Designing a Fuzzy Scheduler for Hard Real-Time Systems

John Yen, Jonathan Lee, Nathan Pfluger, and Swami Natarajan
Department of Computer Science
Texas A&M University
College Station, TX 77843

Abstract

In hard real-time systems, tasks have to be performed not only correctly, but also in a timely fashion. If timing constraints are not met, there might be severe consequences. Task scheduling is the most important problem in designing a hard real-time system, because the scheduling algorithm ensures that tasks meet their deadlines. However, the inherit nature of uncertainty in dynamic hard real-time systems increases the problems inherent in scheduling. In an effort to alleviate these problems, we have developed a *fuzzy scheduler* to facilitate searching for a feasible schedule. A set of fuzzy rules are proposed to guide the search. The situation we are trying to address is the performance of the system when no feasible solution can be found and therefore certain tasks will not be executed. We wish to limit the number of important tasks that are not scheduled.

1 Introduction

Real-time scheduling is a problem which is the key part of designing the operating system for a hard real-time system, and is thus tightly dependent on the architecture of the target system.

Basically, there are two types of real-time systems[2], *soft* real-time systems, and *hard* real-time systems, see figure 1. In soft real-time systems, tasks are performed by the system as fast as possible, but they are not constrained to finish by specific times. The only constraint on the system is to minimize response time. On the other hand, in hard real-time systems, tasks must be performed before their deadlines or there might be severe consequences.

To further break down this taxonomy, hard real-time scheduling can be classified into two categories, static [3], and dynamic [4, 6, 5, 7, 8]. A static real-time scheduler computes schedules for tasks off-line and requires complete prior knowledge of a set of tasks' characteristics such as arrival time, computation time, deadline and so on. A dynamic approach, on the other hand, calculates the schedules on-line and allows tasks to be dynamically invoked. Although static approaches have low run-time cost, they are inflexible and cannot adapt to a changing environment or to an environment whose behavior is not completely predictable. When new tasks are added to a static system, the schedule for the entire system must be recalculated,

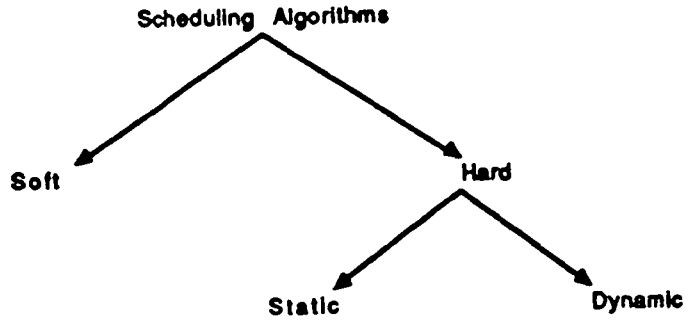

Figure 1: A Taxonomy of Real-Time Scheduling Algorithms

which is expensive in terms of the time and cost. In contrast, dynamic approaches involve higher run-time costs, but, because of the way they are designed, are flexible and can easily adapt to changes in the environment.

Our motivation to develop a fuzzy logic based approach to the dynamic scheduling problem are two fold. First, in a dynamic hard real-time system, not all the characteristics of tasks (e.g., precedence constraints, resource requirements, etc.) are known *a priori*. For example, the arrival time for the next task is unknown for aperiodic tasks. To be more precise, there is an inherit uncertainty in hard real-time environment which will worsen scheduling problems (e.g. arbitrary arrival time, and uncertain computation time). Characteristics of a task that may be uncertain include expected next arrival time, criticality, or importance of the task, system load and/or predicted load of individual processors, and run time, or more specifically average vs. worst-case run time.

Second, there is the possibility of system overload. In the case of overload we want to degrade gracefully by ensuring that the most important tasks are run first, thus allowing an amount of flexibility in the scheduler under adverse conditions to determine which tasks are run and which are not.

Therefore, our goal is to develop an approach to hard real-time scheduling that can be applied to a dynamic environment involving a certain degree of uncertainty and a possibility of overload situations. In this paper, we concentrate on a hard real-time system on a nonpreemptable uniprocessor system with a set of independent tasks. These tasks will have arbitrary arrival times and will be characterized by worst-case computation time and task criticality.

Therefore, we have developed a *fuzzy scheduler* that includes the following features. First, the scheduling process is treated as a search problem, as suggested by [7, 8], in which the search space consists of a tree where the root is an empty schedule, an intermediate node is a partial

schedule, and a leaf is a complete, though not necessarily feasible, schedule. Second, a set of fuzzy rules are used to guide the search of a feasible schedule. A feasible schedule is one that schedules all the tasks so that they may meet their deadlines.

In case no feasible schedule can be found, we then want the scheduler to ensure that its schedules the tasks according to some intelligent heuristic. Some possible heuristics include scheduling the most tasks, scheduling the most important tasks, etc. We also wish to include the possibility of more intelligent heuristics, such as schedule the most important tasks, but only if it allows most of the tasks to be executed.

A background about hard real-time schedule is introduced in the next section. An overview of our fuzzy scheduler and an example set of rules for one type of overload heuristic are presented in section 3. An outline of the benefits and applications for our scheduler is given in section 4. An example to demonstrate the feasibility of our approach is in section 5. Finally, we summarize the advantages and disadvantages of our approach.

2 Background on Hard Real Time Scheduling

The function of a scheduling algorithm is to determine, for a given set of tasks, whether a schedule for executing the tasks exists such that the timing, precedence, and resource constraints for the tasks are satisfied, and to calculate such a schedule if one exists. A schedule is said to be feasible if it contains all the tasks, and all tasks will meet their deadlines. A scheduling algorithm is said to be optimal if it finds a feasible schedule whenever one exists for a given set of tasks.

Most of the work in hard real-time scheduling in the early 70's is accredited to Liu and Layland[3]. In that paper, two algorithms were discussed, tested, and declared to be optimal. These algorithms are the RMS, rate monotonic scheduler, and EDF, earliest deadline first. The largest problem with these algorithms is the set of restrictions placed on the problem set that they solve. Later, another dynamic algorithm, least laxity, was also proposed and found to be optimal. For the case of least laxity and EDF, optimal is defined to be that if there is a feasible schedule for a set of tasks, then these algorithms will find one.

Stankovic and Ramamritham wanted to broaden the areas covered be real-time systems to include intelligent schedulers working on distributed systems [4, 6, 5, 7, 8]. Their method for designing a hard real-time system on a distributed system was to associate with each node in the distributed system a local scheduler. The function of the local scheduler was to receive tasks from the system and attempt to guarantee them to be run on this node. Those tasks that could not be guaranteed were then sent to another node. The method of sending tasks was through a bidding system, where each of the nodes bid for a task depending on its current state and predicated amount of free time. The basic rational behind their approach is the notion of a "guarantee algorithm". An algorithm is said to guarantee a newly arriving task if the algorithm can find a schedule for all the previously guaranteed tasks and the new task, such that each

task finishes by its deadline.

The main problem with this approach was determining a fast uniprocessor scheduling algorithm, that was both adaptive and intelligent. The method that they developed was to transform the scheduling problem into a tree search problem, where the root of the tree was an empty schedule, an intermediate vertex was a partial schedule, and the leaves were complete schedules. It can be proven that if a partial schedule is found to be infeasible, i.e., all tasks currently scheduled do not meet all the deadlines, then any complete schedule derived from this schedule will still be infeasible.

Although determining infeasibility of partial schedules cut down on the number of nodes to be searched, the problem was still NP-complete. The next step to making this method economical and intelligent was to derive a heuristic function which helped guide the search. The earliest heuristic functions simulated EDF and Minimum Processing Time first. Later more complex functions were tested and found to be better than simple heuristic functions.

Another related work regarding the integration of the importance (i.e., criticality) and the deadline of a task in hard real-time scheduling is addressed in [1]. In this paper, they adopt a similar approach to us by adding criticality as one of the major factors considered into their heuristic function to guide the search of feasible schedule.

Our work is an attempt to extend the notion of using a heuristic function for guiding the search in an intelligent manner. The goal of our work is to consider a complicated situation involving several major factors (e.g. deadline, criticality, and earliest starting time). Due to the use of fuzzy logic in representing our guidance mechanisms, they will be easy to express, comprehend, and modify.

3 A Fuzzy Scheduling Approach

3.1 A Scheduling Approach Based on Fuzzy Logic

In dynamic hard real-time scheduling, the nature of the task involves a certain degree of uncertainty, which increases the difficulty of developing a feasible and reliable scheduling algorithm. In order to alleviate the problems associated with hard real-time scheduling, we first treat the scheduling problem as a searching problem. We then developed a set of fuzzy rules to guide the search for a feasible schedule.

We have designed our system to handle a set of aperiodic tasks with arbitrary arrival times. In addition, any periodic task is considered to be a series of aperiodic tasks, each of which is an instance of that periodic task, and are denoted by T(x), where x = 0, 1, 2, ..., n. This method allows us to handle both periodic and aperiodic tasks equally, while still gaining benefits about knowing some of the tasks arrival times.

The major factors considered in our approach to determine the scheduling are task deadline,

criticality, and earliest start time. A deadline is a specific time by which a task must be finished. A task's criticality is the importance of the task. The earliest starting time for a task is the earliest time that a task can submit itself to the scheduler. The earliest start time for an aperiodic task is the current time, while the earliest start time for a periodic task will be a known future time computable from the task's characteristics.

The inputs of these parameters are fuzzified and represented as linguistic variables. The computation of these variables must be done every time the scheduler is executed, due to the dynamic nature of the system. Fuzzy rules are then applied to those linguistic variables to compute the *level value* for deciding which task will be selected to be scheduled next.

The linguistic variables for the three parameters chosen are:

- Task deadline: early, medium, late.
- Task criticality: important, average, unimportant.
- Task earliest starting time: early, medium, late.

To compute the results, we perform a reasoning process using fuzzy rules. The format of our fuzzy rules begins with IF as the left-hand side and ends with THEN as right-hand side. In the left-hand side, we use both basic and modified linguistic variables for the above-mentioned factors, while, in the right-hand side we assign a fuzzy number as a level value of that particular task. For example,

- IF the incoming task has an early deadline, an important criticality, and a medium earliest starting time, THEN assign level ~7.

As a result of the inference, several fuzzy rules will be initiated. We will then combine the fuzzy conclusions of all the rules that are initiated to produce a fuzzy variable which represents the level of the task. This variable will then be defuzzified to produce a crisp level to be compared to the other tasks for the purpose of choosing which task to schedule next.

For example, a scheduling snapshot starting at time 7 has three tasks T1, T2, T3 with the following characteristics:

Task	Earliest-Start-Time	Deadline	Computation-time	Criticality	Level-value
T1	10	15	3	5	~5
T2(1)	20	32	6	10	~3
T3	7	14	3	10	~10

Comparing the deadline of three tasks, we interpret T3 as early, T1 as medium, T2(1) as late. The criticality of T3 is important, T1 is medium and T2(1) is important. According to the example set of rules (in section 3.2), the level value of each task is: T3 (~10), T1 (~5), and T2(1) (~3). Therefore, the task T3 will be scheduled first, and then T1, and then finally T2(1).

3.2 Example Set of Rules

Fuzzy rules can be expressed as English sentences or as a set of if-then clauses using linguistic variables. One possible set of rules is:

- if deadline is early
  - if criticality is important ⇒ level ~10
  - criticality is average
    - * if earliest starting time is early ⇒ level ~9
    - * if earliest starting time is medium ⇒ level ~8
    - * if earliest starting time is late ⇒ level ~7
  - if criticality is unimportant
    - * if earliest starting time is early ⇒ level ~8
    - * if earliest starting time is medium ⇒ level ~7
    - * if earliest starting time is late ⇒ level ~6
- if deadline is medium
  - if criticality is important
    - * if earliest starting time is early ⇒ level ~6
    - * if earliest starting time is medium ⇒ level ~5
    - * if earliest starting time is late ⇒ level ~4
  - if criticality is average ⇒ level ~5
  - if criticality is unimportant ⇒ level ~4
- if deadline is late
  - if criticality is important ⇒ level ~3
  - if criticality is average ⇒ level ~2
  - if criticality is unimportant ⇒ level ~1

We chose to treat deadline as the most important principle behind choosing a task for scheduling because the major purpose of hard real-time scheduling is to meet the deadline. After this came criticality, and then earliest starting time. We felt that when a task must be scheduled, it is important to consider the tasks that must be done immediately. After that, more critical tasks are considered so that they are scheduled even when some of the tasks fail to be scheduled.

4 Benefits and Applications

There are several benefits of our approach: (1) robustness because of the boundary conditions of scheduling parameters are represented as a part of fuzzy subset; (2) flexibility due to easy integration of other requirements into the fuzzy rule set; (3) simplicity in term of understandability and using less rules; and (4) the intelligent scheduling system that may be derived using fuzzy logic.

By using fuzzy logic, the rules for determine task order are concise, intelligible, and easily modified. The rules that we have derived are based on basic rule of thumb theories about task scheduling and the properties we wish our system to show.

Recently, the development of fuzzy logic chip has a major progress. The Microelectronics Center of North Carolina successfully completes the fabrication of the world's fastest fuzzy logic chip. In the architecture of our fuzzy scheduler, a fuzzy logic chip can be used to implement part of fuzzy scheduler.

The use of a uniprocessor scheduling algorithm may be limited due to the distributed nature of today's systems. However, our approach, when used with a method of offloading unscheduled tasks to other nodes, may be useful for distributed systems. In future work, we hope to apply fuzzy logic to the problem of inter-processor communication and load balancing.

5 An Example

To demonstrate our system we have constructed an example scenario (Figure 2). The time frame for this example is limited to 35 for simplicity. The scheduling snapshots for different starting times are given to illustrate our approach:

In figure 3, tasks T1 and T2 are periodic tasks. T1 has four instances, T1(0), T1(1), T1(2) and T1(3). T2 has two instances, T2(0) and T2(1). T1(0) will be scheduled first, then T2(0), T1(1), T1(2), T2(1), and T1(3), in that order. In figure 4, task T2(0) continues to occupy the processor until it finishes at time 9 because we assume non-preemption for all tasks. T3 will take over and execute for 3 units of time. T1(1) and T1(2) will be scheduled before T2(1) because of the level values assigned. Finally, T1(3) will be executed. In figure 5, task T3 continues to execute until it finishes at time 12. T1(1) will be executed next, followed by T4 for 3 units of time. And then T2(1), T1(3) will take over.

In figure 6, task T1(1) continues to execute until 15. T4 will take over and finish at 18, which is followed by T1(2). T5 will be executed at time 21 and completed by 25. T2(1) and T1(3) are scheduled to be executed then. In figure 7, the last snapshot, T4 continues to execute up to time 18. T6 with the highest level value will be scheduled next and finished at time 20. Task T1(2) will be scheduled next. T5 should be scheduled next and finished up by 27. But by doing so, T5 will *miss its deadline*, therefore, T5 will have to be offloaded. Finally, T2(1) and T(3) will be scheduled.

Periodic Tasks

Task	Period	Deadline	Computation-time	Criticality
T1	10	5	3	5
T2	20	12	6	10

Aperiodic Tasks

Task	Arrival-time	Deadline	Computation-time	Criticality
T3	7	14	3	10
T4	12	18	3	1
T5	13	25	4	1
T6	16	21	2	20

Figure 2: An example Scenario

Task	Earliest-Start-Time	Deadline	Computation-time	Criticality	Level-value
T1(0)	0,10,20,30	5,15,25,35	3	5	~9
T2(0)	0,20	12,32	6	10	~3

Figure 3: Scheduling snapshot at time 0

Task	Earliest-Start-Time	Deadline	Computation-time	Criticality	Level-value
T1(1)	10,20,30	15,25,35	3	5	~5
T2(1)	20	32	6	10	~3
T3	7	14	3	10	~10

Figure 4: Scheduling snapshot at time 7

Task	Earliest-Start-Time	Deadline	Computation-time	Criticality	Level-value
T1(1)	12,20,30	15,25,35	3	5	~9
T2(1)	20	32	6	10	~3
T4	12	18	3	1	~4

Figure 5: Scheduling snapshot at time 12

Task	Earliest-Start-Time	Deadline	Computation-time	Criticality	Level-value
T1(2)	20,30	25,35	3	5	~5
T2(1)	20	32	6	10	~3
T4	13	18	3	1	~8
T5	13	25	4	1	~4

Figure 6: Scheduling snapshot at time 13

Task	Earliest-Start-Time	Deadline	Computation-time	Criticality	Level-value
T1(2)	20,30	25,35	3	5	~5
T2(1)	20	32	6	10	~2
T5	16	25	4	1	~4
T6	16	21	2	20	~10

Figure 7: Scheduling snapshot at time 16

Notice that although a feasible schedule was not found for time 16, there was an intelligent choice as to which task was not scheduled. Task T5 not only had a late deadline, but was also the least critical task present.

6 Conclusion

Due to the fact that there is an inherit amount of uncertainty in dynamic hard real-time systems which increases the problems inherent in scheduling, there is a need to develop a flexible scheduler. We have presented a fuzzy scheduler for hard real-time systems in which we treat the scheduling problem as a search problem, utilize a set of fuzzy rules to guide the search for a feasible schedule, and the scheduler is triggered by a newly arrival task.

The main advantage of our system is that an intelligent choice is made during overload situations to determine which task or tasks cannot be scheduled. This allows the system to gracefully degrade when overloaded.

The current scope of our work is confined to uniprocessor systems In the future, we plan to (1) address the utilization of the existing schedule when a new task arrives, (2) address the issues of considering and predicting the load of individual processors, (3) investigate the possibility of using fuzzy logic chip as the scheduling co-processor, and (4) extend to distributed systems using either a focused addressing or bidding algorithm to offload tasks that can not be scheduled locally.

References

[1] S. R. Biyabani, J. A. Stankovic, and K. Ramamritham. "The Integration of Deadlines and Criticaleness in Hard Real-Time Scheduling". *RTSS'88*, pages 152–160, Dec 1988.

[2] S. Cheng, J.A. Stankovic, and K. Ramamritham. "Scheduling Algorithms for Hard Real-Time Systems – A Brief Survey".

[3] C.L. Liu and J.W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". *J. ACM*, pages 46–61, Jan 1973.

[4] K. Ramamritham and J. A. Stankovic. "Dynamic Task Scheduling in Hard Real-Time Distributed Systems". *IEEE Software*, pages 65–75, July 1984.

[5] J.A. Stankovic and K. Ramamritham. "The Design of the Spring Kernel". *RTSS'87*, pages 146–157, Dec 1987.

[6] J.A. Stankovic, K. Ramamritham, and S. Cheng. "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems". *IEEE Trans. Comp.*, pages 1130–1143, Dec 1985.

[7] W. Zhao, K. Ramamritham, and J.A. Stankovic. "Preemptive Scheduling Under Time and Resource Constraints". *IEEE Trans. Comp.*, pages 949–960, Aug 1987.

[8] W. Zhao, K. Ramamritham, and J.A. Stankovic. "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems". *IEEE Trans. on Software Eng.*, May 1987.