operations. N: Number of Objects in the cache and I: Number of the classes in C-LRU [10, 19, 20].

	Hit/Miss	Insert	Delete	Update
LRU	$O(1)$	$O(1)$	$O(1)$	$O(1)$
SLRU	$O(1)$	$O(1)$	$O(1)$	$O(1)$
LRU-K	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$
LFU	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$
LFF	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$
GDS	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$
C-LRU	$O(1)$	$O(\log I)$	$O(\log I)$	$O(1)$

References

[1] www.free.definition.com.

[2] National Lab of Applied Network Research (NLANR). http://ircache.nlanrr.net/.

[3] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. Dbproxy: A Dynamic Data Cache for Web Applications. In *Proceedings of the 19th International Conference on Data Engineering*, pages 821–831, Bangalore, India, March 2003. IEEE Computer Society.

[4] M. F. Arlitt, R. Friedrich, and T. Jin. Workload Characterization of a Web Proxy in a Cable Modem Environment. In *Proceedings of ACM SIGMETRICS*, volume 27, pages 25–36, 1999.

[5] C. Cunha, A. Bestavaros, and M. Crovella. *Characteristics of WWW Clients-based Traces*. Technical Report TR-05-010, Boston University Department of Computer Science, August 1999.

[6] P. Cao and S. Irani. Cost-aware WWW Proxy Caching Algorithms. In *Proceedings of USENIX*, pages 193–206, Monterey, CA, December 1997.

[7] S. Celik. Adaptive Caching in Proxy Servern. Master's thesis, RWTH Aachen, Lehr- und Forschungsgebiet Leistungsbewertung und Verteilte Systeme, 2003.

[8] G. Chen, C.-L. Wang, and F. C. M. Lau. P-jigsaw: A Cluster-based Web Server With Cooperative Caching Support. *Concurrency and Computation: Practice and Experience*, 15(7-8):681–705, 2003.

[9] J. Gettys, T. Berners-Lee, and H. F. Nielsen. Replication and Caching Position Statement. http://www.w3.org/Propagation/activity.html, August 1997.

[10] B. R. Haverkort, R. El Abdouni Khayari, and R. Sadre. A Class- Based Least-Recently Used Caching Algorithm for WWW Proxies. In *Computer Performance Evaluation-Modelling Techniques and Tools*, pages 273–290. Lecture Notes in Computer Science 2324, Springer Verlag, Berlin, 2003.

[11] Boudewijn R. Haverkort. Model-based Self-Configuration for Quality of Service. Technical report, Department of Electrical Engineering, Mathematics and Computer Science, Chair for Design and Analysis of Communication Systems, University of Twente, 2004.

[12] H. Hemmati and R. Jalili. Self-reconfiguration in Highly Available Persative Computing Systems. In *Automatic and Trusted Computing*, pages 289–301. Lecture Notes in Computer Science, Springer Verlag, Heidelberg, 2008.

[13] S. Hosseini-Khayat. Improving Object Caching Performance Through Selective Placements. In *Parallel and Distributed Computing and Networks (PDCN)*, pages 262–265, 2006.

[14] R. El Abdouni Khayari. Ansatz zur Selbst-Rekonfiguration von Caches in Web Proxies. MMB-Mitteilungen, Heft 46, Herbst 2004. In German.

[15] R. EL Abdouni Khayari, M. Best, and A. Lehmann. Impact of Document Types on the Performance of Caching Algorithms in WWW Proxies: A Trace Driven Simulation Study. In *In Proceedings for the IEEE 19th International Conference on Advanced Networking Applications*, pages 737–742, Taipeh, Taiwan, 25-30 March 2005.

[16] R. El Abdouni Khayari, P. Fellinger, A. Musovic, and A. Lehmann. An Adaptive Scheduling Algorithm for Web Server: Impact Analysis of Web Workload. 2009 (to appear).

[17] R. El Abdouni Khayari, R. Sadre, and B. Haverkort. Fitting World-Wide Web Request Traces with the EM-Algorithm. *Performance Evaluation*, 52(2–3):175–191, April 2003.

[18] R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. Fitting World-Wide Web Request Traces with the EM-Algorithm. In R. van der Mei and F. Huebner-Szabo de Bucs, editors, *Proceedings of SPIE*, volume 4523, pages 211–220, Denver, USA, August 2001.

[19] R. El Abdouni Khayari, R. Sadre, and B.R. Haverkort. A Class-Based Least-Recently Used Caching Algorithm for WWW Proxies. In *Proceedings of the 2nd Polish-German Teletraffic Symposium*, pages 295–306, Gdansk, Poland, September 2002.

[20] Rachid El Abdouni Khayari. *Workload-Driven Design and Evaluation of Web-Based Systems*. ISBN: 3-89959-073-2. Der andere Verlag, Osnabrueck, Germany, 2003.

[21] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta Algorithm for Hierarchical Web Caches. In *Proceedings of the IEEE IPCCC*, Phoenix, Arizona, USA, April 2004.

[22] P. Lorenzetti and L. Rizzo. Replacement Policies for a Proxy Cache. *IEEE/ACM Transactions on Networking*, 8(2):158–170, 2000.

[23] M. Arlitt and C. Williamson. *Internet Web Servers: Workload Charactarization and Performance Implications*. volume 5, pages 631–645, IEEE/ACM Transactions on Networking, October 1997.

[24] S. V. Nagaraj. *Web Caching and Its Applications*. Kluver Academic Publishers, 2004.

[25] Netcraft. *Web Server Survey*. http://www.netcraft.co.uk/survey/, August, 2002.

[26] P. Cao and S. Irani. *Cost-Aware WWW Proxy Caching Algorithms*. volume 5, pages 193–206, Monterey, December 1997.

[27] A. Raabe. *Describung and Simulating Dynamic Reconfiguration in SystemC Exemplified by a Dedicated 3D Collision Detection Hardware*. Dissertation, Rheinische Friedrich Wilhelms Universitaet Bonn, April 2008.

[28] R. Rojas. *Neural Networks- A Systematic Introduction*. Springer Verlag, Berlin, New York, 1996.

[29] G. Santhanakrishnan, A. Amer, and P. K. Chrysanthis. Self-Tuning Caching: The Universal Caching Algorithm. *Software: Practice and Experience. Special Isuue: Experiments with Auto-Adaptive and Reconfigurable Systems*, 36(11-12):1179–1188, August 2006.

[30] Maarten Wegdam. *Dymnamic Reconfiguration and Load Distribution in Component Middleware*. PhD thesis, University of Twente, Netherlands, 2004.

[31] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. In *Proceedings of the ACM SIGCOMM Conference*, pages 293–305. ACM Press, August 1996.

[32] D. Zeng and F. Wang. Efficient Web Content Delivery Using Proxy Caching Techniques. *IEEE Transactions on Systems, Mn and Cybernetics*, 34(3):270–280, 2004.