Cálculo del Peor Tiempo de Respuesta en Sistemas de Tiempo Real con Limitados Recursos Computacionales

José M. Urriza[1], Javier D. Orozco[1,2], Ricardo Cayssials[1,2]

Universidad Nacional del Sur[1] / CONICET[2]
Bahía Blanca, Argentina
{jurriza@criba.edu.ar}

Resumen: La necesidad de reducir los costos computacionales de los Sistema Operativos de Tiempo Real (*RTOS*) se ha incrementado, aún cuando los microprocesadores son cada vez más potentes. Es continua la aparición de nuevos dispositivos de propósito dedicado que cuentan con un *RTOS* para poder administrar los recursos requeridos por las aplicaciones que soportan. A fin de garantizar la planificabilidad de un sistema o bien garantizar la admisibilidad de una nueva tarea en un sistema dinámico de tiempo real, resulta necesario conocer el tiempo de respuesta del sistema crítico de tiempo real. En este trabajo se aborda la reducción del costo computacional en el algoritmo de dicho cálculo.

Palabras Clave: RTS, Schedulability, Response Time Analysis, *RM*, *DM*.

1 Introducción

Existen numerosos Sistemas Operativos de Tiempo Real (*RTOS*). Entre los más conocidos podemos nombrar a: *ChibiOS/RT, eCos, FreeRTOS, Fusion RTOS, Nucleus RTOS, OSE, OSEK, QNX, RT-11, RTEMS, RTLinux, Talon DSP RTOS, Transaction Processing Facility y VxWorks*, entre otros.

El más difundido, probablemente, es el desarrollado por *Wind River Systems*, *VxWorks*, por ser utilizado en: *The Spirit and Opportunity Mars Exploration Rovers, The Mars Reconnaissance Orbiter, The Deep Impact space probe, Stardust (spacecraft), The Boeing 787 airliner, The Boeing 747-8 airliner, The BMW iDrive system, Linksys WRT54Gxx wireless routers*, por mencionar algunos dispositivos que cuentan con el *RTOS* de esta empresa. Entre sus principales atributos está la sencillez del mismo y que puede garantizar que un Sistema de Tiempo Real (*STR*) duro pueda ser planificado por la disciplina de prioridades fijas *Rate Monotonic* (*RM*).

Para validar la planificabilidad del conjunto de tareas, este *RTOS* acota el máximo factor de utilización admisible para el sistema, mediante una cota superior presentada en 1973 por Liu & Layland en [1], cuya principal ventaja es poseer un costo

computacional (*CC*) despreciable a cambio de proveer resultados muy pesimistas lo que redundan en un sobredimencionamiento del sistema. La elección de este método por *Wind River Systems* ([2]) probablemente se debe precisamente a su bajo *CC* y a que, los nuevos métodos desarrollados poseen un *CC* que en comparación con Liu & Layland es elevado. Si bien esta aseveración es real, también revela un campo fértil para explorar la disciplina a fin de obtener técnicas de análisis con bajo *CC* que, aún cuando resulten más elevados que la cota de Liu & Layland, este incremento se vea compensado por un mejor aprovechamiento de los recursos existentes o bien por un mejor tratamiento de sistemas con requerimientos más complejos.

A continuación se realiza una breve introducción a los *STR*. Se puede definir informalmente a un *STR* como aquél que necesita terminar una tarea o trabajo, antes de un determinado tiempo denominado vencimiento. Estos se caracterizan por poseer entre sus requerimientos funcionales el tiempo. Luego, es aceptada por la comunidad de la disciplina la definición de Stankovic ([3]) 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"*.

Dependiendo del vencimiento de las tareas, los *STR* 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*. El segundo permite que se pierda algunos vencimientos, por lo cual se los denominan *blandos*. Por último, la mayor difusión de los *STR* ha requerido tipificar a aquellos que permiten sólo una determinada cantidad de pérdidas especificadas bajo algún criterio estadístico. Estos son denominados *firmes.*

En los *STR duros*, la pérdida de un vencimiento en una tarea puede tener consecuencias graves para la integridad del sistema y posiblemente de su entorno en el caso de que interactúe fuertemente con el mismo: aviónica, robótica, etc. A fin de garantizar que todas las tareas terminen antes de su vencimiento es necesario realizar un análisis de la *planificabilidad* del sistema. Si el test es exitoso, se dice que el sistema es *planificable* y se garantiza el cumplimiento de todas sus *constricciones temporales*.

En [1], se considera la planificabilidad de sistemas mono-recurso y multitarea. Se entiende como mono-recurso por ejemplo a un único microprocesador o un medio físico de transmisión, el cual sólo puede ser utilizado por una tarea o mensaje 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,* establece un orden lineal sobre el 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 y sus interdependencias. Usualmente se considera que las tareas son 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 **S**(n) de n tareas se encuentra especificado por **S**(n)={(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)}.

En [1] se demostró que el peor esquema de generación, para un sistema mono-recurso, es aquél en el cual todas las tareas solicitan ser ejecutadas en el mismo instante 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* **S**(n) determina el nivel de utilización del recurso.

En [1], se garantiza la planificabilidad del *STR* mediante el cálculo de una cota, cuyo *CC* es del orden al número de tareas. Si el *FU* resulta menor o igual a la cota mencionada, se garantiza la planificabilidad del *STR*.

El cálculo de la misma se define por: $L_n = n.(2^{1/n} - 1)$ con $FU < L_n$

Se observa que la expresión anterior tiende al logaritmo natural de 2 (ln 2 = 0,6931…) para un número elevado de tareas. Esto nos indica que un *STR* con un factor de utilización mayor puede aun ser planificable pero no se puede garantizar su planificabilidad por este método.

Se han desarrollado números test de planificabilidad desde 1973. En 1982, Leung ([4]), define *Deadline Monotonic* (*DM*), pero no se define un test de planificabilidad y en 1991 Audsley ([5]) presenta una solución a este problema. En 1986, Joseph y Pandya exponen un método de iterativo de *Punto Fijo* (*PF*) para evaluar una condición necesaria y suficiente que valide la factibilidad del *STR* utilizando un planificador por *RM* ([6]). Diversos trabajos han sido publicados con soluciones equivalentes en [7, 8, 9, 10]. En 1998, Sjödin ([11]) incorpora una mejora al test de Joseph que consistente en comenzar la iteración de la tarea i+1 en el instante en donde el método de Joseph encontró el peor tiempo de respuesta de la tarea i, más el tiempo de ejecución de la tarea i+1.

En 2004, Bini ([12]) propone un nuevo método denominado Hyperplanes Exact Test (HET) para determinar la planificabilidad de un *STR*. En dicho test, si bien parte de un novedoso método, las pruebas realizadas para contrastar con el método presentado en este trabajo exhiben *CC* mayores, aún al método de Sjödin ([11]), cuando se lo evalúa de la misma forma que la empleada en el presente trabajo.

En este trabajo se aborda la reducción del costo computacional en el algoritmo para determinar la planificabilidad del *STR* y se compara su *CC* con los algoritmos propuestos en la literatura de tiempo real.

El trabajo está organizado de la siguiente manera: en la Sección 2 se describen los métodos iterativos de *PF* utilizados para determinar la planificabilidad de un sistema de tiempo real y se prueban teoremas para mejorar la búsqueda de dichos *PFs*. En la Sección 3 se compara el *CC* del método propuesto con el *CC* de los algoritmos propuestos en la literatura de tiempo real. Las conclusiones son presentadas en la Sección 4.

2 Métodos Iterativos de Punto Fijo

Desde 1986, la mayoría de los tests de planificabilidad desarrollados para las disciplinas de prioridades fijas *RM* o *DM*, se basan en aplicar un método de *punto fijo* para garantizar de forma necesaria y suficiente la factibilidad del sistema.

Por definición, un *PF* de una función f, es un número *t*, tal que $t = f(t)$. Como en este caso la ecuación de *PF* es función del tiempo, un *punto t* y un *instante t*, son expresiones equivalentes.

El método de *PF*, para un *STR*, fue desarrollado por primera vez por Joseph y Pandya ([6]). En éste, se prueba que no existe una construcción analítica que resuelva este tipo de problemas y sólo es posible calcularlo mediante un cálculo iterativo.

El método de Joseph se inicializa en *el peor estado de carga* ([1]) ($t = 0$) donde todas las tareas se instancian simultáneamente. Como se demuestra en [6], el resultado que se obtiene es el peor *tiempo de respuesta* de la tarea *i*, de un subconjunto de tareas **S**(*i*). A continuación se presenta la misma ecuación con sólo un cambio de nomenclatura y el agregado de un superíndice (*q*) para indicar en qué iteración se encuentra:

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

La solución, si existe, es sólo válida para un *STR* del tipo *duro* planificado por *RM* o *DM*, cuando se encuentra un *PF* ($t_i^{q+1} = t_i^q$) y si el tiempo de respuesta para atender la tarea *i* es igual o menor que el vencimiento de la misma ($t_i^q \le D_i$).

2.1 Algoritmo Iterativo

A continuación se presenta el análisis del método iterativo de Joseph a fin de proponer a partir de este análisis, un algoritmo que obtengan el mismo *PF* con un *CC* menor.

Supongamos un *STR* planificable $S(n) = \{(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)\}$. Cuando se aplica el método de [6] a una tarea *i*, con $i \in 1 < i \le n$, ocurrirán $m+1$ iteraciones antes de llegar a un *PF*, con $m \ge 0$. A continuación se realiza una traza genérica:

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

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

Dado que es una ecuación de *PF* y todo *PF* es un atractor, si el subsistema S(i) es planificable, la ecuación posee como mínimo un *PF*. El proceso iterativo evolucionará hacia el primer *PF* en la cadena ascendente, como lo demuestra el Teorema de Kleene y en [13]. Alcanzado el *PF* en t^m, el resultado de la iteración *m*-1 (t^m) debe ser igual al resultado de la iteración *m* (t^{m+1}).

A continuación se realiza un análisis de la ecuación $\lceil t/T_j \rceil C_j$ y de las condiciones en las cuales el método de *PF* finaliza. El cálculo de $\lceil t/T_j \rceil$ determina la cantidad de instanciaciones de la tarea *j* en el intervalo [0,*t*). Luego, $\lceil t/T_j \rceil C_j$ indica el tiempo de ejecución acumulado de todas las instancias de la tarea *i* hasta el tiempo t , también denominado carga de trabajo de la tarea *j* ($W_j(t)$). Por lo tanto, bajo la premisa de que el planificador no puede dejar el recurso ocioso mientras existan instancias de tareas para ejecutar, la ecuación encontrará un *PF* cuando la sumatoria de todos los tiempos de ejecución de todas las instancias de las tareas, sea igual al tiempo necesario para ejecutarlas.

Para obtener el tiempo donde tendrá lugar la próxima instanciación de la tarea *j*, después del tiempo t , se debe multiplicar a $\lceil t/T_j \rceil$ por el período de la tarea (T_j). Si se calcula $\lfloor t/T_j \rfloor T_j$ se obtiene el tiempo donde arribó la instancia más cercana, que corresponde al tiempo t . Por lo tanto, en el intervalo $[\lfloor t/T_j \rfloor T_j, \lceil t/T_j \rceil T_j)$, el cálculo de la ecuación $\lceil t/T_j \rceil C_j$ con cualquier t dentro de dicho intervalo, posee siempre el mismo valor.

Debido a lo analizado, es conveniente indicar a la carga de trabajo de la tarea *j* en el tiempo *t* de la iteración *q* como:

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

Si se reemplaza en el desarrollo de la traza, partiendo de un valor de semilla t^0 , nos queda:

$$t^1 = A_1^0 + .. + A_{i-1}^0 + C_i \ \ldots \qquad t^m = A_1^{m-1} + .. + A_{i-1}^{m-1} + C_i \qquad t^{m+1} = A_1^m + .. + A_{i-1}^m + C_i$$

El desarrollo a partir de la iteración 1, reemplazado en cada uno de los cálculos de los A_j^q , quedan en función de las cargas de trabajos de todas las tareas en la iteración 0:

Iteración 1:	Iteración 2:	Iteración *m:*
$A_1^1 = \left\lceil \frac{A_1^0 + .. + A_{i-1}^0 + C_i}{T_1} \right\rceil C_1$	$A_1^2 = \left\lceil \frac{A_1^1 + .. + A_{i-1}^1 + C_i}{T_1} \right\rceil C_1$	$A_1^m = \left\lceil \frac{A_1^{m-1} + .. + A_{i-1}^{m-1} + C_i}{T_1} \right\rceil C_1$
$A_2^1 = \left\lceil \frac{A_1^0 + .. + A_{i-1}^0 + C_i}{T_2} \right\rceil C_2 \ \ldots$	$A_2^2 = \left\lceil \frac{A_1^1 + .. + A_{i-1}^1 + C_i}{T_2} \right\rceil C_2 \ \ldots$	$A_2^m = \left\lceil \frac{A_1^{m-1} + .. + A_{i-1}^{m-1} + C_i}{T_2} \right\rceil C_2 \ \ldots$
$A_{i-1}^1 = \left\lceil \frac{A_1^0 + .. + A_{i-1}^0 + C_i}{T_{i-1}} \right\rceil C_{i-1}$	$A_{i-1}^2 = \left\lceil \frac{A_1^1 + .. + A_{i-1}^1 + C_i}{T_{i-1}} \right\rceil C_{i-1} \ \ldots$	$A_{i-1}^m = \left\lceil \frac{A_1^{m-1} + .. + A_{i-1}^{m-1} + C_i}{T_{i-1}} \right\rceil C_{i-1}$

El algoritmo se detiene cuando se ha encontrado un *PF* en el cual, en el intervalo $[t^{m-1}, t^m]$, los pares A_j^{m-1}, A_j^m con $1 \le j \le i-1$ son iguales, debido que se cumple que $t = f(t)$ y consecuentemente

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

Si el *STR* no fuera *factible* para ser planificado por *RM* o *DM*, el algoritmo o bien no converge o su convergencia se encuentra en un tiempo mayor al vencimiento de la tarea analizada. Por lo tanto, el intervalo de inspección del algoritmo se reduce desde el *instante crítico* al vencimiento ($(0, D_i]$).

2.2 Mejoras al algoritmo iterativo.

En la sección anterior se establece que, si la Ecuación (1) tiene una solución (*PF*) las

iteraciones *m*-1 y *m* tendrán los pares A_j^{m-1}, A_j^m con $1 \le j \le i-1$ iguales. Por lo tanto, si no se cumple esta condición, el algoritmo realizará al menos una iteración más para encontrar el *PF*.

En el algoritmo de Joseph, en cada iteración de la Ecuación (1), la inicialización de la variable *t*, se realiza con el valor obtenido en la iteración anterior. Es por ello que A_j^q no afecta al cálculo de los $A_{j+1}^q,....,A_{i-1}^q$. Fundamentalmente, entre cada iteración, la variable *t*, se inicializa con la sumatoria de:

$t^{q+1} = t^q + \sum_{j=1}^{i-1} A_j^q - A_j^{q-1}$ para $q > 0$ que sale de tomar a la Ecuación (1) en dos iteraciones sucesivas:

$$t^{q+1} - C_i - \sum_{j=1}^{i-1} A_j^q = 0 \quad \text{y} \quad t^q - C_i - \sum_{j=1}^{i-1} A_j^{q-1} = 0$$

Consecuentemente, el método requiere de sucesivas iteraciones para que los cambios que ocurren en la iteración actual afecten al cálculo de los siguientes $A_{j+1}^q,....,A_{i-1}^q$. Ejemplificando, en las ecuaciones donde se presenta la traza (Iteración 1), no se utiliza el calculo de A_1^1 en el calculo de A_2^1 y así sucesivamente, pero sí se utilizan todos los A_j^1 de la primera iteración para calcular todos los A_j^2 de la segunda iteración. Surge la pregunta: ¿Es posible utilizar los $A_1^q,...,A_j^q$ con $j < i-1$ ya calculados, para calcular los siguientes $A_{j+1}^q,...,A_{i-1}^q$ que pertenecen a la misma iteración? Sí.

La mejora que se propone al algoritmo de Joseph para reducir el *CC*, es inicializar a la variable *t*, dentro de la misma iteración, con un nuevo valor. Cada vez que $A_j^q > A_j^{q-1}$, se sabe que en la iteración *q* no se encontrará el *PF* que satisface a la Ecuación (1) y consecuentemente el algoritmo no se detendrá. Por lo tanto, como se demuestra en el *Teorema 1*, cuando se cumple que $A_j^q > A_j^{q-1}$, es válido incrementar *t* con la diferencia $A_j^q - A_j^{q-1}$. El nuevo valor (t^{q+}) se utiliza para calcular el siguiente A_{j+1}^q y así sucesivamente. Por lo tanto, inicializamos a *t* con:

$$t^{q+} = t^q + A_j^q - A_j^{q-1} \qquad (\mathbf{2})$$

De esta manera, es posible calcular los siguientes $A_{j+1}^q,...,A_{i-1}^q$ de la iteración *q*, con el cálculo de los primeros $A_1^q,...,A_j^q$ de la misma iteración y no de la anterior. Esta mejora del algoritmo permitirá que sólo se necesiten *p* iteraciones para alcanzar al mismo *PF* con $p \le m$

A continuación se presenta una traza de cómo se desarrolla después de la iteración 0, en la cual el algoritmo se inicializa adoptando una semilla para la variable t^0, y contando consecuentemente con $A_1^0,...,A_{i-1}^0$. Se marcan con $\square$ para resaltar lo establecido.

Iteración 1:

$$\boxed{A_1^1} = \left\lfloor \frac{A_1^0 + .. + A_{i-1}^0 + C_i}{T_1} \right\rfloor C_1$$

$$A_2^1 = \left\lfloor \frac{\boxed{A_1^1} + .. + A_{i-1}^0 + C_i}{T_2} \right\rfloor C_2 \ \ldots$$

$$A_{i-1}^1 = \left\lfloor \frac{\boxed{A_1^1} + .. + A_{i-1}^0 + C_i}{T_{i-1}} \right\rfloor C_{i-1}$$

Iteración 2:

$$\boxed{A_1^2} = \left\lfloor \frac{A_1^1 + .. + A_{i-1}^1 + C_i}{T_1} \right\rfloor C_1$$

$$A_2^2 = \left\lfloor \frac{\boxed{A_1^2} + .. + A_{i-1}^1 + C_i}{T_2} \right\rfloor C_2 \ \ldots$$

$$A_{i-1}^2 = \left\lfloor \frac{\boxed{A_1^2} + .. + A_{i-1}^1 + C_i}{T_{i-1}} \right\rfloor C_{i-1}$$

Iteración *p:*

$$\boxed{A_1^p} = \left\lfloor \frac{A_1^{p-1} + .. + A_{i-1}^{p-1} + C_i}{T_1} \right\rfloor C_1$$

$$A_2^p = \left\lfloor \frac{\boxed{A_1^p} + .. + A_{i-1}^{p-1} + C_i}{T_2} \right\rfloor C_2 \ \ldots$$

$$A_{i-1}^p = \left\lfloor \frac{\boxed{A_1^p} + .. + A_{i-1}^{p-1} + C_i}{T_{i-1}} \right\rfloor C_{i-1}$$

Obsérvese en la traza que en la iteración 1, los $A_1^1, A_2^1, ..., A_{i-2}^1$ son utilizados para el cálculo de A_{i-1}^1.

***Teorema 1**:*

Dado un STR factible, el método propuesto converge al mismo PF que el método de Joseph ([6])*.*

Prueba:

Sea t_{PF} el tiempo donde ocurre el primer *PF*. Entonces dada una semilla t^0, con $t^0 \leq t_{PF}$ el método de Joseph converge hacia el menor *PF* (teorema de Kleene demostrado en [6]). Un nuevo algoritmo que no encuentre el mismo *PF*, sólo puede ser posible si se salta la cuenca de atracción de t_{PF}. Para que esto ocurra con una función monótona creciente y determinística, se debe obtener con el algoritmo un $t > t_{PF}$ y caer en la cuenca de atracción de otro *PF* o no converger. Pero los incrementos a la variable *t,* dentro de la iteración, se obtienen con la misma función $\left(\lceil t^q/T_j \rceil C_j\right)$ y son utilizados por el método de Joseph en próximas iteraciones (Ecuación (3)). Como el nuevo algoritmo propone que con cada $A_j^q > A_j^{q-1}$ se incremente t^q con $A_j^q - A_j^{q-1}$, el cálculo de A_j^q queda calculado con t^q, por lo cual $A_j^q(t^q) \leq A_j^q(t^{q+})$ y se debe esperar hasta la próxima iteración para que A_j^q de la tarea *j* sea calculada con t^{q+}. Entonces, si cada vez que se encuentra un par $A_j^q > A_j^{q-1}$ se incrementa t^q, haciendo $t^{q+} = t^q + A_j^q - A_j^{q-1}$, el mismo se encuentra acotado inferior y superiormente por la función de Joseph calculada en t^q y en t^{q+}

$$C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t^q}{T_j} \right\rceil C_j \leq t^{q+} \leq C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t^{q+}}{T_j} \right\rceil C_j \quad \text{con } t^{q+} = t^q + A_j^q - A_j^{q-1} \text{ y } 1 \leq j \leq i-1$$

Al estar acotado este cálculo por la función de Joseph, a la cual se puede iterar desde el punto t^{q+} y encontrar aun el menor *PF*, no es posible que el algoritmo presentado obtenga un tiempo que saltee al menor *PF*.□

***Teorema 2**:*

Dado un STR factible, el método propuesto converge a una velocidad mayor o igual que el método de Joseph ([6])*.*

Prueba:

En una iteración *q* con t^q distinto de un *PF*, existe al menos un $A_j^{q-1} \neq A_j^q$ con $1 \leq j \leq i-1$. Este incremento produce, en el algoritmo de Joseph, un incremento en t^q que se computará recién en la próxima iteración (Ecuación (3)), pudiendo resultar que, si para dicho valor de t^{q+1}, existe un $A_y^q \neq A_y^{q+1}$ con $j < y \leq i-1$, se habrán computado dos iteraciones para aproximarse al *PF* en $t^{q+2} = t^q + (A_j^q - A_j^{q-1}) + (A_y^{q+1} - A_y^q)$. En el algoritmo propuesto ello ocurre en la misma iteración debido a que el cálculo de A_y^q se realiza con $t^{q+} = t^q + A_j^q - A_j^{q-1}$, resultando luego $t^{q++} = t^{q+} + A_y^q - A_y^{q-1} = t^q + A_j^q - A_j^{q-1} + A_y^q - A_y^{q-1}$.□

3 Resultados Experimentales

Las simulaciones consisten en contar cuántas unidades A_j^q se requiere calcular por el algoritmo propuesto (*RTA2*) y los métodos propuestos por Sjödin ([11]) denominado *RTA* (Response Time Analysis) y por Bini ([12]) denominado *HET*.

El A_j^q es el invariante del método propuesto y también del método ([11]). Para el método propuesto por Bini, se puede presentar un invariante de uno o dos cálculos del tipo A_j^q, por lo cual, contando cada vez que ocurra este tipo de cuenta los resultados de los tres métodos son comparables. Posteriormente se calcula el promedio total de cuantas unidades A_j^q por factor de utilización se utilizaron para saber el tiempo de respuesta de cada tarea del sistema.

Los grupos de tareas se dividieron en dos. En el primero, la elección del período se realizó de manera aleatoria con una distribución uniforme (*DU*) y el tiempo de ejecución de cada tarea también fue aleatorio con una *DU*. El segundo grupo está constituido por subgrupos de tareas con períodos del mismo orden cada uno de ellos, una distribución exponencial (*DE*) para cada subgrupo y tiempos de ejecución aleatorios con una *DU* en forma similar a los sistemas utilizados en los trabajos [14, 15 y 16]. Por ejemplo, para el grupo comprendido entre 25-10000 se subdividió en tres grupos de 25 a 100, de 100 a 1000 y de 1000 a 10000.

En el método de Bini se evaluó para una *DE* en grupos, obteniéndose resultados elevados comparados a los otros métodos y además posee un comportamiento constante a las variaciones del *FU*, por lo que fue eliminado de las gráficas y sólo se mostró en las representaciones comparativas. Los factores de utilización de estos grupos se encuentran comprendidos entre el 70% y 95%, en saltos de 5%, con una tolerancia de ±0.5%. Además, por cada factor de utilización se evaluaron como mínimo 10000 *STR* en cada *FU*. Los *FU* inferiores al 70%, no se tuvieron en cuenta, dado que por debajo del ≈70% (*ln* 2*100) la cota de Liu y Layland ([1]) garantiza la planificabilidad de los *STR* cuando el número de tareas es infinito. El número de tareas de los grupos fue de 10, 20 y 50.

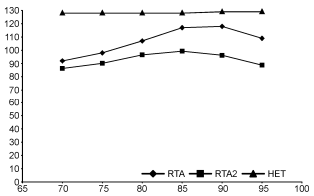

Figura 1 *DU* con 10 tareas períodos 25-10000

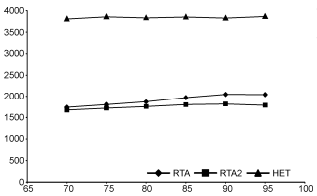

Figura 3 *DU* con 50 tareas períodos 25-10000

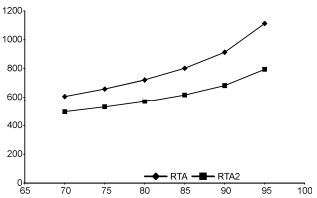

Figura 5 *DE* con 20 tareas períodos 25-10000

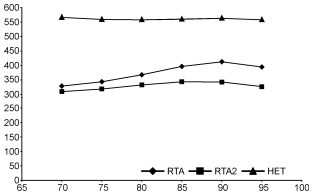

Figura 2 *DU* con 20 tareas períodos 25-10000

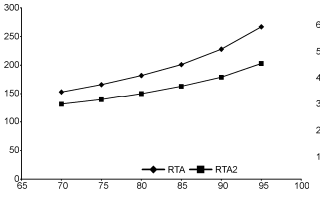

Figura 4 *DE* con 10 tareas períodos 25-10000

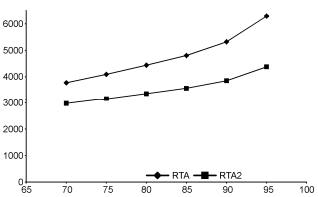

Figura 6 *DE* con 50 tareas períodos 25-10000

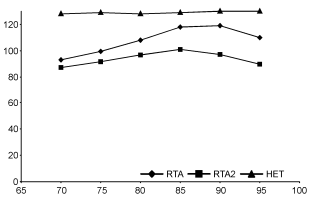

Figura 7 *DU* con 10 tareas períodos 25-100000

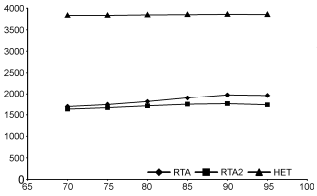

Figura 9 *DU* con 50 tareas períodos 25-100000

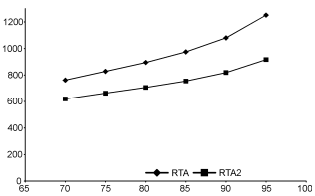

Figura 11 *DE* con 20 tareas períodos 25-100000

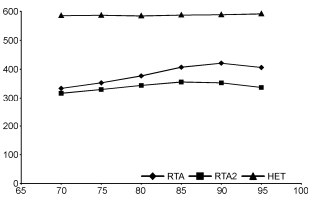

Figura 8 *DU* con 20 tareas períodos 25-100000

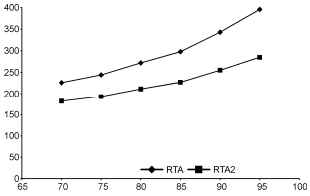

Figura 10 *DE* con 10 tareas períodos 25-100000

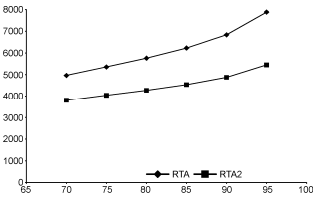

Figura 12 *DE* con 50 tareas períodos 25-100000

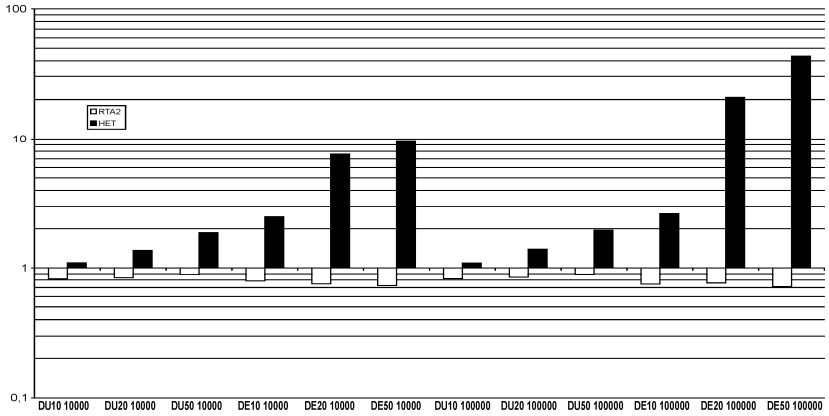

Figura 13 Comparación entre método presentado y el HET tomando como base el RTA.

4 Conclusiones

El método presentado, cuando es comparado con el método presentando por Sjödin en [11], es posible obtener reducciones promedios de un 10% a un 30% dependiendo del tipo de sistema. Esta mejora en el *CC* es considerada importante si sólo razonamos que el gasto para obtenerla es un arreglo de dimensión *n*-1. Sin embargo, el perfeccionamiento en el algoritmo puede ser utilizado en otro tipo de métodos como el presentado en [17] para cálculo de *Slack Stealing*, o en métodos Test de planificabilidad que sólo busquen establecer de manera precisa una condición necesaria y suficiente y no el tiempo de respuesta de la tarea.

En trabajos futuros, se seguirá investigando la posibilidad de aplicar este algoritmo a otros tipos de métodos y desarrollando nuevos con esta mejora.

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, pp. 46-61, 1973.

[2] M. Barabanov, "Best Practices on Wind River Real-Time Core Application Development," Wind River Systems 2008.

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

[4] 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.

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

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

[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] M. Sjödin and H. Hansson, "Improved Response-Time Analysis Calculations," presented at IEEE 19th Real-Time Systems Symp., 1998.

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

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

[14] 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, Internal Report 1995.

[15] 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.

[16] 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.

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