A Fuzzy Algorithm for Real-Time Scheduling of Soft Periodic Tasks

Mojtaba Sabeghi,[†] *and Mahmoud Naghibzadeh*[††]*,*

Ferdowsi University of Mashhad, Mashhad, Iran

Summary

Many scheduling algorithms have been studied to guarantee the time constraints of real-time processes. Scheduling decision of these algorithms is usually based on parameters which are assumed to be crisp. 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. We have proposed a new fuzzy algorithm called highest fuzzy priority first. The performance of this algorithm is compared with the well-known earliest deadline first algorithm as well as least laxity first algorithm through simulation. The simulations were divided into two parts. Firs, the tasks were considered to be preemptable, and second tasks were assumed to be non-preemptable. Simulation results show that this fuzzy approach outperforms the EDF and LLF. 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 real time scheduling, EDF, LLF, MFDF, MFLF.

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, 3]. 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. As stated in [1] 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 [5].

There are a plenty of real-time scheduling algorithms that are proposed in the literature. Each of these algorithms bases its decision on certain parameter while attempting to schedule tasks to satisfy their time requirements. Some algorithms use parameters that are determined statically such as the Rate Monotonic algorithm that uses the request interval of each task as its priority [7, 15]. Others use parameters that are calculated at run time. Laxity and deadline are among those parameters that are the most considered. Laxity says the task execution must begin within a certain amount of time while deadline implies the time instant at which its execution must be completed [2].

In the following, there are descriptions of two famous algorithms which are commonly used in real-time systems and are proved to be optimal for uniprocessor systems when the system load factor is less than one. System load factor is defined as follow:

$$L=\sum_{i=1}^{n} e_i / r_i$$

Earliest Deadline First (EDF) is a dynamic algorithm that does not require processes to be periodic. Whenever a process needs the CPU time, it announces its presence and its deadline. This algorithm keeps a list of running processes that is sorted on deadlines. It always runs the first process on the list that is, the one with the closest deadline. When a new process becomes ready, the algorithm first checks its deadline. If this deadline occurs before the currently running process, then the algorithm preempts the current one and starts the new process.

The Least-Laxity-First (LLF) scheduling algorithm assigns higher priority to a task with the least laxity. The algorithm, however, is impractical to implement because laxity tie results in the frequent context switches among the tasks [4].

Static scheduling works perfect when there is enough information in advance about what has to be done, but dynamic scheduling does not have this restriction. Although, the dynamic 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 objectives. 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 constraints through fuzzy constraints. The same approach has been applied in [12, 14].

The scope of the paper is confined to scheduling of periodic tasks in soft real-time systems with fuzzy constraints. The rest of the paper is organized as follow. In section II the fuzzy inference system is discussed. Section III covers the proposed model and section IV contains the experimental results. Conclusion and future works are debated in Sections V.

2. Fuzzy Inference Systems

A fuzzy inference system (FIS) tries to derive answers from a knowledgebase by using a fuzzy inference engine. The inference engine which is considered to be the brain of the expert systems provides the methodologies for reasoning around the information in the knowledgebase and formulating the results.

Fuzzy logic is an extension of Boolean logic dealing with the concept of partial truth that 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 the 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.

The membership function of a fuzzy set corresponds to the indicator function of the classical sets. It can be expressed in the form of 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].

Fuzzy Inference Systems are conceptually very simple. An FIS consists of an input stage, a processing stage, and an output stage. The input stage maps the inputs, such as deadline, execution time, and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a result for each. It then combines the results of the rules. Finally, the output stage converts the combined result back into a specific output value [6].

As discussed earlier, the processing stage, which is called the 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 subsystems have dozens of rules. These rules are stored in a knowledgebase. An example of fuzzy IF-THEN rules is: IF *deadline* is *critical* then *priority* is *very high*, in which *deadline* and *priority* are linguistics variables and *critical* and *very high* are linguistics terms. The five steps toward a fuzzy inference are as follows:

- fuzzifying inputs
- applying fuzzy operators
- applying implication methods
- aggregating outputs
- defuzzifying results

Bellow is a quick review of these steps. However, 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 the Fuzzy Inference Subsystem (FIS), the results from each rule must be combined in order to make the final decision. Aggregation is the process by which the fuzzy sets that represent the outputs of each rule are processes into a single fuzzy set. The input for the defuzzification process is the aggregated output fuzzy set and the output is then a single crisp value [6]. 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 crisp valued output.

There are two common inference methods [6]. The first one is called Mamdani's fuzzy inference method proposed in 1975 by Ebrahim Mamdani [8] and the second one 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's 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. Firstly, it is computationally efficient, which is an essential benefit to real-time systems. Secondly, 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. The third, advantage of Sugeno type inference is that it is well-suited to mathematical analysis. However, in this paper we will not consider these techniques as we will propose an alternative method in the following.

3. The Proposed Model

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

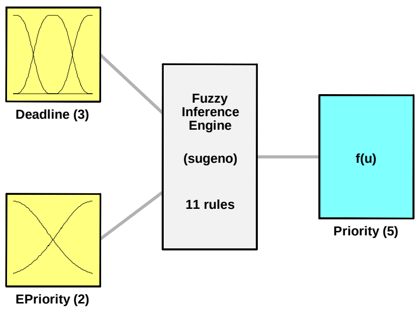

Fig.1. Inference system block diagram.

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 request interval, as rate monotonic algorithm does [3]. 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, two situations are recognized: First, by using laxity as a secondary parameter and, second, by replacing the laxity parameter with deadline. In fact, two algorithms are suggested: one with laxity as the second parameter. This algorithm is called MFLF[1]. The other algorithm is with deadline as the second parameter. This one is called MFDF[2].

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

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.

[1] Minimum fuzzy laxity first
[2] Minimum fuzzy deadline first

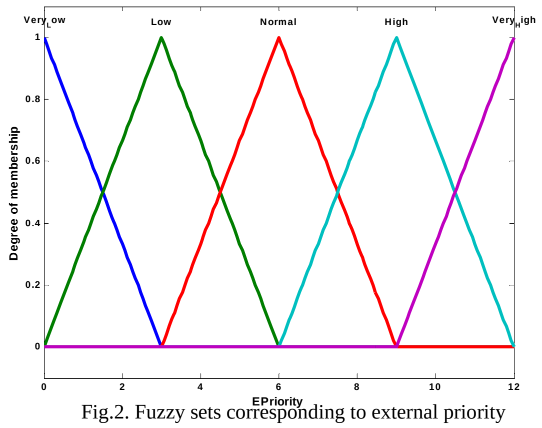

Fig.2. Fuzzy sets corresponding to external priority

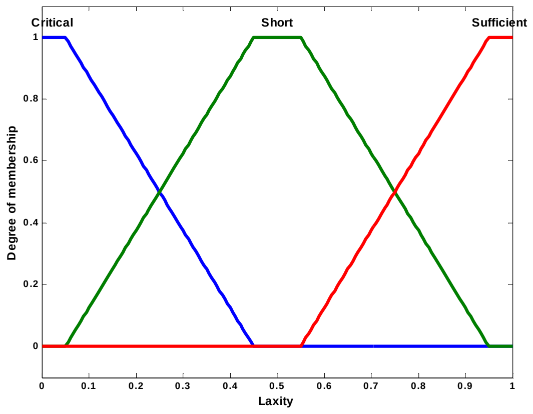

Fig.3. Fuzzy sets corresponding to laxity

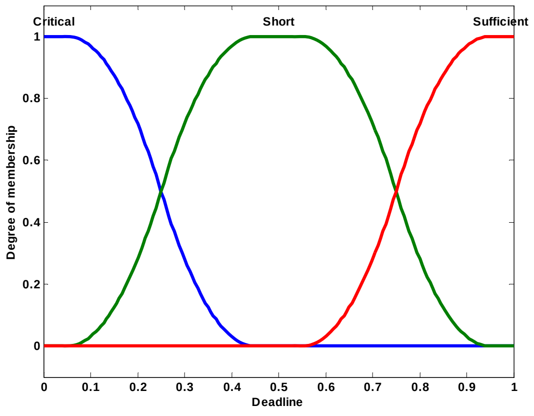

Fig.4. Fuzzy sets corresponding to deadline

We have produced 11 rules for our proposed system. 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 Algorithms

The MFLF algorithm is as follows:

```
Algorithm MFLF

Loop
    1. For each task T, feed its external priority
    and laxity 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 (laxity, deadline,
etc)
End loop
```

The algorithm for the MFDF is similar to the MFLF with laxity replaced by deadline.

4. Experimental Results

As we discussed earlier, in simulation two different situations have been recognized. First, evaluating the systems when the tasks are preemptable and when the task are non-preemptable.

One of the main objectives of scheduling algorithms is to decide whether the currently executing task should be allowed to continue its execution to completion or it is has had enough CPU time for the moment and should be suspended for the benefit another task. A preemptive scheduler can suspend the execution of currently executing request in favor of a more urgent request. However, a non-preemptive scheduler executes the currently running task to completion before selecting another request to be executed, no matter what the relative urgency of tasks is. Sometimes, the non-preemptive approach is favored because: (1) in many practical real-time systems, scheduling problems such as I/O scheduling and properties of device make preemption impossible or prohibitively expensive, (2) non-preemptive scheduling algorithms are easier to implement than preemptive algorithms, and can exhibit dramatically lower overhead at run-time, and (3) non-preemptive scheduling on a uniprocessor systems naturally guarantees exclusive access to shared resources and data, thus this causes eliminating both the need for synchronization and its associated overhead [17].

In this paper, first we will present our simulation results for preemptive scheduling and then the non-preemptive one will be discussed.

4.1 Preemptive Fuzzy Scheduling

The simulation consists of two parts. First, the system was examined for the case where the system load factor is less than one. Second, the system was observed in overloaded conditions. These divisions are suggested because, first, both EDF and LLF algorithms has been proved to be optimal in situations where the system load factor is less than one. The results of this phase shows whether or not the simulation is performed correctly. A correct simulation will reveal that there is no task misses for either of EDF and LLF algorithms. At the same time, it will show whether or not our algorithms perform as well as the EDF and LLF. Second, recall that soft real-time systems, as their definition implies, can tolerate some deadline misses. In real situations, there is no guarantee for soft real-time systems not to be overloaded. Evaluating systems in overloaded conditions is important in comparing the behavior of our scheduling algorithms with the existing EDF and LLF algorithms. As it was discussed earlier, LLF is impractical to implement so we decided to use a modified version of it that solves the problem of frequent context switches. This modified algorithm is fully discussed in reference [4] and is proved to be optimal.

To compare these algorithms, we need to automatically generate some sample systems. The system generation methods will be discussed later.

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.

Response time, which is defined as the 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.

CPU utilization is also an important metric because the main goal of a scheduling algorithm is to assign and manage system resources so that a good utilization is achieved.

Yet another metric, which is considered in our study, is the number of missed deadlines from the class of highest priority tasks. This corresponds to the external priority being *very high*.

4.1.1 Comparison in Non-overloaded Conditions

This comparison was mainly performed to show the correctness of the simulations. To do the evaluation, 2500 test cases with load factors less than one were generated. In each test case, the number of tasks and the corresponding execution time and request interval randomly generated.

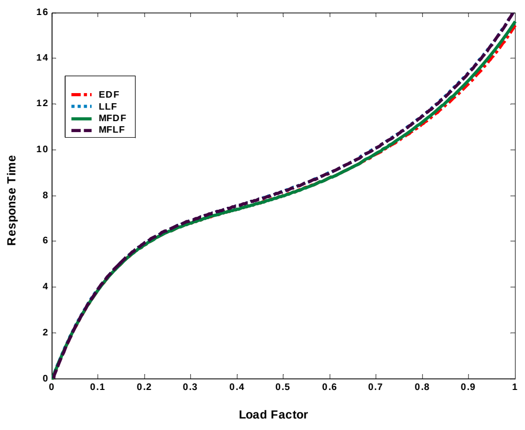

Fig.5. Response time in non-overloaded conditions

For this simulation phase, the goal is to compare average response time. As Figure 5 states all four algorithms show approximately the same performance with respect to the response time. The results are exactly what we have expected.

4.1.1 Comparison in Overloaded Conditions

Comparison parameters which are used here are average response time, number of tasks missing their deadlines, number of preemptions, and CPU utilization.

The simulation was done on 2500 test cases. These test cases were randomly generated. In each test case, the number of tasks and the corresponding execution time and request interval 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) [7].

As Figure 6 states, when the load factor is less than one, all the algorithms have the similar performance. However, when the system becomes overloaded, the response time of both EDF and LLF is much tardier than MFLF and MFDF.

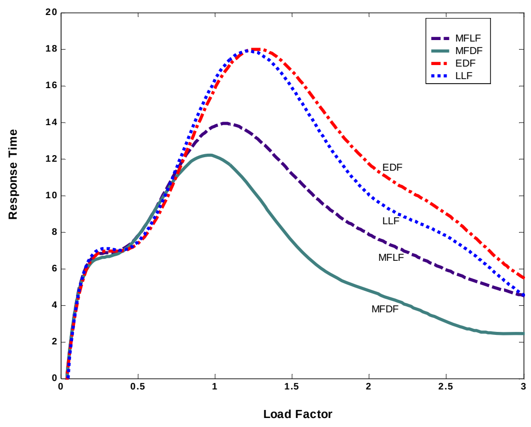

Fig.6. Response time in overloaded conditions

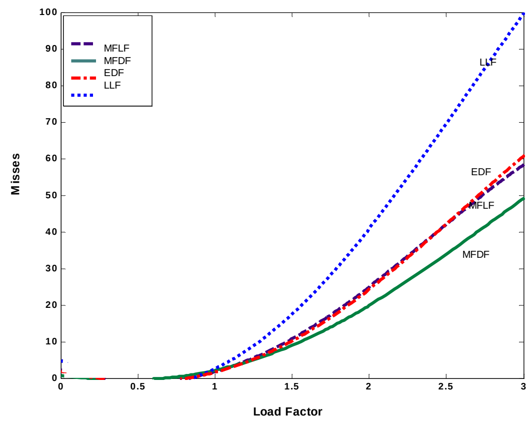

Fig.7. Number of Misses

Figure 7 states that for load factors less than one the number of misses is zero. This is because it has already been proved that any system with a load factor less than or equal to one runs safe under either of EDF and LLF.

Fortunately, MFLF and MFDF perform as well as either of EDF and LLF. In this case, the number of misses is exactly zero for all four algorithms. Because in drawing diagrams some curve fitting techniques is used, it seems that number of misses for algorithms when the load factor is a little bit less than one is a positive number. However, we have examined the numerical results and confirm that the number of misses is exactly zero.

When the load factor is more than one the MFDF has the best performance and MFLF has a performance similar to EDF. The LLF has the worse performance among all four algorithms.

As the Figure 8 shows, there is an opposite relation between the numbers of preemptions on the one hand and response time on the other hand. As the response time gets better number of preemptions comes to worse value.

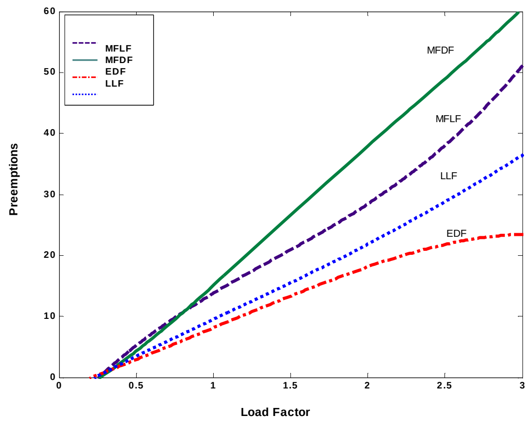

Fig.8. Number of Preemptions

MFDF that has the best performance with respect to response time has a larger number of preemptions. But there is something good about it, and that is, its behavior is predictable as it acts in a linear way. Having higher number of preemptions is reasonable because it eventually leads to having better response time and also better CPU utilization. There should be a balance between the number of preemptions and other factors. Reference [11] argues why such a balance is needed.

Figure 9 demonstrates that with the fuzzy methods CPU utilization is much higher than non-fuzzy methods. When the load factor is about 3, the MFDF and MFLF use about 80 percent of CPU time while EDF uses 60 percent of CPU time and the LLF just uses about 20 percent of CPU time.

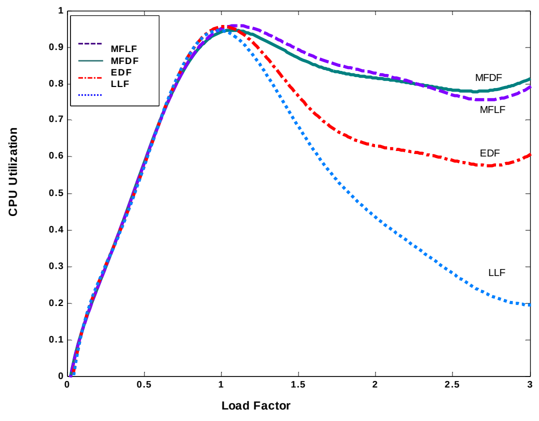

Fig.9. CPU Utilization

Considering the number of missed deadlines from the class of highest priority tasks, Figure 10 shows that both MFDF and MFLF perform much better than EDF and LLF. Comparing Figure 10 with Figure 7 shows that in load factor 3 about 80 percent of missed deadlines in both EDF and LLF are from the class of highest priority tasks while in MFDF and MFLF just about 30 percent of misses are among highest priority tasks. This is because external priority is considered as a decision parameter in the latter two algorithms. It should be mentioned that highest priority tasks in this simulation as discussed earlier, are those with shorter request intervals. These kinds of tasks since their deadline is too short may miss their deadline easier than the others. This is why in EDF and LLF about 80 percent of misses are among these tasks.

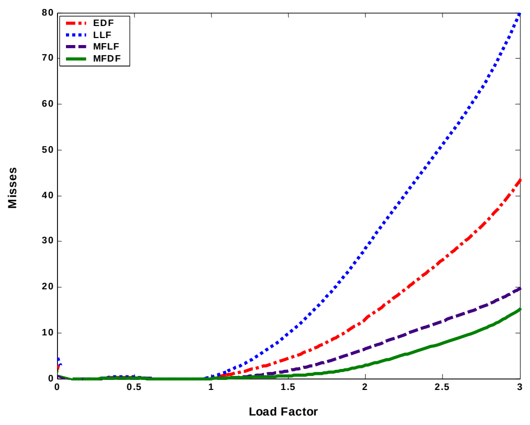

Fig.10. Number of missed deadlines from the class of highest priority tasks

4.1 Non-preemptive Fuzzy Scheduling

In this part of our experiments we only compared EDF with MFDF, because as it was shown in [16] the laxity is not a promising factor in non-preemptive fuzzy scheduling.

In real situations there is no guarantee for soft real-time systems not to be overloaded. Therefore, a soft real-time system, as its definition implies, can tolerate some deadline misses. Evaluating systems in overloaded situations is especially important in comparing the behavior of our scheduling algorithms with the existing well-known EDF algorithm. Overloaded condition is when the system load factor is higher than one.

To compare these algorithms, we need some performance metrics. Performance metrics must be carefully chosen to reflect the real characteristics of a system. The performance metrics in this part of the simulation is like the one for preemptive simulation.

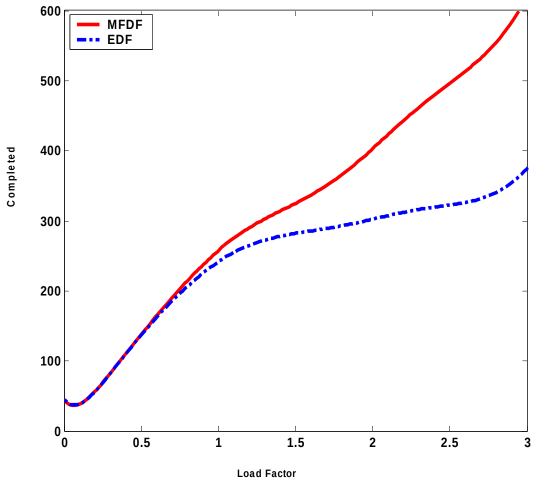

Fig.11. Number of successfully met deadlines

To do the evaluation, 2500 test cases with load factors less than three were generated. In each test case, the number of tasks and the corresponding execution time and request intervals were randomly generated. The external priority of a task was assigned according to the rate monotonic principle (i.e., tasks with shorter request interval are given higher priorities) [7].

Figures 11 and 12 show the number of successfully met deadlines and missed deadlines, respectively. As it is expected, an algorithm with less number of misses must have more competed tasks. These two diagrams reveal that MFDF has a better performance over EDF, especially when the load factor is more that one.

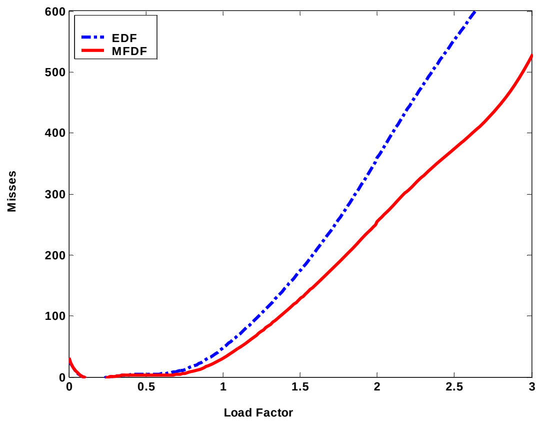

Fig.12. Number of missed deadlines

As Figure 13 shows, CPU utilization of both algorithms is approximately the same. For load factors higher than one, both algorithms use almost 100 percent of the CPU time.

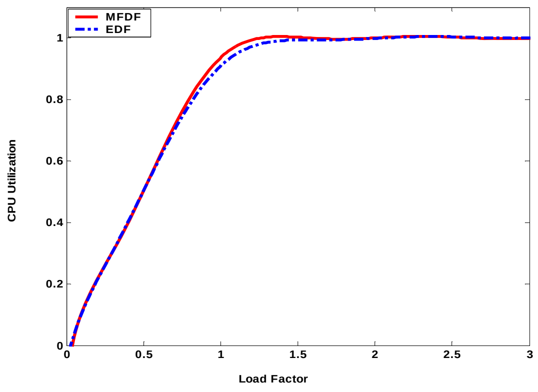

Fig.13. CPU Utilization

Figure 14 demonstrate that response time of MFDF is better when load factor becomes greater than 1.7. However, for load factors between 0.6 and 1.7, EDF shows better response time.

Considering the number of missed deadlines from the class of highest priority tasks, Figure 15 shows that HFPF algorithm performs much better than EDF. Comparing Figure 15 with Figure 12 reviles that for load factor 3 about 56 percent of missed deadlines in EDF algorithm are from the class of highest priority tasks while in HFPF just about 40 percent of misses are among the highest priority tasks. This is because external priority is considered as a decision parameter in the latter algorithm. It should be mentioned that highest priority tasks in this simulation as discussed earlier, are those with shorter request intervals. These kinds of tasks pretend to miss their deadline easier than others.

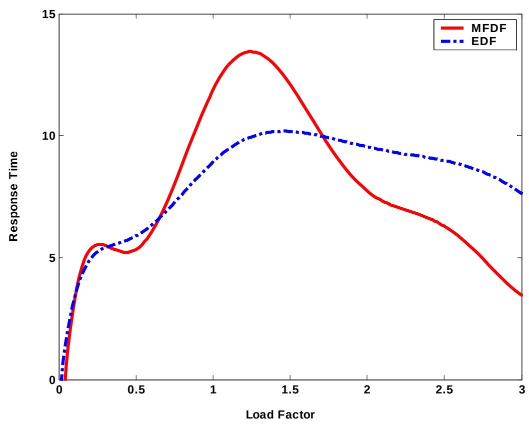

Fig.14. CPU Utilization

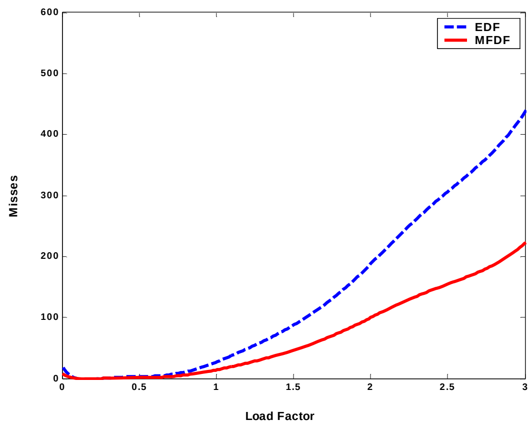

Fig.15. Number of missed deadlines from the class of highest priority tasks

5. Conclusion

Using the fuzzy concept in real-time scheduling, as it was shown, has the following advantages: (1) it better utilizes system resources such as CPU, (2) it decreases the number of missing deadlines, (3) it improves the system response time, and (4) it serves more important tasks better.

In the future, for improving the time complexity of the system, rule reduction techniques are going to be applied to the system. Also, to improve performance, adjusting membership functions with adaptive methods of inference is required [10, 13].

References

[1] Ramamritham K., Stankovic J. A., Scheduling algorithms and operating systems support for real-time systems, Proceedings of the IEEE, Vol. 82, No. 1, pp. 55--67, January 1994.

[2] 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

[3] Sha, L. and Goodenough, J. B., Real-Time Scheduling Theory and Ada, IEEE Computer, Vol. 23, No. 4, pp. 53-62 (April 1990).

[4] Oh S.-H., Yang S.-M., A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks, *rtcsa*, p. 31, Fifth International Conference on Real-Time Computing Systems and Applications (RTCSA'98), 1998.

[5] Tanenbaum A. S., Modern Operating Systems, Second Edition, Prentice-Hall, 2001.

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

[7] 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.

[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] Hamidzadeh B., Shekhar S., Specification and Analysis of Real-time Problem Solvers, IEEE Transactions on Software Engineering, Vol. 19, pages 788-803, 1993

[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] Simon D, Training fuzzy systems with the extended Kalman filter, Fuzzy Sets and Systems, Vol. 132, No. 2, 1, pp. 189-199, December 2002.

[14] Sabeghi M., Naghibzadeh M., Taghavi T., A Fuzzy Algorithm for Real-Time Scheduling of Soft Periodic Tasks on Multiprocessor Systems, IADIS International Conference on Applied Computing, February 2006

[15] Naghibzadeh M, Kim K. H. , A Modified Version of Rate-Monotonic Scheduling Algorithm and its Efficiency Assessment, Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'02), 2002.

[16] Sabeghi M., Naghibzadeh M., Deadline vs. Laxity as a Decision Parameter in Fuzzy Real-Time Scheduling, 2nd IEEE International Conference on Information & Communication Technologies: From Theory to Applications, April 2006

[17] Jeffay K., Stanat D. F., Martel C.U., On non-preemptive scheduling of periodic and sporadic tasks, In Proceedings of the 12th IEEE Symposium on Real-Time Systems, pp129-139, 1991.