Fast Schedulability Test for Fixed Priority Discipline

José M. Urriza, Ricardo Cayssials and Javier D. Orozco
Departamento de Ingeniería Eléctrica y de Computadoras
Universidad Nacional del Sur - CONICET
8000 Bahía Blanca, Argentina
{jurriza, iecayss, ieorozco}@criba.edu.ar

***Abstract*:** The schedulability test is a necessary analysis that allows designers to guarantee that a real-time system will meet its deadlines during runtime. Since 1973, some methods, based on bounds of necessary conditions, have been used, such as the Liu and Layland's bound, the Bini's bound in 2001, and exact conditions as the one proposed by Joseph in 1986, extended by other authors. However, the exact analysis requires a high computational cost, which is much higher when non-unitary execution times as well as fractional execution times are considered. In this paper, we present an exact schedulability test with a low computational cost that can be implemented online in embedded systems.

Keywords: Schedulability analysis, Real-Time systems

1. Introduction

In real-time systems it is necessary to guarantee that results will be produced on time. In [1], Stankovick *et al.* proposed the most used definition of a real-time system: *"In Real-Time computing the correctness of the system depends not only on the logical results of the computation but also on the time at which the results are produced"*.

The system, from a real-time point of view, is modelled as a set $\mathbf{\Gamma}$ of n periodic tasks to be executed on a processor. Each task i is characterised by its period, T_i, its deadline, D_i, and its execution time, C_i. Tasks are invoked periodically and the processor must be assigned C_i time units to task i in order to complete it.

Schedulability analyses allow designers to assure that a real-time system will meet its temporal constraints during runtime. If the schedulability test of a system is successful, then it is said that the system is *schedulable*. Otherwise, it is said *non-schedulable*.

The total utilization factor of a real-time system, denoted UF, determines how much the processor is utilized to execute the real-time tasks of the system. It is defines as:

$$UF = \sum_{i=1}^{n} \frac{C_i}{T_i} .$$

In 1973, Liu and Layland [2] found an upper bound condition for the total utilization factor to assure the schedulability of a real-time system. This upper bound presents a very low complexity but it is very restrictive in the sense that it does not cover a great deal of schedulable real-time systems.

In [3], Joseph *et al.* presented a necessary and sufficient schedulability test based on the worst case of response of a real-time task. Because it is an exact condition, we can determine the schedulability of every real-time system, but its computational cost is high because it has to be calculated in an iterative way. Joseph's method complexity does not allow it to be applied online. Several authors extended the results of Joseph and proposed alternative algorithms to determine the schedulability of a real-time system ([4, 5, 6]).

In 2001, Bini *et al.* [7] proposed a sufficient upper bound that improves the Liu and Layland's upper bound. The complexity is low, but it does not cover all the schedulable real-time systems.

Currently, dynamically reconfigurable real-time systems, sensor networks and mobile computing, need to determine the schedulability of the system during runtime. Consequently, new schedulability methods are required to determine the exact schedulability of a real-time system with a low computational cost, in order to apply them online with a low system overhead. Upper bounds present low complexity but they are very restrictive on determining the schedulability of a system, and consequently, the efficiency is very poor. On the other hand, current exact conditions are very precise, but their computational cost does not allow them to be applied online.

2. Contributions of this paper

In this paper we present an exact schedulability test with a very low computational cost. It can be applied online in embedded systems that require determining the schedulability of the system during runtime. We compare our schedulability analysis with the most efficient schedulability method used currently in real-time literature [3].

The rest of the paper is organised as follows: Section 3 describes the real-time process model used in this paper and the real-time schedulability analysis proposed by Joseph. In Section 4, a set of lemmas is proposed to analyse the schedulability of a real-time system based on determining the slack that a task can tolerate. A low computational cost algorithm based on lemmas of Section 3 is proposed in Section 5. In Section 6, the algorithm is compared with the one proposed in [3]. Conclusions are drawn in Section 7.

3. Real-Time analysis

Real-time systems theory allows analyzing the temporal properties of a set of concurrent tasks. The general process model of a real-time system consists of a set $\mathbf{\Gamma}$, of *n* periodic and non-periodic tasks. Each task is characterised by either its period in case of periodic tasks, or its minimum interarrival time for non-periodic ones, T_i, deadline, D_i, worst-case execution time, C_i.

$\mathbf{\Gamma}=\{ \text{task } i = (T_i, D_i, C_i)\}$

Each time that a task requires the processor to be executed, it is said that the task is either *invoked,* or an *invocation takes place*. When a task invocation is completed, it will not require to be executed until the next invocation.

Only one task can run on a processor at the same time. The selection of the task is done by the *scheduler* that applies a priority discipline among the tasks of the systems that are requiring to be executed. The scheduler is one of the main tasks of a real-time operating system (*RTOS*). The fixed priority (*FP*) discipline is one of the most important in real-time, and most of *RTOS* implemented it. In a *FP* discipline each task is assigned with a priority in the design time and it remains fixed during runtime. The schedulability analysis proposed in this paper is valid when a *FP* discipline is implemented.

A necessary and sufficient scheduling analysis conditions exist to guarantee that the temporal requirements will be satisfied in a real-time system. The most advanced technique computes the worst-case response time of any task. This is done by building the critical instance, named the *worst-case of load*, as the time instant when a task will suffer the maximum interference from higher priority tasks. This actually happens when those higher priority tasks are released simultaneously. The worst-case response time, denoted R_i, of a task *i* can be computed by finding the smallest $R_i > 0$ that is a solution to the following fix point equation:

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

This can be solved by forming a recurrence with $R_i^0 = 0$ and stopping when $R_i^n = R_i^{n+1}$ or when $R_i^n > D_i$. The computational cost of this method is high and consequently it is not suitable to be implemented online.

In this paper, we propose a schedulability test that is not based on calculating the worst response time of each task, and consequently, its computational cost is dramatically reduced.

4. A low cost schedulability analysis

In this section we propose a schedulability analysis based on Slack Stealing techniques. The maximum slack is defined as the maximum delay that a real-time task can withstand without missing its deadline.

In [8], it was proved that a task *i* that has supported its maximum slack is completed, either just before its deadline or just before the release of a higher priority task. The set of values in which we can get the maximum slack of a task *i* at time *t* is named *inspection set*, and denoted $\mathbf{I}_i(t)$. The set $\mathbf{I}_i(t)$ covers an interval in which the maximum slack takes place. We prove that a real-time task is schedulable when it can support a slack greater than or equal to cero. Consequently, we can find the maximum slack by calculating on a reduced set of values, instead of recursively searching for the worst response time of the task. This reduces the computational cost of the schedulability test.

The maximum slack that task *i* can tolerate at the critical instant is ([8]):

$$k_i(t^*) = t^* - \sum_{j=1}^{i} C_j \left\lceil \frac{t^*}{T_j} \right\rceil \qquad \forall t^* \in \mathbf{I}_i(t) \qquad (2)$$

Applying Equation (2) we can get the maximum slack that a task *i* can support and consequently determine whether the task *i* is schedulable or not. The Equation (2) does not require a recursive algorithm to be solved but an evaluation over the set of values of $\mathbf{I}_i(t)$.

In next section we reduce both the interval that the set $\mathbf{I}_i(t)$ covers and the number of values in which Equation (2) should be evaluated.

4.1. Reducing the number of evaluation points

In this section we reduce the number of values in which the Equation (2) should be evaluated and we prove that finding a slack greater than or equal to zero, guarantees that the task is schedulable.

The inspection set contains values that can be discarded previously to evaluate Equation (2), in order to reduce the complexity of the calculus. To do this, Lemma 1 proves that a task that has been delayed its maximum slack finishes within the interval $(D_i/2, D_i]$, and consequently, we reduce 50% the number of values of the set $\mathbf{I}_i(t)$ in which Equation (2) should be evaluated.

Lemma 1:

Being $\mathbf{\Gamma}$ a schedulable real-time system, then the completion time of a task *i*, that supports the maximum slack after a critical instant, denoted V_i, belongs to the subinterval $[(D_i + C_i)/2, D_i]$.

Proof:

The minimum completion time of a task *i* occurs when the maximum slack that task *i* can support, is equal to zero ($S_i(0) = 0$). In such a case, we can assure that the completion time of task *i* is greater than or equal to $R_{i-1}+C_i$, where R_{i-1} is the worst response time for task *i*-1. Hence, the minimum completion time V_i ($\wedge V_i$) occurs when:

$$\wedge V_i = R_{i-1} + C_i \tag{3}$$

On the other hand, if task *i* supports its maximum slack and finishes at time $\wedge V_i$, then the interval $(\wedge V_i, D_i]$ is completely used to execute the tasks of the subsystem $\mathbf{\Gamma}(i\text{-}1)$, and its length is less than or equal to R_{i-1} (otherwise subsystem $\mathbf{\Gamma}(i\text{-}1)$ would leave an empty interval that could be used to introduce more slack to task *i*):

$$R_{i-1} \geq D_i - \wedge V_i \tag{4}$$

From Equations (3) and (4), we can assure that the minimum completion time for task *i* is given for:

$D_i - (\wedge V_i) \leq R_{i-1} \Rightarrow D_i \leq 2(\wedge V_i) - C_i$

and

$\wedge V_i \geq (D_i + C_i)/2$

then

$V_i \geq (D_i + C_i)/2$ □

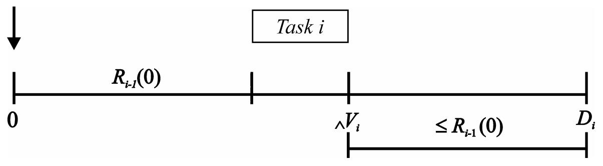

Fig. 1. If maximum slack of task *i* is equal to 0 then $D_i - \wedge V_i \leq R_{i-1}$.

The next lemma proves that any slack greater than or equal to zero within the interval $[(D_i + C_i)/2, D_i]$, guarantees that the system is schedulable under a *FP* discipline.

Lemma 2:

If there exist $t^* \in \mathbf{I}_i$ and $t^* \cup [(D_i + C_i)/2, D_i]$ such that the slack of task *i* ($k_i(t^*)$) is greater than or equal to 0, then task *i* is FP-schedulable, otherwise it is non-schedulable.

Proof:

For Lemma 1, the completion time with the maximum slack is within the interval $[(D_i + C_i)/2, D_i]$. If $k_i(t^*) \geq 0$ then there was enough time to execute completely the task *i*. Otherwise, the system is oversaturated, and consequently, it is non-schedulable under a FP discipline.□

From Lemma 1 and Lemma 2 we can assure that, when a task supports its maximum slack, its complexion time will be closer to its deadline. Under this premise, we can deduce that values $t^* \in \mathbf{I}_i(t)$ closer to D_i, will have more chances to satisfy Equation (2). In the next section, we propose a low computational cost algorithm based on this premise which is useful to determine the schedulability of a real-time system.

5. Schedulability Algorithm

In this section we propose an algorithm to determine whether a task can support a slack greater than or equal to zero, in order to determine its schedulability. This algorithm avoids calculating the worst response time of the task, and, as a result, its computational cost is reduced.

Previous section proved that values of $\mathbf{I}_i(t)$ that satisfy Equation (2) are closer to D_i. Therefore, in order to increase the chances to find a value that satisfies Equation (2), it should be evaluated backwards for each value of $\mathbf{I}_i(t)$, until the first slack greater than or equal to zero is found. If no value is found, then, by Lemma 2, the system is non-schedulable.

The next example shows how this method works. Also, it is compared with the one proposed by Joseph.

Example:

Let's propose the real-time system on table 1.

Task	T_i	C_i	D_i
1	3	1	3
2	4	1	4
3	6	1	6

Table 1.

Figure 2 shows how tasks are executed under the worst case of load.

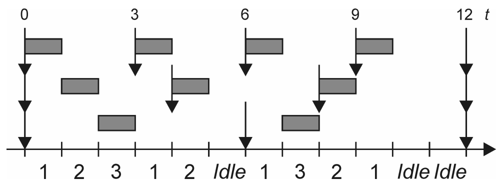

Fig. 2.

Table 2 shows the iterative calculus of Joseph method. It needed 4 iterations to determine the schedulability of the system.

Task	R_i^n	$R_i^{n+1} = C_i + \sum_{j=1}^{i-1} C_j \left\lceil \frac{R_i^n}{T_j} \right\rceil$
2	1	$1 + 1.\left\lceil 1/3 \right\rceil = 2$
2	2	$1 + 1.\left\lceil 2/3 \right\rceil = 2$
3	2	$1 + 1.\left\lceil 2/3 \right\rceil + 1.\left\lceil 2/4 \right\rceil = 3$
3	3	$1 + 1.\left\lceil 3/3 \right\rceil + 1.\left\lceil 3/4 \right\rceil = 3$

Table 2. Joseph's Method

Table 3 shows the schedulability test using the method proposed. It needed 2 iterations to determine the schedulability of the system.

Task	***Interval***	***t****	$k_i = t^* - \sum_{j=1}^{i} C_j \left\lceil \frac{t^*}{T_j} \right\rceil$
2	[2, 4]	4, 3	$4 - \left(1 * \left\lceil \frac{4}{3} \right\rceil + 1 * \left\lceil \frac{4}{4} \right\rceil\right) = 1 \geq 0$
3	[3, 6]	6, 4, 3	$6 - \left(1 * \left\lceil \frac{6}{3} \right\rceil + 1 * \left\lceil \frac{6}{4} \right\rceil + 1 * \left\lceil \frac{6}{6} \right\rceil\right) = 1 \geq 0$

Table 3. Fast Algorithm Method.

It can be noted that the number of iterations may be dramatically reduced and consequently the computational cost is much lower.

5.1. Complexity of the method

The complexity of the method is the cardinality of the set I that can be expressed as:

$$|\mathbf{I}_i| = \sum_{j=1}^{i-1}\left(\left\lceil \frac{d_{i,tc}}{T_j} \right\rceil - \left\lceil \frac{(D_i + C_i)/2}{T_j} \right\rceil\right) + 1$$
$$|\mathbf{I}| = \sum_{z=1}^{i} |\mathbf{I}_z| \tag{5}$$

Equation (5) allows us to get the complexity of the calculus in advance. As a result, the method can be applied during runtime just when the complexity and the load of the system allow it. The worst case occurs when system is non-schedulable, and therefore, the complexity is equal to:

$$O\left(\frac{D_n}{2.T_1} n^2\right). \tag{6}$$

6. Experimental Results

In this section we compare our schedulability method with the one proposed by Joseph in [3]. We consider real-time systems with 10, 20 and 50 tasks. Real-time systems were randomly generated with periods ranged from 25 to 10000 slots, and total utilization factors ranged from 70% to 90% in steps of 5% and for 93%, 95% - 99%. The tasks' deadlines were set equal to tasks' periods. We simulated 10000 real-time systems for each total utilization factor and we recorded the average computational cost.

Fig. 3. Comparison of the number of calculus required for both methods for systems with 50 tasks.

Figures 3 and 4, show the results obtained. We can note that our method presents a lower computational cost than the one proposed by Joseph. When utilization factor is low, our method finds a slack greater than cero in the first values of $\mathbf{I}_i(t)$. When utilization factor increases, both methods need to iterate over a 503 Tm(h)-5.9(9u)0.3n3[(. 56.43TD8 value with slack equal to or greater than cero in the first attempts of the inspection set. When utilisation factor increases, then both methods have to try more values in order to find values that determine that the system is schedulable. The worst-case for both methods occurs when system is non-schedulable and all values should be discarded in order to assure it.

7. Conclusions

In this paper we present a schedulability test with a low computational cost. The performance was compared with the most traditional method in the real-time literature. We show that the computational cost of our method is much lower than the one proposed by Joseph. Consequently, this schedulability test can be used in embedded real-time systems in which real-time parameters change during runtime, execute aperiodic tasks, dynamic task assignment in distributed real-time systems, and fault tolerance mechanisms.

References

[1] J. A. Stankovic, "Misconceptions About Real-Time Computing: A Serius Problem for Next-Generations Systems," *IEEE Computer*, vol. Octubre, pp. 10-19, 1988.

[2] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, pp. 46-61, 1973.

[3] M. Joseph and P. Pandya, "Finding Response Times in Real-Time System," *The Computer Journal (British Computer Society)*, vol. 29, pp. 390-395, 1986.

[4] J. P. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," Department of Statistics, Carnegie-Mellon, Pitsburg, USA, Internal Report 1987.

[5] N. C. Audsley, A. Burns, M. F. Richarson, and A. J. Wellings, "Hard Real-Time Scheduling: The Deadline Monotonic Approach," presented at Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, GA, USA 1991.

[6] J. Santos, M. L. Gastaminza, J. D. Orozco, D. Picardi, and O. Alimenti, "Priorities and Protocols in Hard Real-Time LANs," *Computer Communications*, vol. 14, pp. 507-514, 1991.

[7] E. Bini, G. Buttazzo, and G. Buttazzo, "A Hyperbolic Bound for the Rate Monotonic Algorithm," *IEEE Transactions on Computer*, vol. 52, pp. 933-942, 2003.

[8] J. M. Urriza, J. D. Orozco, and R. Cayssials, "Fast Slack Stealing methods for Embedded Real Time Systems," presented at 26th IEEE International Real-Time Systems Symposium (RTSS 2005) - Work In Progress Session, Miami, EEUU, 2005.