# Diseño y Verificación de Sistemas de Tiempo Real Heterogéneos

Doctorado en Ciencias de la Computación

Departamento de Ciencias e Ingeniería de la Computación de la Universidad Nacional del Sur.

**Autor**: Francisco Ezequiel Páez (FI-UNPSJB) (fep@ing.unp.edu.ar)

Fecha de Defensa: 12/03/2021

**Director**: Javier D. Orozco (DIEC-UNS) (jadorozco@gmail.com)

**Codirectores**: José M. Urriza (FI–UNPSJB) (josemurriza@gmail.com) y Pablo Fillottrani (DCIC–UNS) (prf@cs.uns.edu.ar)

## RESUMEN

La tesis aporta nuevos métodos de evaluación de planificabilidad y de administración del tiempo ocioso. Estos son utilizados en el diseño e implementación de Sistemas de Tiempo Real Heterogéneos, donde se requiere una planificación conjunta y eficientemente de tareas críticas y no-críticas. Los métodos propuestos se evalúan mediante simulaciones e implementaciones sobre placas de desarrollo y Sistemas Operativos de Tiempo Real, verificando así su factibilidad práctica.

***Palabras clave:*** STR, Planificación, Slack Stealing.

## CONTEXTO

Este documento es un resumen de la Tesis "Diseño y Verificación de Sistemas de Tiempo Real Heterogéneos", presentada para la obtención del título de Doctor en Ciencias de la Computación, en el Departamento de Ciencias e Ingeniería de la Computación de la Universidad Nacional del Sur. La fecha de defensa de la tesis fue el día 12 de marzo del año 2021. El desarrollo del doctorado contó con el apoyo de una beca doctoral cofinanciada del CONICET y la Secretaría de Ciencia, Tecnología e Innovación Productiva del Chubut, con lugar de trabajo en la sede Puerto Madryn de la Facultad de Ingeniería de la Universidad Nacional de la Patagonia San Juan Bosco.

## 1. INTRODUCCIÓN

En un Sistema de Tiempo Real (STR) los resultados deben producirse antes de un instante denominado *vencimiento*. Los STR son utilizados clásicamente en situaciones donde un fallo puede producir pérdidas económicas, daños ambientales o incluso pérdidas humanas, como sistemas de control industrial, aviónica o equipamiento médico. Por lo tanto, para garantizar la predictibilidad y determinismo de ejecución, su desarrollo emplea hardware y software de efectividad

comprobada, con diseños e implementaciones estáticas que suelen ser *ad-hoc*. En la actualidad los STR se aplican en una multitud de dispositivos como electrodomésticos, teléfonos móviles o automóviles. Estas aplicaciones contienen tareas críticas, que deben cumplir estrictamente sus vencimientos, y tareas con restricciones temporales relajadas o inexistentes. Un STR de este estilo se denomina *STR heterogéneo* (STRH), donde las tareas críticas se conocen como Tareas de Tiempo Real (TTR) y las tareas no-críticas como Tareas de No-Tiempo Real (TNTR).

Un STRH debe garantizar que se cumplan los vencimientos de las TTR y ofrecer también una calidad de servicio adecuada a las TNTR. Para esto es necesario aprovechar el tiempo ocioso que deja la ejecución de las TTR, mediante técnicas que lo calculen y administren de manera exacta en tiempo de ejecución. Este tiempo ocioso puede ser utilizado para otros objetivos como el ahorro de energía, la tolerancia a fallas o computación imprecisa. Diversos métodos han sido propuestos para este fin, muchos de los cuales subutilizan los recursos computacionales o cuentan con un costo computacional (CC) que restringe su uso en la práctica. Reducir el CC de las técnicas de administración de tiempo ocioso exacto evita el sobredimensionamiento del hardware, con el ahorro económico correspondiente. En la actualidad, la capacidad de cómputo de los microcontroladores permite implementar este tipo de técnicas con una sobrecarga tolerable.

En la tesis se desarrollan nuevas técnicas de evaluación de planificabilidad y de uso del tiempo ocioso, con el objetivo de que los STRH puedan ser eficaces en la planificación de sus tareas. Los métodos propuestos son implementados y ejecutados en placas de desarrollo para verificar su viabilidad práctica.

## 2. CONCEPTOS BÁSICOS DE STR

Según la criticidad del vencimiento, los STR se clasifican como *duros* o *críticos* (no toleran perdidas de vencimientos), *blandos* (permiten algunas pérdidas) o *firmes* (tipifican las pérdidas según algún criterio). El cumplimiento de los vencimientos se verifica *a priori* mediante un *test de planificabilidad*. En sistemas mono-recurso (por ejemplo, con un solo CPU), el peor estado de carga ocurre cuando todas las tareas solicitan simultáneamente acceso al recurso y se lo denomina *instante crítico*. Si todas las tareas cumplen con sus vencimientos a partir de este instante entonces el STR es *planificable* [1]. Un STR se suele modelar como un conjunto de tareas periódicas [1], caracterizadas mediante un periodo ($T$), un vencimiento relativo ($D$) y un peor caso de tiempo de ejecución ($C$). Un STR compuesto por *n* tareas se describe como:

$$S(n) = \{\tau_i = (C_i, T_i, D_i) \; \forall 1 \leq i \leq n\}$$

Dado que las tareas son periódicas, el Factor de Utilización (FU) del STR es el porcentaje de tiempo que el recurso se encuentra ocupado y se calcula como $\sum_{i=1}^{n}(C_i/T_i)$. El orden de ejecución de las tareas se determina mediante un *algoritmo de planificación*, que puede ser *estático* (la planificación se realiza fuera de línea o al inicializar el sistema) o *dinámico* (la decisión se toma en tiempo de ejecución en base a prioridades asignadas a las tareas). En estos últimos las *prioridades* pueden ser *fijas* (no se modifican una vez asignadas) o *dinámicas* (pueden variar en tiempo de ejecución). El algoritmo de planificación dinámico con prioridades fijas más utilizado es Rate Monotonic (RM) [1], que asigna las prioridades de manera inversamente proporcional a los períodos. Su ejecución es predecible, es de fácil implementación y es *óptimo* entre todos los algoritmos de prioridades fijas.

## 3. JITTER Y PLANIFICABILIDAD

El tiempo que transcurre entre el arribo de una tarea y su activación por parte del planificador se denomina *jitter de activación*. Suponer que una tarea *J* arriba en el instante *t*, pero sufre un *jitter de activación* de *j* unidades. Por lo tanto, la tarea será puesta en la cola de tareas listas para ejecutar en el instante $t + j$. Si en dicho instante se activa una tarea *I* de mayor prioridad, la ejecución de la tarea *J* se pospone hasta que la tarea *I* finalice. Esto genera una interferencia adicional a *J*, reduciendo el tiempo disponible para su ejecución. Por lo tanto, las tareas que presentan *jitter de activación* pueden no cumplir sus vencimientos a causa de la interferencia adicional de tareas de mayor prioridad. Estas situaciones deben analizarse para garantizar la planificabilidad del STR. Sin embargo, los *test* que tienen en cuenta el *jitter de activación* analizan la planificabilidad a partir de un *instante crítico* generado por el *jitter de activación*. Como no siempre existe tal *instante crítico*, STR planificables pueden ser identificados como no-planificables.

La tesis presenta un método y un algoritmo de búsqueda del *peor instante crítico con jitter* (PICJ). Con el mismo se realizan simulaciones sobre conjuntos de 10, 20 y 50 tareas, con FU entre el 10% y el 90%. Los resultados muestran que el porcentaje de sistemas con un PICJ conformado por todas las tareas decrece exponencialmente al aumentar el FU. Esto indica que considerar el *peor caso de jitter de activación* puede causar una evaluación incorrecta de la planificabilidad del STR. Dado que estos análisis son un requisito necesario en el diseño de STR críticos se propone como trabajo futuro desarrollar un método que identifique los valores reales de *jitter de activación* que pueden causar un PICJ, para utilizarlos en el análisis de planificabilidad.

## 4. TEST DE PLANIFICABILIDAD

En la tesis se presenta un nuevo método de evaluación de planificabilidad exacto y de bajo CC para las disciplinas de prioridades fijas RM [1] y Deadline Monotonic (DM) [2]. El método está basado en la búsqueda de un Punto Fijo (PF), técnica propuesta originalmente en [3] e independientemente en [4]. La búsqueda del PF de una tarea $i$ utiliza la siguiente fórmula iterativa:

$$t_i^{q+1} = t_i^q + \sum_{j=1}^{i-1}\left(A_j^q - A_j^{q-1}\right) \quad donde\ A_j^q = \left\lceil \frac{t_i^q}{T_j} \right\rceil$$

$$con \quad t_i^{q+1} \le D_i, \quad 0 \le q \le m \quad y \quad 1 \le j \le i-1$$

El PF en $t_i^q$ existe sí y sólo sí $A_j^q = A_j^{q-1}$ para todo $1 \le j \le i-1$. Si el primer PF se encuentra en un instante $t_i^{q+1} = t_i^q \le D_i$, entonces la tarea es planificable y su peor caso de tiempo de respuesta es $R_i = t_i^q$. Caso contrario, la tarea no es planificable. La semilla inicial es $t_i^0 = R_{i-1} + C_i$ [5] y se actualiza durante la sumatoria como $t_i^{q+} = t_i^q + A_j^q - A_j^{q-1}$ [6]. Si se encuentra un PF para todas las tareas en un instante menor o igual a su vencimiento, se garantiza que el STR es planificable.

Para reducir el número de invariantes a calcular, se aprovecha que el término $A_j^q$ es una función monótona creciente a saltos, cuyo valor se mantiene constante durante el período de la tarea $j$. Sea entonces $I_j^q$ el máximo instante hasta el que $A_j^q$ mantiene su valor. Luego, dada una iteración $q+1$, si $t_j^{q+1} \le I_j^q$ entonces $A_j^{q+1} = A_j^q$ y no es necesario calcular $A_j^{q+1}$. Caso contrario se calcula y actualiza $I_j^q$. Esta mejora es aplicable en la evaluación de planificabilidad de cualquier política de prioridades fijas y requiere un arreglo del orden del número de tareas, que almacene los valores $I_j^q$.

Mediante el anterior resultado se obtuvieron los siguientes teoremas, que evitan la necesidad de una iteración adicional para verificar el PF y permiten, bajo ciertas condiciones, realizar el cálculo directo del peor caso de tiempo de respuesta:

**Teorema 1 (3.1 en la Tesis)**: Sea un STR de $n$ tareas. Si al finalizar la iteración $q$ del cálculo del peor caso de tiempo de respuesta de la tarea $i$, se tiene que $I_j^q \ge t_j^{q+1}$ para todo $1 \le j < i$, entonces el algoritmo iterativo ya ha encontrado un PF en $t_i^{q+1}$.

**Teorema 2 (3.2 en la Tesis)**: Dada una tarea $i$ sea $t_i^0 = R_{i-1} + C_i$. Si $I_j^0 \ge t_j^0$ para todo $1 \le j < i$ entonces $R_i = t_i^0$ y es un PF.

Durante el cálculo del peor caso de tiempo de respuesta de una tarea $i$, al calcular $A_j^q$ y actualizar $I_j^q$, es posible que el nuevo valor de

la semilla, $t_i^{q+} = t_i^q + A_j^q - A_j^{q-1}$, exceda el nuevo intervalo de validez, esto es $t_i^{q+} > I_j^q$. Por lo tanto, no se encontrará un PF en la siguiente iteración. Para evitar iteraciones adicionales, el cálculo del término $A_j^q$ se reemplaza mediante el propuesto en el siguiente teorema:

**Teorema 3 (3.3 en la Tesis)**: Dada una iteración $q$, el valor $t_i^{q+}$ que maximiza el incremento de la carga de trabajo de la tarea $j$ a partir de la semilla $t_i^q$, puede calcularse como:

$$t_i^{q+} = t_i^q + A_j^{q*} - A_j^{(q-1)*} \quad con\ A_j^{q*} = \left\lceil \frac{t_i^q - A_j^{(q-1)*}}{T_j - C_j} \right\rceil C_j$$

En base a estas mejoras, se desarrollaron dos nuevos métodos de evaluación de planificabilidad, RTA3 [7] (mejoras 1 y 2) y RTA4 [8] (aplica todas las mejoras). Para evaluar sus rendimientos en comparación con otros métodos (RTA [5], RTA2 [6] y HET2 [9, 10]), se realizaron simulaciones y ejecuciones en una placa de desarrollo mbed LPC1768 (ARM Cortex-M3 a 96 Mhz con 32 KiB de RAM), registrando el número de términos $A_j^q$ requeridos para evaluar la planificabilidad de un STR y el tiempo de ejecución en microsegundos. Los nuevos métodos presentan el mejor desempeño, con un CC acotado entre $\theta(n \cdot \log(n))$ y $\theta(n^2)$. En la Figura 1 pueden observarse los resultados para STR con 10 tareas y distribución uniforme de periodos.

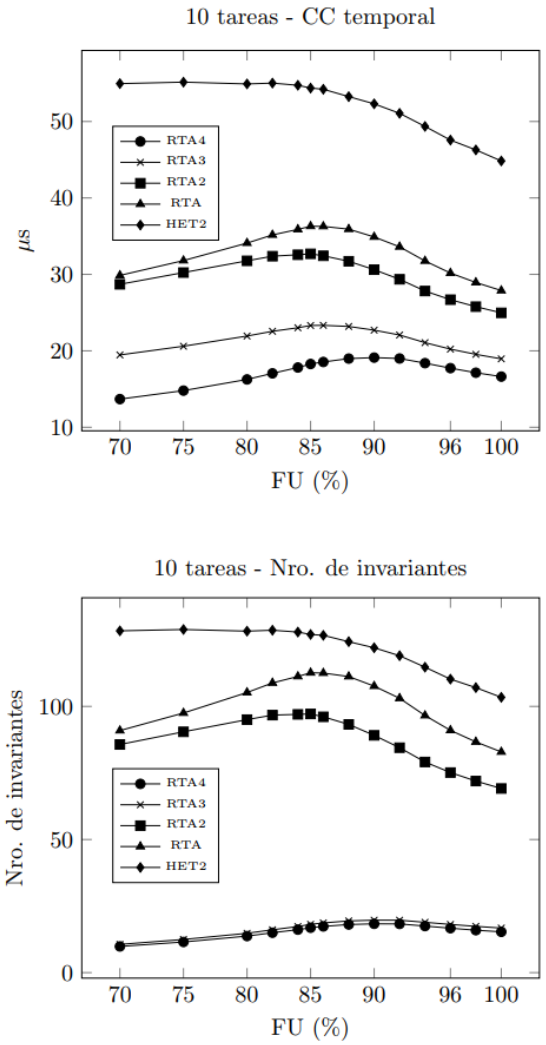


Figura 1: CC al evaluar la planificabilidad de STR de 10 tareas en microsegundos y número de invariantes calculadas.

## 5. MÉTODOS DE SLACK STEALING

Los métodos de Slack Stealing (SS) [11-13] permiten identificar y adelantar una parte del tiempo ocioso futuro disponible, en STR planificados mediante disciplinas de prioridades fijas como RM o DM. El tiempo ocioso que puede ser aprovechado sin

comprometer la planificabilidad del STR se denomina Slack Disponible (SD). El cálculo del SD puede realizarse fuera de línea o en tiempo de ejecución y ser aproximado o exacto. El SD puede ser utilizado para la planificación de STRH, como también para el ahorro de energía o la tolerancia a fallas.

Las primeras técnicas de SS presentan un elevado CC que imposibilita su uso en tiempo de ejecución [14, 15]. Sin embargo, aportes recientes [16-18] reducen este CC, permitiendo el uso de estos métodos en línea. Las mejoras al proceso iterativo de búsqueda de un PF presentados en la sección anterior son aplicables en los métodos de SS, reduciendo aún más el CC. El cálculo en línea del SD de una tarea $i$ en un instante $t_c$ se realiza de la siguiente manera [16]:

$$s_i(t_c,t) = t - t_c - \sum_{j=i}^{n} \left\lceil \frac{t}{T_j} \right\rceil C_j - \sum_{j=i}^{n} \left( \left\lfloor \frac{t_c}{T_j} \right\rfloor C_j - c_i(t_c) \right)$$

$$S_i(t_c) = \max_{t \in [x_i(t_c), d_i(t_c)]} s(t_c,t) \quad \textit{en el menor } t.$$

Este cálculo se realiza en diversos instantes dentro de un intervalo de búsqueda, siendo el SD el mayor valor calculado en el instante $t$ más pequeño. La mejora al proceso iterativo se aplica al primer término de la sumatoria, que es la carga de trabajo de la tarea $j$ en el instante $t$ ($A_j^q$). El costo espacial adicional es el de un arreglo del orden del número de tareas, para almacenar los términos $A_j^q$.

Se modificaron los métodos de SS publicados en [17] (HeuristicSlack) y [16] (FixedSlack). Para evaluar la mejora, se contabilizó mediante simulaciones y ejecuciones en una placa de desarrollo mbed LPC1768 el número de techos y pisos requeridos para el cálculo del SD. Se evaluaron STR de 10, 20 y 50 tareas, con FU del 10% al 90% y distribuciones de periodos uniformes y por grupos. Los métodos modificados presentan un menor CC. Como resultado ilustrativo, se presenta los resultados para STR de 50 tareas con distribución de periodos por grupos (Figura 2).

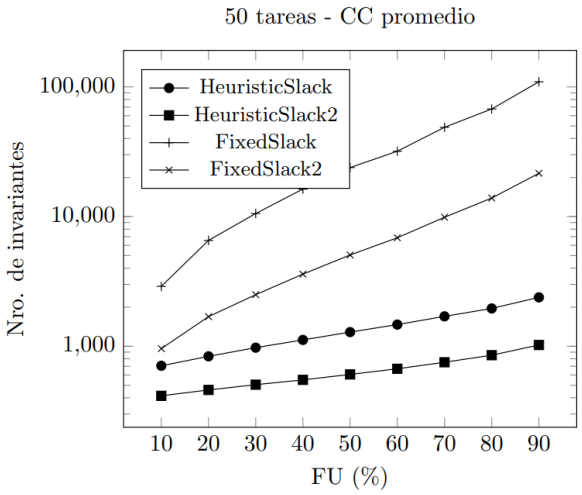


Figura 2: CC de los métodos de SS antes y después de la mejora en el cálculo de la carga de trabajo para STR de 50 tareas.

## 6. SLACK STEALING EN UN SOTR

Para comprobar la factibilidad de uso en tiempo de ejecución de los métodos de SS, en la tesis se presenta una implementación de

estos en el Sistema Operativo de Tiempo Real (SOTR) FreeRTOS, de código abierto y ampliamente utilizado. La implementación permite utilizar el SD para la planificación de un conjunto heterogéneo de tareas.

Se agregaron estructuras de datos para administrar los parámetros y colas de tareas adicionales (clasificación de tareas en TTR y TNTR, tareas suspendidas por falta de SD, etc.) y un API para el desarrollador. En el núcleo de FreeRTOS se modificaron las siguientes funciones:

- `vTaskDelayUntil()`: utilizada para implementar tareas con periodicidad estricta. Se agregó el cálculo del SD y la reanudación de la ejecución de TNTRs suspendidas por falta de SD.
- `xTaskIncrementTick()`: rutina de servicio de la interrupción de reloj. Se agregó la gestión de contadores de SD, la suspensión de la ejecución de las TNTRs si no existe SD y un mecanismo de control de vencimientos para las TTRs.

Por defecto FreeRTOS utiliza una política de planificación *primero en llegar primero en ser atendido* (FCFS) apropiativa por prioridades, que garantiza la ejecución de la tarea con mayor prioridad lista para ejecutar. Sin modificar este mecanismo, se altera la composición de la cola de tareas listas para ejecutar, asignando las primeras *M* prioridades a las TNTR. Estas estarán presentes en la cola de tareas listas sí y sólo sí existe suficiente SD. De esta manera, las TNTR son ejecutadas si y sólo si no comprometen la planificabilidad de las TTR.

Se evaluó el CC introducido por las modificaciones midiendo el número de ciclos de CPU que requiere el cambio de contexto al finalizar la ejecución de cada TTR, donde ocurre el cálculo del SD. Las pruebas se realizaron sobre una placa mbed LPC1768. La interrupción de reloj se configuro en 1 ms, dando a cada *time slice* aproximadamente 96000 ciclos de CPU. Se empleo el método Fixed2 (ver sección anterior). Se evaluaron tres configuraciones: sin modificar FreeRTOS, calculando el SD en tiempo de inicialización y en línea. Los resultados muestran que el CC adicional de las modificaciones, sin tener en cuenta el cálculo del SD, es constante independientemente del FU. Incluyendo el cálculo en línea del SD, el costo en el peor caso no supero el 2,5% del *quantum*. En la Figura 3 se presentan estos resultados.

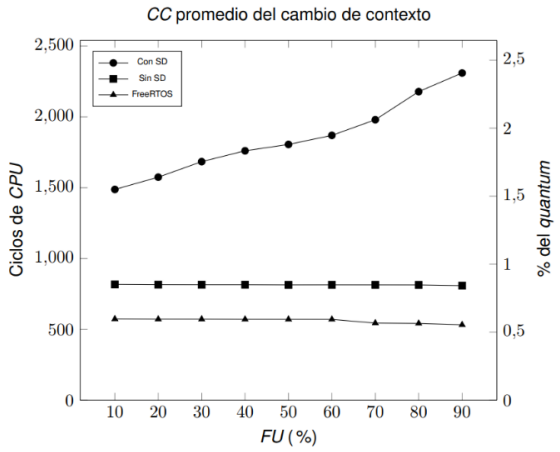


Figura 3: Comparativa del CC en ciclos de CPU del cambio de contexto de FreeRTOS.

La implementación permite verificar que la ejecución de los métodos SS propuestos es factible en un SOTR, con un CC aceptable para la ejecución de conjuntos heterogéneos de tareas. Sin embargo, podría ser utilizado para otros fines como tolerancia a fallas o ahorro de energía.

## 7. TAREA PLANIFICADORA

En general, los SOTR ofrecen un número limitado de políticas de planificación, como planificadores apropiativos por prioridades dinámicas. Dado que modificar el código de un SOTR no es siempre deseable o posible, se presenta en la tesis una solución mediante una *tarea planificadora* que permite implementar un planificador heterogéneo sin necesidad de modificar el SOTR. La tarea planificadora (TP) debe ejecutarse ante ciertos eventos durante la ejecución del SOTR (arribo de una tarea, finalización, bloqueo, suspensión o interrupción), en los cuales debe tomar una decisión de planificación. El SOTR debe proveer mecanismos para reanudar y suspender tareas, cambiar sus prioridades e interceptar eventos.

La solución propuesta se organiza en dos módulos. El primer módulo se encarga del mecanismo de planificación: implementa la TP como una tarea del sistema, su activación ante los eventos correspondientes provee las estructuras de datos requeridas y la interface de programación (API) para el desarrollador. El segundo modulo implementa las políticas de planificación, por ejemplo, Dual Priority [19], EDF [1] o RM con SS. De esta manera se desacopla el mecanismo de la política implementada.

El diseño se implementó sobre FreeRTOS y se evaluó el CC ejecutando diversos STR en una placa de desarrollo mbed LPC1768. Se evaluaron STR de 10 tareas, con FU de 10% al 90%, con distribuciones de periodos uniforme y por grupos. Las pruebas muestran que el CC introducido por la TP no es excesivo, aunque debe evaluarse en cada caso particular si es aceptable. La Figura 4 presenta los resultados para periodos distribuidos uniformemente entre 25 y 1000 ms. El diseño propuesto es lo suficientemente flexible para ser adaptado a otros SOTR para implementar políticas de planificación heterogénea.

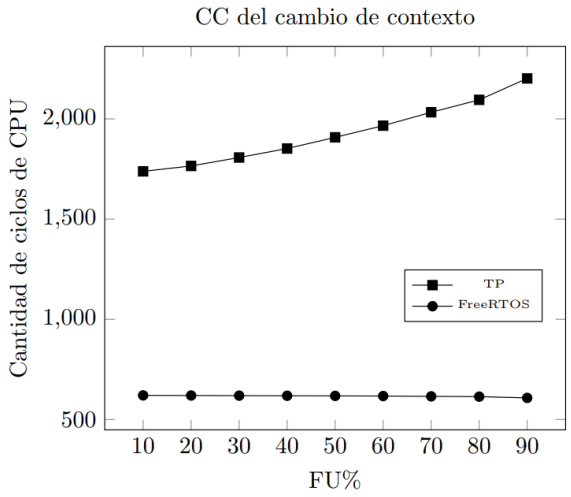


Figura 4: Comparativa del CC del cambio de contexto de FreeRTOS con y sin la TP, para STR de 10 tareas y distribución de periodos uniforme.

## 8. CONCLUSIONES

Los STRH requieren de métodos de administración de tiempo ocioso y de evaluación de planificabilidad para la admisión de tareas en tiempo de ejecución. Para que sea factible su empleo en la práctica, estos métodos deben presentar un CC reducido, para que la sobrecarga introducida en el sistema sea aceptable. Los métodos desarrollados en la tesis reducen el CC de los métodos exactos de evaluación de planificabilidad y de cálculo del tiempo ocioso mediante SS. Estas mejoras son verificadas mediante simulaciones, pero también mediante implementaciones en plataformas de desarrollo, cuantificando así el costo de ejecución en la práctica. Los métodos exactos de evaluación de planificabilidad de bajo costo pueden utilizarse para la aceptación en línea de nuevas tareas sin comprometer la planificabilidad del STRH, dinamizando su composición. Aunque los métodos de SS presentados son empleados en la planificación heterogénea para brindar atención prioritaria a tareas no-críticas, estos pueden ser aplicados para el ahorro de energía o la tolerancia a fallas. De esta manera, múltiples líneas de investigación y desarrollo futuro son posibles a partir de los resultados presentados en este trabajo.

## 9. REFERENCIAS

[1] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM,* vol. 20, no. 1, pp. 46-61, 1973.

[2] 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.

[3] M. Joseph and P. Pandya, "Finding Response Times in Real-Time System," *The Computer Journal (British Computer Society),* vol. 29, no. 5, pp. 390-395, 1986.

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

[5] M. Sjödin and H. Hansson, "Improved Response-Time Analysis Calculations," in *IEEE 19th Real-Time Systems Symp.*, IEEE, Ed., Dec. 1998, pp. 399-409.

[6] J. M. Urriza, J. D. Orozco, R. Cayssials, and L. Schorb, "Reduced Computational Cost in the Calculation of Worst Case Response Time for Real Time Systems," (in English), *Journal of Computer Science & Technology,* vol. 9, no. 2, pp. 72-81, 2009.

[7] J. M. Urriza, F. E. Paez, J. D. Orozco, and R. Casyssials, "Computational Cost Reduction for Real-Time Schedulability Tests Algorithms," *IEEE Latin America Transactions,* vol. 13, no. 12, pp. 3714-

3723, 2015, doi: 10.1109/TLA.2015.7404899.

[8] J. M. Urriza, F. E. Páez, M. Ferrari, R. Cayssials, and J. D. Orozco, "A New RM/DM Low Cost Schedulability Test," in *Eight Argentine Symposium and Conference on Embedded Systems*, Buenos Aires, 2017, pp. 13-18.

[9] E. Bini, G. Buttazzo, and G. Buttazzo, "A Hyperbolic Bound for the Rate Monotonic Algorithm," *IEEE Transactions on Computer,* vol. 52, no. 7, pp. 933-942, 2003.

[10] R. I. Davis, A. Zabos, and A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems," *IEEE Transactions on Computers,* vol. 57, no. 9, pp. 1261-1276, 2008, doi: 10.1109/TC.2008.66.

[11] S. Ramos-Thuel and J. P. Lehoczky, "Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Priorirys Systems using Slack Stealing," in *Real-Time Systems Symposium*, IEEE, Ed., 7-9 Dec. 1994, pp. 22-33.

[12] J. P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," in *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, EUA, 1992, pp. 110-123.

[13] S. Ramos-Thuel and J. P. Lehoczky, "On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems," in *Real-Time Systems Symposium*, 1993, pp. 160-171.

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

[15] T.-S. Tia, J. W.-S. Liu, and M. Shankar, "Algorithms and Optimality of Scheduling Soft Aperiodic Requests in Fixed Priority Preemtive Systems," *The International Journal of Time-Critical Computing Systems,* vol. 10, no. 1, pp. 23-43, January 1996.

[16] J. M. Urriza, F. E. Paez, R. Cayssials, J. D. Orozco, and L. Schorb, "Low Cost Slack Stealing Method for RM/DM," *International Review in Computers and Software (IRECOS),* vol. 5, no. 6, pp. 660-667, 2010.

[17] J. M. Urriza, "Factibilidad de Sistemas de Tiempo Real con Requerimientos Heterogéneos," Doctor, Departamento de Ingeniería Eléctrica y Computadoras, Universidad Nacional del Sur, Bahía Blanca, 2008.

[18] J. M. Urriza, R. Cayssials, and J. D. Orozco, "A Fast Slack Stealing Method for embedded Real-Time Systems," Dep. de Ing. Eléctrica y Computadoras, Universidad Nacional del Sur, Argentina., Bahía Blanca, Internal Report May 31 2005.

[19] R. Davis and A. Wellings, "Dual priority scheduling," in *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, 5-7 Dec 1995 1995, pp. 100-109, doi: 10.1109/REAL.1995.495200.