Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance

Steve Vestal
steve.vestal@honeywell.com
Honeywell Labs
Minneapolis, MN 55418

Abstract

This paper is based on a conjecture that the more confidence one needs in a task execution time bound (the less tolerant one is of missed deadlines), the larger and more conservative that bound tends to become in practice. We assume different tasks perform functions having different criticalities and requiring different levels of assurance. We assume a task may have a set of alternative worst-case execution times, each assured to a different level of confidence. This paper presents ways to use this information to obtain more precise schedulability analysis and more efficient preemptive fixed priority scheduling. These methods are evaluated using workloads abstracted from production avionics systems.

1 Introduction

Schedulability analysis only guarantees that deadlines are met to the degree that the assumptions of the model are guaranteed to hold in the real system. A particularly problematic assumption is that no task execution time exceeds the specified worst case execution time (WCET) for that task. In practice, determining an exact WCET value for a task is very difficult and remains an active area of research[8]. The WCET parameter used for analysis is usually a conservative upper bound that exceeds the true WCET.

In many applications, the consequences of missing a deadline vary in severity from task to task. In RTCA DO 178B, for example, system safety analysis assigns to each task a criticality level (ranging from A to D), where erroneous behavior by a level A task might cause loss of aircraft but erroneous behavior by a level D task might at worst cause inconvenient or suboptimal behavior.

This paper is based on a conjecture that the higher the degree of assurance required that actual task execution times will never exceed the WCET parameters used for analysis, the larger and more conservative the latter values become in practice. For example, at low criticalities the worst time observed during tests of normal operational scenarios might be used. At higher criticalities the worst time observed during more exhaustive tests specifically constructed for this purpose might be used. At the highest criticality, some code flow analysis and worst-case instruction cycle counting might be done.

This paper presents modified preemptive fixed priority (PFP) schedulability analysis algorithms that assume the timing guarantees for a task must be based on WCET values that have the same level of assurance as required for that task. We also present algorithms that take this into account when assigning scheduling priorities. We include a comparative evaluation of these algorithms, based on data obtained from production avionics systems.

How can we have a highly assured worst-case execution time for a piece of low assurance software? Defects that may impact timing (e.g. infinite loops) are not assured to be absent to the degree required. In current practice, worst-case exection times having higher assurance than the software itself are obtained by time partitioning (run-time enforcement). In time partitioned PFP systems, when a task completes earlier than its enforced budget, the remaining time becomes available for other tasks[2]. In many applications, this behavior is relied upon when determining whether lower criticality tasks will meet their deadlines. This paper presents a structured way to use and analyze this in time partitioned PFP systems.

2 Multi-Criticality Analysis

Using traditional notation, τ_i for $i = 1..n$ denotes a set of periodic tasks. T_i and D_i specify the period and deadline respectively for task τ_i.

Let $\mathcal{L} = \{A, B, C, D\}$ be an ordered set of design assurance levels, where A is the highest design assurance level. Let L_i specify a design assurance level for τ_i (normally determined during system safety analysis). Let C_{il} for $l \in \mathcal{L}$ specify a set of compute times

1052-8725/07 $25.00 © 2007 IEEE
DOI 10.1109/RTSS.2007.47

for task τ_i, where l is the degree of assurance that true execution times for τ_i will not exceed C_{il}. We assume that $D_i \leq T_i$ and $C_{iA} \geq C_{iB} \geq C_{iC} \geq C_{iD}$ for all i.

For each task τ_i we want to assure to level L_i that τ_i never misses a deadline. We assume this level of assurance is achieved when the analysis is carried out using compute times having the same level of assurance.

Preemptive fixed priority (PFP) scheduling is a widely-used uni-processor discipline. For each task τ_i, a distinct scheduling priority is specified by $\rho_i \in 1..n$. Among all tasks that have been dispatched but not completed, the processor is executing the task having the numerically smallest priority (which, by convention, is called the highest scheduling priority although it is numerically smallest).

Here we modify the Joseph-Pandya worst-case response time algorithm to perform multi-criticality analysis[6] (the same approach could be applied to other PFP analysis algorithms[9]). Let R_i denote the worst-case response time of task τ_i, which can be iteratively computed as the least fixed point of the equation

$$R_i = \sum_{j:\rho_j \leq \rho_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_{jL_i} \tag{1}$$

We attempt to give brief insight into this formula as follows. It has been shown that the worst-case response time occurs when a task is dispatched simultaneously with all tasks having higher priority[5]. This equation sums all the work dispatched for a task and all higher priority tasks starting at this so-called critical instant, up until the earliest point in time at which all such work has been completed. This earliest point is the least fixed point for R_i. This is computed by first assigning a variable $R_i := C_{iL_i}$, then repeatedly evaluating the right-hand-side to update R_i until either the value of R_i no longer changes or $R_i > D_i$. In the latter case, the task set is said to be unschedulable or infeasible using the given priority assignment.

3 Multi-Criticality Scheduling

A deadline monotonic priority assignment (one in which scheduling priorities are assigned in order of deadline, higher scheduling priority to lower deadline) is known to be optimal given fixed periods and compute times. However, it is not optimal for the multi-criticality scheduling problem. An example is $T_1 = D_1 = 2, L_1 = B, C_{1B} = 1, C_{1A} = 2$; and $T_2 = D_2 = 4, L_2 = A, C_{2B} = 1, C_{2A} = 1$. If task 1 is assigned highest priority, then task 2 sees a processor that already has 100% of its available level A time set aside for task 1. However, the system is feasible (as determined by the previous multi-criticality analysis algorithm) if task 2 is assigned highest priority.

Period transformation is a technique originally developed to provide graceful degredation under overload conditions by modifying a workload so that higher criticality tasks have higher scheduling priorities[7]. If a high-criticality task has a longer period than a low-criticality task, run-time time slicing is used to schedule the higher-criticality task as if it had a smaller period and execution time. For each task τ_j where $D_j = T_j$ and there is some other remaining task having lower criticality but smaller period, transform the period and WCET to $T'_j = T_j/n, C' = C_j/n$ for an integer n that is just sufficient to reduce the period to a value at or below the period of every other task having lower criticality. (The algorithm as stated here assumes deadlines are equal to periods, but it can be extended to handle preperiod deadlines[3].)

Period transformation can improve multi-criticality schedulability. We compute the transformed WCET from C_{iL_i}, the WCET assured to the same level as the task. The value C_{iL_i}/n is used as a run-time time slice. When a transformed task is executed, some number of the initial transformed dispatches always consume this maximum allowed time-slice. We assume the task will nominally complete before accumulating a total of C_{iL_i} execution time by the end of its original untransformed period. This means at some point one of the transformed dispatches will complete early, after which the remaining transformed dispatches will have zero execution time until the end of the original untransformed period.

We modify the multi-criticality analysis equation 1 as follows to take this behavior into account. At each iteration of the fixed-point computation, if a task τ_j has been transformed, then

$$\left\lceil \frac{R_i}{(T_j/n)} \right\rceil \tag{2}$$

is the number of transformed dispatches up to R_i. Each block of n such dispatches corresponds to a complete execution of the original untransformed task, which we assume actually requires only C_{jL_i} rather than C_{jL_j}. When summing the total work of τ_j that might preempt τ_i, use C_{jL_i}/n for the first

$$n \left\lfloor \frac{R_i}{T_j} \right\rfloor \tag{3}$$

transformed dispatches (blocks of transformed dispatches that complete the original untransformed task), and C_{jL_j} for the remaining (transformed dispatches that may execute until the current time-slice has been exhausted).

If τ_i has itself been transformed, then the modified equation 1 actually gives the worst-case response

time of the first transformed dispatch of τ_i, not the response time for the original untransformed task. The equation would need to be further modified to add idle intervals resulting from enforced time-slicing of τ_i. However, a test of schedulability is sufficient for our immediate purposes, and we omit these extensions here.

A second multi-criticality scheduling technique, which can be used either as an alternative to or in combination with period transformation, is to apply Audsley's priority assignment algorithm to the multi-criticality scheduling problem[1]. This algorithm is based on the following two observations, which we state without proof.

lemma 1: The worst-case response time for a task τ_i can be determined (e.g. by equation 1) by knowing which subset of tasks has higher priority than τ_i but without otherwise knowing what their specific priority assignments are.

lemma 2: If task τ_i is schedulable given a particular priority assignment, then it remains schedulable if it is assigned a higher priority (other task priorities remaining fixed).

The algorithm begins with no task having an assigned priority. Priorities are assigned from lowest to highest scheduling priority, so that the first task assigned a priority will be the lowest priority task. At each step, a task is selected from among those still lacking a priority assignment and is assigned the next higher priority. The task selected at each step is any one of those that will be schedulable if assigned the next higher priority. By lemma 1, this can be determined without knowing the exact priorities that will be assigned to the remaining tasks, only that the remaining tasks will eventually receive a higher priority. The algorithm terminates unsuccessfully at any step in which none of the remaining tasks are schedulable, or successfully when all tasks have been assigned a priority. Lemma 2 assures us that the algorithm will find a feasible priority assignment if any such exists; the schedulable tasks that are not picked at any step cannot be made unschedulable by waiting and assigning them higher priorities at later steps.

If Audsley's algorithm is modified to use multi-criticality schedulability analysis at each step, the result is a priority assignment algorithm for the multi-criticality scheduling problem. At each step, among all feasible choices, we selected the one having the greatest critical scaling factor (critical scaling factors are explained in the next section).

If any of the priority assignment algorithms we considered could not otherwise distinguish between two tasks, then the tie was broken by assigning the higher priority to the task having higher criticality.

task	T_i	L_i	measured	allocated	margin
P1 40hz	25	B	1.06	1.4	32.1%
P1 20hz	50	B	3.09	3.9	26.2%
P2 20hz	50	B	2.7	2.8	3.7%
P3 20hz	50	B	1.09	1.4	28.4%
P4 40hz	25	A	0.94	1.1	17%
P4 20hz	50	A	1.57	1.8	14.6%
P4 10hz	100	A	1.68	2.0	19%
P4 5hz	200	A	4.5	5.3	17.8%
P5 20hz	50	B	2.94	3.7	25.9%
P5 10hz	100	B	1.41	1.8	27.7%
P5 5hz	200	B	6.75	8.5	25.9%
P6 20hz	50	D	5.4	5.4	0%
P6 5hz	200	D	2.4	2.4	0%
P7 20hz	50	D	0.94	1.3	38.3%
P7 5hz	200	D	1.06	1.5	41.5%
P8 40hz	25	D	2.28	2.3	0.9%
P8 10hz	100	D	4.75	4.8	1.1%
P8 5hz	200	D	12.87	13	1%
P9 10hz	100	D	0.47	0.6	27.7%
PA 20hz	50	C	1.24	1.9	53.2%
PB 20hz	50	D	1.62	2.4	48.1%
utilization			80.4%	93%	21.4%

Table 1: Example Multi-Criticalty Workload

4 Evaluation

Discussions with individuals from various Honeywell sites indicated that execution time measurements obtained from instrumented platforms were the primary but not only data used to determine worst-case execution time parameters. Testing was influenced by the design assurance level of a task, e.g. at higher levels, more effort was spent analyzing program flows and developing additional tests specifically to assure worst-case execution paths were exercised. Special steps were sometimes taken to deal with caches, e.g. invalidating a cache at context swaps. In some cases experience or special factors associated with a particular application domain were taken into account, sometimes including added safety margins. Detailed flow analyses and instruction cycle counting were used in a few critical localized areas, e.g. context swap code in the operating system.

We obtained processor workload data from three time-partitioned avionics systems, developed by three different applications groups, and hosted on two different platform configurations (different processor, RTOS, and cross-development environment). Table 1 shows an abstract workload of 17 tasks extracted from one of these data sets. Measured is the largest observed execution time across a series of tests. Allocated is the execution time limit enforced in the time-partitioned implementation and used in schedulability analysis for certification purposes. Margin is the difference between allocated and measured, expressed as a percentage of the measured. The bottom row shows that utilization computed using measured versus al-

	Workload 1		Workload 2		Workload 3	
method	Δ^*	increase	Δ^*	increase	Δ^*	increase
deadline monotonic priority traditional analysis	1.08	–	1.24	–	1.07	–
deadline monotonic multi-criticality analysis	1.20	11%	1.26	2%	1.09	2%
multi-criticality Audsley's multi-criticality analysis	1.20	11%	1.26	2%	1.09	2%
transformed & deadline monotonic multi-criticality analysis	1.20	11%	1.76	42%	1.70	59%
transformed & multi-criticality Audsley's multi-criticality analysis	1.20	11%	1.76	42%	1.70	59%

Table 2: Comparative Evaluation Results

located values is 80.4% versus 93% respectively. On average, allocated budgets were about 21.4% higher than the largest measured execution time. About 54% of the allocated utilization was at level C, the rest was spread roughly evenly across levels A through D.

The remaining two workloads of 78 and 65 threads had allocated versus measured utilizations of 80.4% vs. 56.8% and 93.1% vs. 58.9%, respectively. On average, the allocated budgets were 71.5% and 33.9% higher than measured WCETs respectively (ignoring a few outliers where the difference exceeded 400%). The 78 thread workload had about 50% of the allocated utilization at level A and the rest roughly evenly spread across the other levels. The third workload had about 45% of the allocated utilization at level A, 29% at level C, and the remainder roughly evenly divided between the other levels.

This data was not collected and categorized to use the methods of this paper, so we had to speculatively construct a multi-criticality workload. In our evaluations, we used the measured value for a task at all criticalities below that of the task itself, and used the allocated value at the task and higher criticalities, except the measured values were always used for the lowest criticality tasks. This is based on the following rationalization. The allocated values are enforced in a time partitioned system, and should be used for the task itself. The measured value is taken to be the nominal WCET and is used when analyzing lower-criticality tasks. The difference reflects the amount of time that in practice will be available to other lower priority, lower criticality tasks during operation.

The metric we use to compare approaches is the critical scaling factor, Δ^*, which is the largest value by which all execution times can be simultaneously multiplied while preserving feasibility[4]. This is a measure of how much additional work could be proportionally added to all tasks while still guaranteeing schedulability, for given schedulability analysis and priority assignment algorithms.

Table 2 shows the comparative results. The increase column gives the percentage increase over the first method shown, which is a traditional deadline monotonic priority assignment used with an analysis based on guaranteed (level A, enforced) worst-case execution times. The second method is a deadline monotonic priority assignment used with our multi-criticality analysis. The third method uses our modified Audsley's algorithm. The next two methods use period transformation and multi-criticality analysis with a deadline monotonic and a modified Audsley's priority assignment, respectively.

All of the benefit for workload 1 was achieved by modified analysis alone, without any change to the way the system would be scheduled. For the remaining two workloads, period transformation combined with the modified analysis provided the greatest benefit, regardless of whether priorities were subsequently assigned using a deadline monotonic or Audsley's algorithm. Note that the benefit is not necessarily related to the average difference between measured and allocated; workload 2 had an average margin of 71.5% and received a 42% benefit, while workload 3 had a smaller average margin of 33.9% and received a larger 59% benefit. Both of these did benefit more than workload 1 with its 21.4% average margin.

5 Remarks

A primary motivator for using these methods would be to provide more software functionality using less hardware. For most embedded computer systems, costs associated with the amount of hardware (e.g. parts cost, size, weight, power) typically dominate engineering development costs over the product life cycle.

A second evaluation metric that could be used is the reduction in development effort that might be achieved. We conjecture that the tighter the WCET

bound, or the more assured the WCET bound, then the more effort required to obtain the bound. One would not in practice obtain WCET bounds at every level of assurance for every task, one would obtain WCET bounds only at the precision and level of assurance required. The use of these methods might therefore enable a reduction in the effort expended to obtain adequately precise and assured WCET parameters.

The evaluations we performed using three abstracted workloads is suggestive of the utility of these methods, but they are not conclusive. We abstracted data for this study from a development process that was not intended to collect data for and use these methods. A better assessment would include a careful consideration of the processes that might be used to determine multi-criticality WCET parameters.

It would be necessary to assure that current certification processes could be acceptably modified to use these methods.

Tractable exact WCET analysis would reduce but not entirely eliminate the utility of these methods. For example, the longest execution paths might be sufficiently infrequent in practice (e.g. error handling paths) that they should be ignored for low-to-moderate criticality tasks. Occasional deadline misses may be tolerable, especially by tasks at lower criticality levels.

We assumed at the beginning that high assurance execution time limits for low assurance software were obtained by time partitioning. However, it might be feasible to do this by design assurance, at least at the lower criticality levels. For example, it might be possible using methods such as enforced coding guidelines, focused code reviews, and testing to assure the worst-case execution time for a task to level C, even though overall task functionality is only deemed assured to level D.

An interesting theoretical question we encountered was: What is a good multi-criticality utilization metric? We considered computing a vector of utilizations (one per design assurance level), computing a utilization using for each task the compute time associated with its own criticality, and computing a utilization using the compute time associated with the highest criticality of any task of equal or lower priority. A vaguely troublesome property of all these metrics is that some workloads may be feasibly scheduled at higher than 100% utilization.

Acknowledgements

The author would like to express his appreciation for the information, insights and comments provided by Curt Bisterfeldt, Ted Bonk, Dennis Cornhill, Matt Diethelm, Wayne King, and Jeff Novacek, all of Honeywell.

References

[1] N. C. Audsley, "Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times," Technical Report YCS 164, Department of Computer Science, University of York, November 1991.

[2] Pam Binns, "A Robust High-Performance Time Partitioning Algorithm: The Digital Engine Operating System (DEOS) Approach," *Digital Avionics Systems Conference*, Orlando, FL, October 2001.

[3] Pam Binns and Steve Vestal, "Message Passing in MetaH Using Precedence-Constrained Multi-Criticality Preemptive Fixed Priority Scheduling," *Life Cycle Software Engineering Conference*, Redstone Arsenal, AL, August 2000.

[4] J. Lehoczky, L. Sha and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-Time Systems Symposium*, 1989, pp 166-171.

[5] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, v20, n1, January 1973.

[6] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, 29(5), 1986.

[7] Lui Sha, John P. Lehoczky and Ragunathan Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling," *IEEE Real-Time Systems Symposium*, 1986.

[8] Peter Puschner and Alan Burns (guest editors), *Real-Time Systems Special Issue: Worst-Case Execution-Time Analysis*, v18, n2/3, May 2000.

[9] Steve Vestal, "Fixed Priority Sensitivity Analysis for Linear Compute Time Models," *IEEE Transactions on Software Engineering*, April 1994.