A Fuzzy Algorithm For Scheduling Real Time Jobs in Multiprocessor Systems

Sameena Naaz
Lecturer Department of Computer Science
Jamia Hamdard, Hamdard University New Delhi
snaaz@jamiahamdard.ac.in

Parul Agarwal
parul_ag1@yahoo.com

ABSTRACT
In this paper we consider the use of fuzzy logic in the scheduling of periodic tasks in soft real-time multiprocessor systems. In real-time systems, tasks have to be performed correctly and timely. Finding minimal schedule in multiprocessor systems with real time constraints is shown to be NP hard. Most researches concerning real-time system scheduling assume 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. The intrinsic uncertainty in dynamic real-time systems increases the difficulties of scheduling problem. Here a fuzzy approach to multiprocessor real-time scheduling is proposed in which the scheduling parameters are treated as fuzzy variables. A simulation is also performed and the results are judged against each other. Experimental results have shown that the proposed fuzzy scheduler creates feasible schedules for homogeneous and heterogeneous tasks. It also considers tasks priorities, which cause higher system utilization and lowers deadline miss time. According to the results, it performs very close to optimal schedule of uni-processor systems. It is concluded that the proposed fuzzy approach is very promising and it has the potential to be considered for future research.

KEYWORDS
Deadline, Priority, Fuzzy scheduling, FGEDF, FPEDF, FGMLF, FPMLF, Real time multiprocessor Systems.

INTRODUCTION
Many applications namely avionics, traffic control, automated factory, and military systems require real-time communication and computation. In real-time systems, all tasks have specific parameters such as deadline, priority, etc. 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. 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. Many real-time systems are hard and missing deadline is catastrophic [2-5], whereas in soft real-time system occasional violation of deadline constraints may not result in a useless execution of the application or calamitous consequences, but decreases utilization. A schedule, which is executing all real-time tasks within their deadlines and all the other constraints are met, is called a feasible schedule.

Real-time scheduling can be classified in two categories, static [6] and dynamic scheduling [7]. Static real-time scheduling algorithm such as Rate Monotonic schedules all real-time tasks off-line using static parameters and requires complete knowledge about tasks and system parameters, while dynamic task scheduler calculates the feasible schedule on-line and allows tasks to be invoked dynamically. These algorithms use dynamic parameters such as deadline and laxity [7,8,9,10]. 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. Scheduling in real-time system involves allocation of CPU and other resources to run corresponding tasks to meet certain timing constraints. Nonetheless, scheduling is more significant in real-time systems than non-real-time systems. In real-time systems, tasks have to be performed correctly and in a timely fashion as well. Tasks are classified as periodic and non-periodic [11,12]. The execution requests of a periodic task repeatedly occur at regular intervals. On the contrary, execution requests of a non-periodic task are unpredictable.

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 non preemptive 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 is needed. Multiprocessor scheduling techniques in real-time systems fall into to general categories: partitioning and global...
scheduling[13]. 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. But finding an optimal assignment of tasks to processors is NP-hard.

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. In this approach a task can migrate from one processor to another during execution.

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. Others use parameters that are calculated at run time. Laxity and deadline are among those parameters that are the most considered.

SCHEDULING ALGORITHMS AND TASK MODEL:
Task Model:
A task is a complete sequence of instructions. Task execution starts when a task is selected by task dispatcher and one of the system’s processors starts to run task’s instructions. Tasks are classified according to their deadline, priority, arrival characteristic, and computation cycles requests.

Scheduling Algorithms:
First-Come-First-Served (FCFS) [14] algorithm selects the task with the earliest arrival time. If system contains periodic tasks, their release time will be considered. This algorithm makes no effort to consider a task’s deadline. Earliest Deadline First (EDF) [15,14] algorithm always chooses the task with the earliest deadline. It has been proved that this algorithm is optimal in a uni-processor system. Since it cannot consider priority and therefore cannot analyze it, this algorithm fails under overloading conditions.

Least Laxity First (LLF) [16] algorithm selects the task that has the lowest laxity among all the ready ones whenever a processor becomes idle, and executes it to completion. This algorithm is non-preemptive and avoids the well-known problem of its preemptive counterpart that sometimes degenerates to a processor-sharing policy. Robust Earliest Deadline (RED)[9,17,15] algorithm calculates residual time and workload of tasks as their schedulability. It has some task rejection mechanism to handle system load when there is no feasible schedule.

FUZZY INFERENCE ENGINE

Fuzzy logic [18,19] is a superset of conventional Boolean logic and extends it to deal with new aspects such as partial truth and uncertainty. Fuzzy inference is the process of formulating the mapping from a given input set to an output using fuzzy logic. The basic elements of fuzzy logic are linguistic variables, fuzzy sets, and fuzzy rules [20]. The linguistic variables’ values are words, specifically adjectives like “small,” “little,” “medium,” “high,” and so on. A fuzzy set is a collection of couples of elements. It generalizes the concept of a classical set, allowing its elements to have a partial membership. The degree to which the generic element “x” belongs to the fuzzy set A (expressed by the linguistic statement x is A) is characterized by a membership function (MF), f_A(x). 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. This operation normalizes all inputs to the same range and has a direct effect on system performance and accuracy.

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. 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, in 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. There are two common inference processes. First is called mamdani’s [21] fuzzy inference method proposed in 1975 by Ebrahim Mamdani and the other is Takagi-Sugeno[22] or simply Sugeno, method of fuzzy inference introduced in 1985. 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.

THE PROPOSED MODEL:
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
For each ready task T (a task which have not been run on Loop Algorithm FPEDF for each CPU
The FPEDF algorithm is as follows:

1. 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 a 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.

The Proposed Algorithm
The FGEDF algorithm is as follows:

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 a 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 FGEDF just by replacing the word deadline by laxity.

The FGEDF algorithm is as follows:
Algorithm FGEDF for each CPU

Loop
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 a scheduling event occurs (a running task finishes, a new task arrives)
3. Update the system states (deadline, etc)

End loop

FGMLF is much the same with FGEDF just by replacing the word deadline by laxity.

The Proposed Algorithm

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 a scheduling event occurs (a running task finishes, a new task arrives)
3. Update the system states (deadline, etc)

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). 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. Now, we will compare the algorithms in each metric. About number of misses, FPMLF has larger amount of misses and FPEDF seems to have fewest numbers of misses the other two algorithms carry out nearly the same. Thus deadline is a better parameter in this case. Comparing number of preemptions, 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. Thus in this case also, deadline is the better parameter. Judging against average response time, FPMLF presents extremely bad results. The other three algorithms have almost the same performance. FPMLF, which was the worst algorithm in the other three metrics, has the best CPU utilization among the other algorithms. This algorithm utilizes almost 100 hundred percent of CPU. Thus it has been seen that partition approach achieves better CPU utilization.

Conclusion:

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 of fuzzy real-time scheduling.

**FUTURE SCOPE:**
In the future, a deeper simulation can be performed and the results of fuzzy approach with the other algorithms be compared.

**REFERENCES:**


