A Fuzzy Algorithm for Scheduling Periodic Tasks on Multiprocessor Soft Real-Time Systems

Mojtaba Sabeghi1,[†] *and Hossein Deldari2*[††]*,*

Ferdowsi University of Mashhad, Mashhad, Iran

Summary

In this paper we consider the use of fuzzy logic in the scheduling of periodic tasks in soft real-time multiprocessor systems. Most researches concerning real-time system scheduling assumes scheduling constraint to be precise. However, in many circumstances the values of these parameters are vague. The vagueness of parameters suggests that we make use of fuzzy logic to decide in what order the requests should be executed to better utilize the system and as a result reduce the chance of a request being missed. Our main contribution is proposing a fuzzy approach to multiprocessor real-time scheduling in which the scheduling parameters are treated as fuzzy variables. A simulation is also performed and the results are judged against each other. It is concluded that the proposed fuzzy approach is very promising and it has the potential to be considered for future research.

Key words:
Multiprocessor real-time scheduling, FGEDF, FGMLF, FPEDF, FPMLF.

Introduction

Real-time systems are vital to industrialized infrastructure such as command and control, process control, flight control, space shuttle avionics, air traffic control systems and also mission critical computations [1]. In all cases, time has an essential role and having the right answer too late is as bad as not having it at all.

In the literature, these systems have been defined as: “systems in which the correctness of the system depends not only on the logical results of computation, but also on the time at which the results are produced” [1]. Such a system must react to the requests within a fixed amount of time which is called deadline.

In general, real-time systems can be categorized into two important groups: hard real-time systems and soft real-time systems. In hard real-time systems, meeting all deadlines is obligatory, while in soft real-time systems missing some deadlines is tolerable.

In both cases, when a new task arrives, the scheduler is to schedule it in such a way that guaranties the deadline to be met. Scheduling involves allocation of resources and time to tasks in such a way that certain performance requirements are met.

These tasks can be classified as periodic or aperiodic. A periodic task is a kind of task that occurs at regular intervals, and aperiodic task occurs unpredictably. The length of the time interval between the arrivals of two consecutive requests in a periodic task is called period.

Another aspect of scheduling theory is to decide whether the currently executing task should be allowed to continue or it has had enough CPU time for the moment and should be suspended. A preemptive scheduler can suspend the execution of current executing request in favor of a higher priority request. However, a nonpreemptive scheduler executes the currently running task to completion before selecting another request to be executed. A major problem that arises in preemptive systems is the context-switching overhead. The higher number of preemptions a system has, the more context switching needed [2].

Schedulability of periodic, preemptive, soft real-time tasks on uniprocessor systems is well understood; simple and efficient algorithms are available and widely used [2, 3, 4, 5].

Nevertheless, for multiple processors these algorithms are not promising. Meeting the deadlines of real-time tasks in a multiprocessor system requires a scheduling algorithm that determines, for each task in the system, in which processor it must be executed, and in which order with respect to the other tasks, it must start its execution.

Multiprocessor scheduling techniques in real-time systems fall into to general categories: partitioning and global scheduling [11]. Under partitioning, each processor schedule tasks independently from a local ready queue. Each task is assigned to a particular processor and is only scheduled on that processor. In contrast, all ready tasks are stored in a single queue under global scheduling. A single system-wide priority space is assumed: the highest priority task is selected to execute whenever the scheduler is invoked. Partitioning is the favored approach because it has been proved to be efficient and reasonably effective when popular uniprocessor scheduling algorithms such as EDF and RM are used [7]. But finding an optimal assignment of tasks to processors is NP-hard.

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

In the global scheme, processors repeatedly execute the highest priority tasks available for execution. It has been shown that using this approach with common priority assignment schemes such as rate monotonic (RM) and earliest deadline first (EDF) can give rise to arbitrarily low processor utilization [7]. In this approach a task can migrate from one processor to another during execution.
In both cases researchers made some significant contributions by those results in better multiprocessor scheduling algorithms.
Although, these algorithms focus on timing constraints but there are other implicit constraints in the environment, such as uncertainty and lack of complete knowledge about the environment, dynamicity in the world, bounded validity time of information and other resource constraints. 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 satisfy objectives. The satisfaction degree can then be used as a parameter for making a decision. One especially straightforward method to achieve this is the modeling of these parameters through fuzzy logic. The same approach is also applied on uniprocessor real-time scheduling in [12, 13].
The scope of the paper is confined to scheduling of soft periodic tasks in multiprocessors real-time systems. 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 *laxity* is *critical* then *priority* is *very high*, which laxity and priority are linguistics variables and critical 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.

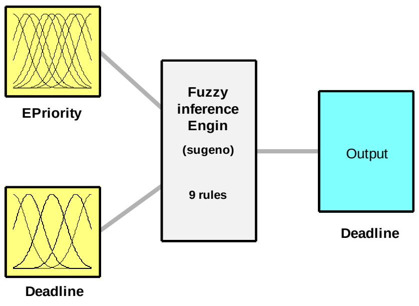

Fig.1. Inference system block diagram.

In the proposed model, the input stage consists of two linguistic variables. The first one is an external priority which is the priority assigned to the task from the outside world. This priority is static. One possible value can be the tasks interval, as rate monotonic algorithm does. For Figure 1, the other input variable is the Deadline. This input can easily be replaced by laxity, wait time, or so on, for other scheduling algorithms. Each parameter may cause the system to react in a different way. The only thing that should be considered is that by changing the input variables the corresponding membership functions may be changed accordingly.

For the simulation purposes, as it is discussed later, four situations are recognized: First, by using laxity as a secondary parameter and, second, by replacing the laxity parameter with deadline and both of them are considered in partitioned and also global scheme. In fact, four algorithms are suggested: FGEDF[1], FGMLF[2], FPEDF[3], and FPMLF[4].

The output if the system is priority that determines which is used as a parameter for making a decision.

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

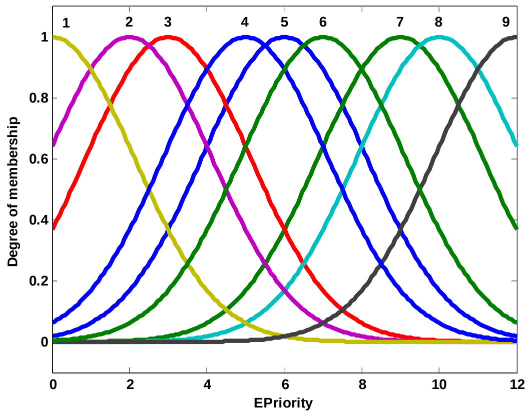

Fig.2. Fuzzy sets corresponding to reference recency

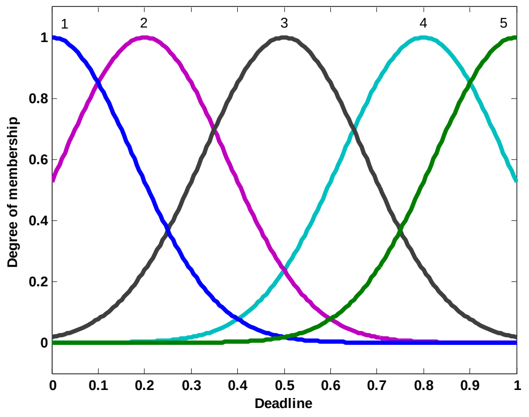

Fig.3. Fuzzy sets corresponding to deadline

[1] Fuzzy Global EDF
[2] Fuzzy Global MLF
[3] Fuzzy Partitioned EDF
[4] Fuzzy Partitioned MLF

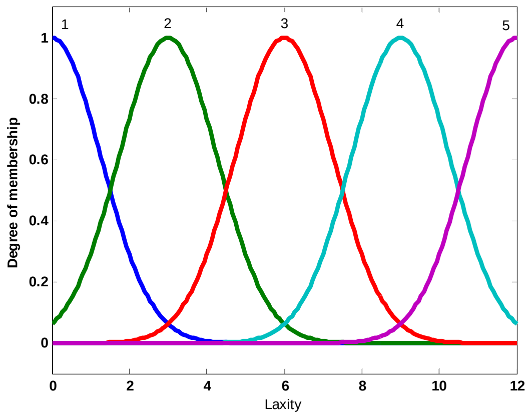

Fig.4. Fuzzy sets corresponding to laxity

The expert determines the shape of the membership function for each linguistic term. 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]. In this paper, we will not consider these techniques. They can be further studied in a separate paper.
Fuzzy rules try to combine these parameters as they are connected in real worlds. Some of these rules are mentioned here:

- If (EPriority is high) and (laxity is critical) then (Priority is very high)
- If (EPriority is normal) and (laxity is critical) then (Priority is high)
- If (EPriority is very low) and (laxity is critical) then (Priority is normal)
- If (EPriority is high) and (laxity is sufficient) then (Priority is normal)
- If (EPriority is very low) and (laxity is sufficient) then (Priority is very low)

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.

3.1 The Proposed Algorithm

The FGEDF algorithm is as follows:

```
Algorithm FGEDF

Loop            // System is running for ever
    For each CPU in the system do the followings:
            1. for each ready task T (a task which is not
            running), feed its external priority and
            deadline into the inference engine. Consider
            the output of inference module as priority of
            task T.
            2. Execute the task with highest priority until
            an scheduling event occurs (a running task
            finishes, a new task arrives)
            3. Update the system states (deadline, etc)
    End
End loop
```

FGMLF is much the same with FGEDF just by replacing the word deadline by laxity.
The FPEDF algorithm is as follows:

```
Algorithm FPEDF for each CPU

Loop
    1. For each ready task T (a task which have not
    been run on another CPU), feed its external priority
    and deadline into the inference engine. Consider the
    output of inference module as priority of task T.
    2. Execute the task with highest priority until an
    scheduling event occurs (a running task finishes, a
    new task arrives)
   3. Update the system states (deadline, etc)
End loop
```

FPMLF is much the same with FPEDF just by replacing the word deadline by laxity.

4. Performance Evaluation

To evaluate our algorithm and to demonstrate its strength, 1024 test cases with different load factors were generated. In each test case, the number of tasks and the corresponding execution time and request interval were randomly generated. Also, each task has been assigned a priority according to the rate monotonic principle (tasks with shorter request interval are given higher priorities) [3]. The behavior of all the four algorithms is compared with each other. Performance metrics, which are used to compare different algorithms, must be carefully chosen to reflect the real characteristics of a system. These metrics are as follows.
Average Response time, which is defined as the average amount of time a system takes to react to a given input, is

one of the most important factors in most scheduling algorithms.

Number of missed deadlines is an influential metric in scheduling algorithms for soft real-time systems.

When task preemption is allowed, another prominent metric comes into existence and that is the number of preemptions. Each of preemptions requires the system to perform a context switching which is a time consuming action.

Yet another metric, which is considered in our study, is the average CPU utilization. The main goal of a scheduling algorithm is to assign and manage system resources so that a good utilization is achieved.

We performed our simulation for systems with different number of CPUs and judge the algorithms against each other in these conditions.

Among the four algorithms FGEDF and FGMLF nearly achieve the same performance in all situations and all metrics. FPMLF performs poorly in number of misses and also average response time, but its performance on CPU utilization and also number of preemption is much better than the others especially when the number of CPUs increase.

Now, we will compare the algorithms in each metric. About number of misses, as Figures 5 to 10 shows, FPMLF has larger amount of misses and FPEDF seems to have fewest numbers of misses the other two algorithms carry out nearly the same. These figures show that deadline is a better parameter in this case.

Comparing number of preemptions, as Figures 11 to 16 demonstrate, FPMLF outperforms the others. FPEDF reaches to a higher number of preemptions as the load factor increases. Numbers of preemptions in FGMLF are a little more than FGEDF. In this case, deadline is the better parameter.

Judging against average response time, as stated in Figures 17 to 22, FPMLF presents extremely bad results. The other three algorithms have almost the same performance.

FPMLF which was the worse algorithm in the other three metrics, Has the best CPU utilization among the other algorithms. This algorithm utilizes almost 100 hundred percent of CPU. As Figures 23 to 28 states, partition approach achieves better CPU utilization.

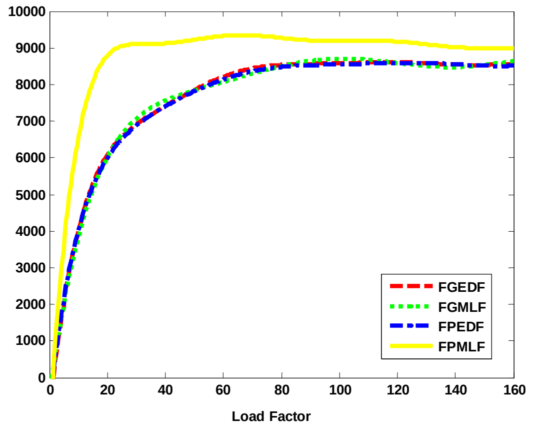

Fig.5. Number of misses for 4 CPUs

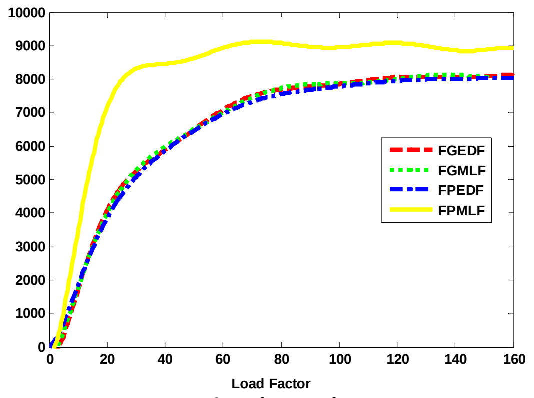

Fig.6. Number of misses for 8 CPUs

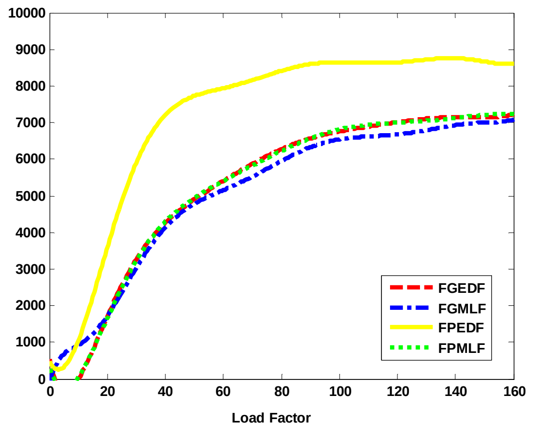

Fig.7. Number of misses for 16 CPUs

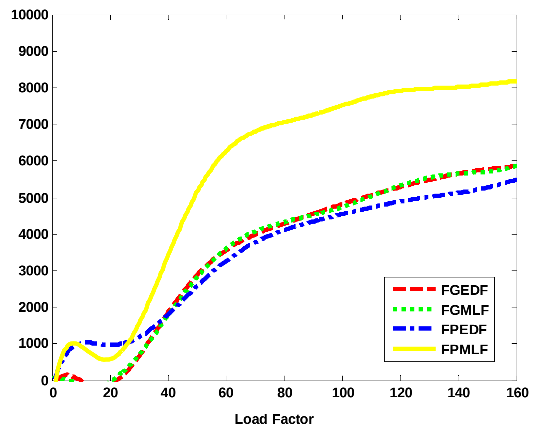

Fig.8. Number of misses for 32 CPUs

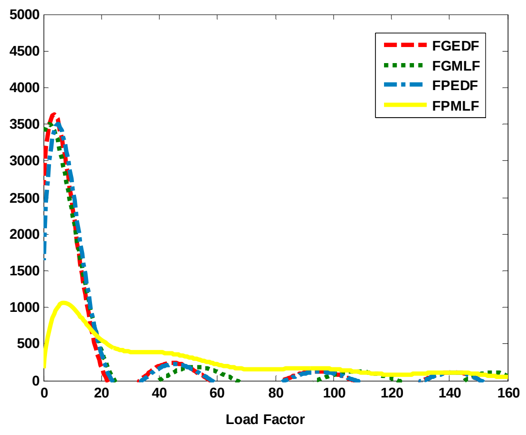

Fig.11. Number of preemptions for 4 CPUs

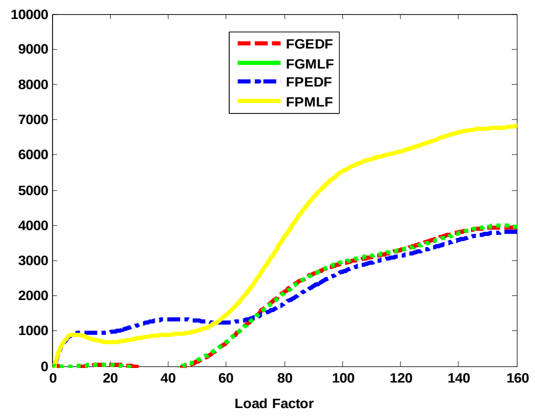

Fig.9. Number of misses for 64 CPUs

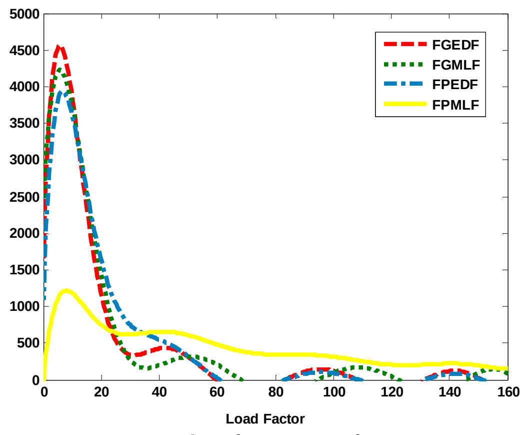

Fig.12. Number of preemptions for 8 CPUs

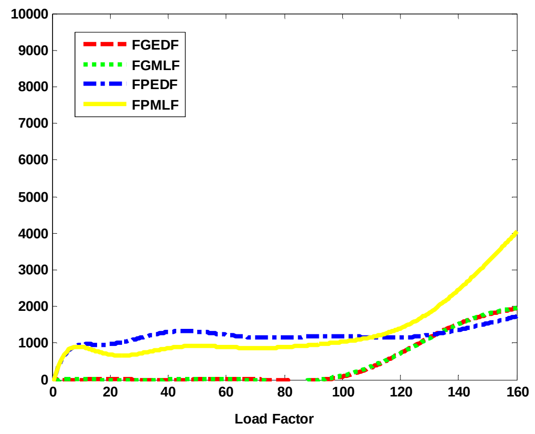

Fig.10. Number of misses for 128 CPUs

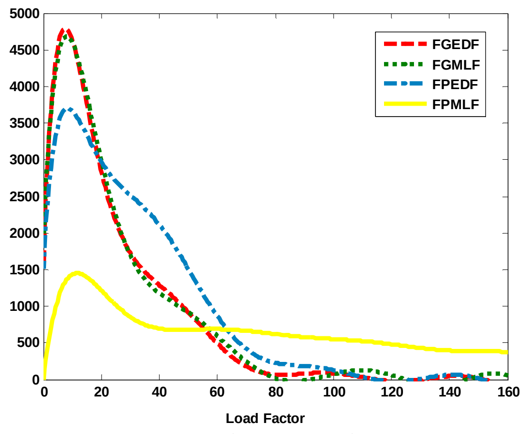

Fig.13. Number of preemptions for 16 CPUs

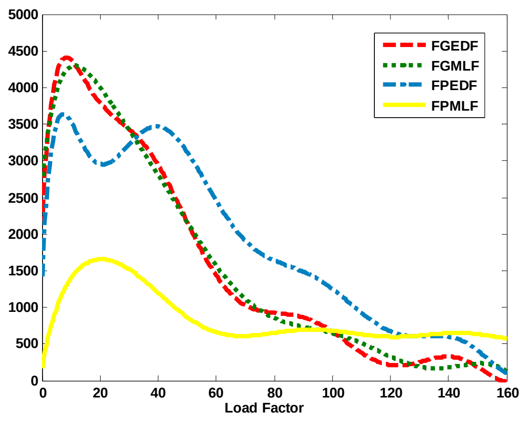

Fig.14. Number of preemptions for 32 CPUs

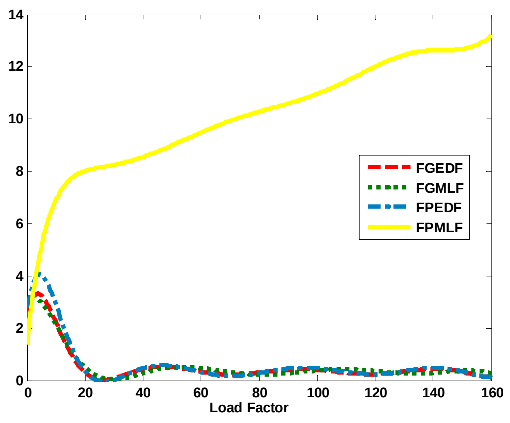

Fig.17. Average response time for 4 CPUs

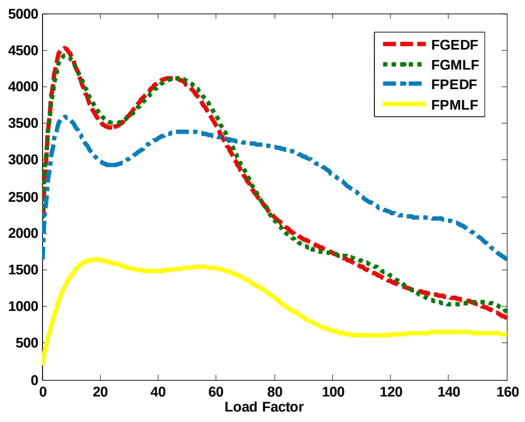

Fig.15. Number of preemptions for 64 CPUs

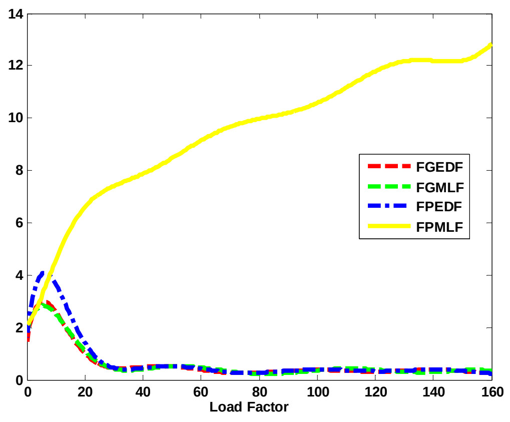

Fig.18. Average response time for 8 CPUs

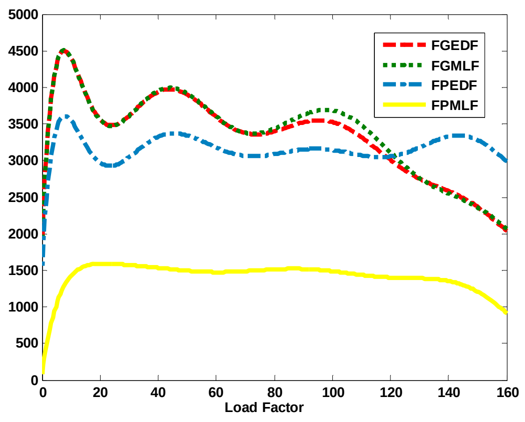

Fig.16. Number of preemptions for 128 CPUs

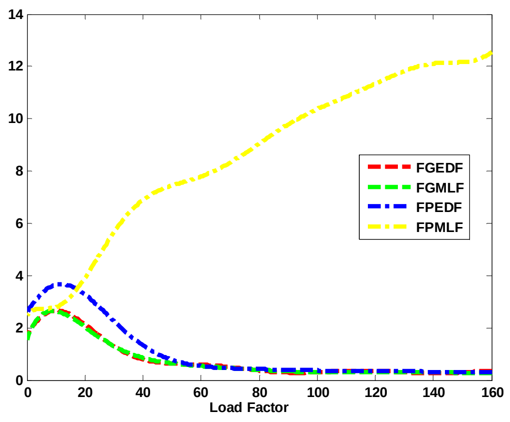

Fig.19. Average response time for 16 CPUs

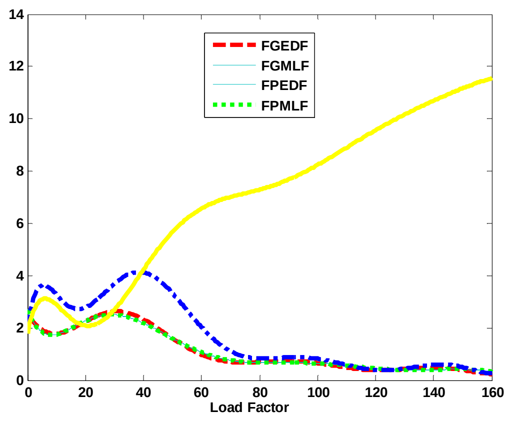

Fig.20. Average response time for 32 CPUs

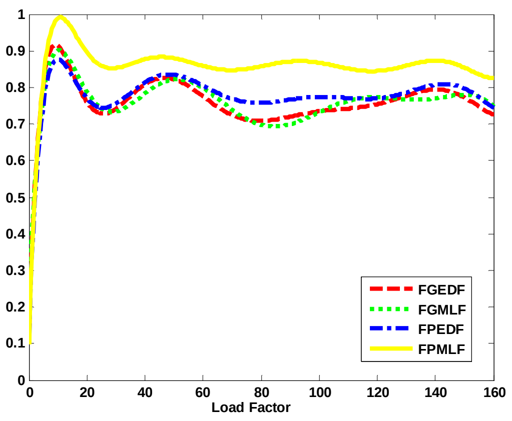

Fig.23. Average CPU utilization for 4 CPUs

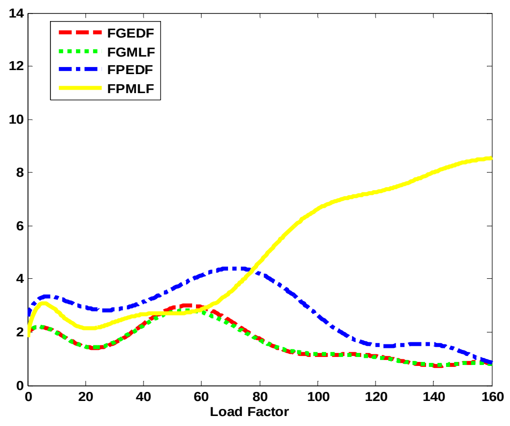

Fig.21. Average response time for 64 CPUs

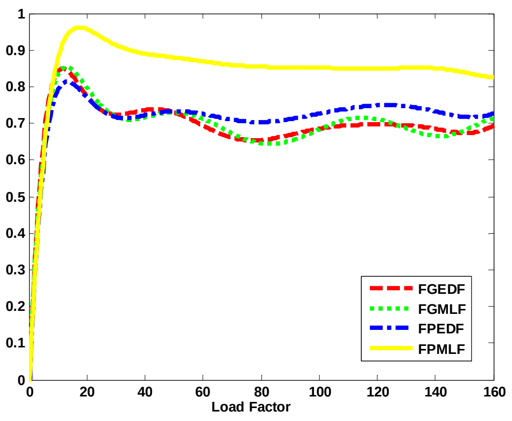

Fig.24. Average CPU utilization for 8 CPUs

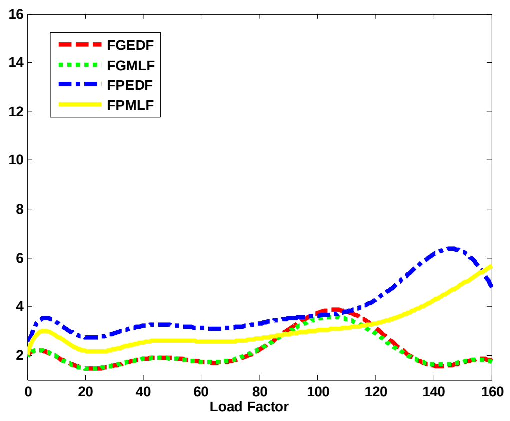

Fig.22. Average response time for 128 CPUs

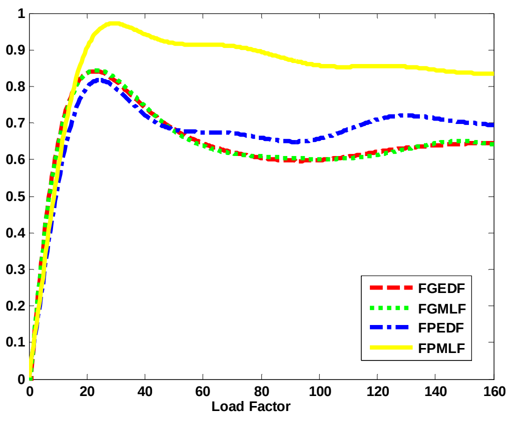

Fig.25. Average CPU utilization for 16 CPUs

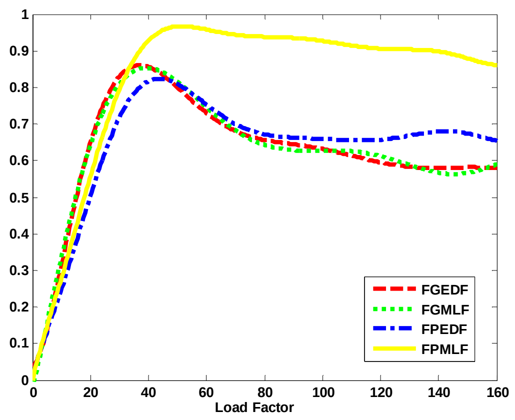

Fig.26. Average CPU utilization for 32 CPUs

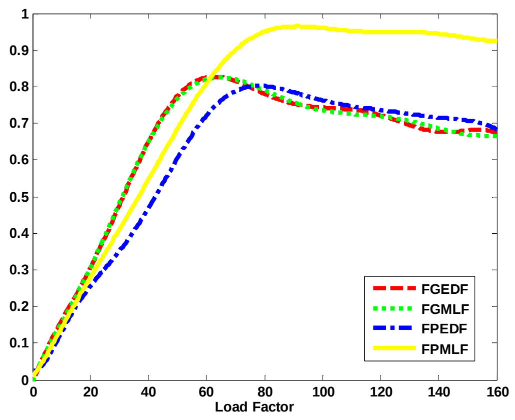

Fig.27. Average CPU utilization for 64 CPUs

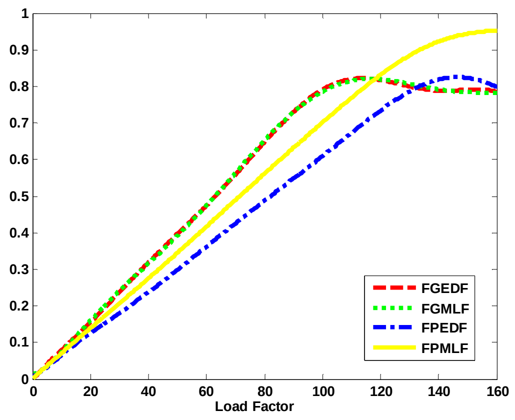

Fig.28. Average CPU utilization for 128 CPUs

4. Conclusion and Future Works

This paper has described the use of fuzzy logic to multiprocessor real-time scheduling. As it was shown, using deadline as a fuzzy parameter in multiprocessor real-time scheduling is more promising than laxity. Also, it seems that partitioning approach almost outperforms global approach in case fuzzy real-time scheduling.
I the future, we will conduct a deeper simulation and compare the results of fuzzy approach with the other algorithms.

References

[1] Ramamritham K., Stankovic J. A., Scheduling algorithms and operating systems support for real-time systems, Proceedings of the IEEE, Vol. 82(1), pp55-67, January 1994.
[2] Goossens J., Richard P., Overview of real-time scheduling problems, Euro Workshop on Project Management and Scheduling, 2004
[3] Liu C. L., Layland J. W., Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM, 20(1):46-61, 1973.
[4] Hong J., Tan X., Towsley D., A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System, *IEEE Trans. on Comp.*, Vol. 38, No. 12**,** Dec. 1989
[5] Sha, L. and Goodenough, J. B., Real-Time Scheduling Theory and Ada, IEEE Computer, Vol. 23, No. 4, pp. 53-62 (April 1990).
[6] Wang Lie-Xin, A course in fuzzy systems and control, Prentice Hall, Paperback, Published August 1996.
[7] Andersson B., Jonsson J. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition, Seventh International Conference on Real-Time Computing Systems and Applications (RTCSA'00), 2000
[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] Lauzac S., Melhem R., Mosse D. Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor, Euromicro Workshopon RealTime Systems, 1998.

[12] Sabeghi M., Naghibzadeh M., Taghavi T., A Fuzzy Algorithm for Scheduling Soft Periodic Tasks in Preemptive Real-Time Systems, International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE), 2005
[13] Sabeghi M., Naghibzadeh M., A Fuzzy Algorithm for Real-Time Scheduling of Soft Periodic Tasks, IJCNS International Journal of Computer Science and Network Security, Vol. 6 No.2 February 2006