# Using Fuzzy Logic to Improve Cache Replacement Decisions

*Mojtaba Sabeghi1,† and Mohammad Hossein Yaghmaee2††,*

Ferdowsi University of Mashhad, Mashhad, Iran

**Summary**
Most researches concerning uniform caching base their replacement decision on just one parameter. This parameter in some cases may not do well because of the workload characteristics. Some others use more than one parameter. In this case, finding the relation between these parameters and how to combine them is another problem. A number of algorithms try to combine their decision parameter with some mathematical equations. But as different workloads have different characteristics, it is not possible to express the parameters relation with an exact mathematical formula. In real world situations, it would often be more realistic to find viable compromises between these parameters. For many problems, it makes sense to partially consider each of them. One especially straightforward method to achieve this is the modeling of these parameters through fuzzy logic. This paper proposes a fuzzy algorithm in which the decision parameters are treated as fuzzy variables. A simulation is also performed and the results are compared with Optimal, LRU and LFU replacement algorithms. The latter two algorithms are the most commonly used algorithms for replacement of cache objects and the first one is a theoretical optimal algorithm. It is concluded that the proposed fuzzy approach is very promising and it has the potential to be considered for future research.

***Key words:***
*Fuzzy logic, page replacement algorithm, LRU, LFU, FPR.*

## Introduction

With the ever-widening speed gap between computing elements and memory units in modern computing systems, Caching finds extensive application in storage systems, databases, Web servers, middleware, processors, file systems, disk drives, redundant array of independent disks controllers, operating systems, and other applications such as data compression and list updating. [1].

Both cache and auxiliary memory handle uniformly sized items called *pages*. Requests for pages go first to the cache. When a page is found in the cache, a *hit* occurs; otherwise, a cache *miss* happens, and the request goes to the auxiliary memory.

In the latter case, a copy is *paged in* to the cache. This practice, called *demand paging*, rules out prefetching pages from the auxiliary memory into the cache. If the cache is full, before the system can page in a new page, it must page out one of the currently cached pages. A *replacement policy* determines which page is evicted.

Under demand paging, the only question of interest is: When the cache is full and a new page must be inserted in the cache, which page should be replaced.

A good policy reduces the number of media accesses by choosing to evict pages that will not be used for some time, thus keeping in memory pages that will be used soon. Notice that choosing well requires that the policy guess whether or not a page is likely to be needed soon. Such a choice demands certain assumptions about how a program is likely to reference its pages in the future, making page replacement a difficult and impractical problem. Because page replacement was initially greeted with such skepticism, researchers analyzed reference behavior and developed important concepts, such as the principle of locality [11] which states that only a small fraction of a program's memory is used most of time. This concept exhibited by many programs made the paging aspect of virtual memory practical.

The basic idea of locality is that data which is referenced is close to other data that has been and will be referenced. Notice that the term close can be interpreted in at least two ways, most commonly in terms of both space and time. Therefore, both spatial and temporal localities are discussed when analyzing reference behavior and designing replacement algorithms. Two pieces of data are temporally local if they are accessed at nearly the same time during program execution. Two pieces of data are spatially local if they are near to one another in the address space.

Belady's MIN is an optimal, offline policy for replacing the page in the cache that is used farthest in the future. This algorithm could cheat by examining future references, therefore avoiding poor decisions [12].

Of course, as mentioned before, in practice, we are only interested in online cache replacement policies that do not demand any prior knowledge of the workload. Page replacement, in real systems, must be done online and the policy must decide which page to evict based only on information it has collected about previously used pages. In other words, a page replacement policy must use past information to predict the future.

Manuscript received March 25, 2006.
Manuscript revised March 30 , 2006.

The LRU policy always replaces the least recently used page. LRU exploits spatial locality in request sequence and the recency property which states recently requested objects are more likely to be requested next than objects that have not been requested recently [1,3].
In use for decades, this policy has undergone numerous approximations and improvements. Three of the most important related algorithms are Clock, WS (working set), and WSClock. So far, LRU and its variants are amongst the most popular replacement policies [2].
The advantages of LRU are that it is extremely simple to implement, has constant time and space overhead, and captures recency or clustered locality of reference that is common to many workloads. One main disadvantage of the LRU algorithm is as follows: While LRU captures the recency features of a workload; it does not capture and exploit the frequency features of a workload. More generally, if some pages are often requested, but the temporal distance between consecutive requests is larger than the cache size, then LRU cannot take advantage of such pages with long-term utility.
LFU is another policy that seems to perform well in web request sequences [3]. LFU evicts the pages in the cache that was requested the fewest number of times. LFU exploits the frequency property of request sequence which states that pages that have been requested more times are more likely to be requested again. This is exactly what we called it temporal locality. As a disadvantages, it does not adapt well to variable access patterns; it accumulates stale pages with past high frequency counts, which may no longer be useful.
The last fifteen years have seen development of a number of novel caching algorithms that have attempted to combine recency and frequency. There is an association between recency and frequency; if a recently used page is likely to be used soon, then such a page will be used frequently. The least recently frequently used (LRFU) Policy is one if them [3, 5].
The LFFU policy associates a value with each block. This value is called CRF (Combined recency and frequency) and quantifies the likelihood that the block will be referenced in the near future. This value is calculated according to a mathematical equation that is going to mix recency and frequency and comes to a single decision parameter.
Although, there is a correlation between recency and frequency but, as experiences show, this correlation is not the same for all kind of workloads. So, describing this relation with an exact mathematical formula is impossible.
In real world situations, it would often be more realistic to find viable compromises between these parameters. For many problems, it makes sense to partially consider each of them. One especially straightforward method to achieve this is the modeling of these parameters through fuzzy logic. Using fuzzy rules we can combine these parameters as they are connected in real worlds.
The scope of the paper is confined to replacement of uniform cache objects with fuzzy logic. The rest of the paper is organized as follow. In section 2 the fuzzy inference systems are discussed. Section 3 covers the proposed model and section 4 contains the experimental results. Conclusion and future works are debated in Sections 5.

## 2. Fuzzy Inference System

Fuzzy logic is an extension of Boolean logic dealing with the concept of partial truth which denotes the extent to which a proposition is true. Whereas classical logic holds that everything can be expressed in binary terms (0 or 1, black or white, yes or no), fuzzy logic replaces Boolean truth values with a degree of truth. Degree of truth is often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision.
Fuzzy Inference Systems (FIS) are conceptually very simple. They consist of an input, a processing, and an output stage. The input stage maps the inputs, such as frequency of reference, recency of reference, and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a corresponding result. It then combines the results. Finally, the output stage converts the combined result back into a specific output value [6].
The membership function of a fuzzy set corresponds to the indicator function of the classical sets. It is a curve that defines how each point in the input space is mapped to a membership value or a degree of truth between 0 and 1. The most common shape of a membership function is triangular, although trapezoidal and bell curves are also used. The input space is sometimes referred to as the universe of discourse [6].
As discussed earlier, the processing stage which is called inference engine is based on a collection of logic rules in the form of IF-THEN statements where the IF part is called the "antecedent" and the THEN part is called the "consequent". Typical fuzzy inference systems have dozens of rules. These rules are stored in a knowledgebase. An example of a fuzzy IF-THEN rule is: IF *Frequency* is *low* then *SwapPriority* is *very high*, which frequency and priority are linguistics variables and low and very high are linguistics terms. Each linguistic term corresponds to membership function.

An inference engine tries to process the given inputs and produce an output by consulting an existing knowledgebase. The five steps toward a fuzzy inference are as follows:

- Fuzzifying Inputs
- Applying Fuzzy Operators
- Applying Implication Methods
- Aggregating All Outputs
- Defuzzifying outputs

Bellow is a quick review of these steps but a detailed study is not in the scope of this paper.

Fuzzifying the inputs is the act of determining the degree to which they belong to each of the appropriate fuzzy sets via membership functions. Once the inputs have been fuzzified, the degree to which each part of the antecedent has been satisfied for each rule is known. If the antecedent of a given rule has more than one part, the fuzzy operator is applied to obtain one value that represents the result of the antecedent for that rule. The implication function then modifies that output fuzzy set to the degree specified by the antecedent. Since decisions are based on the testing of all of the rules in an FIS, the results from each rule must be combined in order to make a decision. Aggregation is the process by which the fuzzy sets that represent the outputs of each rule are combined into a single fuzzy set. The input for the defuzzification process is the aggregated output fuzzy set and the output is a single value. This can be summarized as follows: mapping input characteristics to input membership functions, input membership function to rules, rules to a set of output characteristics, output characteristics to output membership functions, and the output membership function to a single-valued output.

There are two common inference processes [6]. First is called Mamdani's fuzzy inference method proposed in 1975 by Ebrahim Mamdani [8] and the other is Takagi-Sugeno-Kang, or simply Sugeno, method of fuzzy inference Introduced in 1985 [9]. These two methods are the same in many respects, such as the procedure of fuzzifying the inputs and fuzzy operators.

The main difference between Mamdani and Sugeno is that the Sugeno output membership functions are either linear or constant but Mamdani's inference expects the output membership functions to be fuzzy sets.

Sugeno's method has three advantages. First it is computationally efficient, which is an essential benefit to real-time systems. Second, it works well with optimization and adaptive techniques. These adaptive techniques provide a method for the fuzzy modeling procedure to extract proper knowledge about a data set, in order to compute the membership function parameters that best allow the associated fuzzy inference system to track the given input/output data. However, in this paper we will not consider these techniques. The third, advantage of Sugeno type inference is that it is well-suited to mathematical analysis.

## 3. The Proposed Model

The block diagram of our inference system is presented in Figure 1.

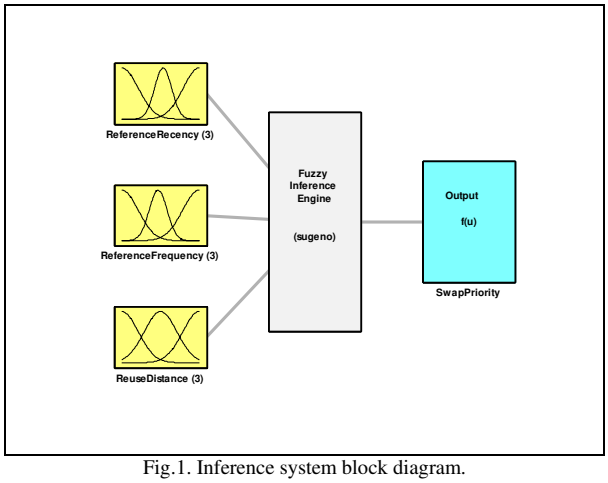


Fig.1. Inference system block diagram.

In the proposed model, the input stage consists of three linguistic variables. The first one is the recency of references which is going to present the spatial locality of references. The second input variable is the frequency of references. This parameter exploits the temporal locality of references. The last input variable is reuse distance which is the distance between two consecutive references to the page. By considering this parameter, we are able to eliminate negative effects caused by only considering recency in weak locality workloads such as sequential scans and loop like patterns [7].

Frequency seems to matter more for larger caches and recency has a more impact with smaller cache sizes. Distance reuse enriches the system to become suitable to weak locality workloads.

With the help of these three parameters this system is going to predicate about the soonness of reference to each page. The output if the system is swap priority that determine which page should be replaced. The page with higher swap priority has little chance to be referenced in near future.

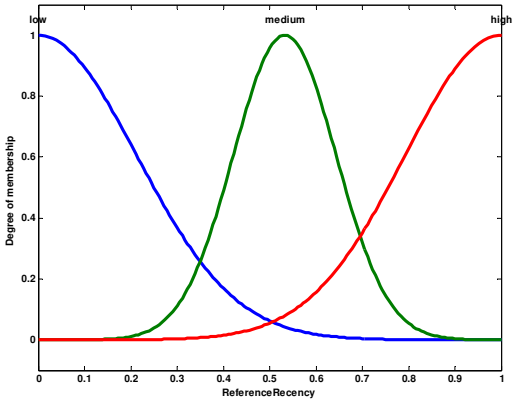


Fig.2. Fuzzy sets corresponding to reference recency

The input variables mapped into the fuzzy sets as illustrated in Figures 2, 3 and 4.

The shape of the membership function for each linguistic term is determined by the expert. It is very difficult for the expert to adjust these membership functions in an optimal way. However, there are some techniques for adjusting membership functions [10, 13]. In this paper, we will not consider these techniques. They can be further studied in a separate paper.

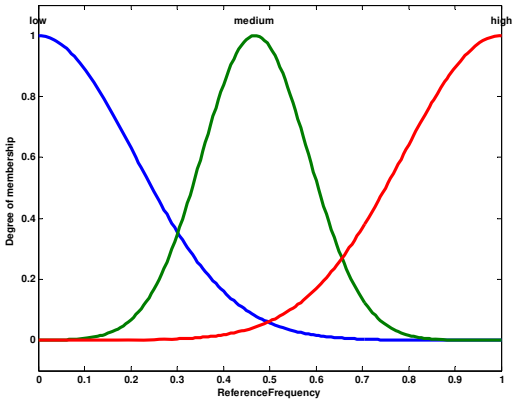


Fig.3. Fuzzy sets corresponding to frequency of references

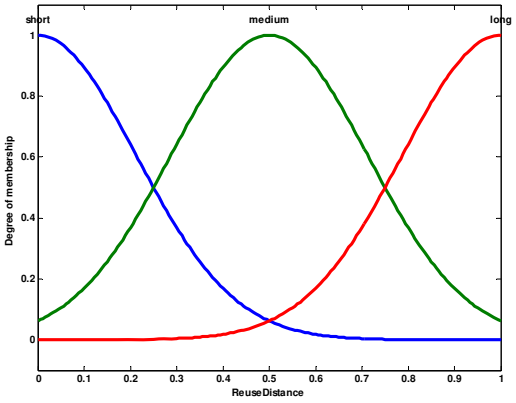


Fig.4. Fuzzy sets corresponding to reuse distance

Fuzzy rules try to combine these parameters as they are connected in real worlds. Some of these rules are mentioned here:

- If (ReferenceRecency is high) and (ReferenceFrequency is high) and (ReuseDistance is short) then (SwapPriority is very low)
- If (ReferenceRecency is low) and (ReferenceFrequency is low) and (ReuseDistance is long) then (SwapPriority is very high)
- If (ReferenceRecency is medium) and (ReferenceFrequency is medium) and (ReuseDistance is medium) then (SwapPriority is normal)

In fuzzy inference systems, the number of rules has a direct effect on its time complexity. So, having fewer rules may result in a better system performance.

## 2.1 The Proposed Algorithm

The FPR algorithm is as follows:

Algorithm FPR

1. For each used page P in cache, feed its recency of reference, frequency of reference and reuse distance into the inference engine. Consider the output of the inference module as swap priority of page P.
2. Swap out the page with the highest swap priority.

## 4. Performance Evaluation

To validate our algorithm and to demonstrate its strength, we use trace-driven simulations with various types of workloads. Results presented in this paper are given for a set of nine workload traces which was used in [4]. These traces were used in previous literature aiming at addressing other algorithms performance.

These traces can be classified to five groups according to their cache access pattern.

- Spatial clustered references: pages are accessed according to the spatial locality of references. It means pages are near to one another in the address space.
- Temporal clustered references: pages are accessed at nearly the same time during the execution.
- Looping references: all pages are accessed repeatedly with a regular interval (period);
- Probabilistic references: each page has a reference probability, and all pages are accessed independently with the associated probabilities.
- Mixed references: A mixture of the others.

A commonly used performance metric for evaluating a replacement policy is its *hit ratio* which is defined as the frequency with which it finds a page in the cache. All the traces are compared with each other according to this metric.

The traces are as follows:

- The first trace, which is called Cpp, is a GNU C compiler pre-processor trace. The total size of C source programs used as input is roughly 11.
- Cs: is an interactive C source program examination tool trace. The total size of the C programs used as input is roughly 9 MB.
- Mud: is a multi-user database application trace with a random access pattern.
- Glimpse: is a text information retrieval utility trace. The total size of text files used as input is roughly 50 MB.
- Sprite**:** is from the Sprite network file system, which contains requests to a file server from client workstations for a two-day period.
- Postgres**:** is a trace of join queries among four relations in a relational database system from the University of California at Berkeley.
- Mulit1: is obtained by executing two workloads, Cpp and Cs, together.
- Muti2: is obtained by executing three workloads Cs, Cpp and Postgres, together.
- Muti3: is obtained by executing four workloads, Cpp, Glimpse, Postgres, Cs, together.

Among these nine traces Cs, Postgres and Glimpse have looping reference pattern. As Figures 7, 8 and 12 state, for looping pattern, the FPR does approximately the same as the LFU but, much better than LRU. Mud has a probabilistic reference pattern. Figure 6 shows that both LFU and LRU are the same and FPR performs better for this pattern. Temporal clustered reference pattern exists just in Cpp trace and as Figure 5 illustrates all the algorithms are near optimal. The sprite reference trace has a sequential pattern. As it was expected and the Figure 13 confirms LRU has the best performance among the others. Figure 9, 10 and 11 show that, FPR can make better decisions in mixed reference patterns that is closer to the real world.

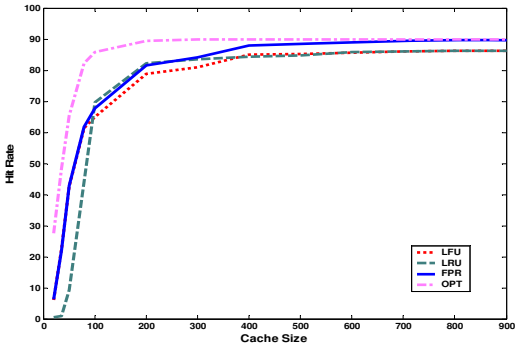


Fig.5. Cpp

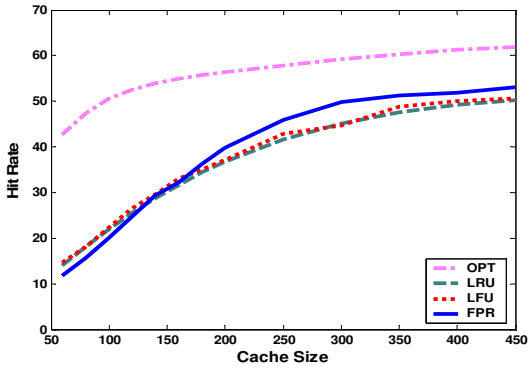

Fig.6. Mud

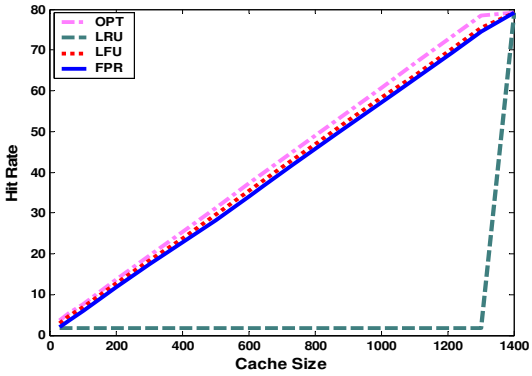

Fig.7. Cs

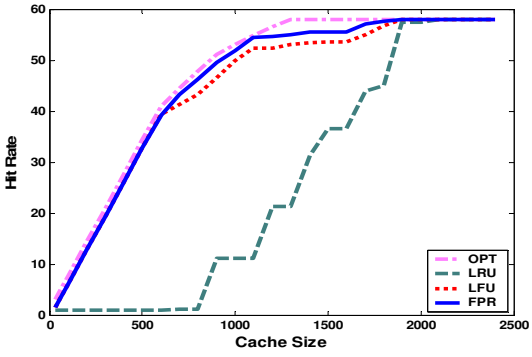

Fig.8. Glimpse

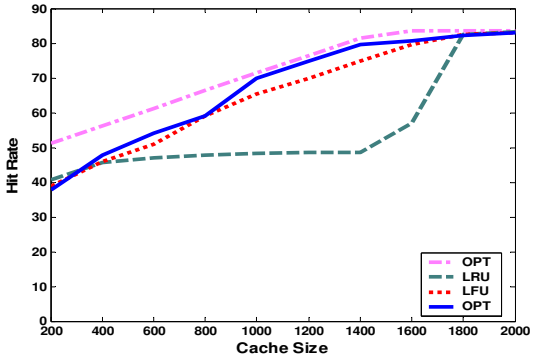

Fig.9. Multi1

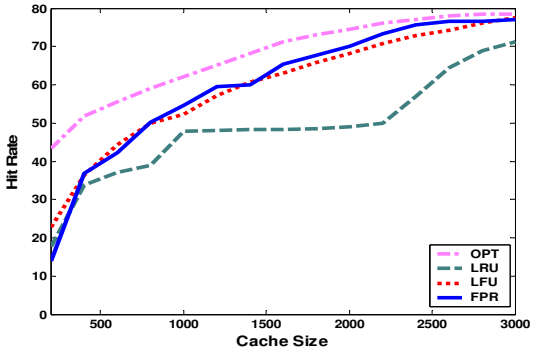

Fig.10. Multi2

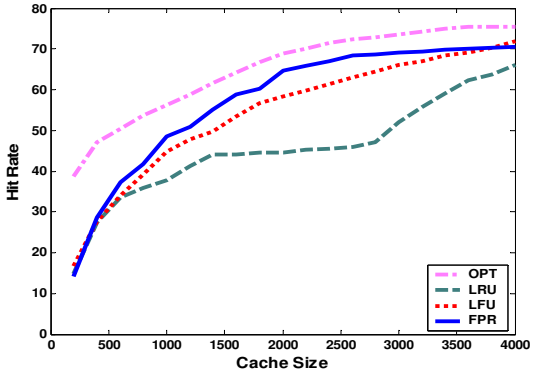

Fig.11. Multi3

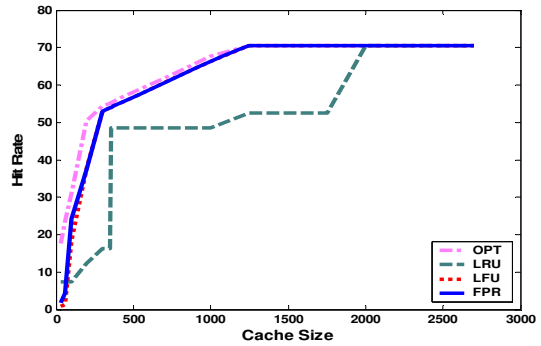


Fig.12. Postgres

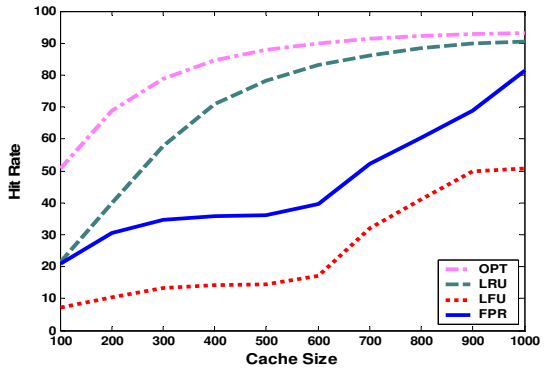


Fig.13. Sprite

## 5. Conclusion and Future Works

This paper has described the use of fuzzy logic to improve cache replacement decisions. Nine different workloads were examined and as it was shown the fuzzy approach has better performance over the LFU and LRU algorithms in eight of them. Results say that, the fuzzy approach is suitable for looping, probabilistic and temporal pattern of reference and it does better in mixed reference patterns.

Despite its success, the proposed fuzzy algorithm is a little bit time consuming and it is because the nature of fuzzy inference process.

In the future, the fuzzy approach will be applied to the distributed cache management. Also a non-uniform version of this algorithm is under publication.

## References

[1] Megiddo N., Modha DS., Outperforming LRU with an adaptive replacement cache algorithm, Computer, 2004.

[2] Bansal S., Modha DS, CAR: Clock with Adaptive Replacement, Proceedings of the USENIX Conference on File and Storage Technologies, 2004.

[3] Lee D., Choi J., Kim J.-H., Min S.L., Cho Y., Kim C.S., Noh S.H., On the Existence of a Spectrum of Policies That Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies, Proceedings ACM SIGMETRICS, 1999.

[4] Choi J., Noh S., Min S., Cho Y., Towards Application/File-Level Characterization of Block References: A Case for Fine-Grained Buffer Management, Proceedings of ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, June 2000.

[5] Lee D., Choi D., Kim J.-H., Noh S. H., Min S. L., Cho Y., Kim C. S., LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies, IEEE Trans. Computers, vol. 50, no. 12, 2001.

[6] Wang Lie-Xin, A course in fuzzy systems and control, Prentice Hall, Paperback, Published August 1996.

[7] Jiang S., Zhang X., LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache performance, SIGMETRICS, 2002.

[8] Mamdani E.H., Assilian S., An experiment in linguistic synthesis with a fuzzy logic controller, International Journal of Man-Machine Studies, Vol. 7, No. 1, pp. 1-13, 1975.

[9] Sugeno, M., Industrial applications of fuzzy control, Elsevier Science Inc., New York, NY, 1985.

[10] Jang, J.-S. R., ANFIS: Adaptive-Network-based Fuzzy Inference Systems, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 3, pp. 665-685, May 1993.

[11] Denning, P. J., The locality principle, Communication of the ACM 48, 7, 2005.

[12] Aho A.V., Denning P. J., and Ullman J. D., Principles of optimal page replacement, J. ACM, vol. 18, no. 1, 1971.

[13] Simon D, Training fuzzy systems with the extended Kalman filter, Fuzzy Sets and Systems, Volume 132, Number 2, 1, pp. 189-199, December 2002.