Un Algoritmo para la Diagramación de Tareas No-Duras mediante el Cálculo del Slack Time Disponible en cada Instante

José Manuel Urriza
Laboratorio de Sistemas Digitales
Universidad Nacional del Sur – CONICET
Bahía Blanca, Argentina, 8000
(jurriza@criba.edu.ar)

Jorge Santos
Laboratorio de Sistemas Digitales
Universidad Nacional del Sur – CONICET
Bahía Blanca, Argentina, 8000
(iesantos@criba.edu.ar)

and

Javier Dario Orozco
Laboratorio de Sistemas Digitales
Universidad Nacional del Sur – CONICET
Bahía Blanca, Argentina, 8000
(jorozco@uns.edu.ar)

Abstract

The execution of a non saturated Real-Rime System leaves idle or empty time-slots called slack. They can be used to execute tasks not belonging to the system but sharing the processor with it. Very often, the quality of service, provided to those tasks can be improved if the slack is reshuffled in such a way that the empty slots are advanced without jeopardizing the real-time tasks. The purpose of this paper is to present a redistributing algorithm for calculating the slack available at every instant. The theoretical foundation is formally proved, the algorithm is described and exemplified and, finally, a comparative evaluation against similar methods proposed by other authors to solve the same problem is carried out.

Keywords: Real Time Systems, Slack Time.

Resumen

Un Sistema de Tiempo Real, STR, no saturado, posee ranuras de tiempo que se encuentran ociosas. Las mismas, que se denominan *slack*, pueden ser utilizadas para ejecutar tareas que no pertenecen al STR. La calidad de servicio, *QoS*, brindado a estas últimas puede en general ser mejorado si el slack es redistribuido de modo tal que las ranuras vaciás se adelanten sin comprometer el cumplimiento de las constricciones de tiempo del STR. En este trabajo se presenta un algoritmo de redistribución que permite calcular en cada instante el slack disponible. Se demuestra formalmente el fundamento teórico, se describe y ejemplifica el algoritmo y, finalmente, se compara con métodos propuestos por otros autores para resolver el mismo problema.

Palabras Claves: Sistema de Tiempo Real, Slack Time

1 Introducción

Los sistemas de tiempo real (STR) son aquellos en los cuales los resultados no sólo deben ser correctos desde un punto de vista aritmético-lógico, sino que además deben ser obtenidos antes de un determinado instante denominado *vencimiento.* Cuando no pueden perderse vencimientos, el sistema se denomina *duro* en oposición a los *blandos* en los cuales, son tolerables algunas pérdidas. Cuando el sistema cumple con todas las constricciones de tiempo, se dice que es *diagramable.*

Siguiendo el modelo multiproceso-monoprocesador del trabajo liminar de Liu y Layland [4], las tareas de tiempo real duro a ser consideradas en este trabajo son periódicas (en consecuencia determinísticas), independientes y apropiables. Cuando dos tareas compiten por el uso del procesador, el conflicto es dirimido de acuerdo a reglas que forman una *disciplina de prioridades*.

El MCM de los períodos se denomina *hiperperíodo*. En virtud de la periodicidad de las tareas, los sucesivos hiperperíodos reproducen exactamente el esquema de generación de tareas que tuvo lugar en el primero.

La fracción del hiperperíodo que dejan libre las tareas duras se denomina *slack*. El mismo puede entonces ser destinado a otros usos, *eg.* la ejecución de tareas estocásticas no duras [5, 11, 12] o bien tareas opcionales asociadas a las duras, de cuya ejecución se deriva una cierta recompensa [1, 3, 6, 9].

En [10] se demostró formalmente que toda disciplina de prioridades que no deje ocioso el procesador, si hay tareas con ejecución pendiente, produce la misma distribución del slack a lo largo del tiempo. Este slack, que aparece naturalmente cuando el procesador está ocioso, se denomina *background*. Es posible que, sin afectar el cumplimiento de las constricciones de tiempo, una redistribución del slack mejore la calidad de servicio (*QoS*) brindado a las tareas no duras. Esto requiere conocer en cada instante cuánto es el slack disponible.

El objeto de este trabajo es presentar un algoritmo *on-line* que permite calcular el slack disponible en cada instante, sin necesidad de conocer los futuros requerimientos aperiódicos. En la Sección 2 se analizan trabajos similares hechos previamente por otros autores. En la Sección 3 se describe el método utilizado para verificar la diagramabilidad del STR. En la Sección 4 se presenta el algoritmo, previa demostración formal de su fundamento teórico. En la Sección 5 se compara con métodos similares y, finalmente en la Sección 6 se extraen las conclusiones y se delinea el trabajo futuro.

2 Trabajos Previos

La redistribución de slack ha sido tratada usando *servidores*, *slack-stealing*, *k-diagramabilidad* y *vencimientos efectivos*. En todos los casos se usa una disciplina de prioridades fijas (por ejemplo Períodos Monotónicos Crecientes) para las tareas duras y una cola del tipo FIFO o aleatorio para las no-duras.

Los servidores [11, 12] son en esencia tareas duras que utilizan el slack para ejecutar tareas no-duras. Dado que una métrica de la calidad de servicio es el retardo o tiempo de respuesta en el procesamiento de estas últimas, el período del servidor debe ser bajo (en general iguala el mínimo período del sistema) con lo cual su capacidad (o tiempo de ejecución), impuesta por las condiciones de diagramabilidad, resulta también en general baja. Estos métodos son *no-golosos* en el sentido de que la capacidad del servidor no necesariamente debe ser usada en forma inmediata. Se dice, por ello, que preservan el ancho de banda.

El método de slack-stealing estático [5] precalcula y tabula la cantidad de slack disponible antes del vencimiento de cada tarea periódica. Durante el procesamiento de las tareas, esos valores son usados para calcular el slack disponible y actualizados al término de la ejecución de cada tarea dura. El método es rígido en el sentido de que los períodos deben ser observados rigurosamente, sin posibilidad de variaciones (*jitter*). El slack-stealing dinámico [2] calcula la cantidad de slack disponible cuando aparece un requerimiento no-duro. Ambos métodos son golosos. El hecho de que el primero es muy rígido y en los dos casos el gasto indirecto (*overhead*) es alto, los hace en general inapropiados para aplicaciones *on-line* del mundo real.

Los métodos de *k*-diagramabilidad [7] se basan en las inversiones de prioridad que pueden soportar las tareas duras. Preservan el ancho de banda y tienen bajo gasto indirecto pero, en general, redistribuyen sólo parte del slack generado en el hiperperíodo.

Tia *et al* [13] realizan dos aportes teóricos importantes, ambos demostrados formalmente. El primero es que es imposible diseñar un algoritmo que minimice el tiempo de respuesta de todas las tareas no duras. Ni tan siquiera un algoritmo *clarividente* (que conozca *a priori* todos los requerimientos no-duros) puede lograr ese grado de optimización. El segundo aporte es que es imposible diseñar un algoritmo *on-line* que minimice el tiempo promedio de respuesta de todas las tareas no-duras. Con esas salvedades, proponen dos algoritmos. El primero minimiza el tiempo de respuesta de la tarea no dura que encabeza su cola (óptimo local). El segundo minimiza el tiempo en el cual se completa el cálculo de todas las tareas no duras que forman parte de la cola en ese momento (óptimo global).

En este trabajo se presenta un algoritmo que permite calcular el slack en cada instante. Previa demostración formal del fundamento teórico, el algoritmo es descripto, ejemplificado y comparado con métodos propuestos por otros autores para resolver el mismo problema.

3 La Disciplina de Periodos Monotónicos Crecientes y el Método de las Ranuras Vacías

En [4] se demuestra que la disciplina de Períodos Monotónicos Crecientes, PMC, es óptima entre las disciplinas de prioridades fijas. En lo que sigue es la disciplina que se usará para diagramar el conjunto de tareas duras. Para determinar la diagramabilidad PMC se usará el Método de las Ranuras Vacías [10]. En el mismo, el tiempo se considera ranurado y la duración de una ranura es tomada como la unidad de tiempo. Las ranuras se notan *t* y se numeran 1, 2, ... La expresión *al comienzo de la ranura t* e *instante t* son equivalentes. El sistema es apropiativo y las tareas pueden ser desalojadas del procesador únicamente al comienzo de la ranura. El peor estado de carga ocurre cuando todas las tareas duras reclaman ejecución en $t = 1$.

Un conjunto $\mathbf{S}(n)$ de n tareas independientes periódicas apropiables se encuentra completamente especificado por $\mathbf{S}(n)=\{(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)\}$, donde C_i, T_i y D_i, indican el máximo tiempo de ejecución, el período o, si es variable, el mínimo tiempo de interarribo y el vencimiento de la tarea i respectivamente. En la mayoría de las aplicaciones de ingeniería $D_i = T_i$. En [10] se demostró formalmente que $\mathbf{S}(n)$ es diagramable por PMC sss:

$$\forall i \in (1,2,....n) \qquad T_i \geq \text{menor } t \mid t = C_i + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil$$

$W_n(t)$ denota la función *trabajo* en el instante *t*. Está definida como

$$W_n(t) = \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil$$

Para $t = M$, en la que M denota el hiperperíodo, la función trabajo es

$$W_n(M) = \sum_{i=1}^{n} C_i \left\lceil \frac{M}{T_i} \right\rceil = \sum_{i=1}^{n} C_i \frac{M}{T_i}$$

y representa, por lo tanto, el número de ranuras necesarias para procesar el sistema duro en el intervalo [1, M]. El slack total en el hiperperíodo, en consecuencia, está dado por

$$ST = M - W_n(M)$$

El tiempo de respuesta de las tareas no-duras puede acortarse si, para su ejecución, se adelanta el slack producido naturalmente como background. Esto requiere conocer en cada instante el slack disponible para ser utilizado a partir de ese momento sin quebrar las constricciones de tiempo. El slack disponible por la tarea j en el instante t será notado $SD_j(t)$. El slack disponible por el sistema en el instante t será notado $SD(t)$.

4 El Cálculo del *SD*(*t*) y su Utilización por el Diagramador

4.1 Fundamento teórico

El fundamento teórico del algoritmo está dado en el lema y en el teorema que siguen. La notación complementaria a usar es:

$C_i(t)$: tiempo ya usado para la ejecución de la última instanciación (pendiente total o parcialmente o ya ejecutada totalmente) de la tarea *i* en el instante *t*.

$t_{INS\ j}$: instante de instanciación de la tarea *j* a ser considerado en el algoritmo. A los efectos de calcular el número de ranuras que la tarea *j* va a ocupar para su ejecución y substraerlas a futuro, hay que distinguir dos casos según que la tarea haya sido completamente ejecutada o no en su presente instanciación.

***Lema*:** El $t_{INS\ j}$ a ser considerado en el algoritmo es $\left\lceil \frac{t}{T_j} \right\rceil T_j + 1$, si la tarea *j* no ha sido totalmente ejecutada en la presente instanciación y $\left\lceil \frac{t}{T_j} \right\rceil T_j + T_j + 1$, si ha completado su ejecución en la presente instanciación

Demostración: Si la tarea *j* no ha completado su ejecución en la presente instanciación, ocupará ranuras antes de su vencimiento, que coincide con la próxima instanciación. Si, en cambio, ha completado su ejecución en la presente instanciación, podrá ofrecer el slack no sólo de ella sino también de la siguiente, cuyo vencimiento está, por lo tanto, alejado otro período

Ejemplo: En la Fig. 1 se representa la generación periódica de una tarea de *C*=1 y *T*=3 en el intervalo [9, 16]. El símbolo ↓ indica que la tarea ha sido instanciada y reclama ejecución. Se desea determinar el slack disponible en *t*=11. Si la tarea todavía no fue ejecutada, requerirá una ranura antes de *t*=13, que es, en consecuencia el t_{INS} que deberá ser considerado. Si la tarea ya fue ejecutada (en *t*=10), no va a requerir ranuras de ejecución en la presente instanciación y podrá incluirse el slack provisto en la siguiente, que se produce en *t*=16. Ambos t_{INS} corresponden a los calculados con las expresiones del lema, 1 y 4, respectivamente. Debe hacerse notar que en el segundo caso el slack es mayor que el período

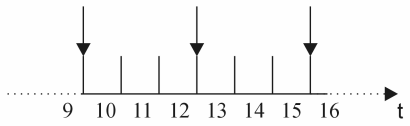

Fig. 1

Básicamente, el cálculo del slack disponible en cada instante consiste en determinar el slack que ofrece cada una de las tareas en un cierto intervalo y elegir el menor. Esto requerirá el cálculo de funciones trabajo entre *t*=1 y un cierto punto a la derecha (*t*>1). Si, por algún motivo, es necesario excluir eventuales requerimientos generados en el extremo derecho mismo del intervalo, la función trabajo deberá calcularse hasta la ranura anterior (extremo derecho del intervalo cerrado menos 1). Si en el ejemplo de la Fig. 1 se desea excluir el requerimiento generado en *t*=10, la función trabajo deberá ser calculada en el intervalo [1, 9].

El slack ofrecido por cada tarea en un cierto intervalo será la diferencia entre el número de ranuras del intervalo y el número de ranuras no vacías, ocupadas por la ejecución de la propia tarea y las de mayor jerarquía.

***Teorema*:** El slack disponible por el sistema en el instante *t* es

$$SD(t) = \underset{j=1}{\overset{n}{Min}}\left(SD_j(t)\right)$$

$$\text{en la que} \quad SD_j(t) = t_{INS\ j} - t - \sum_{i=1}^{j}\left(C_i\left\lceil \frac{t_{INS\ j}-1}{T_i}\right\rceil - \left(C_i\left\lceil \frac{t}{T_i}\right\rceil - C_i + C_i(t)\right)\right) \qquad (2)$$

Demostración: En la expresión (2), $(t_{INS\ j} - t)$ es la cantidad de ranuras que contiene el intervalo [*t*, $t_{INS\ j}$-1]. A esto habrá que restar el número de ranuras no vacías. Este será la diferencia entre la función trabajo en [1, $t_{INS\ j}$-1] y la función trabajo en [1, t]. En esta última, sin embargo, habrá que restar del C_i de la instanciación presente el $C_i(t)$ de las tareas ya ejecutadas. Cuando la sumatoria se resta del número de ranuras en el intervalo, se obtiene el slack que puede aportar la tarea *j* sin perturbar la ejecución de las tareas de mayor prioridad. El mínimo $SD_j(t)$ es el $SD(t)$

Lema:

$$\sum_{i=1}^{n}\left(C_i\left\lceil\frac{t}{T_i}\right\rceil - C_i\right) = \sum_{i=1}^{n}\left(C_i\left\lfloor\frac{t-1}{T_i}\right\rfloor\right)$$

Demostración: Inmediata de las definiciones de las funciones techo y piso

El lema es utilizado para simplificar el cálculo efectivo. En lugar de calcular el techo y substraerle C_i, se calcula el piso directamente.

4.2 El algoritmo de diagramación

El algoritmo que debe obedecer el diagramador (que forma parte del sistema operativo) surge naturalmente del teorema. Sus pasos están descriptos en el siguiente pseudocódigo:

Decisión del Diagramador ante la llegada de una Tarea No Dura, TND
IF *TND > 0*
  THEN *; La cola de tareas no dura posee tareas.*
    SD(t) = Valor Muy Grande *; No esperado que ocurra.*
    SUM = 0
    FOR *j=1* ***TO*** *n*
      IF $C_j(t) = C_j$ *;Estado de la tarea j.*
        THEN *;La tarea j se encuentra ejecutada en su totalidad.*
          $t_{INS\,j}(t) = \left\lceil\frac{t}{T_j}\right\rceil T_j + T_j + 1$ *; Instanciación posterior a la siguiente.*
        ELSE *; se encuentra no ejecutada o solo parcialmente.*
          $t_{INS\,j}(t) = \left\lceil\frac{t}{T_j}\right\rceil T_j + 1$ *; Instanciación siguiente.*
      END IF
      FOR *i=1* ***TO*** *j* *; Cálculo de la función de trabajo de la*
        $SUM = SUM + C_i\left(\left\lceil\frac{t_{INS\,j}-1}{T_i}\right\rceil - \left\lfloor\frac{t-1}{T_i}\right\rfloor\right) - C_i(t)$ *; tarea i en el intervalo* [*t* , $t_{INS\,j}$-1]
      NEXT
    NEXT
    $SUM = t_{ins\,j} - t - SUM$
    IF *SD(t) > SUM* *; Obtención del mínimo SD(t)*
      THEN
        SD(t) = SUM
    END *IF*
    Ejecuta Tareas No Duras
  ELSE
    Ejecuta Tareas Duras
END IF

Conviene notar que a los efectos de disminuir el gasto indirecto, el slack disponible es calculado sólo cuando alguna tarea no-dura reclama ejecución. Si efectivamente el slack se destina a ello, su valor debe decrementarse en una unidad por cada ranura utilizada. El mismo procedimiento debe observarse si una ranura vacía disponible queda vacía por no haber sido utilizada para ejecutar una tarea no-dura.

4.3 Ejemplo

A los efectos de no polarizar la presentación a favor del método propuesto, se toma como ejemplo uno de los conjuntos de tareas utilizados en [13]. Su especificación está dada en la Tabla 1

Tabla 1

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

En la Fig 2 está representada la evolución del STR. Sobre el eje horizontal están indicadas las tareas ejecutadas. Bajo dicho eje se consignan los instantes t = 1, 2, ... Las ranuras vacías, notadas e, aparecen como background en t = 6, 11 y 12.

Como ilustración, se detalla a continuación el cálculo del slack en t = 6.

$$j = 1,\ t_{INS.1} = 10$$

$$10 - 6 - \left(1\left(\left\lceil \frac{9}{3} \right\rceil - \left\lfloor \frac{5}{3} \right\rfloor\right) - 1\right) = 3$$

$$j = 2,\ t_{INS.2} = 13$$

$$13 - 6 - \left(1\left(\left\lceil \frac{12}{3} \right\rceil - \left\lfloor \frac{5}{3} \right\rfloor\right) - 1\right) + \left(1\left(\left\lceil \frac{12}{4} \right\rceil - \left\lfloor \frac{5}{4} \right\rfloor\right) - 1\right) = 4$$

$$j = 3,\ t_{INS.3} = 13$$

$$13 - 6 - \left(1\left(\left\lceil \frac{12}{3} \right\rceil - \left\lfloor \frac{5}{3} \right\rfloor\right) - 1\right) + \left(1\left(\left\lceil \frac{12}{4} \right\rceil - \left\lfloor \frac{5}{4} \right\rfloor\right) - 1\right) + \left(1\left(\left\lceil \frac{12}{6} \right\rceil - \left\lfloor \frac{5}{6} \right\rfloor\right) - 1\right) = 4$$

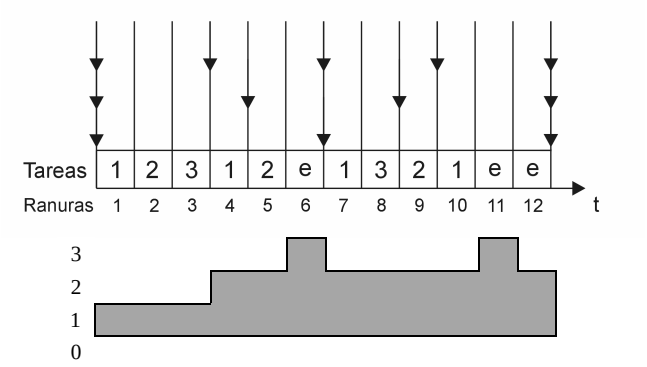

Fig. 2

En la parte inferior de la figura se representa, grisado, el slack disponible calculado de acuerdo al algoritmo propuesto. Cabe hacer notar que en [13] el slack calculado para t = 6, es de sólo 2 ranuras. En t=11 y 12 hay slack proveniente del siguiente hiperperíodo.

En la Fig 3 se representa el adelantamiento de las ranuras vacías de background a las posiciones t = 7, 8, sin que el sistema duro deje de cumplir con las constricciones de tiempo. A partir de t = 6, en consecuencia, tres ranuras sucesivas podrían destinarse a la ejecución de tareas no-duras. Si se hiciera tal cosa, en t = 6, 7 y 8, el slack disponible por aplicación del método bajaría a 2, 1 y 0, respectivamente, con lo cual se habría agotado el disponible en ese hiperperíodo.

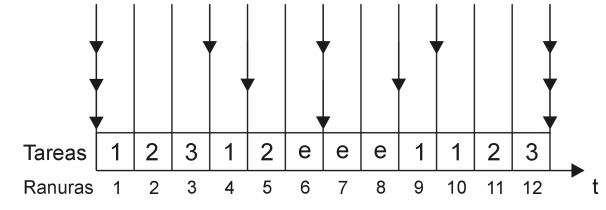

Fig. 3

5 Evaluación comparativa

La principal ventaja del método propuesto en este trabajo es que en cada instante *t* permite determinar exactamente y usar todo el slack disponible no sólo en *t* sino también en los instantes inmediatos sucesivos. Esta ventaja no es compartida por los métodos de *k*-diagramabilidad, que trabajan con cotas, ni por los métodos basados en servidores, que sólo ejecutan tareas aperiódicas hasta agotar la capacidad de aquéllos.

En el ejemplo anterior, el método propuesto encuentra que en $t = 6$ el slack disponible es 3 y, en consecuencia, puede usar las ranuras 6 a 8. Esto permitiría ejecutar una tarea aperiódica de tiempo de ejecución $C = 3$ que se hubiera generado en el instante 6. El tiempo de respuesta es el mínimo posible dado que la ejecución de la tarea comienza apenas generada y se completa en ranuras corridas.

En el método de *k*-diagramabilidad con detección de singularidades simples [7], el valor calculado de *k* es 1. Como se detecta que las ranuras 6 y 7 son singularidades, se ejecutan en ellas las dos primeras ranuras de la tarea aperiódica. La siguiente singularidad se detecta recién en la ranura 12 con lo cual la tarea aperiódica recién termina de ejecutarse en ese momento. En el método de *k*-diagramabilidad con detección de singularidades múltiples [7], los k_i valen 2, 1 y 1, respectivamente. En este ejemplo en particular, la tarea aperiódica sería ejecutada también en las ranuras 6, 7 y 12. La desventaja principal de estos métodos frente al aquí propuesto es que los *k* son cotas calculadas para el peor estado de carga y no puede sacarse pleno partido de las singularidades generadas en tramos más aliviados. Esto es especialmente grave en casos particulares como el del ejemplo, en el cual el hiperperíodo es un múltiplo pequeño del período máximo.

Al analizar la diagramabilidad del sistema se encuentra que es imposible mantener las constricciones de tiempo si se incorpora otra tarea dura de período 3, ni aún con un $C = 1$. Tampoco puede incorporarse una tarea dura de período 4. Recién cuando se llega a un $T = 5$, es posible incorporar una tarea de $C = 1$. Esta tarea sería el servidor en cualquiera de los métodos basados en ellos. La tarea aperiódica sería ejecutada también en las ranuras 6 y 11 (por servidor) y 12 (por ranura libre).

El algoritmo propuesto en [13], según lo ilustra el ejemplo dado en el propio trabajo, sólo detecta un slack de valor 2 en el instante 6 y, como consecuencia, en este caso particular la tarea aperiódica también comenzaría a ejecutarse en las ranuras 6, 7 y terminada en una posterior a la 8. Como puede verse no se minimiza el tiempo de respuesta.

Una métrica usual en la calidad del servicio prestado (QoS) es el retardo que sufren las tareas aperiódicas, medido como el intervalo entre el instante en que se generan y solicitan ejecución y el instante en que terminan de ser ejecutadas. En el ejemplo anterior, la tarea aperiódica se genera en t=6 y requiere tres ranuras para ser ejecutada. Con el método aquí propuesto se termina de ejecutar en t=8, con ambos métodos de *k*-diagramabilidad y con el método del servidos en t=12 y con el método de Tia en t=9. Los retardos respectivos son 3, 7, 7 y 4 ranuras respectivamente. La mejora con respecto al mejor de los métodos que le siguen es pues del 33%.

Por supuesto, lo anterior se consigna sólo a modo de ejemplo. Una evaluación comparativa cuantitativa que pueda considerarse válida requiere simulaciones masivas con conjuntos aleatorios de tareas que sean representativos del mundo real. Ese es el objeto de un futuro trabajo.

6 Conclusiones

Se ha presentado un algoritmo para el cálculo del slack disponible en cada instante de la evolución de un STR. La importancia del mismo radica en el hecho de que una adecuada redistribución del slack disponible en el hiperperíodo permite mejorar la QoS de las tareas no-duras que comparten el procesador con el STR.

El fundamento teórico ha sido formalmente demostrado y se ha hecho una comparación evaluativa, también teórica, con métodos similares previos propuestos por otros autores para resolver el mismo problema. La comparación revela que el algoritmo permite detectar más slack y presenta, en consecuencia, un mejor comportamiento. Tanto el algoritmo como su evaluación han sido ilustrados con un ejemplo tomado de uno de los trabajos previos para evitar una polarización a favor del propuesto.

La futura línea de trabajo incluye evaluaciones por simulaciones experimentales y la formalización de algunas propiedades que permitan simplificar la complejidad del cálculo.

Agradecimiento

Los autores agradecen las sugerencias formuladas por uno de los revisores anónimos.

Referencias

[1]. Aydin, H., Audsley, N. C., R. I. Davis and A. Burns, “Mechanism for enhancing the flexibility and utility of hard real-time systems”, *Proc. 15th IEEE Rea-Time System Symposium*, pp. 12-21, 1994.

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

[3]. Dey, J. K. and J. Kurose, “On line scheduling policies for a class of IRIS (Increasing reward with increasing service) real-time tasks”, 46, 7, pp. 802-813, 1986.

[4]. Liu C. L. and J. W. Layland, “Scheduling algorithms for multiprogramming in hard real-time environment”, *J. ACM*, 20, 1, pp. 46-61, 1973.

[5]. Lehoczky J. P. y Ramos Thuel S., “An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed Priority Preemptive Systems”, *Proceedings of the Real Time Systems Symposium pp 110-123 1992.*

[6]. Melhem, R., D. Mossé and P. Mejía-Alvarez, “Optimal reward-based scheduling for periodic real-time tasks”, *IEEE Transactions on Computers*, 50, 2, pp. 111-130, 2001

[7]. Orozco J., R. Santos, J. Santos and R. Cayssials, “Taking advantage of priority inversions to improve the processing of non-hard real-time tasks in mixed systems”, *Proc. WIP 21st IEEE Real-Time Systems Symposium*, pp. 13-16, 2000.

[8]. Santos, J. and J. Orozco, "Rate monotonic scheduling in hard real-time systems", *Information ProcessingLetters*, 48, pp. 39-45, 1993.

[9]. Santos, R. M., J.Urriza, J. Santos and J. Orozco, "Heuristic use of singularities for on-line scheduling of real-time mandatory/reward-based optional systems", *Proc. 14th Euromicro Comference on Real-Time Systems*, 2002.

[10]. Santos, J. *et al*, "Priorities and Protocols in Hard Real-Time LANs", *Computer Communications*, 14, 9, 507-514, 1992.

[11]. Sprunt, B. Sha, L. y Lehoczky, J. P., “Aperiodic Task Scheduling for Hard Real Time Systems”, *The Journal of Real Time Systems Vol 1 pp 27-60 1989*

[12]. Strosnider, J. K., J. P. Lehoczky and L. Sha, The Deferable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", *IEEE Transactions on Computers*, 44, 1, 75-91, 1995

[13]. Tia, Too-Seng, Jane W.-S Liu y Mallikarjun Shankar, “Algorithms and Optimality of Scheduling Soft Aperiodic Requests in Fixed-Priority Preemptive Systems”, *Real Time Systems – Kluwer Academic Publishers – Vol. 10 N°1 Enero 1996*