Computational Cost Reduction for Real-Time Schedulability Tests Algorithms

J. M. Urriza, F. E. Páez, J. D. Orozco, *Member, IEEE* and R. Cayssials, *Member, IEEE*

***Abstract*— This paper focuses on reducing the computational cost of iterative algorithms used to evaluate the schedulability of Real-Time Systems. These algorithms calculate the worst case response time of each task. Through simulations it is found that the proposed new algorithm produces a significant reduction in the average temporal cost, reaching in some cases a reduction of $O(n^2)$ to $O(n.\log(n))$, with respect to classical response time evaluation algorithms.**

***Keywords*— Real-Time Systems, Rate Monotonic, Deadline Monotonic, Response Time Analysis**

I. INTRODUCCION

EN LOS *Sistemas de Tiempo Real* (*STR*), los resultados deben producirse antes de un determinado tiempo denominado *vencimiento*. Diversos métodos han sido propuestos para garantizar, en tiempo de diseño, que los vencimientos serán satisfechos durante la ejecución. Sin embargo, estos pueden poseer un *Costo Computacional* (*CC*) elevado, lo que conspira contra su uso en tiempo de ejecución. Para que puedan ser aplicados de manera eficiente en tiempo de ejecución, deben poseer un *CC* espacial y temporal acotado. Numerosos trabajos se centran en la reducción del *CC*, abordando este problema desde diversos puntos de vista.

Métodos como el cálculo del *peor caso de tiempo de respuesta* (*WCRT*), mecanismos basados en *Servidores* [1, 2, 3, 4], *Slack Stealing* [5, 6, 7, 8, 9, 10], y algunos métodos de admisión de tareas en tiempo de ejecución, etc., poseen un *CC* elevado que limitan su aplicabilidad en tiempo de ejecución, o imponen restricciones sobre el conjunto de tareas, que condicionan el diseño del planificador en forma pesimista. Esto resulta en *STR* poco tolerantes a cambios dinámicos de diversa índole. En otros casos, la reducción del *CC* se logra produciendo un resultado aproximado, con la consiguiente incertidumbre que esto produce en las futuras acciones, obteniendo una eficiencia menor a la deseada.

Por otro lado, un método exacto de bajo *CC* permitiría tomar decisiones en tiempo de ejecución que permitan una administración dinámica de los recursos del sistema, manteniendo no sólo la planificabilidad sino la eficiencia. Consecuentemente, un bajo *CC* permite a los diseñadores desarrollar y ofrecer una nueva heterogeneidad de aplicaciones, con una mejor calidad de servicio, nuevas prestaciones y mejor utilización de los recursos.

En dispositivos de Tiempo Real, en general, es necesario emplear un *Sistema Operativo de Tiempo Real* (*RTOS*), para administrar los recursos del sistema. Una de las funciones del *RTOS* es la planificación y temporización de las tareas de tiempo real. Esta función es llevada a cabo por el planificador. Por lo general existe, además, un conjunto de requerimientos no funcionales como métodos de tolerancia a las fallas, atención de tareas aperiódicas, optimización en el consumo de energía, entre otros, que son implementados como extensiones al algoritmo básico de planificación. Consecuentemente, métodos con un bajo *CC* permiten tiempos reducidos de cambios de contexto (*CS*) y menores tiempos de inicialización y ejecución del *RTOS*. Esto conlleva a que el sistema tenga más recursos disponibles para la ejecución de las tareas de tiempo real, que es su principal función.

En este trabajo se presenta un nuevo test de planificabilidad, basado en una mejora de los métodos de verificación iterativos. Esta mejora puede ser aplicada también en algunos de los métodos clásicos antes mencionados, reduciendo el *CC* temporal de los mismos.

A. Introducción a los Sistemas de Tiempo Real

La definición más aceptada en la disciplina, fue realizada por Stankovic ([11]): "*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* se pueden clasificar, según la tolerancia del sistema a la pérdida de vencimientos en las tareas, en tres tipos. En el primer tipo, no se permite que ninguna tarea pierda su vencimiento, y se los denominan *STR duros* o *críticos*. En el segundo tipo, se permite que algunos vencimientos se pierdan, por lo cual se los denominan *blandos*. Por último, la mayor difusión de los *STR* ha requerido tipificar aquellos que permiten sólo una determinada cantidad de pérdidas, especificadas bajo algún criterio particular, como por ejemplo un vencimiento de cada diez. A estos *STR* se los denomina *firmes* o *débiles*. Ejemplos de estos tipos de *STR* son sistemas de transacción *online*, centrales telefónicas, redes de datos, sistemas multimedia (por ejemplo, video a demanda), etc. [12, 13].

En los *STR* de tipo *duro*, la pérdida de un vencimiento tiene

J. M. Urriza, Universidad Nacional de la Patagonia San Juan Bosco, Puerto Madryn, Argentina, josemurriza@unp.edu.ar

F. E. Paez, Universidad Nacional de la Patagonia San Juan Bosco, Puerto Madryn, Argentina, fpaez@unpata.edu.ar

J. D. Orozco, Universidad Nacional del Sur, Bahía Blanca, Argentina, jorozco@uns.edu.ar

R. Cayssials, Universidad Nacional del Sur y Universidad Tecnológica Nacional Facultad Regional Bahía Blanca, Argentina, rcayssials@frbb.utn.edu.ar

consecuencias adversas para la integridad del sistema, y posiblemente de su entorno. Ejemplos de éstos se pueden encontrar en aviónica, robótica, etc. Consecuentemente, a fin de garantizar que todas las tareas terminen antes de su vencimiento, es necesario realizar un análisis de *planificabilidad*. Si es exitoso, se dice que el sistema es *planificable* y garantiza el cumplimiento de todas sus *constricciones temporales*.

Liu y Layland, establecieron en [14] el marco de trabajo en el cual se basan la mayoría de los trabajos de la disciplina. En éste se consideraron sistemas mono-recurso y multitarea. Se entiende como mono-recurso a un único procesador, canal, etc., el cual sólo puede ser utilizado por una tarea / mensaje a la vez.

Existen dos formas de asignar 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* establece un orden lineal sobre un conjunto de tareas, permitiéndole al *planificador* determinar, en cada instante de activación, qué tarea utilizará el recurso compartido.

Para formalizar el análisis de planificabilidad, resulta necesario modelar al conjunto de tareas en base a sus requerimientos temporales e interdependencias. Usualmente se considera que las tareas son periódicas, independientes y apropiables. Una tarea periódica, es aquella que después de un determinado tiempo fijo solicita nuevamente ejecución. La tarea se dice que es independiente cuando no requiere de la ejecución de 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.

En lo que sigue, la notación utilizada será la siguiente:

C_i : Peor caso de tiempo de ejecución de la tarea i.
D_i : Vencimiento relativo de la tarea i.
T_i : Período de la tarea i.
l_i : Instante en el que se invoca la tarea i.
t^q : Tiempo en la iteración q.

Generalmente, los parámetros de una tarea i son su C_i, T_i y D_i. Luego, un conjunto $\mathbf{S}(n)$ de n tareas se especifica como $\mathbf{S}(n) = \{(C_1, T_1, D_1), (C_2, T_2, D_2), \ldots, (C_n, T_n, D_n)\}$. Si bien este modelo de tareas es reducido, se considera suficiente para el análisis presentado.

En [14] se demostró que el peor esquema de generación, para un sistema mono-recurso, es aquél en el cual todas las tareas solicitan ejecución en el mismo instante y se le denominó *instante crítico* o *peor estado de carga*. Se demostró también que si este estado es planificable, el *STR* lo es para cualquier otro estado, bajo la disciplina de prioridades utilizada.

El factor de utilización (U) de un conjunto de tareas $\mathbf{S}(n)$, determina el nivel de utilización del recurso. Se calcula como:

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

B. Trabajos previos

En 1986, Joseph y Pandya [15] fueron los primeros en introducir un algoritmo iterativo para el cálculo del *WCRT*. Entre los métodos más relevantes, similares al propuesto, se pueden citar los desarrollados en [1, 13, 16, 17, 18, 19] entre otros. No obstante, solo en [13, 18, 19] se plantean nuevas formas de realizar el test.

En [15], el algoritmo iterativo presentado consiste en encontrar el *Punto Fijo* (*PF*) de la ecuación (1), para cada tarea partiendo de una semilla igual a cero ($t^0 = 0$). Diversos trabajos han sido publicados con soluciones equivalentes en [13, 16, 17, 20].

$$t_i^{q+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t_i^q}{T_j} \right\rceil C_j \quad \text{con } D_i \le t_i^q \tag{1}$$

En 1998, Sjödin [13] incorpora una mejora al test de Joseph y Pandya, que consistente en emplear, como semilla inicial para la iteración de la tarea i+1, el instante en donde se encontró el *WCRT* de la tarea i (R_i), más el peor caso de tiempo de ejecución de la tarea i+1 ($t^0 = R_i + C_{i+1}$).

En [18], Bini propone un nuevo método denominado *Hyperplanes Exact Test* (*HET*). Este método es recursivo y las simulaciones realizadas por los autores no evalúan el número de invariantes calculados, con lo cual su comparación con otros métodos es dificultosa. Sin embargo, en el presente trabajo se compara igualmente con los métodos existentes y el propuesto.

En [19], se plantea una mejora al algoritmo iterativo, en la cual se basa parte del presente trabajo. En el mismo, se realimenta la iteración con las soluciones parciales obtenidas en cada paso del cálculo iterativo.

II. CÁLCULO DEL WCRT

A continuación se realiza una introducción a los algoritmos iterativos presentados en [15, 19], los cuales son la base del presente trabajo para reducir el *CC*.

Por definición, un *PF* de una función f, es un número t, tal que $f(t) = t$. En este caso, la ecuación de *PF* es función del tiempo, entonces un punto t y un instante t, son expresiones equivalentes. Además, un *PF* es también un atractor al cual converge la Ecuación (1) [21].

En [15], Joseph y Pandya prueban que no existe una construcción analítica que resuelva este tipo de problemas, y presentan una solución mediante un algoritmo iterativo. Para una mejor comprensión, se realiza un cambio de nomenclatura a la ecuación presentada en [15], y se agrega un superíndice q para indicar la iteración actual en que se encuentra el algoritmo (Ecuación 1).

El operador $\lceil\ \rceil$ representa la operación techo. En la evolución del proceso iterativo, es posible que se detenga cuando se encuentra un *PF* ($t_i^{q+1} = t_i^q$) para un instante $t_i^q \le D_i$. Si esto ocurre, se obtiene el *WCRT* (R_i) de la tarea *i*. Caso contrario, si la iteración continúa hasta llegar a algún valor $t_i^q > D_i$, se debe detener al algoritmo, dado que la tarea, en su peor caso de ejecución, no puede finalizar antes de su vencimiento, y por lo tanto el *STR* no es planificable.

A. Algoritmo Iterativo

En lo que sigue, se presenta el algoritmo introducido en [15] y la mejora propuesta en [19].

Sea un *STR* S(n) = {(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)}. Se supone el cálculo del *WCRT* para una tarea *i*, con $i \in 1 < i \le n$, y que el *STR* es planificable, a fin de realizar una presentación más sencilla. El método comienza con una semilla $t^0 = 0$ como en [15]. Sin embargo, también es válido comenzar en una semilla distinta, como la presentada en [13] ($t^0 = R_{i-1} + C_i$), lo cual reduce el intervalo de iteración.

Cuando se aplica el algoritmo, si el sistema es planificable, ocurrirán m+1 iteraciones antes de llegar a un *PF*, con $m \ge 0$. Dado que el subsistema **S**(i) es planificable, la ecuación posee como mínimo un *PF* en el intervalo $(0, D_i]$. El proceso iterativo evolucionará hacia el primer *PF* en la cadena ascendente, como es demostrado por el Teorema de Kleene y en [22].

A continuación se realiza una traza genérica de la ecuación (1), de acuerdo al algoritmo presentado en [15]:

Iteración 0	$\left\lceil \frac{t^0}{T_1} \right\rceil C_1 + \left\lceil \frac{t^0}{T_2} \right\rceil C_2 + ... + \left\lceil \frac{t^0}{T_{i-1}} \right\rceil C_{i-1} + C_i = t^1$
…….	…….
Iteración m-1	$\left\lceil \frac{t^{m-1}}{T_1} \right\rceil C_1 + \left\lceil \frac{t^{m-1}}{T_2} \right\rceil C_2 + ... + \left\lceil \frac{t^{m-1}}{T_{i-1}} \right\rceil C_{i-1} + C_i = t^m$
Iteración m:	$\left\lceil \frac{t^m}{T_1} \right\rceil C_1 + \left\lceil \frac{t^m}{T_2} \right\rceil C_2 + ... + \left\lceil \frac{t^m}{T_{i-1}} \right\rceil C_{i-1} + C_i = t^{m+1}$

Alcanzado el *PF* en t^m, el resultado de la iteración m-1 es igual al resultado de la iteración m ($t^m = t^{m+1}$).

La ecuación $\lceil t/T_j \rceil C_j$ indica el peor caso de carga de trabajo para la tarea *j*, en el intervalo $[0, l_j + D_j]$ y posee siempre el mismo valor en el intervalo $(l_j, l_j + D_j]$. Sea entonces, A_j^q el valor de la carga de trabajo de la tarea *j* hasta el instante *t* en la iteración *q*:

$$A_j^q = \left\lceil \frac{t^q}{T_j} \right\rceil C_j \ \text{ con } \ 0 \le q \le m \ \text{ y } \ 1 \le j \le i-1$$

Se hace notar que A_j^q es el invariante del algoritmo [13, 15]. Luego, si se reemplaza en el desarrollo de la traza anterior, queda:

Iteración 0	$A_1^0 + A_2^0 + ... + A_{i-1}^0 + C_i = t^1$
…….	…….
Iteración m-1	$A_1^{m-1} + A_2^{m-1} + ... + A_{i-1}^{m-1} + C_i = t^m$
Iteración m	$A_1^m + A_2^m + ... + A_{i-1}^m + C_i = t^{m+1}$

El método detendrá su iteración cuando se cumpla la igualdad $t^m = t^{m+1}$. Esto sólo ocurrirá cuando $t^m = A_1^{m-1} + A_2^{m-1} + ... + A_{i-1}^{m-1} + C_i = A_1^m + A_2^m + ... + A_{i-1}^m + C_i = t^{m+1}$, y sólo es posible sí $A_1^{m-1} = A_1^m, A_2^{m-1} = A_2^m, ..., A_{i-1}^{m-1} = A_{i-1}^m$. Por lo tanto, si no se cumple esta condición, el algoritmo realizará al menos una iteración más para encontrar el *PF*.

Consecuentemente, en los métodos tradicionales, para cada iteración de la Ecuación (1), la inicialización de la variable *t* se realiza con el valor obtenido en la iteración anterior ($t^{q+1} = f(t^q)$). Es por ello que A_j^q no afecta al cálculo de los $A_{j+1}^q, ..., A_{i-1}^q$ subsiguientes.

Sin embargo, si se toman dos iteraciones sucesivas y se igualan, se puede observar que:

$$\begin{aligned} & t^{q+1} - C_i - \sum_{j=1}^{i-1} A_j^q = 0, \ \ t^q - C_i - \sum_{j=1}^{i-1} A_j^{q-1} = 0 \\ & t^{q+1} = t^q + \sum_{j=1}^{i-1} A_j^q - A_j^{q-1} \\ & t^{q+1} = t^q = A_1^q - A_1^{q-1} + A_2^q - A_2^{q-1} + ... + A_{i-1}^q - A_{i-1}^{q-1} \end{aligned} \tag{2}$$

En [19], se demuestra que sí $A_j^q \ne A_j^{q-1}$, es posible calcular A_{j+1}^q con $t_{j+1}^q = t_j^q + A_j^q - A_j^{q-1}$, y así sucesivamente. Por lo tanto, es posible generar una nueva semilla cada vez que $A_j^q \ne A_j^{q-1}$, sin esperar hasta la próxima iteración para incrementar el valor de *t*. Esta mejora en el algoritmo iterativo, permite reducir el número de invariantes calculados en hasta un 30% [19], al disminuir el número de iteraciones necesarias. Sin embargo, este método eleva el *CC* espacial a un vector de orden lineal $O(n)$, al tener que almacenar cada A_j^q para ser comparado con A_j^{q-1}, lo cual es un costo aceptable.

III. Algoritmo *RTA3*

Las mejoras presentadas en la sección anterior disminuyen el *CC* temporal al reducir el número de iteraciones requeridas para hallar el *PF*. A continuación se introduce una mejora basada en la reducción del número de invariantes calculados en cada iteración. El siguiente análisis, no contempla el *offset,* el *jitter* o el *bloqueo*, pero puede ser extendido de manera similar que en los trabajos antes mencionados, como en [16].

El algoritmo presentado en [19] requiere almacenar los valores A_j^q correspondientes a la última iteración *q*. Luego, si para la iteración *q* se tiene que $A_j^q = A_j^{q-1}$, la semilla no es actualizada, y el valor numérico de A_j^q sigue siendo el mismo. Esto se debe a que A_j^q es una función monótona creciente a saltos, y su valor no varía dentro del intervalo $(l_j, l_j + D_j]$. Si t^q se encuentra dentro de este intervalo, se sabe que $A_j^q = A_j^{q-1}$, por lo cual el cálculo de A_j^q es redundante.

Sea entonces, $I_j^{q-1} = \lceil t^{q-1}/T_j \rceil T_j$ el máximo instante hasta el

cual A_j^{q-1} no incrementa su valor. Luego para la iteración *q*, sólo es necesario calcular el valor de A_j^q sí y sólo si $t^q > I_j^q$, de lo contrario $A_j^q = A_j^{q-1}$. El valor de I_j^q se actualiza cada vez que se calcula una nueva carga de trabajo. El incremento del *CC* espacial del algoritmo es lineal con respecto al número de tareas, lo cual es admisible.

Por otra parte, el algoritmo puede ser utilizado para cualquier asignación con prioridades fijas. Sin embargo, si se utiliza *RM* con tiempos de ejecución siguiendo una distribución uniforme, y el cálculo de la carga de trabajo se realiza de la tarea de menor prioridad que se está evaluando, a la tarea de mayor prioridad, se logra una leve reducción promedio del *CC*. Esto se debe a que el cálculo de la tarea de menor prioridad, es decir de mayor periodo en *RM*, conlleva a que posea un tiempo de ejecución mayor, por lo cual logra que el algoritmo evolucione más rápidamente que empezando de la tarea de mayor prioridad y menor periodo.

El siguiente teorema permite terminar la iteración si todos los I_j^q con $1 \le j \le n$ son menores o iguales al t^{q+1} final.

Teorema 1:

Sea un STR de n tareas, si finalizando alguna iteración se encuentra que todos los I_j^q *con* $1 \le j \le n$, *son menores o iguales a* t^{q+1} *el algoritmo ya se encuentra en un PF.*

Prueba:

Si todos los I_j^q son menores a t^{q+1}, en la siguiente iteración ningún I_j^{q+1} cambiará y no se incrementará ningún A_j^{q+1}, consecuentemente el proceso iterativo ya se encuentra en un *PF* dado que será $t^{q+1} = t^q$. □

A. Algoritmo

Los valores iniciales de los A_j e I_j para todas las tareas, son precargados en la inicialización con los valores de $A_j = C_j$ y $I_j = T_j$. En las pruebas experimentales realizadas, el *CC* temporal es levemente mayor al implementar el Teorema 1. Por ello, no fue agregado en el algoritmo. A continuación se presenta el pseudocódigo:

```
function rta3(rts):
    t+ = 0
    t = C1
    WCRT1 = C1
    for i = 2 to n
        t+ = t + Ci
        do
            t = t+
            for j = i-1 to 1
                if t+ > Ij then
                    tmp = ⌈t+/Tj⌉
                    a = tmp * Cj
                    t+ = t+ + a − Aj
                    if t+ > Di then
                        return false
                    end if
                    Aj = a
                    Ij = tmp * Tj
                end if
            end for
        while t ≠ t+
        WCRTi = t
    end for
    return true
end function
```

B. Ejemplo

Se adopta la siguiente anotación: *A* es el valor A_j^q e *I* el valor I_j^q. Si $I_j^q \neq I_j^{q-1}$ se utiliza la notación a., si $I_j^q = I_j^{q-1}$ entonces se emplea la notación b., y $A = A_j^q = A_j^{q-1}$.

a. Si $I_j^q \neq I_j^{q-1} \Rightarrow \left\lceil \frac{t}{T_j} \right\rceil^{A,I} . C_j$, b. Si $I_j^q = I_j^{q-1} \Rightarrow {}^{A,I}A$

Sea el sistema $S(4)=\{(1,4,4),(2,5,5),(1,6,6),(1,12,12)\}$. Se calcula el valor del *WCRT* para la tarea 4. Como el *WCRT* de la tarea 3 es 4, la semilla para comenzar la iteración de la tarea 4, es el instante *t*=5. Se realiza el mismo ejemplo para los algoritmos de *Sjödin*, *RTA2* y *RTA3*.

1) Sjödin

Iteración 0 $t_4^0 = 5 \rightarrow t_4^1 = 1 + \left\lceil \frac{5}{4} \right\rceil^{4,8} .2 + \left\lceil \frac{5}{5} \right\rceil^{1,5} .1 + \left\lceil \frac{5}{6} \right\rceil^{1,6} .1 = 7$

Iteración 1 $t_4^1 = 7 \rightarrow t_4^2 = 1 + \left\lceil \frac{7}{4} \right\rceil^{4,8} .2 + \left\lceil \frac{7}{5} \right\rceil^{2,10} .1 + \left\lceil \frac{7}{6} \right\rceil^{2,12} .1 = 9$

Iteración 2 $t_4^2 = 9 \rightarrow t_4^3 = 1 + \left\lceil \frac{9}{4} \right\rceil^{6,12} .2 + \left\lceil \frac{9}{5} \right\rceil^{2,10} .1 + \left\lceil \frac{9}{6} \right\rceil^{2,12} .1 = 11$

Iteración 3 $t_4^3 = 11 \rightarrow t_4^4 = 1 + \left\lceil \frac{11}{4} \right\rceil^{6,12} .2 + \left\lceil \frac{11}{5} \right\rceil^{3,15} .1 + \left\lceil \frac{11}{6} \right\rceil^{2,12} .1 = 12$

Iteración 4 $t_4^5 = 12 \rightarrow t_4^5 = 1 + \left\lceil \frac{12}{4} \right\rceil^{6,12} .2 + \left\lceil \frac{12}{5} \right\rceil^{3,15} .1 + \left\lceil \frac{12}{6} \right\rceil^{2,12} .1 = 12$

2) RTA2

Iteración 0 $t_4^0 = 5 \rightarrow t_4^1 = 1 + \left\lceil \frac{5}{4} \right\rceil^{4,8} .2 + \left\lceil \frac{5}{5} \right\rceil^{1,5} .1 + \left\lceil \frac{5}{6} \right\rceil^{1,6} .1 = 7$

Iteración 1 $t_4^1 = 7 \rightarrow t_4^2 = 1 + \left\lceil \frac{7}{4} \right\rceil^{4,8} .2 + \left\lceil \frac{7}{5} \right\rceil^{2,10} .1 + \left\lceil \frac{8}{6} \right\rceil^{2,12} .1 = 9$

Iteración 2 $t_4^2 = 9 \rightarrow t_4^3 = 1 + \left\lceil \frac{9}{4} \right\rceil^{6,12} .2 + \left\lceil \frac{11}{5} \right\rceil^{3,15} .1 + \left\lceil \frac{12}{6} \right\rceil^{2,12} .1 = 12$

Iteración 3 $t_4^3 = 12 \rightarrow t_4^4 = 1 + \left\lceil \frac{12}{4} \right\rceil^{6,12} .2 + \left\lceil \frac{12}{5} \right\rceil^{3,15} .1 + \left\lceil \frac{12}{6} \right\rceil^{2,12} .1 = 12$

3) RTA3

Iteración 0 $t_4^0 = 5 \rightarrow t_4^1 = 1 + \left\lceil \frac{5}{4} \right\rceil^{4,8} .2 + \left\lceil \frac{5}{5} \right\rceil^{1,5} .1 + \left\lceil \frac{5}{6} \right\rceil^{1,6} .1 = 7$

Iteración 1 $t_4^1 = 7 \rightarrow t_4^2 = 1 + \overset{4,8}{4} + \left\lceil \frac{7}{5} \right\rceil .1 + \left\lceil \frac{\overset{2,10}{8}}{6} \right\rceil .1 = 9$

Iteración 2 $t_4^2 = 9 \rightarrow t_4^3 = 1 + \left\lceil \frac{\overset{6,12}{9}}{4} \right\rceil .2 + \left\lceil \frac{\overset{3,15}{11}}{5} \right\rceil .1 + \overset{2,12}{2} = 12$

El algoritmo *RTA3* funciona de manera similar que *RTA2*, salvo por el cálculo de A_j^q. En la iteración 0, se calcula cada uno de los A_j^q. Durante la iteración 1, como $t_4^1 = 7$ es menor a $I_1^0 = 8$, no es necesario calcular A_1^1, por lo cual $I_1^1 = I_1^0$ y $A_1^1 = A_1^0$. Luego, como $A_2^1 \neq A_1^0$, entonces $t_4^1 = t_4^1 + A_2^1 - A_2^0 = 8$. El algoritmo continúa de la misma manera.

Como se puede observar en el ejemplo, el método de Sjödin calcula 15 invariantes, *RTA2* calcula 12 y *RTA3* sólo calcula 7. Sin embargo, para *RTA3*, cuando se calcula el tiempo de respuesta de la tarea 3, son calculados los valores de A_2^0, I_2^0 y A_3^0, I_3^0 y no es necesario calcularlos para el tiempo de respuesta de la tarea 4. Consecuentemente, solo es necesario calcular 5 invariantes cuando el método realiza un análisis de planificabilidad del *STR* completo.

El teorema presentado en la sección anterior, elimina la iteración 3 para *RTA3*. Dado que ningún I_j^q se incrementa, ningún A_j^q puede variar, y de esta manera se sabe que el proceso iterativo se encuentra en un *PF*.

IV. Resultados Experimentales

A continuación se presenta el desempeño del algoritmo *RTA3*, evaluado en conjunto con los métodos *RTA2* [19], Sjödin [13] y *HET* [18]. Se emplearon dos métricas. La primera consiste en el número promedio de invariantes calculados para determinar la planificabilidad de un *STR*. El invariante para los algoritmos evaluados es el número de operaciones techo o piso. La segunda métrica es el tiempo promedio que insume cada algoritmo en determinar la planificabilidad de un *STR*.

Las simulaciones fueron ejecutadas en dos plataformas. La primera plataforma son dos PC, con Windows 7 Professional (i7 2600) y Linux Ubuntu 12.04 LTS (i7 860). La segunda plataforma es una placa de desarrollo *mbed* NXP LPC1768, con un micro controlador basado en ARM Cortex-M3 (96 Mhz).

Se evaluó el algoritmo propuesto con *STR* de 10, 20, 50 y 100 tareas, con distribuciones uniformes (*DU*) y exponenciales en grupos (*DEG*), y factores de utilización de 70%, 75%, 80%, 82%, 84%, 85%, 86%, 88%, 90%, 92%, 94%, 96% y 98%. Para la *DU* se crearon cuatro grupos de *STR*, con periodos para las tareas entre 25 y 1000; 25 y 10000; 25 y 100000 y 25 y 1000000. Para la *DEG* se generaron grupos de 25 y 10000; 25 y 100000 y 25 y 1000000, con subgrupos de 25 a 100, 100 a 1000, 1000 a 10000, 10000 a 100000, distribuidos uniformemente, según corresponda, en cada grupo. Se generaron 10000 *STR* por cada *FU*, para ser simulados bajo la disciplina *RM*.

Para *x86*, a fin de minimizar la interferencia del Sistema Operativo (*SO*) en las mediciones de tiempo, se evaluó cada *STR* 100 veces, promediando el tiempo de ejecución, en nanosegundos (*nseg*). Las simulaciones se ejecutaron con la máxima prioridad que el *SO* permitía asignar. Los algoritmos fueron programados en *C*, y compilados con *GCC* 4.7.3.

Para las simulaciones en la placa de desarrollo *mbed*, cada sistema se computó 10 veces, promediando luego los tiempos de ejecución en microsegundos (μseg). Los algoritmos fueron programados en *C*, y compilados mediante Sourcery CodeBench Lite Edition (*GCC* 4.8.2), para plataformas ARM.

En ambas plataformas se empleó el máximo nivel de optimización que ofrece *GCC* (-O3).

Por motivos de espacio, se presentan los gráficos de las simulaciones para 100 tareas, pudiendo encontrarse en el anexo, en http://www.rtsg.unp.edu.ar el resto de las figuras.

1) *Tiempo promedio de ejecución en mbed LPC 1768*

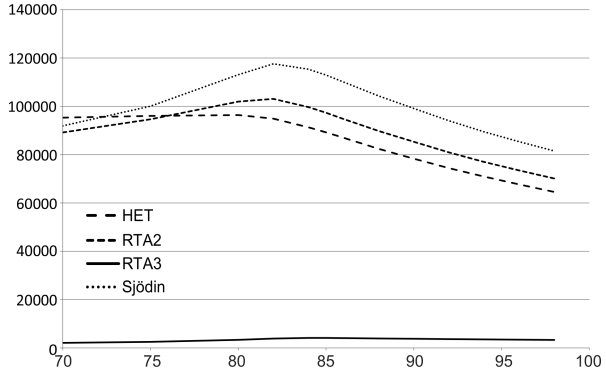

Figura 1. *mbed* LPC1768, 100 Tareas, *DU* 25-1K (μseg vs. *FU*).

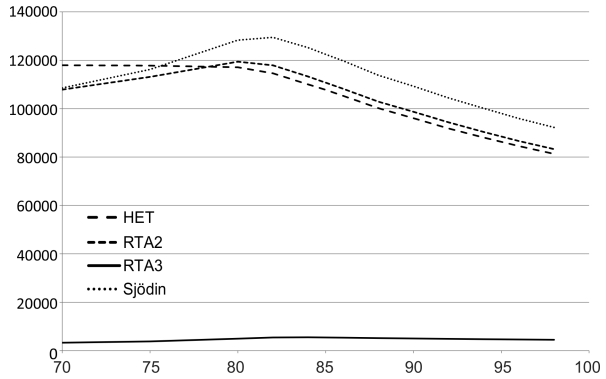

Figura 2. *mbed* LPC1768, 100 Tareas, *DU* 25-10K (μseg vs. *FU*).

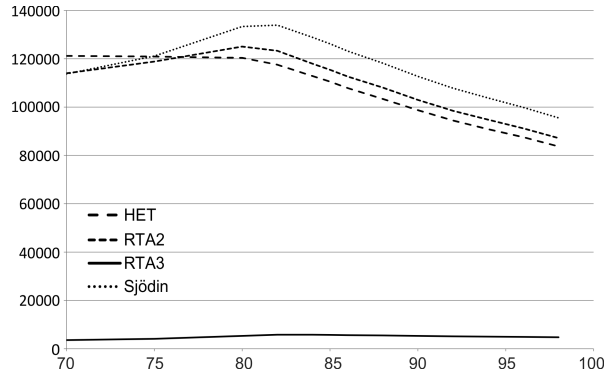

Figura 3. *mbed* LPC1768, 100 Tareas, *DU* 25-100K (*µseg* vs. *FU*).

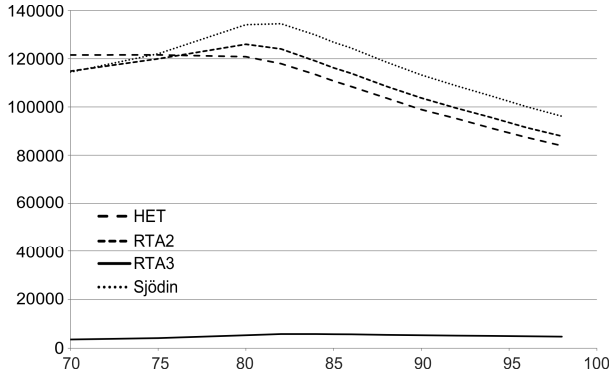

Figura 4. *mbed* LPC1768, 100 Tareas, *DU* 25-1000K (*µseg* vs. *FU*).

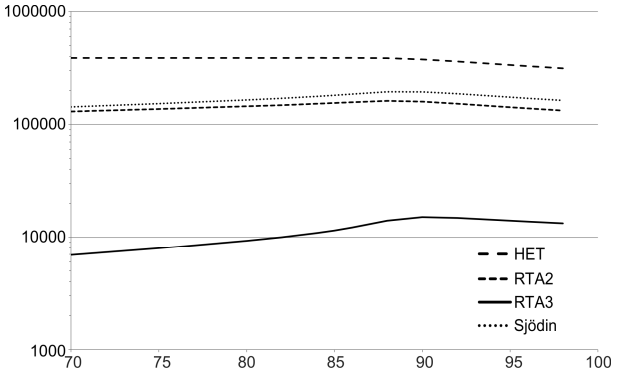

Figura 5. *mbed* LPC1768, 100 Tareas, *DEG* 25-10K (*µseg* vs. *FU*).

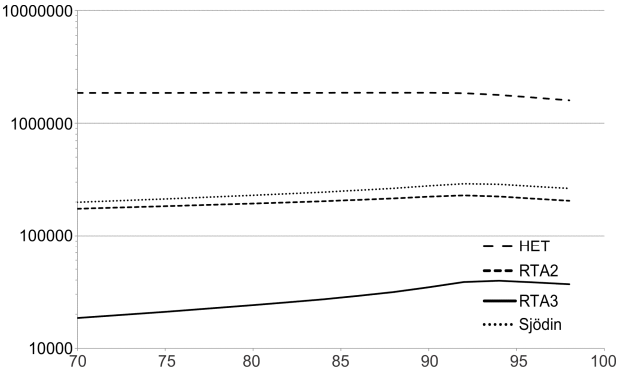

Figura 6. *mbed* LPC1768, 100 Tareas, *DEG* 25-100K (*µseg* vs. *FU*).

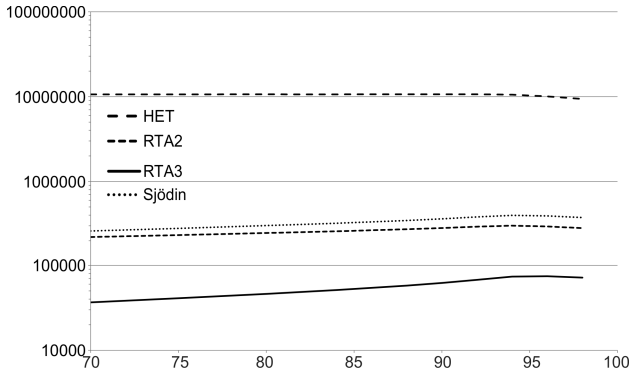

Figura 7. *mbed* LPC1768, 100 Tareas, *DEG* 25-1000K (*µseg* vs. *FU*).

2) *Tiempo promedio de ejecución en x86.*

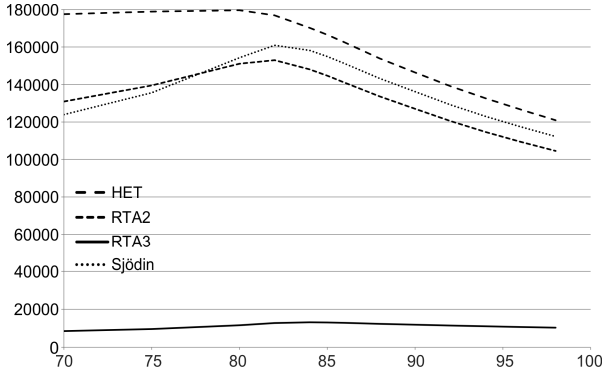

Figura 8. *x86*, 100 Tareas, *DU* 25-1K (*nseg* vs. *FU*).

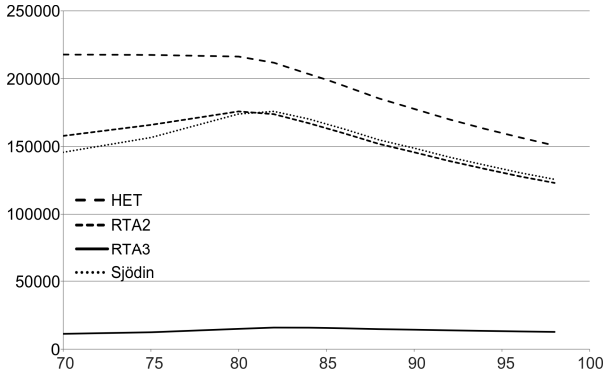

Figura 9. *x86*, 100 Tareas, *DU* 25-10K (*nseg* vs. *FU*).

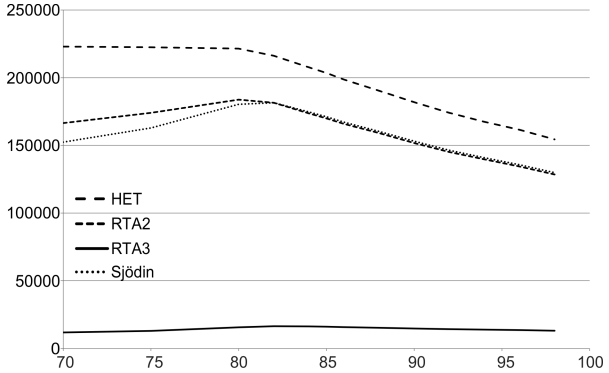

Figura 10. *x86*, 100 Tareas, *DU* 25-100K (*nseg* vs. *FU*).

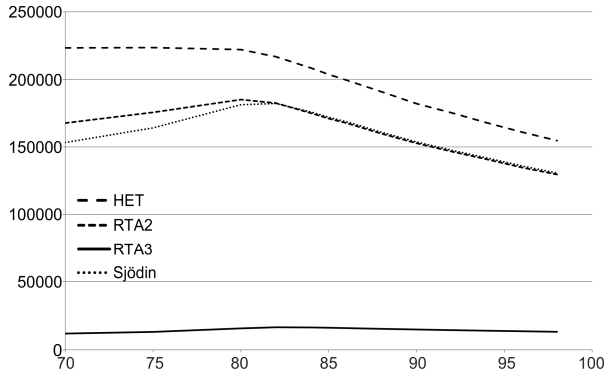

Figura 11. *x86*, 100 Tareas, *DU* 25-1000K (*nseg* vs. *FU*).

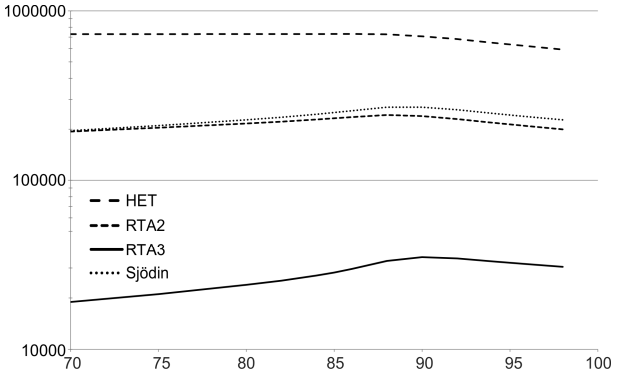

Figura 12. *x86*, 100 Tareas, *DEG* 25-10K (*nseg* vs. *FU*).

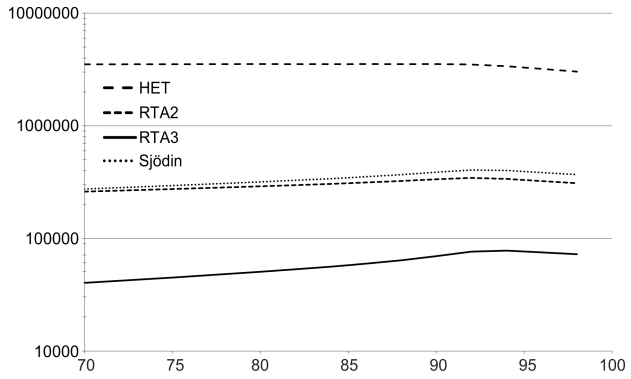

Figura 13. *x86*, 100 Tareas, *DEG* 25-100K (*nseg* vs. *FU*).

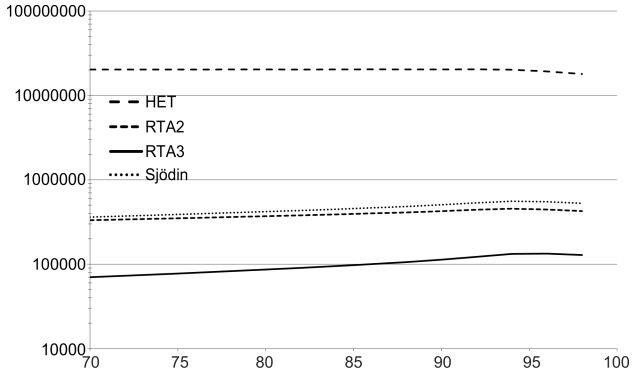

Figura 14. *x86*, 100 Tareas, *DEG* 25-1000K (*nseg* vs. *FU*).

3) *Número promedio de invariantes calculadas*

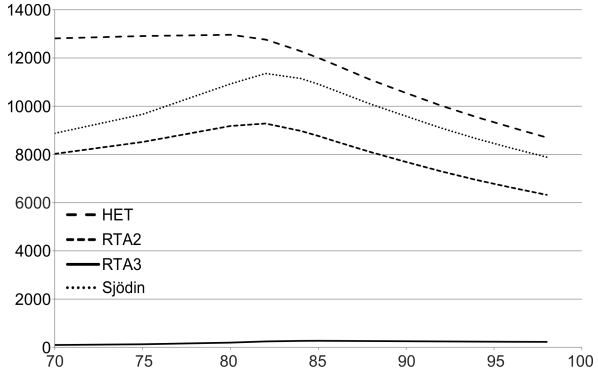

Figura 15. *mbed* LPC1768, 100 Tareas, *DU* 25-1K (*CC* vs. *FU*).

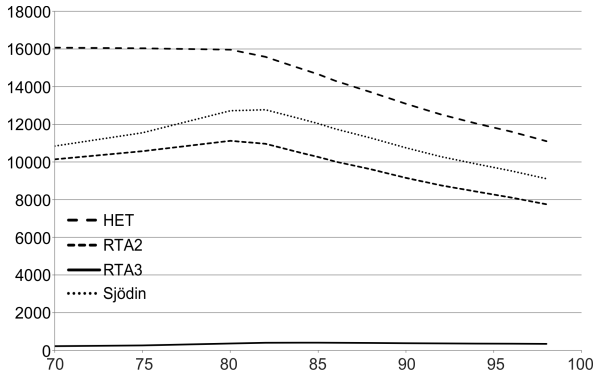

Figura 16. *mbed* LPC1768, 100 Tareas, *DU* 25-10K (*CC* vs. *FU*).

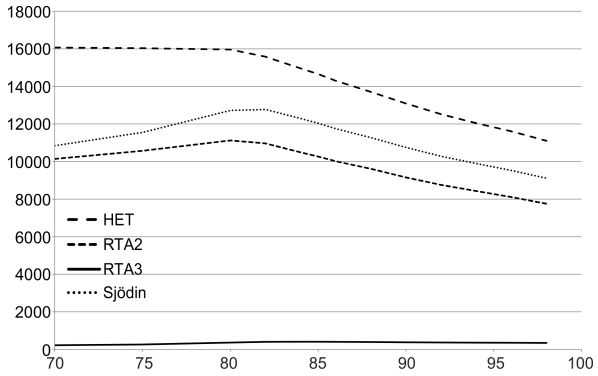

Figura 17. *mbed* LPC1768, 100 Tareas, *DU* 25-100K (*CC* vs. *FU*).

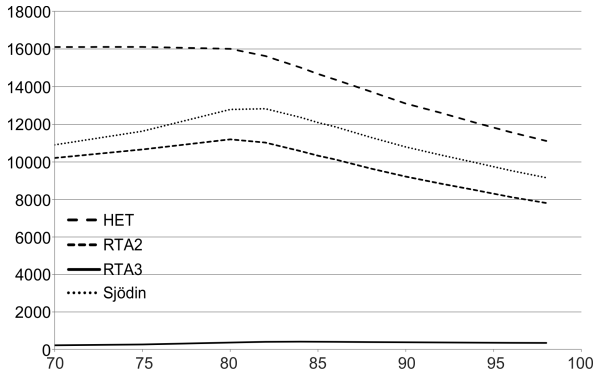

Figura 18. *mbed* LPC1768, 100 Tareas, *DU* 25-1000K (*CC* vs. *FU*).

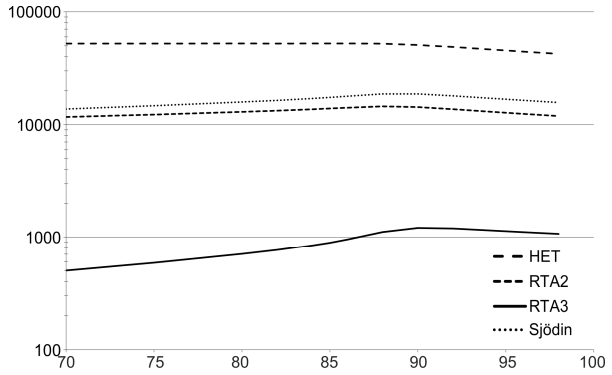

Figura 19. *mbed* LPC1768, 100 Tareas, *DEG* 25-10K (*CC* vs. *FU*).

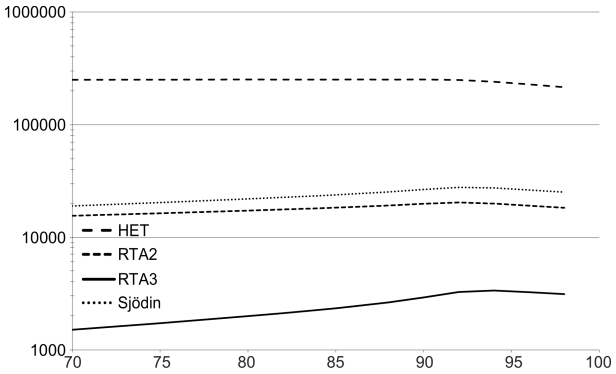

Figura 20. *mbed* LPC1768, 100 Tareas, *DEG* 25-100K (*CC* vs. *FU*).

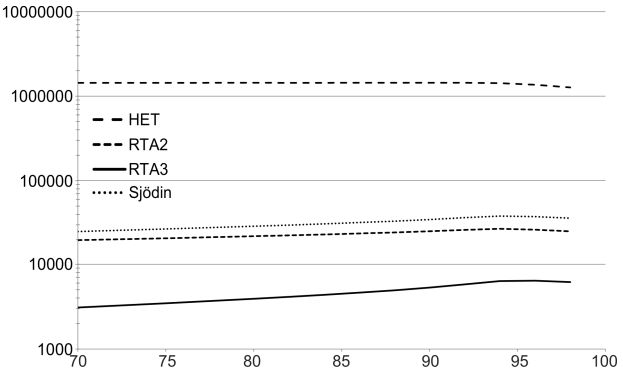

Figura 21. *mbed* LPC1768, 100 Tareas, *DEG* 25-1000K (*CC* vs. *FU*).

V. Análisis de los Resultados

En la sección anterior se exhiben los resultados del algoritmo propuesto en comparación con los más relevantes en la literatura. En estos, se puede ver el promedio del tiempo consumido por el test de planificabilidad por factor de utilización, sobre una placa *mbed* y *x86*, y el *CC* medido en invariantes calculados, para los diversos sistemas generados.

Es claro que todas las simulaciones siguen el mismo patrón de curvas. Para *RTA2*, *RTA3* y *Sjödin*, se incrementa el *CC* hasta cierto punto, en donde empiezan a encontrar sistemas que resultan no planificables al analizar las últimas tareas, lo que conlleva a que el método se detenga y el costo en invariantes y temporal se vea reducido por ello.

A continuación, se presenta un análisis asintótico del *CC* temporal, buscando determinar cuál es el orden computacional promedio de los algoritmos simulados.

En complejidad computacional, lo deseable es obtener algoritmos que resuelvan problemas en un orden *constante* $O(1)$, *logarítmico* $O(\log n)$ o *lineal* $O(n)$. Sin embargo, esto no siempre es posible. Algoritmos que resuelvan con un orden $O(n*log(n))$ o $O(n^2)$ son también soluciones aceptables. No obstante, cuando el algoritmo debe trabajar en tiempo de ejecución, cuanto menor sea el *orden computacional*, menor es la sobrecarga que se introduce al sistema.

Teniendo como base a los conjuntos de 10 tareas, a continuación se determina si el crecimiento promedio del *CC* temporal de los sistemas simulados, para los conjuntos de 20, 50 y 100 tareas, sigue un patrón de orden computacional conocido o bien se encuentra acotado.

Sea P_n el promedio resultante del *CC* temporal del conjunto de n tareas (con n = 20, 50, 100), divido por el *CC* temporal del conjunto de 10 tareas, para cada U.

Se considera que los algoritmos siguen un crecimiento de orden $O(n*log(n))$ o $O(n^2)$, a partir del conjunto de 10 tareas. En consecuencia, si se sigue alguno de estos órdenes, el resultado puede ser expresado como una función de n, donde k es una constante:

$$P_n/P_{10} = k.n.\log(n) \quad \text{o} \quad P_n/P_{10} = k.n^2$$

Si se despeja el valor de k, ésta debe mantenerse constante para los conjuntos de 20, 50 y 100 tareas para poder afirmar que el orden computacional promedio sigue alguno de los dos presentados.

$$k_{\log} = P_n/(P_{10}.n.\log(n)) \quad \text{o} \quad k_{n^2} = P_n/(P_{10}.n^2)\,.$$

En la Tabla I se presentan los resultados del análisis asintótico para *mbed*, y en la Tabla 2 para *x86*.

Del análisis asintótico, se puede observar que los métodos *RTA2* y *Sjödin* siguen un $O(n^2)$, dado que se ajustan a la función cuadrática con muy buena precisión. Por otro lado, *HET* tiene un $O(n^2)$ o mayor. A juicio de los autores, esto se debe a que su funcionamiento es mediante la inspección de las posibilidades de instanciación de las tareas, lo cual crea un árbol de inspección, que llega a ser muy grande cuando existen tareas con distintos órdenes de magnitud en el mismo sistema. Es por ello que, para las distribuciones exponenciales, y por grupos de tareas de distintos órdenes de magnitud, su eficiencia se ve disminuida. Por otro lado, se hace notar que, para los mismos sistemas, el método *HET* tiene una mejora en el tiempo de ejecución promedio en la plataforma *mbed*, al emplear los niveles de optimización de *GCC*, que en *x86*.

El análisis asintótico del algoritmo *RTA3*, no se ajusta siempre a ninguna de las dos funciones de manera precisa. Sin embargo, es claro que es mayor o igual a $O(n*log(n))$ y menor a $O(n^2)$, llegando a ajustarse para el caso *DU* 25-1K para *mbed* a un $O(n*log(n))$.

TABLA I. ANÁLISIS ASINTÓTICO – MBED.

Métodos	*RTA3*			*RTA2*			*HET*			*Sjodin*		
Conjuntos	P_n	$k_{\log}$	k_{n^2}	P_n	$k_{\log}$	k_{n^2}	P_n	$k_{\log}$	k_{n^2}	P_n	$k_{\log}$	k_{n^2}
DU 25-1K 20	2.6177	0.1006	0.0065	4.1069	0.1578	0.0103	4.2769	0.1644	0.0107	4.1267	0.1586	0.0103
DU 25-1K 50	8.6197	0.1015	0.0034	25.5380	0.3006	0.0102	26.1839	0.3082	0.0105	25.8467	0.3043	0.0103
DU 25-1K 100	20.3547	0.1018	0.0020	100.4403	0.5022	0.0100	97.4119	0.4871	0.0097	103.0482	0.5152	0.0103
DU 25-10K 20	2.6866	0.1032	0.0067	4.1688	0.1602	0.0104	4.3995	0.1691	0.0110	4.1739	0.1604	0.0104
DU 25-10K 50	9.4624	0.1114	0.0038	26.4080	0.3109	0.0106	28.1229	0.3311	0.0112	26.3239	0.3099	0.0105
DU 25-10K 100	24.1808	0.1209	0.0024	105.0169	0.5251	0.0105	111.3755	0.5569	0.0111	104.4915	0.5225	0.0104
DU 25-100K 20	2.6993	0.1037	0.0067	4.1857	0.1609	0.0105	4.4193	0.1698	0.0110	4.1877	0.1609	0.0105
DU 25-100K 50	9.6408	0.1135	0.0039	26.8469	0.3160	0.0107	28.5136	0.3357	0.0114	26.6759	0.3140	0.0107
DU 25-100K 100	24.9880	0.1249	0.0025	107.7875	0.5389	0.0108	113.5259	0.5676	0.0114	106.3657	0.5318	0.0106
DU 25-1000K 20	2.7004	0.1038	0.0068	4.1909	0.1611	0.0105	4.4181	0.1698	0.0110	4.1900	0.1610	0.0105
DU 25-1000K 50	9.6515	0.1136	0.0039	26.9249	0.3170	0.0108	28.5497	0.3361	0.0114	26.7112	0.3144	0.0107
DU 25-1000K 100	25.0278	0.1251	0.0025	108.2235	0.5411	0.0108	113.7857	0.5689	0.0114	106.5796	0.5329	0.0107
DEG 25-10K 20	2.8091	0.1080	0.0070	4.0178	0.1544	0.0100	6.2188	0.2390	0.0155	4.0895	0.1572	0.0102
DEG 25-10K 50	10.3346	0.1217	0.0041	25.0031	0.2943	0.0100	43.6500	0.5138	0.0175	25.8290	0.3041	0.0103
DEG 25-10K 100	26.1819	0.1309	0.0026	99.5738	0.4979	0.0100	172.0286	0.8601	0.0172	104.7431	0.5237	0.0105
DEG 25-100K 20	2.9980	0.1152	0.0075	3.9782	0.1529	0.0099	11.0450	0.4245	0.0276	4.0622	0.1561	0.0102
DEG 25-100K 50	13.0495	0.1536	0.0052	24.7478	0.2913	0.0099	106.5918	1.2548	0.0426	25.8936	0.3048	0.0104
DEG 25-100K 100	38.3946	0.1920	0.0038	98.8979	0.4945	0.0099	450.3865	2.2519	0.0450	105.6293	0.5281	0.0106
DEG 25-1000K 20	3.4010	0.1307	0.0085	4.0746	0.1566	0.0102	27.8903	1.0719	0.0697	4.2372	0.1628	0.0106
DEG 25-1000K 50	15.3263	0.1804	0.0061	24.7696	0.2916	0.0099	406.3678	4.7837	0.1625	26.3388	0.3101	0.0105
DEG 25-1000K 100	49.6275	0.2481	0.0050	99.3483	0.4967	0.0099	1771.3777	8.8569	0.1771	107.6836	0.5384	0.0108

TABLA II. ANÁLISIS ASINTÓTICO – *X86*.

Métodos	*RTA3*			*RTA2*			*HET*			*Sjodin*		
Conjuntos	P_n	$k_{\log}$	k_{n^2}	P_n	$k_{\log}$	k_{n^2}	P_n	$k_{\log}$	k_{n^2}	P_n	$k_{\log}$	k_{n^2}
DU 25-1K 20	0.7398	0.0284	0.0018	3.3494	0.1287	0.0084	4.2955	0.1651	0.0107	3.5011	0.1346	0.0088
DU 25-1K 50	2.6643	0.0314	0.0011	20.1100	0.2367	0.0080	25.2594	0.2973	0.0101	20.9137	0.2462	0.0084
DU 25-1K 100	6.8417	0.0342	0.0007	78.3600	0.3918	0.0078	92.5728	0.4629	0.0093	82.2072	0.4110	0.0082
DU 25-10K 20	0.7791	0.0299	0.0019	3.5034	0.1346	0.0088	4.3674	0.1678	0.0109	3.6049	0.1385	0.0090
DU 25-10K 50	2.9498	0.0347	0.0012	21.6707	0.2551	0.0087	27.0802	0.3188	0.0108	21.8845	0.2576	0.0088
DU 25-10K 100	7.9015	0.0395	0.0008	85.2856	0.4264	0.0085	105.8194	0.5291	0.0106	85.5702	0.4279	0.0086
DU 25-100K 20	0.7914	0.0304	0.0020	3.5651	0.1370	0.0089	4.3883	0.1686	0.0110	3.6572	0.1405	0.0091
DU 25-100K 50	3.0266	0.0356	0.0012	22.3484	0.2631	0.0089	27.4786	0.3235	0.0110	22.4353	0.2641	0.0090
DU 25-100K 100	8.1731	0.0409	0.0008	88.7811	0.4439	0.0089	107.9347	0.5397	0.0108	88.1050	0.4405	0.0088
DU 25-1000K 20	0.7939	0.0305	0.0020	3.5803	0.1376	0.0090	4.3875	0.1686	0.0110	3.6707	0.1411	0.0092
DU 25-1000K 50	3.0370	0.0358	0.0012	22.4739	0.2646	0.0090	27.5061	0.3238	0.0110	22.5358	0.2653	0.0090
DU 25-1000K 100	8.2022	0.0410	0.0008	89.3907	0.4470	0.0089	108.1568	0.5408	0.0108	88.5699	0.4428	0.0089
DEG 25-10K 20	0,6281	0,0241	0,0016	2,1516	0,0827	0,0054	6,2290	0,2394	0,0156	2,3068	0,0887	0,0058
DEG 25-10K 50	2,4715	0,0291	0,0010	13,0727	0,1539	0,0052	42,0256	0,4947	0,0168	14,0869	0,1658	0,0056
DEG 25-10K 100	6,6855	0,0334	0,0007	51,6434	0,2582	0,0052	163,1925	0,8160	0,0163	56,4486	0,2822	0,0056
DEG 25-100K 20	0,5686	0,0219	0,0014	1,6355	0,0629	0,0041	11,4556	0,4402	0,0286	1,7947	0,0690	0,0045
DEG 25-100K 50	2,5482	0,0300	0,0010	9,9924	0,1176	0,0040	106,0161	1,2480	0,0424	11,1463	0,1312	0,0045
DEG 25-100K 100	7,7296	0,0386	0,0008	39,5979	0,1980	0,0040	440,8387	2,2042	0,0441	44,9580	0,2248	0,0045
DEG 25-1000K 20	0,6200	0,0238	0,0015	1,5080	0,0580	0,0038	29,2490	1,1241	0,0731	1,7157	0,0659	0,0043
DEG 25-1000K 50	2,8423	0,0335	0,0011	9,0098	0,1061	0,0036	417,1601	4,9107	0,1669	10,4191	0,1227	0,0042
DEG 25-1000K 100	9,2986	0,0465	0,0009	35,8310	0,1792	0,0036	1788,5722	8,9429	0,1789	42,0844	0,2104	0,0042

VI. CONCLUSIONES Y TRABAJOS FUTUROS

Se presentó un nuevo *test de planificabilidad* para prioridades fijas (*RM* o *DM*), mediante el cálculo del peor caso de tiempo de respuesta, que mejora en eficiencia a los métodos tradicionales en el *CC* asociado a su cálculo. Este método es

factible de implementación en tiempo de ejecución, para diversas funciones, como la aceptación de tareas de tiempo real que arriben para ser atendidas de manera dinámica, tolerancia a las fallas, ejecución de servidores asincrónicos, etc. Además, algunos métodos de *Slack Stealing* [23] utilizan básicamente el mismo algoritmo iterativo, con lo cual los resultados del presente trabajo también aplican a éstos, y será una futura línea de investigación.

AGRADECIMIENTOS

A Joe Bungo y al *ARM University Program* por su aporte de placas *mbed* y la inclusión en el programa.

REFERENCES

[1] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Enviroments," in *IEEE Real-Time Systems Symposium*, 1987, pp. 261-270.

[2] B. Sprunt, J. P. Lehoczky, and L. Sha, "Exploiting Unused Periodic Time For Aperiodic Service Using The Extended Priority Exchange Algorithm," in *IEEE Real-Time Systems Symposium*, Huntsville, Alabama, USA, 1988, pp. 251-258.

[3] L. Sha, B. Sprunt, and J. P. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems," *The Journal of Real-Time Systems,* vol. 1, pp. 27-69, 1989.

[4] G. Fohler, T. Lennvall, and G. Buttazzo, "Inproved Handling of Soft Aperiodic Tasks in Offline Scheduled Real-Time Systems using Total Bandwidth Server," in *8th IEEE International Conference on Emerging Technologies & Factory Automation*, Nice France, 2001, pp. 151-157.

[5] 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.

[6] 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.

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

[8] T.-S. Tia, J. W. Liu, and M. Shankar, "Aperiodic Request Scheduling in Fixed-Priority Preemptive Systems," Department of Computer Science, University of Illinois, Internal Report UIUCDCS-R-94-1859, 1994.

[9] 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.

[10] R. I. Davis, "Approximate Slack Stealing Algorithms for Fixed Priority Pre-Emptive Systems," Real-Time Systems Research Group, University of York, York, England, Internal Report 1994.

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

[12] J. W. S. Liu, *Real-Time Systems*: Prentice Hall, 2000.

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

[14] 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.

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

[16] 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.

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

[18] E. Bini and C. B. Giorgio, "Schedulability Analysis of Periodic Fixed Priority Systems," *IEEE Trans. on Computers,* vol. 53, pp. 1462-1473, November 2004.

[19] 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," *Journal of Computer Science & Technology,* vol. 9, pp. 72-81, 2009.

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

[21] J. M. Urriza, R. Cayssials, and J. D. Orozco, "Modelado de Sistemas de Tiempo Real Planificados por RM o DM: Caracterización y Análisis," in *XXXIV Conferencia Latinoamericana de Informática, CLEI 2008*, Santa Fe, Argentina, 2008, pp. 1435-1444.

[22] Z. Manna, S. Ness, and J. Vuillemin, "Inductive Methods for Proving Properties of Programs," *Communications of the ACM,* vol. 16, pp. 491-502, August 1973.

[23] 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, pp. 660-667, 2010.

José M. Urriza received the Electronic Engineering degree and the Doctorate in Engineering from the Universidad Nacional del Sur in 2000 and 2008 respectively. His research focuses in the schedulability of heterogeneous Real-Time Systems. He teaches Networks and Data Transmission, and Embedded and Real-Time Systems in the Universidad Nacional de la Patagonia San Juan Bosco (UNPSJB). He is founder and director of the Real Time System Group in UNPSJB.

Francisco E. Páez received his degree in Computer Science from the Universidad Nacional de la Patagonia San Juan Bosco (UNPSJB) in 2010. He teaches as Assistant in Operating Systems, and Embedded and Real-Time Systems. He is part of Real Time System Group in the UNPSJB. His areas of interest include real-time systems, operating systems and process and memory management.

Javier D. Orozco received the Electrical Engineering degree and the Doctorate in Engineering from the Universidad Nacional del Sur in 1984 and 1998 respectively. His research is focused on the scheduling of real-time systems. He teaches Logical Design at the under-graduate level and is co-professor of Real Time Open Dynamic Systems at the graduate level. He is presently the Head of the RTS group in Universidad Nacional del Sur.

Ricardo Cayssials received the Electrical Engineering degree and the Doctorate in Engineering from the Universidad Nacional del Sur in 1993 and 1999 respectively. In 2001, he was granted a postdoctoral scholarship by The University of York, England, which visited again at the beginning of 2003. His main research area is the schedulability analysis of hard and weakly real time for both multi-user-single-resource and multi-user-multi-resource systems. His second research area is Programmable Logic. Because of his work, the Digital Systems Laboratory is a member of the ALTERA University Program. He teaches Design of Highly Complex Logical Circuits.