Fuzzy Logic Based Adaptive Hierarchical Scheduling for Periodic Real-Time Tasks

Tom Springer
University of California, Irvine
Center for Embedded Computer Systems
tspringe@uci.edu

Steffen Peter
University of California, Irvine
Center for Embedded Computer Systems
st.peter@uci.edu

Tony Givargis
University of California, Irvine
Center for Embedded Computer Systems
givagis@uci.edu

ABSTRACT
In this paper, we present a new scheduling approach for real-time tasks in an embedded system. Our method utilizes hierarchical scheduling to provide a resource based allocation scheme while using a fuzzy logic based feedback scheduler to react to environmental changes within the application. The primary goal is to provide a scheduling mechanism that can adapt to overload conditions but still present a level of service while enforcing the temporal isolation between independent applications. The scheduler then considers this level of service to make scheduling decisions based upon a task's service requirements, such as criticality or timeliness. Implemented in VxWorks on a uniprocessor-based platform results show that our adaptive approach provides significant advantages, during overload conditions, over traditional fixed-priority scheduling schemes.

Categories and Subject Descriptors
C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and embedded systems.

General Terms
Algorithms, Performance, Reliability

Keywords
Real-time systems, hierarchical scheduling, fuzzy logic, real-time operating systems.

1. INTRODUCTION
Current embedded systems are becoming considerably more complicated and they are expected to handle increasingly diverse applications. No longer are they considered special-purpose computing environments but are evolving into more general-purpose type platforms in terms of their processing and workload requirements. These increasingly diverse applications present new challenges for traditional real-time scheduling mechanisms in that applications can have conflicting objectives. For example, one application may be more concerned with screen update response as opposed to whether a single update is missed. While a mission critical application, such as a navigational task, cannot afford to miss even a single update.

The problem is that traditional real-time scheduling mechanisms do not map well to these diverse types of applications specifically during a processing fault or during periods of computational overload. Faults can occur from longer than unexpected task execution time or from programming errors which can lead to the starvation for all lower-priority tasks. An overload can occur as the result of too many tasks being admitted into the system resulting into what is known as the "domino effect" where all tasks except the newly admitted one miss their deadlines.

EWiLi'15, October 8th, 2015, Amsterdam, The Netherlands.

The challenge is that many embedded systems are expected to perform continuous operations in potentially harsh environments and execute at least a subset of critical operations during a fault or overload condition. In order to enforce these strict timing constraints required by critical functions during a fault condition a form of temporal isolation is needed so that corresponding timing requirements are respected. During an overload event the system needs to be able to dynamically adapt to the current load so that system performance can degrade gracefully.

As a solution to these challenges our work utilizes hierarchical scheduling to provide the temporal isolation for real-time tasks by enforcing their timing constraints. The hierarchical scheduling framework (HSF) originally proposed by researchers [1] is a component based technique for scheduling complex real-time systems. The initial idea in applying this approach is that relatively simple components can be used to create larger and more complex systems. In this way, the timing constraints of individual components can be verified, a type of divide and conquer approach. Therefore, by extending this framework we can then schedule each application (i.e. component) such that their timing constraints are satisfied. However, the current limitation with a traditional HSF-based approach is that the scheduling parameters for each component are assigned statically. Unfortunately, in a dynamic system the resource demand for each component can vary significantly especially during periods of overload. It is for this reason that we present an adaptive mechanism where the component parameters can adapt to environmental changes in the system. In this way, the system can degrade gracefully in the presence of computational overload while still maintaining a level of serviceability for critical applications.

For this work we apply a novel approach where the component parameters adapt based upon a value-based heuristic instead of a deadline based policy. This value-based approach is applied because authors in [5] have presented the limitations of a deadline based model for real-time scheduling and have concluded that a value-based approach can more accurately represent the cost or benefit of meeting or missing a deadline. The challenge is in assigning this value metric because in the event of an overload we want to degrade the performance gracefully by ensuring that tasks are provided at least some minimum level of service. Therefore, during an overload when the current schedule is unfeasible we want the scheduler to schedule tasks according to some intelligent heuristic. Some possible heuristics would include scheduling the most important tasks first while still maintaining some level of timeliness for the less important tasks. Our approach is to utilize a heuristic function for guiding the scheduling decisions in a complicated situation where multiple factors may need to be considered such as deadlines, task criticality or task response times.

In this paper we present a new adaptive hierarchical scheduler for real-time systems (AHS-RT) that provides timing guarantees for critical tasks and a minimum level of service for non-critical tasks during overload conditions. Our approach is to utilize fuzzy logic for the guidance mechanisms because they prove to be easier to express, comprehend and modify than other heuristic functions.

The remainder of this paper is organized as follows. Section 2 provides an overview of the hierarchical scheduling framework used by our scheduling mechanism. Section 3 discusses related work and Section 4 provides an overview of the hierarchical scheduler (AHS-RT). Section 5 presents the simulations we used to provide comparisons between our scheduling approach and traditional fixed priority scheduling. In Section 6 we conclude with future work and the research summary.

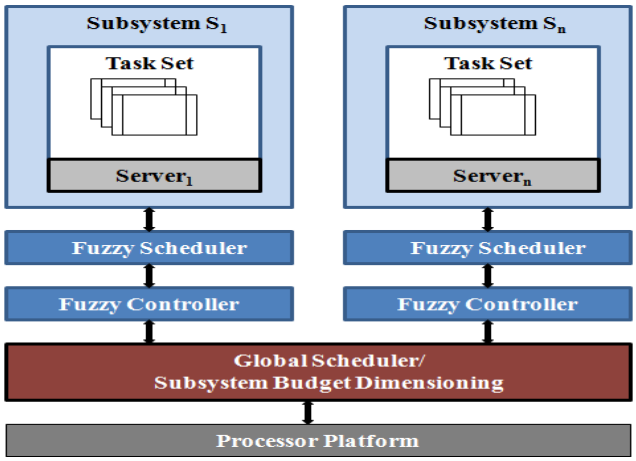

Figure 1: AHS-RT Architecture

2. BACKGROUND

This section provides a background of the terminology used in the paper as well as an overview of hierarchical scheduling provided as a reference for the overall architecture of adaptive hierarchical scheduling.

2.1 Hierarchical Scheduling Framework

Hierarchical scheduling provides a framework for scheduling multiple real-time applications on a single processor which is modeled as a system *S*. Each system may consist of multiple applications (subsystems S_i) such that $S_i \in S$. Each *subsystem* consists of a number of real-time *tasks*. Each subsystem is associated with a *periodic server* which provides the temporal isolation between subsystems. The execution of tasks is performed using a two-level hierarchical scheduling policy: *global* and *local*. The global scheduling policy determines which subsystem has access to the processor while the local scheduling policy determines which task should actually execute (Figure 1).

2.2 Task Model

We consider a task set $\Gamma_s = \{\tau_1, \tau_2, \ldots \tau_n\}$, such that each task τ_i is defined as (T_i, C_i, D_i, L_i) where T_i is defined as the task period, C_i denotes the task worst case execution time (WCET), D_i is the relative deadline and L_i represents the task criticality value. It is assumed that each task τ_i is a constrained task such that $C_i \leq D_i \leq T_i$. The criticality value L_i represents the importance or weight of the task as it relates to other tasks in the set. The criticality value along with the deadline and period are used by the fuzzy inference engine to make scheduling decisions by the local scheduler.

2.3 Subsystem Model

Each subsystem consists of a task set Γ_{si} such that $S_i \leftarrow \Gamma_{si}$. The subsystem is modeled as a periodic task so a subsystem can be scheduled in a similar way as a simple real-time periodic task. The subsystem is defined as $S_i = (P_i, Q_i, L_i)$ where P_i represents the subsystem period, Q_i represents the subsystem budget and L_i represents the subsystem criticality level. Similar to the task model the service value L_i is used to make scheduling decisions at the subsystem level. Note that during overload conditions the subsystem with the highest criticality level is granted its full budget at the possible expense of lower criticality subsystems.

2.3.1 Periodic Server

The virtual server is invoked with the corresponding subsystem period P_i. If there are any ready tasks within the subsystem then they execute until they complete or the server's budget Q_i is exhausted. If there are no ready tasks to execute or no higher priority subsystem needs to utilize some of the server's budget during an overload condition then the capacity is idled away as if a background task were running. After a server's budget is exhausted the server suspends the execution of the subsystem until the capacity is replenished at the start of the next period. For this work we choose a periodic server as the fixed priority server algorithm, in part because the simpler design has less overhead but also because authors in [2] have shown it to dominate other fixed-priority server algorithms.

2.4 Fuzzy Systems

The scheduler and the controller of AHS-RT are based upon fuzzy-logic heuristics. The fuzzy logic based approach was chosen because of its strength in dealing with dynamic environments involving a certain degree of uncertainty. The fuzzy system is defined as having *n* inputs $x_i \in X_i$, where $i = 1,2, \ldots, n$ and X_i, is the collection of numbers for x_i (universe of discourse for x_i) and one output $y \in Y$, where Y is the universe of discourse for y (multiple input single output fuzzy system). The inputs x_i and output y are *crisp* values (i.e. real numbers). The structure of the fuzzy system consists of three stages; *fuzzication* stage, *inference* stage and the *defuzzication* stage. The fuzzication stage converts the crisp input values into fuzzy sets to be used by the inference stage. The inference stage uses the rules defined in the rule base to convert these fuzzy sets into other fuzzy sets that represent the recommendations of the various rules in the rule base. The defuzzication stage combines these fuzzy recommendations to provide a crisp output.

3. RELATED WORK

Hierarchical scheduling framework (HSF) was initially proposed by researchers [1][4][6] as a means to reduce the scheduling complexity for open source embedded systems. Resource partitioning [7] was introduced as a general technique for limiting the effects of overruns in tasks with variable execution times. This resource reservation technique can then be applied by hierarchical schedulers to provide the temporal isolation between subsystems

for more predictable behavior, improved reusability and composability. However, the current limitation with HSF is that in order to determine the resource reservations all tasks parameters must be known *a priori* and fixed during run-time. The problem is that accurate task information may not be known or hard to derive at run-time. Additionally, in order to account for overload conditions the system may need to be over-engineered which could lead to significant underutilization during nominal load periods.

In [8] [9] [10] authors proposed a feedback mechanism to account for the dynamic behavior when the task parameters may not be fully known. The approach was for the scheduler to maximize the CPU utilization, avoid system overload and distribute the computing resource evenly among tasks. By incorporating feedback the scheduler reacts to changes in the workload then tries to keep the overall utilization as close as possible to a desired set point typically using a type of control mechanism, such as a proportional integral derivative (PID) controller. Related work [11] [12] adjusts the resource allocation on-line based upon a quality-of-service (QoS) scheme where a certain level of service is provided in cases overload. However, the primary objective of this approach is control performance and not necessarily minimizing the number of missed deadlines.

Authors in [14] took a slightly different approach in that they based their scheduler on a benefit based model. Their approach was to schedule the tasks using a traditional deadline based scheduling policy until a potential fault was detected and before an overload condition could occur. After a fault is detected the scheduler switches to a benefit based scheduler that considers task importance, system state and timeliness to schedule tasks. Authors in [13] also took a similar approach in adaptive scheduling except they manipulated the task period of other tasks to achieve the desired level of performance.

Other research [15] [16] [17] treated the uncertainty of varying execution times as a multi-criteria optimization problem then applied fuzzy logic to derive a feasible schedule. Their approach was to treat various task parameters, such as deadline, start time or execution time, as inputs to the fuzzy scheduler then perform fuzzy analysis to assign a task priority value. Additional work [18] utilized fuzzy logic as a means for tuning a feedback controller to provide optimal resource utilization through task period re-adjustment.

Recent work [19] extended hierarchical scheduling to provide an adaptive hierarchical framework for managing overruns in tasks with varying execution times. Their approach was to utilize a feedback control mechanism for adapting the resource allocation by adjusting the amount of budget assigned to a subsystem. By adjusting the budgets at run-time the framework can better adapt to changes in the workload.

Our approach in AHS-RT is similar to the work in [19] in that we also utilize hierarchical scheduling for determinism and temporal isolation. However, AHS-RT differs in how the local scheduling and global scheduling is performed. Local scheduling is based upon a fuzzy scheduler which is more adept at making scheduling decisions when the task parameters are vague. Research by authors in [17] demonstrated that fuzzy logic based approaches outperform traditional deadline based policies such as earliest deadline first (EDF). In AHS-RT global scheduling also uses a feedback controller but the controller is based upon a fuzzy logic heuristic instead of a PID controller. Because fuzzy logic can better tolerate imprecision thereby providing improved run-time flexibility.

4. AHS-RT Architecture

This section describes the overall architecture (see Figure 1) of the AHS-RT scheduling framework which consists of a two-level hierarchical scheduling framework. The root-level contains the global scheduler which manages how subsystems (i.e. applications) are allocated on the processor. While the node-level contains the local scheduler which manages how tasks are scheduled on the processor.

4.1 Global Scheduling

At run-time the global scheduler chooses the highest priority subsystem that has tasks ready to run. The priority is based upon the subsystem period P_i so the shorter the period the higher the subsystem priority. Therefore if the priority of $S_j > S_i$ then S_j would be scheduled first with its full budget then S_i would be scheduled next with its full budget unless an overload condition is detected. In the event of an overload a higher criticality subsystem may request a budget change at the possible expense of a lower criticality subsystem which may or may not be a lower priority subsystem.

The logical approach may be to re-assign budgets based upon subsystem priority. However, during an overload event studies have shown [3] that a value-based approach offers considerable advantages over traditional deadline-based approaches. For this reason, during an overload event the global scheduler of AHS-RT temporarily switches from a deadline-based scheduling policy to a value-based scheduling policy. Instead of the highest priority subsystem receiving their full budget the subsystem with the highest criticality level L_i will receive their entire budget. Therefore, the global scheduler redistributes budgets based upon the criticality level which means lower criticality subsystems yield their budgets to higher criticality subsystems. This greedy approach can lead to starvation, even for some high priority subsystems, but this is acceptable in that during overload conditions the highest criticality subsystems are considered superior to lower criticality subsystems.

4.1.1 Detecting Overloads

An overload condition is based upon the overall subsystem utilization which is defined as:

$$U_T = \sum_{\forall S_s \in S} \frac{Q_s}{P_s}$$

and because we are using RM then an overload condition is determined by $U_T \leq m(2^{1/m} - 1)$, where m is the number of subsystems. An overload can occur because a subsystem requests a budget change in order to adapt to a fault or missed deadline within a task of an individual application. A budget change does not necessarily mean that the system is overloaded just that there is the potential for an overload condition to exist. Consider some unallocated system utilization denoted as U_T' such that $U_T + U_T' \leq m\left(2^{1/m} - 1\right)$, and then this extra utilization could be temporarily reallocated to the subsystem requesting the additional budget. However, if there are not sufficient resources to satisfy all the budget requirements then the system is considered overloaded which implies that a budget reallocation needs to be performed.

4.1.2 Budget Reallocation

After the full budget has been allocated to the highest criticality subsystem the lower criticality budgets needs to be re-dimensioned. The next lower criticality subsystems are then assigned budgets based upon the remaining utilization. The algorithm and description for budget dimensioning is provided below. The budget dimensioning algorithm (Algorithm 1) works

Algorithm 1 AHS-RT Budget Dimensioning

Input: The subsystem subset S_{sj} with criticality levels less than the task set T_s

Output: A new budget parameter that maintains a schedulable system.

1: FOR each $S_i \in S_{sj}$
2: $Q_i = NewBudget();$
3: WHILE not Schedulable (S_s, S_i) DO
4: $Q_i = FindNewBudget(S_i, Q_i)$
4: END WHILE
6: END FOR

by iterating through all the subsystems S_i in the subset $S_{s,j}$ of lower criticality subsystems. In line 2 the new budget is calculated based upon the remaining system utilization. A schedulability test (line 3) is then performed on the modified budget. If the modified budget renders the system unschedulable then a new budget value is attempted based upon the previous failed value. The algorithm continues to reduce the budgets of lower criticality subsystems until a schedulable system is found.

4.2 Local Scheduling

The local scheduling of AHS-RT consists of two primary components; a fuzzy logic based scheduler and a fuzzy logic based feedback controller. The scheduler selects the task to execute on the processor derived from the fuzzy rules based approach to real-time scheduling. The feedback controller gathers system state information for subsystem budget management to maximize utilization and minimize missed deadlines.

4.2.1 Fuzzy Scheduler

At run-time the fuzzy scheduler selects the highest priority task that is ready for execution on the processor. The priority of the task is determined by several parameters: task deadline, task criticality and task execution time. The task deadline is the time before the task should be completed. The task criticality relates to the consequences of missing a deadline. The task execution time is the worst-case execution time for that task. These parameters are then *fuzzified* and represented as linguistic variables (i.e. a word used to describe a variable). Fuzzy rules are then applied to the linguistic variables to compute the service value. The linguistic values for the three parameters are defined as: *task deadline* (early, on-time, late), *task criticality* (hard, firm, soft) and *CPU time* (very low, low, normal, high, very high). Fuzzy rules are then applied to create a *fuzzy conclusion* for computing the priority level. Figure 2 illustrates the linguistic variables used by the inference stage of the fuzzy scheduler.

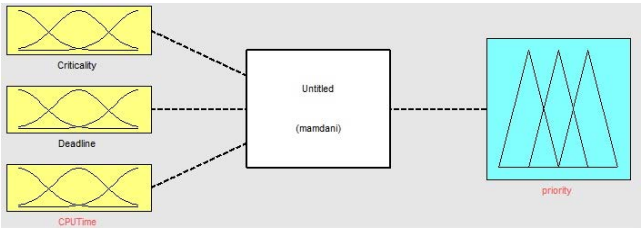

Figure 2: AHS-RT Inference block system rules

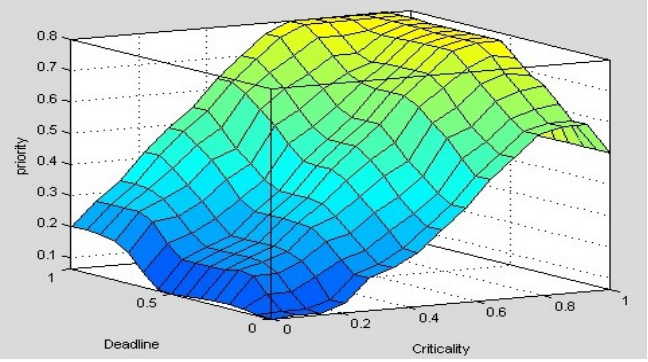

Figure 3: AHS-RT Decision Surface

Some of the fuzzy rules for the scheduler inference mechanism are listed as an example here:

- *If (CPU Time is high) and (deadline is late) and (criticality is hard) then (Priority is very high)*
- *If (CPU Time is normal) and (deadline is on-time)and (critically is firm) then (Priority is normal)*
- *If (CPU time is low) and (deadline is early) and (criticality is soft) then (Priority is low).*

These fuzzy conclusions are then combined to produce a fuzzy variable that represents the criticality level of the task. The variable is then defuzzified to create a value that is compared to other tasks to determine which task should be scheduled next. The decision surface illustrates the crisp output value (priority) that is obtained based upon the input parameters (See Figure 3).

The fuzzy scheduler algorithm (Algorithm 2) iterates through all the tasks τ_i in the task set for a particular subsystem and for each task passes the deadline (D_i), criticality value (L_i) and starting time (T_i) into the fuzzy inference engine. The output from the inference function is a crisp value used to assign a priority to each task and stored in a priority array ($Priority_i$). The task with the highest priority is then executed until some scheduling event occurs (task completion, new task instance arrives or server budget exhaustion). The system status is then updated and if a task misses its deadline such as server budget exhaustion then the deadline miss is reported to the feedback controller which could trigger a budget reallocation across the system.

Algorithm 2 AHS-RT Fuzzy Scheduler

Input: The task set Γ_{si} of subsystem S_{sj}

Output: The "crisp" criticality level (priority) of each task based upon the system state.

1: WHILE Loop
2: FOR each $\tau_i \in \Gamma_{sj}$
3: $Priority_i = FuzzyInferenceEngine(D_i, T_i, L_i);$
4: END FOR
5: $\tau_i = FindHighestPriorityTask(\Gamma_{sj});$
6: $ScheduleTask(\tau_i);$
7: $UpdateSystemStates(\Gamma_{sj});$
8: END LOOP

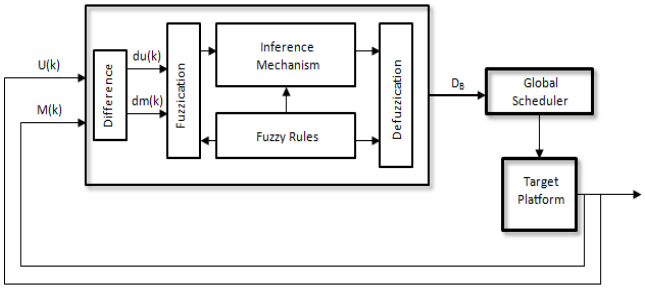

Figure 4: Internal structure of the feedback controller

4.2.2 Fuzzy Feedback Controller

The feedback controller in AHS-RT is similar to the FC-UM algorithm [20] in that both the miss-ratio and utilization are monitored. The reference inputs for miss-ratio M_s and unused budget U_s are both set to zero. At each sampling instant the miss-ratio *M(k)* and the unused subsystem budget *U(k)* are fed back into the controller. These values are then compared to their respective set points to determine the difference where *du(k)* represents the utilization error and *dm(k)* represents the miss-ratio error. The output from the fuzzy controller is the budget dimensioning factor D_B. As part of a typical fuzzy controller (Figure 4) we need to specify meaningful linguistic values and membership functions for each input and output variable. The input to the controller are miss ratio M_s and task utilization ratio U_s defined as a triangular membership functions. The input linguistic values are *utilization* (very low, low, normal, high, very high) and *deadline misses* (zero, small, medium, high). The output linguistic values is *bandwidth adjustment* (none, very small, small, medium, big and very big).

Some of the fuzzy rules for the controller inference mechanism are listed as an example here:

- *If (misses are zero) and (utilization is normal) then (bandwidth adjustment is none)*
- *If (misses are small) and (utilization is high) then (bandwidth adjustment is small)*
- *If (misses are high) and (utilization is high) then (bandwidth adjustment is high)*

4.3 Task Scheduling Example

To demonstrate AHS-RT we have provided an example scheduling scenario. Note for illustration purposes we are only considering one subsystem. So, the primary purpose of this example is present how the fuzzy scheduler manages tasks within the context of one subsystem.

Consider the task set and subsystem listed in Tables 1 and 2. Table 3 describes the scheduling of tasks at the first scheduling event where tasks τ_1, τ_2 and τ_3 all have the same initial starting time but since τ_1 has the nearest deadline it is assigned the highest priority by the fuzzy scheduler. Therefore, τ_1 is allowed to execute until completion then at time unit *t3* task τ_2 executes until time unit *t5* when the subsystem's budget expires. At time unit *t10* (see Table 4) the subsystem's budget is replenished where the tasks can continue execution. At this time the fuzzy scheduler performs a re-ordering of task priorities to reflect the system state. Task τ_2 is assigned the highest priority because the start time is the earliest and the deadline is the closest.

Table 1: Subsystem Parameters

Subsystem	P_s	Q_s	L_i
S_1	10	5	10

Table 2: Task Parameters

Task	T_i	C_i	D_i	L_i
τ_1	10	2	10	5
τ_2	15	5	15	10
τ_3	20	3	20	10

Table 3: Scheduling snapshot at time 0

Task	$Start_i$	D_i	C_i	L_i	$Prio_i$
τ_1	0,10,20,30	10,20,30,40	2	5	~9
τ_2	0,15,30	15,30	5	10	~5
τ_3	0,20,40	20, 40	3	10	~3

Table 4: Scheduling snapshot at time 10

Task	$Start_i$	D_i	C_i	L_i	$Prio_i$
τ_1	20,30	20,30,40	2	5	~5
τ_2	15,30	15,30	2	10	~9
τ_3	20	20, 40	3	10	~7

Table 5: Scheduling snapshot at time 20

Task	$Start_i$	D_i	C_i	L_i	$Prio_i$
τ_1	20,30	30,40	2	5	~10
τ_2	30	30	5	10	~5
τ_3	40	40	3	10	~3

Note that at time unit *t12* task τ_2 will complete execution but task τ_3 will be scheduled over τ_1 even though both tasks have the same relative deadline and start time. This is because τ_3 was assigned a higher priority by the fuzzy controller because τ_3 was defined to be a higher criticality task than τ_1. Also note, due to subsystem budget exhaustion at time unit *t15* task τ_1 will miss its deadline which would trigger a budget reallocation request to the fuzzy controller for an increase in the subsystem budget. Finally, at time unit *t20* (see Table 5) the scheduler re-orders the task priorities where once again τ_1 will be assigned the highest priority.

4.4 Subsystem Reallocation Example

Consider the following subsystems with parameters presented in Table 6 which is used to illustrate how a subsystem is scheduled by AHS-RT.

Table 6: Subsystem Parameters

Subsystem	P_s	Q_s	L_i
S_1	12	4	10
S_2	15	3	8
S_3	20	4	5

Table 7: Budget Reallocation Snapshot

Subsystem	P_s	Q_s	L_i	$DU(k)$	$DM(k)$	D_B
S_1	12	3	10	-0.2	0.1	~4.0
S_2	15	3	8	0.0	0.0	~3.0
S_3	20	5	5	0.0	0.0	~5.0

Suppose that at some scheduling instant subsystem S_1 has a current budget $Q_1 = 3$ but due to a deadline miss the fuzzy controller recommends a budget increase to 4. Also suppose that S_2 and S_3 report no deadline misses or under utilization so the fuzzy controller recommends no budget changes. However, the increased budget of S_1 causes the schedulability test to fail because $U_T > m\left(2^{1/m} - 1\right)$ so now the criticality level L_i is considered and since S_1 has the highest criticality level it is granted the full budget. After $Q_1 = 4$ the budget dimensioning algorithm is performed to redistribute the remaining utilization. Initially, the budgets for S_2 and S_3 will be $Q_2 = 3$ and $Q_3 = 0$ then a successful schedulability test will be performed. Next the budget for S_3 will be $Q_3 = 5$ but the schedulability test will fail. Since the system is no longer schedulable the budget for S_3 will now be $Q_3 = 4$. This time the system is schedulable so the adjusted budgets are reallocated to their respective subsystems.

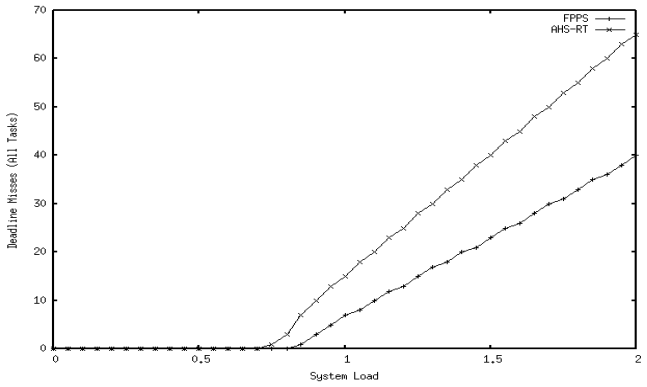

Figure 5: Number of Deadline Misses (All Tasks)

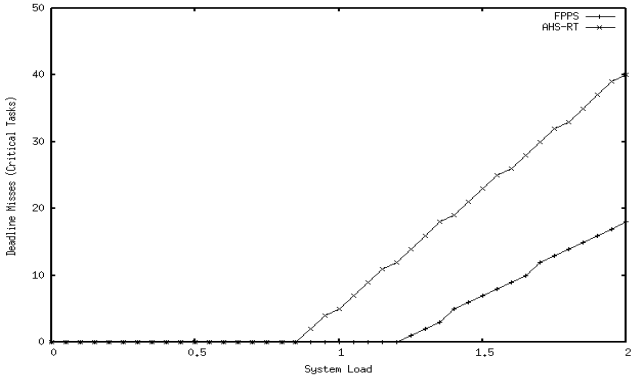

Figure 6: Number of Deadline Misses (Highest Criticality Tasks)

5. SIMULATION

AHS-RT was implemented as part of the VxWorks 6.9 real-time operating system (RTOS). The simulations were executed using the *SIMNT vxsim* simulator. For evaluation purposes we ported the SNU Real-Time Benchmark Suite [22] to compare deadline misses. The SNU real-time benchmark suite contains small C programs used for worst-case execution time analysis. The programs are mostly numeric and DSP algorithms. In order to represent the periodic task model of an embedded system a subset of the programs in the benchmark suite were chosen and assigned arbitrary task rates and criticality levels. Illustrated in Figure 5 both AHS-RT and the VxWorks native fixed-priority preemptive scheduler (FPPS) are comparable as long as the load factor is below ~0.70 which corresponds with the lower bound for priority based algorithms. Notice that AHS-RT experiences significantly fewer deadline misses than FPPS when the system starts to become overloaded (> ~0.70). Also note that AHS-RT manages overload more effectively in that it does not start to report deadline misses until closer to a ~0.80 load factor. Another important observation depicted in Figure 6 is that AHS-RT manages deadline misses much more effectively than FPPS for higher criticality tasks. Notice that AHS-RT does not even start to report deadline misses until close to a ~1.25 load factor while FPPS starts to report deadlines as early as ~0.85. Clearly, AHS-RT is the superior scheduling mechanism as compared to FPPS specifically during periods of overload.

6. CONCLUSIONS/FUTURE WORK

In this paper we considered the problem of how to schedule tasks with varying levels of criticality on a uniprocessor to more effectively adapt to computational changes. Those changes were managed by hierarchical scheduling to provide the temporal isolation between tasks. The efficient scheduling of tasks was accomplished using a fuzzy based heuristic which has been proven to be more effective than traditional deadline based approaches especially during periods of overload. The results are a demonstrated reduction in deadline misses for all tasks during periods of overload as compared to traditional fixed priority based scheduling mechanisms. As further confirmation for the practicality for this approach we implemented AHS-RT as part of the VxWorks RTOS.

Future work includes evaluating the additional overhead AHS-RT incurs in VxWorks as compared to the traditional scheduler. Additionally, we would like to extend AHS-RT into a multi-core environment and consider semi-independent tasks where subsystems would have to share a mutual resource such as a semaphore.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under NSF grant number 1136146

REFERENCES

[1] Z. Deng and J. W.-S. Liu, “Scheduling real-time applications in an open environment,” (*RTSS*’97), pp. 308–319.

[2] Davis, R.I.; Burns, A., "Hierarchical fixed priority pre-emptive scheduling," *(RTSS’05)*

[3] Saini, G., "Application of fuzzy logic to real-time scheduling," *Real Time Conference, 2005.*

[4] I. Shin and I. Lee, “Periodic resource model for compositional real-time guarantees,” *(RTSS ’03).*

[5] C.D. Locke and H. Tokuda, “A Time-Value Driven Scheduling Model for Real-Time Operating Systems”, *Proc. Symp. on Real-Time Systems*, Nov. 1985.

[6] F. Zhang and A. Burns, “Analysis of hierarchical EDF pre-emptive scheduling,” *in Proc. of the 28th IEEE International Real-Time Systems Symposium* (RTSS’07)

[7] A. Mok, X. Feng and D. Chen, “Resource partition for real-time systems,” in *Proc of the 7th Real-Time Technology and Applications Symposium* (RTAS’01), 2001

[8] J. Stankovic, C. Lu, S. Son and G. Tao, “The case for feedback control in real-time scheduling,” *in Proc. of the 11th Euromicro Conference on Real-Time Systems* (ECRTS ’99).

[9] C. Lu, J. Stankovic, G. Tao and S. Son, “Design and evaluation of a feedback control EDF scheduling algorithm,” *in Proc. of the 20th IEEE (RTSS’99).*

[10] C. Lu, J. Stankovic, S. Son and G. Tao, “Feedback control real-time scheduling: Framework, modeling and algorithms,” *Real-Time Systems*, vol. 23, pp 85-126, 2002.

[11] T. Abdelzaher, E. Atkins and K. Shin, “QoS negotiation in real-time systems and its application to flight control,” *in Proc. of the IEEE (RTSS’97)*.

[12] R. Rajkumar, C. Lee, J. Lehoczky and D. Siewiorek, “A resource allocation model for QoS management, “ *in Proc. of the IEEE Real-Time Technology and Applications* 1997.

[13] S.P. Dwivedi, "Adaptive Scheduling in Real-Time Systems Through Period Adjustment", *CoRR*, 2012.

[14] Richardson, P.; Sarkar, S., "Adaptive scheduling: overload scheduling for mission critical systems," *(RTSA)* 1999

[15] J. Yen, J. Lee, N. Pfluger, and S. Natarajan. "Designing a fuzzy scheduler for hard real-time systems." (1992).

[16] L. Jonathan, A. Tiao, and J. Yen. "A fuzzy rule-based approach to real-time scheduling." In *Fuzzy Systems, In Proc. of the 3rd IEEE Conference*. IEEE, 1994.

[17] S. Mojtaba, and M. Naghibzadeh. "A Fuzzy algorithm for real-time scheduling of soft periodic tasks." *IJCSNS International Journal of Computer Science and Network Security* 6.2A (2006):

[18] X. Feng, X. Shen, L. Liu, Z. Wang, and Y. Sun. "Fuzzy logic based feedback scheduler for embedded control systems." In *Advances in Intelligent Computing*, 2005.

[19] Khalilzad, N.M.; Behnam, M.; Nolte, T., "Adaptive hierarchical scheduling framework: Configuration and evaluation," *(ETFA), 2013.*

[20] L., Chenyang, J. Stankovic, H. Son, and G. Tao. "Feedback control real-time scheduling: Framework, modeling, and algorithms*." *Real-Time Systems* (2002).

[21] M. Behnam, T. Nolte, I. Shin, M. Asberg. Towards Hierarchical Schedling in VxWorks. (*OSPERT)* 2008

[22] SNU Real-Time Benchmark, http://www.cprover.org