Optimality of (D-J)-monotonic Priority Assignment

A. Zuhily

May 9, 2006

Abstract

Release jitter of tasks may affect their schedulability. They may have more chances to miss their deadlines if their priorities are assigned according to Deadline Monotonic(DM). (D-J)-monotonic is given instead. It is proved in this report that (D-J)-monotonic is an optimal priority assignment algorithm when the deadlines of tasks are less than or equal to their periods.

Introduction

Release Jitter, RJ, of a task is the maximum deviation of successive task releases from its period. Tindell in his thesis[5] addressed the problem of RJ; it occurs when the shortest time between successive releases of a task is shorter than the normal repeating time between arrivals of a task. The following scenario is given to illustrate the problem:

> Suppose there are, on different processors, two related tasks : periodic one j and sporadic one i with same period (i.e. $T_j = T_i$). Task j calls i as soon as j has finished its execution. For some reason, j has not finished its first execution till the end of its period; while it executes at the very begining of its next period. So, i, in this case, is released twice within its period instead of once (i.e the time between the two successive releases is less than the usual minimum inter arrival time of the sporadic task i).

It is obvious that as a result of this scenario, the number of interference from task i would increase. The two popular situations of the RJ problem are: when related tasks are executing on different processors, and because of the granularity of the system timer.

It was found that neither deadline monotonic nor rate monotonic priority assignment are optimal in case of jitter[5]. So, priorities could be assigned according to the optimal priority assignment technique that depends on feasibility. Audsley[1], in his report, covered that technique, which is explained, in summary, as following. For a task set $\phi = \{\tau_1, \tau_2, ...\tau_n\}$, first, attempt to find a task τ_A that is feasible at priority level $j = n$. Next, find a feasible task at priority $j = n - 1$. Successively, feasible tasks will be found at priorities n to 1. If a feasible task, at priority level i, could not be found, no feasible priority assignment function exists. Full details can be found in Audsley's report[1].

However, Burns, Tindell and Wellings[3] mentioned that priorities should be assigned according to (D-J)-monotonic as DM is no longer optimal. No proof of the optimality of (D-J)-monotonic has been subsequently provided. This report gives a proof of the optimality of (D-J)-monotonic priority assignment, when $D \leq T$.

Notations that are Used

R_i - Worst-case response time of the i^{th} task.

C_i - Worst-case execution time of the i^{th} task.

D_i - Deadline of the i^{th} task.

T_i - period of the i^{th} task.

J_i - Release jitter of the i^{th} task.

P_i - Priority of the i^{th} task.

$(D-J)-monotonic$ Priority assignment where the task that has lower value of D-J will take higher priority. In other words

$$D_i - J_i < D_j - J_j \;\Rightarrow\; P_i > P_j.$$

n - number of tasks in the system.

Feasibility Analysis

Exact worst case response time analysis, that is introduced by Joseph and Pandya[4], is used to prove the schedubility of a system. Audsley and his colleagues [2], in 1993, improved the analysis and gave a simple way to solve these equations using a recurrence relationship.

$$R_i = C_i + (\sum_{j=1}^{j=i-1} \lceil \frac{R_i}{T_j} \rceil C_j)$$

The previous equation means that Worst Case Response Time (WCRT) of a task is the execution time of that task added to the interferences from all higher priority tasks. Where number i=1 represents the highest priority, followed by i=2, ...and so on until i=n which represents the lowest priority.

However, when tasks suffer from release jitter, the response time equation is enhanced to include release jitter. Equation (1) represents the response time formula when tasks suffer from release jitter

$$R_i = C_i + (\sum_{j=1}^{j=i-1} \lceil \frac{R_i + J_j}{T_j} \rceil C_j) \quad (1)$$

Optimality of (D-J)-monotonic

Theorem

(D-J)-monotonic priority assignment is optimal in the sense that if any task set, Q, is schedulable by priority scheme, W, is also schedulable by (D-J)-monotonic.

Proof

To prove the optimality of (D-J)-monotonic, the priorities of Q (as assigned by W) will be transformed until the ordering is (D-J)-monotonic while preserving schedulability. Let i and j be two tasks with successive priorities in Q such that under W: $P_i > P_j$ and $D_i - J_i > D_j - J_j$. Define scheme W' to be identical to W except that tasks i and j are swapped. Schedulability of all tasks that have higher priorities than i or less priorities than j are not affected by swapping the two tasks i and j. Moreover, schedulability of task j will also not be affected by the swap since it will have higher priority than before and therefore, it will suffer less interference. So, what is left is to show that task i is still schedulable under W' .

Let R_j^W be the response time of task j under W scheme, and $R_j^{W'}$ be the response time of task j under W' scheme. It can be seen that $R_j^W \leq D_j - J_j$ because j is schedulable and in the worst case may not be released until t = J_j. In addition, it is given that $D_j - J_j < D_i - J_i \leq D_i \leq T_i$. Therefore, task i only interferes once during the execution of j (under W). So, worst case response time of task j can be split, under W scheme into

$$R_j^W = C_j + C_i + \sum_{k \in S} \lceil \frac{R_j^W + J_k}{T_k} \rceil C_k. \tag{2}$$

Where, S is the set of tasks that have higher priority than i in the set under W(which is equal to the set of higher priority than j under W').

Equation(2) can be written as

$$R_j^W - C_j = C_i + \sum_{k \in S} \lceil \frac{R_j^W + J_k}{T_k} \rceil C_k. \tag{3}$$

Response time equation of the task i under W' scheme is given by

$$R_i^{W'} = C_i + \sum_{k \in hp(i)} \lceil \frac{R_i^{W'} + J_k}{T_k} \rceil C_k$$

hence,

$$R_i^{W'} = C_i + \lceil \frac{R_i^{W'} + J_j}{T_j} \rceil C_j + \sum_{k \in S} \lceil \frac{R_i^{W'} + J_k}{T_k} \rceil C_k. \quad (4)$$

Lemma 1

R_j^W is a solution of equation(4).

proof

R_j^W is going to be substituted in the right side of the equation(4).

$C_i + \lceil \frac{R_j^W + J_j}{T_j} \rceil C_j + \sum_{k \in S} \lceil \frac{R_j^W + J_k}{T_k} \rceil C_k =$

$R_j^W - C_j + \lceil \frac{R_j^W + J_j}{T_j} \rceil C_j =$

$R_j^W - C_j + C_j =$

R_j^W

That is because of the Equation(3) and because task j is schedulable under W scheme, so $R_j^W < D_j - J_j$; which means $R_j^W + J_j < D_j < T_j$. So, $\lceil \frac{R_j^W + J_j}{T_j} \rceil = 1$. □

Therefore, R_j^W is a solution of the equation of $R_i^{W'}$; which means that $R_i^{W'} \leq R_j^W$.

On the other hand, we have that $R_j^W \leq D_j - J_j$ as well as $D_j - J_j < D_i - J_i$.

Therefore,

$$R_i^{W'} < (D_i - J_i)$$

which concludes that i is schedulable after swapping.

Now, priority scheme W' can be transformed to W'' by choosing two more tasks that are in the wrong order for (D-J)-monotonic, and swapping them. Each such swap preserves schedulability. Eventually, there will be no more tasks to swap and priority ordering will be exactly as (D-J)-monotonic ordering. Hence, (D-J)-monotonic is optimal. □

Example

The following example shows how (D-J)-monotonic priority assignment improves the schedulability of a system. Suppose a system with two tasks as in Table 1. It can be seen from that table that

$Task$	C	J	D	T	$D-J$
τ_1	6	3	13	14	10
τ_2	3	12	20	25	8

Table 1: Tasks Description's

assigning priorities according to DM leads to τ_1 having higher priority than τ_2 and τ_2 misses its first deadline. Figure 1 illustrates that situation.

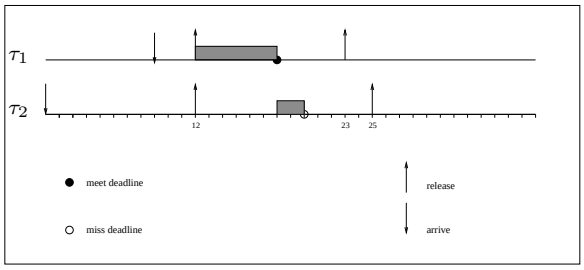

Figure 1: execution of τ_1 and τ_2 according to deadline monotonic priority assignment

On the other hand, τ_1 and τ_2 will meet their deadlines, if (D-J)-monotonic priority assignment is considered; where τ_2 in that case will have higher priority than τ_1. Figure 2 illustrates how both tasks become schedulable in the case of (D-J)-monotonic priority assignment is considered.

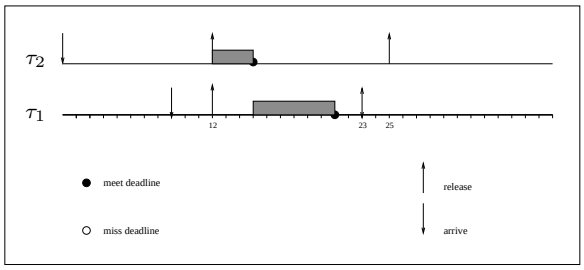

Figure 2: execution of τ_1 and τ_2 according to (D-J)-monotonic priority assignment

Applying Equation(1) calculates WCRT of tasks τ_1 and τ_2, considering (D-J)-monotonic priority assignment. Therefore, WCRT of τ_2 is:

$R_{\tau_2} = 3$.

While WCRT of τ_1 is

$R_{\tau_1} = 6 + \lceil \frac{6+12}{25} \rceil 3 = 9$.

$R_{\tau_1} = 6 + \lceil \frac{9+12}{25} \rceil 3 = 9$.

So, $R_{\tau_1} = 9$ which means τ_1 is schedulable.

Conclusion

Deadline monotonic priority assignment is not optimal when tasks suffer release jitter. Priorities can be assigned according to (D-J)-monotonic instead of DM. Proof of optimality of (D-J)-monotonic is given in this report. The proof is given when deadlines of tasks are less than or equal to their periods. So, schedulability of a system is improved when (D-J)-monotonic priority assignment is employed.

Bibliography

[1] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start time. Technical Report YCS 164, University of York, 1991.

[2] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying New Scheduling Theory to Static Priority Preemptive Scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

[3] A. Burns, K. Tindell, and A. J. Wellings. Effictive analysis for engineering real-time fixed priority schedulers. *IEEE Transactions On Software Engineering*, 21(5):475–480, 1995.

[4] M. Joseph and P. Pandya. Finding Response Times in a Real-Time system. *The Computer Journal*, 29(5):390–395, October 1986.

[5] K.W. Tindell. *Fixed Priority Scheduling Of Hard Real-Time Systems*. PhD thesis, University of York, 1993.