A Fuzzy Algorithm for Web Caching[*]

Maria Carla Calzarossa
Dipartimento di Informatica e Sistemistica
Università di Pavia
I-27100 Pavia, Italy
mcc@unipv.it

Giacomo Valli
Dipartimento di Informatica e Sistemistica
Università di Pavia
I-27100 Pavia, Italy
giavl@tin.it

ABSTRACT

The large popularity of web services and applications makes their performance very critical. Reducing the latency to retrieve web pages has become a real challenge. Caching is widely employed for this purpose. In this paper, we address web caching by studying replacement policies based on fuzzy logic. We propose an algorithm that applies a set of fuzzy control rules to identify the pages to evict from the cache. We study the performance of our algorithm via trace-driven simulations using traces collected on various proxy servers.

Keywords
Web caching, fuzzy logic, replacement algorithm, performance evaluation.

1. INTRODUCTION

Quality of Service and performance experienced by end users are critical issues in the web domain. It is widely recognized that slow web sites are the primary source of user dissatisfaction. As web services and applications are gaining larger and larger popularity, reducing the latency to retrieve web pages has become a real challenge. Caching is a mechanism widely employed for this purpose. The idea of caching is to store popular objects "closer" to the user who requests them such that they can be retrieved faster. Caching has also the effect of reducing the load of the web servers and the traffic over the network.

[*]This work was supported by the Italian Ministry of Education, University and Research (MIUR) under the FIRB and PRIN programmes and by the University of Pavia under the FAR programme.

Web caching can be implemented at different levels, i.e., client, server, network. The web browser and the web server are responsible for caching at the client and at the server side, respectively. Proxy servers are used for caching at the network level.

A proxy server acts as an intermediary between clients and web servers. Many organizations use proxy servers in front of their LANs to save network bandwidth and speedup web pages retrieval by serving the requests locally. Upon receiving requests originating by multiple clients, a proxy server checks its cache to see whether it can serve these requests without accessing the original web servers. In the case of a cache miss, that is, the requested page is not stored in the proxy cache or it has expired, the proxy server forwards the request to the web server. Once the page is returned by the server, the proxy sends it back to the client and stores a copy in its local cache for future requests. If the cache is full, one or more pages have to be evicted to store the newly accessed page.

The efficiency and performance of proxy caches mainly depend on their design and management. Replacement policies play a key role for the effectiveness of caching. The goal of replacement policies is to make the best use of the available resources by dynamically selecting the pages to be cached or evicted.

Replacement policies have been extensively studied and many papers appeared in the literature (see e.g., [1], [2], [3], [4], [5], [6], [8], [11], [12], [14], [15]). A few policies, e.g., Least Recently Used (LRU), Least Frequently Used (LFU), are direct extensions of the traditional replacement algorithms typical of the operating system domain. Other policies, e.g., Greedy

Dual–Size (GDS), have been explicitly designed for web environments.

Many generalizations of both traditional and web specific policies have been proposed. In [2], the LRU algorithm is generalized such as to take into account access costs and expiration times. A generalization of the GDS algorithm that incorporates short term temporal locality and long term popularity of web request streams is presented in [5]. In [11] randomized algorithms have been applied for approximating any existing web cache scheme. The algorithms sample a few pages and replace the least useful pages of the sample. In [14], the replacement strategies are addressed in the framework of a model for optimizing the content of the web cache. The model is based on a genetic algorithm or an evolutionary programming scheme.

In this paper, we address web caching replacement policies in the framework of the fuzzy logic. Our choice is motivated by the need to base the replacement process on both qualitative and quantitative information. Our algorithm takes into account the characteristics and the properties of the workloads of proxy servers and applies some qualitative reasoning to identify the pages to evict from the cache. The variables describing each web page are first "fuzzified". A set of fuzzy control rules is then applied and their outputs are "defuzzified" as to identify the pages to evict.

The paper is organized as follows. Section 2 introduces the fuzzy rules and describes the main steps of the fuzzy algorithm. Section 3 discusses the performance achieved by the algorithm using trace-driven simulations with traces collected on various proxy servers. Finally, Section 4 draws a few conclusions and outlines future research directions.

2. THE FUZZY ALGORITHM

As previously stated, the replacement algorithm proposed in this paper relies on fuzzy logic (see e.g., [7], [9], [13]). When a cache miss occurs and the cache is full, the algorithm determines the pages to evict by computing for each page in the cache a figure of merit, namely, its probability of replacement. Among the pages ranked according to their probability of replacement, the algorithm chooses the pages with the highest rank. In details, the first step towards the construction of the algorithm deals with its design, that is:

- identification of the input and output variables;
- definition of the Memberships Functions of each input and output variable;
- construction of the rule base.

The proper choice of process state input variables and control output variables is essential to the characterization of the operation of a fuzzy system. Hence, as a preliminary step towards the construction of the fuzzy algorithm, we have analyzed various proxy servers of the NLANR cache hierarchy [10] with the aim of understanding the properties and dynamic behavior of their workloads. As a result of these analyses, we have chosen three input variables to represent the process state. These variables describe each web page in terms of its size (`Size`), access frequency (`Frequency`), i.e., number of accesses, and access recency (`Time`), i.e., time elapsed since last access. As output variable, we have chosen the probability of replacement (`RP`) of each page.
For each of these variables, we have defined the fuzzy sets with the Membership Functions (MFs) describing the degree of membership of the variable to the corresponding fuzzy set.

Figures 1, 2 and 3 show the MFs of the three input variables. As can be seen, we used MFs having triangular or trapezoidal shapes. There are three MFs associated with the variables `Size` and `Frequency`. `LOW, MEDIUM, HIGH` have been used as descriptive linguistic values, i.e., labels, for these MFs.
To describe the variable `Time` we have chosen five MFs. This was because the algorithm requires a finer control of this variable. The corresponding descriptive labels are `VERY LOW, LOW, MEDIUM, HIGH, VERY HIGH`. The centers and the left and right limits of the MFs have been obtained as a result of the analysis of the proxy workloads. Note that the values shown

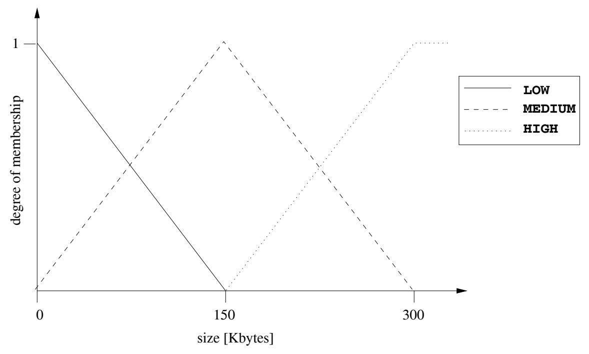

Figure 1: Membership Functions of the variable `Size`.

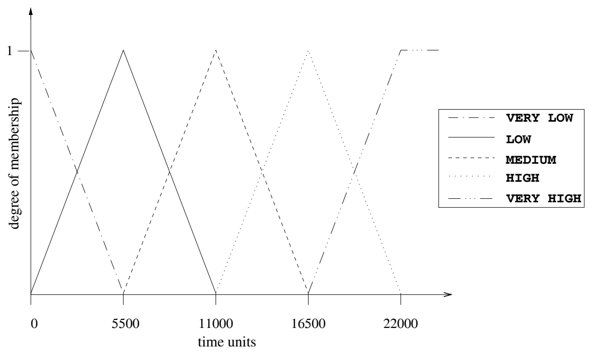

Figure 3: Membership Functions of the variable `Time`.

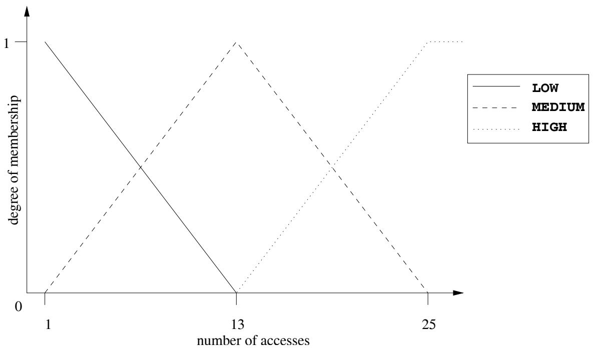

Figure 2: Membership Functions of the variable `Frequency`.

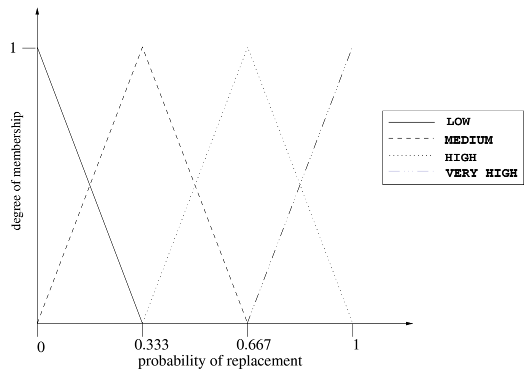

Figure 4: Membership Functions of the variable `RP`.

in Fig. 3 correspond to the initializations as the centers of these MFs have to be dynamically updated according to the values computed for all the pages in the cache.

Figure 4 shows the MFs of the output variable, that is, the probability of replacement `RP`. As can be seen, four MFs, with descriptive labels `LOW, MEDIUM, HIGH, VERY HIGH`, have been associated with this variable.

Having defined the MFs, we construct the rule base. The rule base consists of fuzzy conditional statements in the form `''if-then''` in which the antecedent is a condition in its application domain and the consequent is a control action for the system under control. Each rule involves in the antecedent one or more variables.

There is no general procedure for deciding on the optimal number of fuzzy control rules and the role of each variable. A number of factors is involved in this decision. In our case, the rules are based on a quantitative understanding of proxy workloads followed by a calibration phase performed via simulations. We have defined two sets of rules. The first set (`Fuzzy20`) consists of 20 rules, whereas the second set (`Fuzzy12`) consists of 12 rules. The rationale behind these rules is to identify a few "well-defined" situations and let the algorithm choose otherwise. The aim of `Fuzzy20` is to keep in the cache the pages that have been accessed "very" recently, and to evict "large" pages. Moreover, among pages with similar size, the rules penalize the pages characterized by a "small" number of accesses or by not being accessed "very" recently. Figure 5 presents the `Fuzzy12` rules. As can be seen, the 12 rules have been designed as to take into account the variable `Size` only when a page is "large", namely, larger than the corresponding average.

```
if (Frequency is LOW) and (Time is VHI) and (Size is MED) then (RP is VHI)
if (Frequency is LOW) and (Time is HIG) and (Size is HIG) then (RP is VHI)
if (Frequency is MED) and (Time is VHI) and (Size is HIG) then (RP is VHI)
if (Frequency is LOW) and (Time is VHI) and (Size is HIG) then (RP is VHI)
if (Frequency is LOW) and (Time is HIG) and (Size is LOW) then (RP is HIG)
if (Frequency is MED) and (Time is HIG) and (Size is LOW) then (RP is MED)
if (Frequency is MED) and (Time is VHI) and (Size is MED) then (RP is HIG)
if (Frequency is MED) and (Time is HIG) and (Size is HIG) then (RP is HIG)
if (Frequency is HIG) and (Time is VHI) and (Size is HIG) then (RP is LOW)
if (Frequency is HIG) and (Time is HIG) and (Size is HIG) then (RP is LOW)
if (Frequency is LOW) and (Time is MED) and (Size is HIG) then (RP is HIG)
if (Frequency is MED) and (Time is HIG) and (Size is MED) then (RP is MED)
```

Figure 5: `Fuzzy12` rules.

Once the design parameters have been defined, the fuzzy algorithm proceeds as follows:

1. measurement of the values of the input data from the proxy server;
2. fuzzification of the crisp input data into fuzzy sets;
3. inference from fuzzy rules;
4. aggregation across the rules and defuzzification of the fuzzy output into a non fuzzy control action.

More specifically, the fuzzification has the effect of scaling and mapping crisp input data into fuzzy sets by means of the corresponding Membership Functions. The input values related to each page are translated into linguistic concepts. For each rule, the antecedent is evaluated and the degree of truth is computed by applying the fuzzy `and` operator, that is, the product. The aggregation process combines the outputs of the rules by applying the `maximum` operator to each descriptive label of the output variable `RP` (i.e., the probability of replacement). The defuzzification transforms these four values into a nonfuzzy control action corresponding to the probability of replacement of the page. The defuzzification used in our algorithm is based on the method of the centroid. The masses, obtained as a result of the aggregation process, have been placed at the three points where the MFs of the output variable `RP` intersect, that is, at the points 0.25, 0.5, and 0.75. Moreover, the mass corresponding to the label `VERY HIGH` has been placed at 1.

As a final step, the pages are ranked according to their probability of replacement. The algorithm then evicts the pages with the highest rank.

Note that in what follows, we denote our fuzzy algorithm by Fuzzy20 and Fuzzy12 according to the set of rules used.

3. PERFORMANCE EVALUATION

The evaluation of the performance of our fuzzy algorithm is based on trace–driven simulations. The simulations use traces of four proxies of the NLANR cache hierarchy [10]. SJ, UC, STARTAP and SV denote the NLANR sites where the traces were collected. The choice of these sites is motivated by the need to assess the performance and the sensitivity of the algorithm under different workloads.
Table 1 summarizes the characteristics of the traces. The traces were recorded during spring and summer 2002. For each trace we report the number of unique pages and their total size (Mbytes). Moreover, we list the percentage of the so–called "one–timers", that is, the pages that are requested only once and, as such, are not worth to be cached. As can be seen, the workloads of the four proxies are very different in terms of intensity and characteristics of the pages being re-

Trace	Period	Requests	Pages	Mbytes	One–timers
SJ	03/04–04/04	5,512,899	2,178,893	26,407.49	0.74
UC	03/11–03/15	2,420,731	957,878	20,689.09	0.75
STARTAP	07/16–07/22	1,595,730	372,440	8,683.03	0.63
SV	08/15–08/21	2,856,904	795,351	22,102.10	0.61

Table 1: Main characteristics of the NLANR traces.

quested. In particular, the proxy UC processes on the average 484,146 requests per day, whereas the workload intensity of the proxy SJ is much smaller. It processes only about 172,278 requests per day. The average size of unique pages is also quite different across the proxies. It is about 28Kbytes for the SV proxy, and 12Kbytes for the SJ proxy. Similar considerations can be drawn for the percentage of one–timers, that is equal to 61% for the SV proxy and to 75% for the UC proxy.

Before using these traces in the simulations, we performed some pre–processing with the aim of excluding the requests corresponding to non-cacheable pages (e.g., cgi–bin, .exe). On the average, we removed about 10% of the requests. For example, we removed 8.16% of the requests of the SJ trace and 14.2% of the requests of the SV trace.

In the simulations, we compare our fuzzy algorithm (Fuzzy20 and Fuzzy12) with the LFU, LRU, and GDS replacement algorithms, as they represent a reference in the framework of web caching. The efficiency of these replacement policies is evaluated as a function of the cache size. In our experiments, the cache size varies from a size of less than 2% to about 19% of the total number of unique bytes in the trace, hereafter denoted as cache capacity.

Two performance metrics, namely, Hit Rate (HR) and Byte Hit Rate (BHR), are used to evaluate the efficiency of the replacement algorithms. The Hit Rate is a standard metrics in the cache domain that measures the fraction of requested pages retrieved in the cache. The Byte Hit Rate is a metrics specific for the web domain in that it takes into account the nonhomogeneity of the sizes of web pages. It is a measure of the fraction of requested bytes retrieved directly from the cache. Note that Fuzzy20 is aimed at maximizing the performance of the cache expressed in terms of the Hit Rate, whereas Fuzzy12 is aimed at maximizing both the Hit Rate and Byte Hit Rate.

Figure 6 shows, as a function of the cache size, the Hit Rate of the algorithms resulting from the simulations with the SJ trace. As can be seen, both Fuzzy20

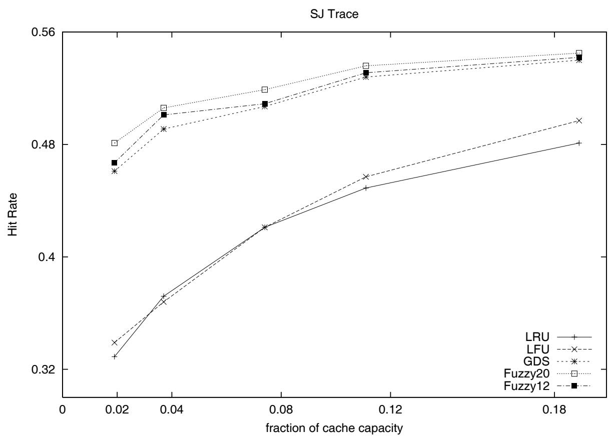

Figure 6: Hit Rate obtained in the experiments with the SJ trace.

and Fuzzy12 outperform the others algorithms under test, especially in the case of small cache size, i.e., less than 10% of the cache capacity, that is the typical size of most proxy caches. Moreover, as the cache capacity increases, the three top curves approach the upper bound HR_∞, equal to 0.548, corresponding to a cache of infinite capacity where no eviction is required. It should also be noted that Fuzzy20 and Fuzzy12 achieve more than 90% of HR_∞ with a cache size as low as the 5% of its capacity.

We noticed that in the experiments performed with the other three traces, the algorithms exhibit a behavior similar to what shown in Fig. 6. As an example, Figure 7 shows the behavior of the HR obtained in the simulations with the STARTAP trace. As can be seen, despite of the characteristics of the workload of this proxy, the performance of Fuzzy20 and Fuzzy12 is as good as the performance of GDS. Note that in this case, the value of HR_∞ is much larger, namely, equal to 0.786.

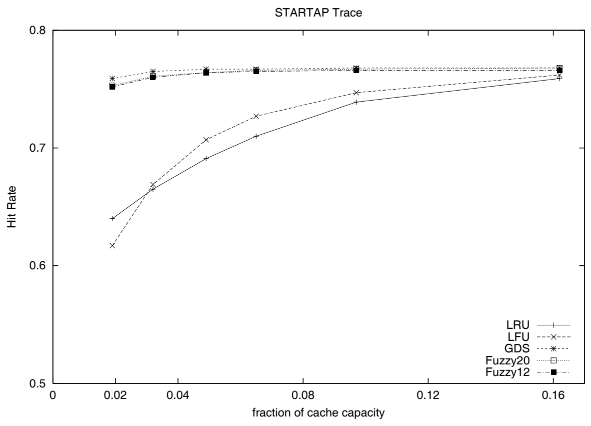

Figure 7: Hit Rate obtained in the experiments with the STARTAP trace.

The analysis of the performance of the algorithms in terms of BHR shows that the algorithms exhibit different behaviors. Figure 8 shows the BHR for simulations of the SJ trace. None of the algorithms outperforms the others. As the cache capacity increases, the LFU achieves the best BHR, even though the BHR of Fuzzy12 is only 0.5% smaller. For this trace the value of BHR_∞, obtained with an infinite cache, is equal to 0.634.

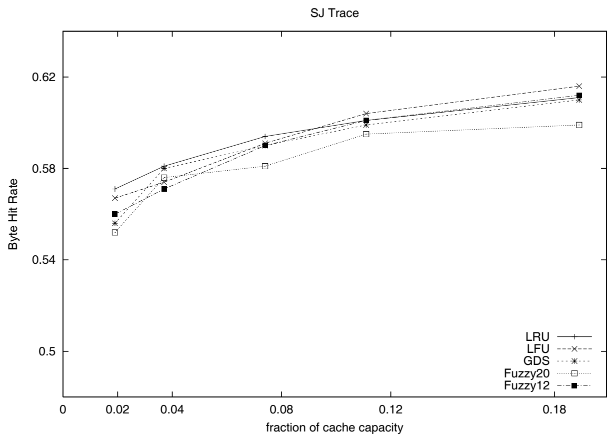

Figure 8: Byte Hit Rate obtained in the experiments with the SJ trace.

Figure 9 shows the BHR obtained in the simulations with the UC trace. As can be seen, Fuzzy12 outperforms the other algorithms, whereas Fuzzy20 is the worst algorithm. The value of its Byte Hit Rate is equal to 0.321 for a cache size equal to 1.7% of the cache capacity and to 0.371 for a cache size equal to 14% of the cache capacity. For this trace the value of BHR_∞ is equal to 0.455. It is also interesting to no-

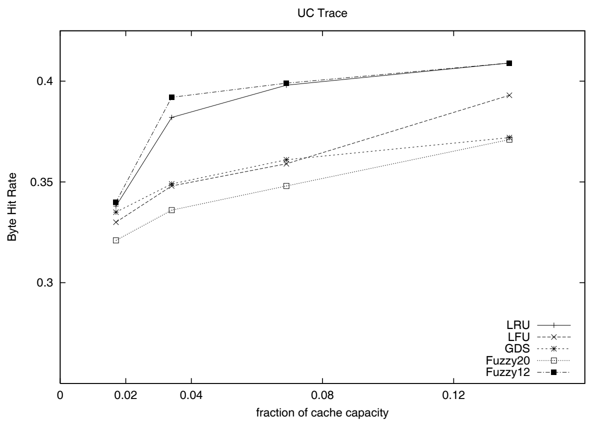

Figure 9: Byte Hit Rate obtained in the experiments with the UC trace.

tice that the relative difference between the best and the worst algorithm, with a cache size of about 3.4% of the cache capacity, is about 15%. This means that the efficiency of proxy caching can be significantly improved by replacement policies aimed at maximizing the Byte Hit Rate.

In summary, we can conclude that both Fuzzy20 and Fuzzy12 achieve good performance in terms of Hit Rate, whereas the efficiency of Fuzzy12 is larger when considering the Byte Hit Rate. Moreover, the performance of our fuzzy algorithm is at least as good as the performance of the GDS algorithm. Furthermore, the results of the simulations have also shown another good property of the fuzzy algorithm, that is, its ability to easily adapt to the characteristics of the workload.

4. CONCLUSIONS

Replacement policies play a key role for the effectiveness of web caching. The replacement algorithm proposed in this paper is based on the fuzzy logic.

The choice of the pages to be evicted from the cache is based on qualitative reasoning that takes into account the page characteristics.

The complexity of our algorithm is of the order of the number of pages in the cache as it evaluates for each page its probability of replacement. However, even though this complexity is larger than the complexity of most of the algorithms proposed in the literature, we do not think it is an issue. The workload of a proxy server is typically I/O bound and the processor is never the bottleneck of the system. Hence, we believe that it is worth to invest a few extra CPU cycles in a replacement policy that helps to save disk and network accesses.
Moreover, the results of our simulations have shown that the fuzzy algorithm achieves good performance even for small cache size, that is, less than 10% of the cache capacity. This means that the fuzzy approach allows a dramatic savings of the disk space to be allocated for caching.

As a future work, we plan to address the caching of dynamic pages in the framework of fuzzy logic. Our aim is to define policies that take into account both the availability of the pages and their attributes, such as, freshness and consistency.

5. REFERENCES

[1] M. Abrams, C.R. Standridge, G. Abdulla, S. Williams, and E.A. Fox. Caching Proxies: Limitations and Potentials. In *Proc. Int. World Wide Web Conference*, pages 119–133, 1995.

[2] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the World Wide Web. *IEEE Trans. on Knowledge and Data Engineering*, 11(1):94–107, 1999.

[3] P. Cao and S. Irani. Cost-Aware WWW Proxy Caching Algorithm. In *Proc. USENIX Symp. on Internet Technologies and Systems*, pages 193–206, 1997.

[4] J. Dilley and M. Arlitt. Improving Proxy Cache Performance: Analysis of Three Cache Replacement Policies. *IEEE Internet Computing*, 3(6):44–50, 1999.

[5] S. Jin and A. Bestavros. GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams. *Computer Communications*, 2(24):174–183, 2000.

[6] S. Jin and A. Bestavros. Popularity-Aware Greedy Dual-Size Web Proxy Caching Algorithms. In *Proc. IEEE ICDCS'00*, pages 254–261, 2000.

[7] C. C. Lee. Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Parts I and II. *IEEE Trans. on Systems, Man and Cybern.*, 20(2):404–435, 1990.

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

[9] T. Munakata and Y. Jani. Fuzzy Systems: An Overview. *Comm. of the ACM*, 37(3):68–76, 1994.

[10] National Laboratory for Applied Network Research (NLANR). http://www.nlanr.net.

[11] K. Psounis and B. Prabhakar. A Randomized Web-Cache Replacement Scheme. In *Proc. IEEE INFOCOM 2001*, pages 1407–1415, 2001.

[12] J. Shim, P. Scheuermann, and R. Vingralek. Proxy Cache Algorithms: Design, Implementation, and Performance. *IEEE Trans. on Knowledge and Data Engineering*, 11(4):549–562, 1999.

[13] T. Terano, K. Asai, and M. Sugeno. *Fuzzy Systems Theory and Its Applications.* Academic Press, 1992.

[14] A. Vakali. Evolutionary Techniques for Web Caching. *Distributed and Parallel Databases - An International Journal*, 11(1):93–116, 2002.

[15] C. Williamson. On Filter Effects in Web Caching Hierarchies. *ACM Trans. on Internet Technology*, 2(1):47–77, 2002.