Test Rápido de Planificabilidad para R.M. o D.M.

José M. Urriza, Ricardo Cayssials y Javier D. Orozco
Departamento de Ingeniería Eléctrica y 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 designer to guarantee that a real-time system will meet their deadlines during runtime. Since 1973 there have been several methods based on bounds that are necessary conditions as the Liu and Layland bound and the Bini bound in 2001, and exact conditions such as the proposed by Joseph and extended by other authors. However, the exact analysis requires a high computational cost. In this paper, we present an exact schedulability test with a low computational cost that can be implemented online in embedded systems.

Keywords: Schedulability Test, RM, DM

Resumen

El test de planificabilidad es la herramienta necesaria para garantizar que un sistema de tiempo-real pueda cumplir con sus restricciones temporales. Desde 1973, se han desarrollando diversos métodos basados en cotas que garanticen dicha condición necesaria, como la de Liu y Layland, la cota de Bini en el 2001 y análisis exactos, como el desarrollado por Joseph y extendido por otros autores. Pero este análisis exacto conlleva una carga computacional elevada. Este trabajo presenta un test exacto, con una carga de computacional baja que puede ser utilizado en tiempo de corrida en sistemas embebidos.

Palabras Claves: Test planificabilidad, RM, DM

1 INTRODUCCIÓN A LOS SISTEMAS DE TIEMPO REAL

Los sistemas de tiempo real (*STR*) se pueden encontrar en casi todas las ramas de las ingenierías, pero en los últimos años se han extendido a la producción masiva. Los sistemas de control, y comunicaciones son ramas en las cuales más se han desarrollado.

Se puede definir informalmente a un *STR* como aquél que necesita terminar una tarea o trabajo, antes de un determinado tiempo. Estos se caracterizan por poseer entre sus parámetros el tiempo y, conjuntamente con las demás, determinan el resultado o comportamiento del sistema. Formalmente es aceptada la definición de Stankovic[1] que dice que: *en los STR los resultados no sólo deben ser correctos aritmética y lógicamente sino que además, deben producirse antes de un tiempo determinado denominado vencimiento.*

Los *STR* dependiendo del vencimiento de la tarea, se pueden clasificar en tres tipos. El primero no permite que ninguna tarea pierda su vencimiento, por lo cual se los denominan *duros* o *críticos* (*hard*). El segundo permite que se pierda algunos vencimientos, por lo cual se los denominan *blandos* (*soft*). Por último, en la actualidad han aparecido sistemas que permiten sólo una determinada cantidad de pérdidas y a estos se los denominan *firmes* (*firm*), por ejemplo, permiten que al menos 9 de 10 alcancen su vencimiento.

En los *STR duros*, la pérdida de un vencimiento en una tarea puede tener consecuencias adversas. En los sistemas en los cuales la vida humana es participe, por ejemplo la aviónica, se le debe garantizar que todas las tareas terminen antes de su vencimiento mediante un *test de diagramabilidad* o *planificabilidad*. Si el test es exitoso, a estos sistemas se los denomina *diagramables* o *planificables* y se dice que han cumplido con todas sus *constricciones de tiempo*.

En 1973, C. L. Liu y James W. Layland [2] realizan el primer aporte significativo para determinar la diagramabilidad de un *STR-duro*. Este trabajo se ha convertido en un trabajo liminar, el cual ha sido la base para todos los trabajos posteriores. Esto se debe a que formalizan el marco de trabajo necesario para garantizar la diagramabilidad de un conjunto de tareas. En [2], se considera al sistema mono-recurso y multitarea. Se entiende como recurso por ejemplo a un único microprocesador o medio físico de transmisión, el cual sólo puede ser utilizado por una tarea a la vez.

Existen dos formas de planificar las tareas al recurso. La primera forma es predeterminada con anterioridad (estática) y la segunda, que es la más utilizada, se basa en asignar una prioridad a cada tarea.

Una *disciplina de prioridades* es una regla implementada en un algoritmo por el cual, el *diagramador* o *planificador*, determina qué tarea (τ_i) tiene prioridad de ejecución en el recurso. El conjunto de tareas se las consideran periódicas, independientes y apropiables. Una tarea periódica, es aquella que después de un determinado tiempo solicita nuevamente ejecución. La tarea se dice que es independiente cuando no necesita el resultado de la ejecución de alguna otra tarea, para su propia ejecución. Finalmente se dice que una tarea es apropiable cuando el *planificador*, puede suspender su ejecución y desalojarla del recurso en cualquier momento.

Generalmente los parámetros de cada tarea, bajo este marco de trabajo, son: su tiempo de ejecución, que se nota como C_i, su período T_i y su vencimiento D_i. Por lo tanto un conjunto $\mathbf{S}(n)$ de n tareas se encuentra especificado por $\mathbf{S}(n)=\{(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)\}$.

Como las tareas son periódicas, el esquema de generación se repite después de un determinado tiempo. Este tiempo se denomina *hiperperíodo* y es el mínimo común múltiplo (*MCM*) de los períodos de las tareas.

En [2] se demostró que el peor esquema de generación, para un mono-recurso, es aquél en el cual todas las tareas solicitan ser ejecutadas en el mismo tiempo y se denomina *instante crítico* o *peor estado de carga*. Se demostró también que, si este estado es planificable, el *STR* es planificable para cualquier otro estado, bajo la disciplina de prioridades utilizada.

El factor de utilización (*FU*) de un conjunto de tareas *duras* $\mathbf{S}(n)$ determina el nivel de utilización del recurso. Este se puede calcular con la siguiente fórmula.

$$FU = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{1}$$

Si el $FU < 1$ se dice que el sistema es *no-saturado*, por lo tanto el recurso tendrá tiempos ociosos en los cuales ninguna tarea requiera ser ejecutada, estos tiempos se denominan *slack* y se podrán utilizar para otros tipos de requerimientos.

Sí $FU = 1$ entonces el sistema se dice *saturado* y no posee tiempos libres.

Sí $FU > 1$ se requiere más tiempo de ejecución que el que se dispone, por lo cual un conjunto de tareas $\mathbf{S}(n)$ con esas características, no es planificable por ninguna *disciplina de prioridades*.

Existen en la actualidad tres *disciplinas de prioridades* básicas para la diagramación de tareas en tiempo real. Estas pueden ser: prioridades fijas, prioridades dinámicas o una combinación de las anteriores. A su vez, las mismas se

diferencian en qué regla implementa el algoritmo diagramador. En prioridades fijas las reglas más utilizadas son Rate Monotonic [2] (*RM*) y Deadline Monotonic [3] (*DM*), en prioridades dinámicas, la más utilizada es Earliest Deadline First [2] (*EDF*) y, por último, una combinación de prioridades fijas y dinámicas es el método Dual Priority [4].

Las condiciones de planificabilidad propuestas en la literatura de tiempo real son complejas y no permiten ser utilizadas en tiempo de corrida debido a la sobrecarga que producen al sistema. Condiciones necesarias han sido propuestas para determinar la planificabilidad de un sistema pero su rigurosidad no cubre muchos sistemas de tiempo real que son planificables. En este trabajo se propone un test de planificabilidad exacto de baja complejidad que puede ser implementado en tiempo de corrida. Se compara la complejidad del test con la de los métodos tradicionales propuestos en la literatura de tiempo real.

El trabajo se organiza de la siguiente manera: en la sección 2 se presenta los trabajos previos. En la sección 3 se presenta el test de planificabilidad que garantiza que el sistema de tiempo real cumplirá con sus restricciones temporales. La sección 4 presenta una heurística para disminuir la complejidad en el test de planificabilidad. En la sección 5 se exponen los resultados experimentales y finalmente en la sección 6 se realizan las conclusiones.

2 TRABAJOS PREVIOS

En [2], Liu presenta una cota que por 13 años fue la única técnica para determinar si un sistema diagramado por *RM* cumplía con sus constricciones de tiempo. Esta cota impone un límite de aproximadamente el 70% (*ln* 2=0,69315) al factor de utilización del sistema cuando el número de tareas tiende a infinito. Esta cota es de simple utilización, dado que si un conjunto de tareas **S**(*n*) con un *FU* es menor a *ln* 2, el sistema es planificable por la disciplina de prioridades *RM*. A continuación se presenta en detalle el cálculo de la cota de Liu:

$$FU \leq n.(2^{\frac{1}{n}} - 1) \tag{2}$$

La cota es una condición necesaria pero no suficiente, por lo cual sistemas de tiempo real con un factor de utilización mayor a *ln* 2 pueden ser planificables por *RM*. Por ejemplo: el sistema con periodos 2, 3, 6 y tiempos de ejecución unitarios, es planificable y es *saturado* (*FU*=1).

En 1986, Joseph y Pandya [5] publicaron un trabajo en el cual se definen las condiciones necesarias y suficientes para que un *STR* pueda ser planificable por *RM*. El método propuesto calculaba, partiendo del instante crítico, el peor tiempo de respuesta de cada tarea. Se probó también que no existía una solución analítica de este tipo de problemas y solo era posible calcularlo de manera iterativa. La solución de la ecuación (3) es válida si $R_i^+ = R_i$ y además es $R_i \leq D_i$.

$$R_i^+ = C_i + \sum_{j=1}^{i-1} C_j \left\lceil \frac{R_i}{T_j} \right\rceil \tag{3}$$

Diversos trabajos han sido publicados con soluciones similares. En 1987, Lehoczky produce un reporte interno [6], que luego en 1989 es publicado en el *RTSS* de *IEEE* [7]. En 1991 y en 1993, Santos [8, 9] publica dos trabajos de diagramabilidad de *RM* para redes en tiempo-real. También en 1993, Audsley [10] publica un trabajo en el cual, además de dar un test de diagramabilidad similar al de Joseph, somete al *STR* a diversas problemáticas de implementación (eg. jitter, bloqueos, precedencias, etc). Se hace notar que en este trabajo determinan que dependiendo del ordenamiento por prioridades de las tareas (*RM* o *DM*), el test de diagramabilidad desarrollado por Joseph sirve para las dos disciplinas.

En 1982, Leung [3], definió *DM*, pero no se presentó un test de diagramabilidad hasta que fue presentado por Audsley ([11]) en 1991.

En 2001, Bini [12] define una cota hiperbólica que mejora un poco la cota de Liu. Esta se calcula de la siguiente manera:

$$\prod_{i=1}^{n} \left(\frac{C_i}{T_i} + 1 \right) \leq 2 \tag{4}$$

Como se puede observar para la cota de Liu [2] y la cota de Bini [12], se necesita calcular el factor de utilización del conjunto de tareas y la productoria respectivamente. Estos cálculos poseen un orden O(*n*). También para la cota de Bini el sistema con periodos 2, 3 y 6 y tiempos de ejecución unitarios no cumple con la cota.

3 TEST DE PLANIFICABILIDAD

En los trabajos previos [5-8, 10, 11], donde se calcula el test exacto, además de garantizar la planificabilidad del *STR*, se obtiene el peor tiempo de respuesta de la tarea para el momento crítico. Para obtenerlo, se necesita que de manera iterativa se busque la solución del sistema. Esta búsqueda convierte a los tests en métodos que poseen una alta carga computacional. Una solución que no busque el peor tiempo de respuesta, pero sí garantice la planificabilidad del sistema tendría una menor carga.

En [13] se presenta un método de *Slack Stealing*, en el cual se determinan en qué tiempos es posible encontrar la solución de las ecuaciones temporales ahí presentadas. Mediante un lema se demuestra que una tarea τ_i, que ha sido retrasada al máximo su ejecución, sólo puede terminar justo antes de que una tarea de prioridad mayor demande ser atendida o antes de que la misma se venza. Por lo tanto se busca el punto donde el Slack es máximo, sin necesidad de estar buscando de manera iterativa cuál es el máximo slack. El conjunto de puntos, donde es posible encontrar la solución se lo denomina $\mathbb{V}_i$. Se presenta a continuación la fórmula del Slack de forma completa y para cuándo es calculado desde $t = 0$.

$$k_i(t^*) = t^* - t - \sum_{j=1}^{i} \left[C_j \left(\left\lceil \frac{t^*}{T_j} \right\rceil - \left\lfloor \frac{t}{T_j} \right\rfloor \right) - c_i(t) \right], \forall t^* \in \mathbb{V}_i$$

$$para \quad t = 0 \tag{5}$$

$$k_i(t^*) = t^* - \sum_{j=1}^{i} C_j \left\lceil \frac{t^*}{T_j} \right\rceil, \forall t^* \in \mathbb{V}_i$$

Con algunos cambios se puede adaptar este método para realizar solamente un test de planificabilidad.

Los puntos del conjunto $\mathbb{V}_i$ deben satisfacer el lema del trabajo en [13]. Partiendo del peor estado de carga, el intervalo donde se encuentran los puntos de $\mathbb{V}_i$ y donde es posible encontrar un slack ≥ 0 es:

$$[A, D_i] \text{ con } \quad A = \sum_{j=1}^{i} C_j$$

A continuación se presenta el siguiente lema considerando que el sistema **S**(*i-1*) tareas es planificable:

Lema 1:

Sí dentro del intervalo $[A, D_i]$ en los tiempos donde las tareas de mayor prioridad se instancia, o en el vencimiento de la tarea τ_i, el slack para alguno de estos tiempo es ≥ 0, entonces el sistema es planificable por *RM* o *DM* Por el contrario, si todos esos tiempos tienen un *slack* < 0 el sistema es no planificable.

Prueba:

El intervalo de búsqueda está limitado por *A*, y el vencimiento de la tarea τ_i (D_i). Si el *slack* ≥ 0, quiere decir que ha cumplido con todas las constricciones de tiempo del subsistema **S**(*i*). Para el caso contrario, que el sistema no tenga en el intervalo ningún punto con *slack* ≥ 0, el sistema no es planificable, debido a que la ejecución del sistema **S**(*i*) en el intervalo [0, D_i] es sobresaturado. □

El *Lema 1* permite disminuir el intervalo de inspección y consecuentemente reduce la complejidad del cálculo de la planificabilidad.

4 HEURÍSTICA

En [8] se probó que un sistema de tiempo real **S**(*n*) partiendo del peor estado de carga, si posee tiempos ociosos, estos se encontrarán sobre el final del intervalo [0, D_n]. Esto, sumado al *Lema 1* permite construir una heurística de búsqueda de donde es posible que el slack sea mayor o igual a cero. Por lo tanto, la búsqueda se inicia de atrás hacia adelante y sólo es necesario para probar que el sistema es planificable el primer *slack* ≥ 0. La forma en que se realiza es calculando

con la ecuación (5), en primer lugar en el vencimiento de la tarea τ_i y luego en cada invocación de las tareas de mayor prioridad, buscando algún punto con *slack* ≥ 0. A continuación se muestra cómo funciona la heurística comparándola con el método de Joseph en un ejemplo.

Ejemplo:

Sea el siguiente sistema de tiempo real

Tarea τ_i	T_i	C_i	D_i
1	3	1	3
2	4	1	4
3	6	1	6

Tabla 1.

En la figura 1 vemos la ejecución de las tareas en el hiperperíodo.

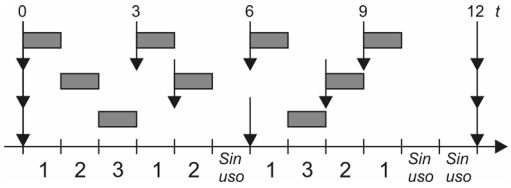

Figura 1.

Como se muestra en la Figura 1, el *STR* es planificable por la disciplina de prioridades *RM*. A continuación se calcula de forma iterativa la planificabilidad por el método de Joseph para dos y tres tareas:

Tarea τ_i	R_i	$R_i^+ = C_i + \sum_{j=1}^{i-1} C_j \left\lceil \frac{R_i}{T_j} \right\rceil$ (3)
2	1	$1+1.\left\lceil \frac{1}{3} \right\rceil = 2$
2	2	$1+1.\left\lceil \frac{2}{3} \right\rceil = 2$
3	2	$1+1.\left\lceil \frac{2}{3} \right\rceil + 1.\left\lceil \frac{2}{4} \right\rceil = 3$
3	3	$1+1.\left\lceil \frac{3}{3} \right\rceil + 1.\left\lceil \frac{3}{4} \right\rceil = 3$

Tabla 2

Ahora se calcula el slack con el valor de *t** en el arribo de las tareas de mayor prioridad o en el vencimiento.

Tarea τ_i	*Intervalo*	*t**	$k_i = t^* - \sum_{j=1}^{i} C_j \left\lceil \frac{t^*}{T_j} \right\rceil$ (5)
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	$6 - \left(1*\left\lceil \frac{6}{3} \right\rceil + 1*\left\lceil \frac{6}{4} \right\rceil + 1*\left\lceil \frac{6}{6} \right\rceil \right) = 1 \geq 0$

Tabla 3

Se puede observar que existe una disminución de las iteraciones necesarias para probar que el sistema es planificable. Es esta ventaja que se quiere aprovechar para bajar el costo computacional.

4.1 Complejidad

La complejidad del cálculo, para el caso que el sistema sea no planificable, se puede saber de antemano, dado que es la cardinalidad del conjunto de puntos $\mathbb{V}$.

$$|\mathbb{V}_i| = \sum_{j=1}^{i-1}\left(\left\lceil \frac{D_i}{T_j} \right\rceil - \left\lceil \frac{A}{T_j} \right\rceil\right) + 1$$
$$|\mathbb{V}| = \sum_{z=1}^{i} |\mathbb{V}_z| \qquad (6)$$

5 RESULTADOS EXPERIMENTALES

En esta sección se compara al método desarrollado en [5] con la heurística propuesta. Para esto se generaron tres grupos de 10, 20 y 50 tareas, de manera similar a las generadas en [14]. En los tres grupos los períodos de las tareas se dividieron en tres subgrupos que van de 25 a 100, de 100 a 1000 y de 1000 a 10000 unidades de tiempo.

El primer grupo está compuesto por 4 tareas del primer subgrupo, 3 del segundo y 3 del tercero. El segundo grupo está compuesto por 7 tareas del primer subgrupo, 7 del segundo y 6 del tercero. El último grupo está compuesto por 17 tareas del primer subgrupo, 17 del segundo y 16 del tercero. A su vez los factores de utilización fueron del 70% al 95% en incrementos de 5%. La precisión es de ± 0,5% por factor de utilización.

El período de las tareas fue elegido aleatoriamente con una distribución exponencial. El vencimiento de las tareas se adoptó igual que el período, por ser el peor caso que se puede adoptar. El tiempo de ejecución de cada tarea se eligió aleatoriamente y fue entero.

Se simularon más de 10000 sistemas por factor de utilización y se registró la carga computacional promedio. La forma en que se realizó fue la siguiente: si el subsistema de 2 tareas era planificable se multiplicaba por la cantidad de iteraciones que requirió para comprobarlo y se pasaba al subsistema de 3 tareas y así sucesivamente de manera que se sumó el número de iteraciones requerido por número de tareas, hasta que resultase planificable o no. Luego se obtuvo el promedio del costo de todos los sistemas simulados. La Figura 2 muestra los resultados obtenidos.

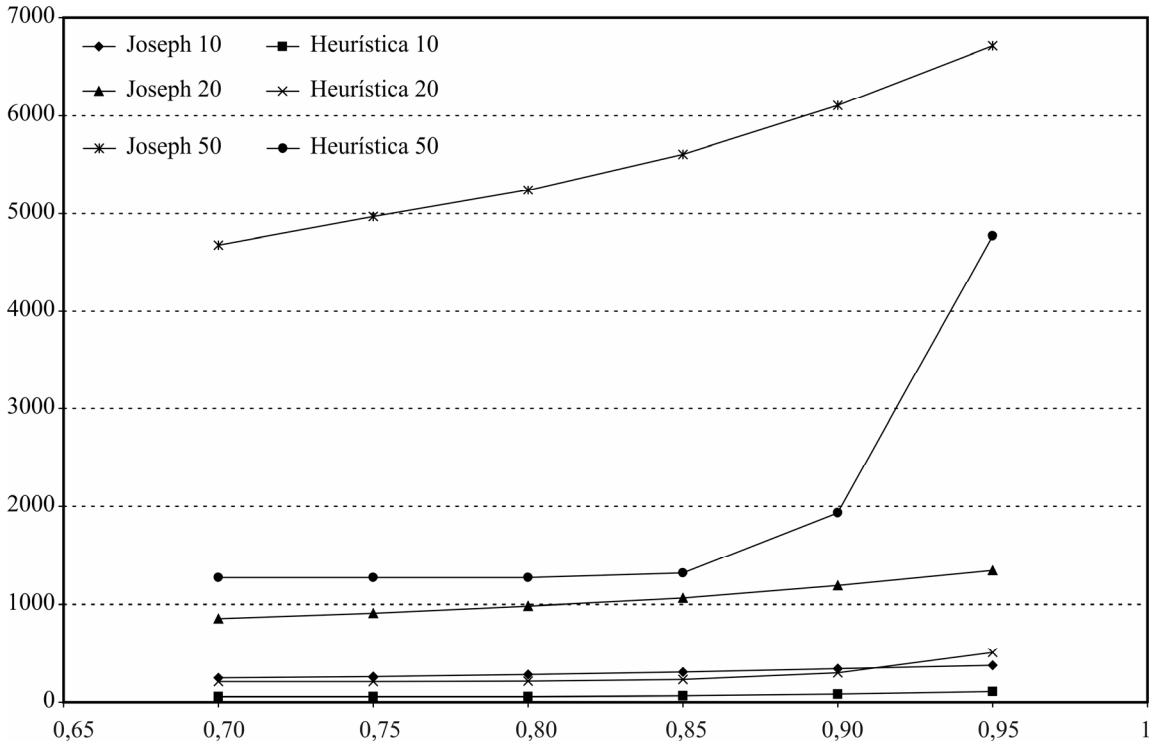

Figura 2. Costo Computacional vs. Factor de Utilización.

Se puede observar que el método aquí presentado obtuvo un menor costo computacional que el método de Joseph para los tres grupos de tareas testeados. Las diferencias son notables para factores de utilización menores al 90% debido

a que la heurística encuentra rápidamente algún punto con *slack* ≥ 0. Para factores de utilización entre el 90% al 100% los dos métodos deben buscar exhaustivamente si los sistemas son planificables, encontrando muchos casos en los cuales no lo son, lo cual incrementa notablemente el costo computacional.

Se resalta el hecho de que para factores de utilización menores al 90% y para los grupos de tareas aquí testeados, en promedio, el valor del costo computacional del cálculo es aproximadamente la suma de la serie:

$$2+3.....+n=\sum_{i=2}^{n} i=\frac{n(n+1)}{2}-1 \tag{7}$$

Esto se debe a que en el cálculo en el vencimiento de la tarea τ_i, el *slack* obtenido es mayor o igual a cero con lo cual, se puede afirmar que el *STR* es planificable y por lo general, se obtiene sólo con una iteración de la ecuación (5).

6 CONCLUSIONES

En este trabajo se presenta un test de planificabilidad de baja complejidad para sistemas de tiempo real. Se comparan los resultados obtenidos con la complejidad de los métodos tradicionales y se comprueba una mejora sustancial mediante del método propuesto.

La complejidad del test permite su utilización en tiempo de corrida en sistemas embebidos para garantizar que las restricciones temporales serán satisfechas cuando el sistema se vea sometido a cambios dinámicos de sus características de tiempo real.

Referencias

[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] J. Y. T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real Time Tasks.," *Perf. Eval. (Netherlands)*, vol. 2, pp. 237-250, 1982.

[4] R. I. Davis, "Dual Priority Scheduling: A Means of Providing Flexibility in Hard Real-Time Systems," Department of Computer Science, University of York, York, England 1995.

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

[6] 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 1987.

[7] J. P. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior," presented at IEEE Real-Time Systems Symposium, 1989.

[8] 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.

[9] J. Santos and J. D. Orozco, "Rate Monotonic Scheduling in Hard Real-Time Systems," *Information Processing Letters*, vol. 48, pp. 39-45, 1993.

[10] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling," *Software Engineering Journal*, vol. 8, pp. 284-292, 1993.

[11] 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.

[12] E. Bini and G. Buttazzo, "A Hyperbolic Bound for the Rate Monotonic Algorithm," presented at 13th Euromicro Conference on Real-Time Systems, 2001.

[13] J. M. Urriza and J. D. Orozco, "Métodos Rápidos para el Cálculo del Slack Stealing Exacto y Aproximado para Aplicaciones en Sistemas Embebidos," Dep. de Ing. Eléctrica y Computadoras, Universidad Nacional del Sur, Argentina., Bahía Blanca 2004.

[14] R. I. Davis, K. W. Tindell, and A. Burn, "Scheduling Slack Time in Fixed-Priority Preemptive Systems," *Proceedings of the Real Time System Symposium*, pp. 222-231, 1993.