# Métodos Rápidos para el cálculo del Slack Stealing Exacto y Aproximado para Aplicaciones en Sistemas Embebidos

José M. Urriza y Javier D. Orozco

Departamento de Ingeniería Eléctrica y Computadoras
Universidad Nacional del Sur / CONICET
8000 Bahía Blanca, Argentina
{jurriza, ieorozco} @criba.edu.ar

**Resumen**: Desde los comienzos de los años 90 se vienen desarrollando diversos métodos aplicables a sistemas de tiempo real (*STR*) duro con el objetivo de aportar flexibilidad a los mismos sin perder la necesaria robustez que debe caracterizarlos. Las sobrecargas producidas por los mecanismos de tolerancias a las fallas, la atención de tareas esporádicas y de tiempo real blando, conjuntamente con el sistema de tiempo real duro, puede comprometer el funcionamiento de este último si el problema no es tratado correctamente. Varias de estas propuestas han encontrado, en los métodos de *Slack Stealing* (*SS*), una herramienta valiosa cuyos resultados se aproximan a los óptimos teóricos. Adicionalmente, la pronta difusión en los últimos años de la computación móvil, introduce un nuevo tipo de requerimiento a los sistemas de cómputo que también conspira con el desempeño del sistema de tiempo real. El desarrollo actual de los dispositivos de almacenamiento de energía no logra cubrir las necesidades simultáneas de capacidad y tamaño requeridos por los equipos móviles por lo que, la economía de energía resulta vital y la misma se encuentra estrechamente vinculada a la administración que se realice, por parte del sistema operativo (*SO*), de los recursos del sistema. Algunas de las propuestas de *SO* de tiempo real para dispositivos móviles, utilizan métodos basados en *Earliest Deadline First* (*EDF*) dada su simpleza de validación de la planificación. A este método, se le asocia un algoritmo de *Dynamic Voltage Scheduling* (*DVS*) a fin de reducir el consumo de energía sin perturbar el comportamiento temporal del sistema. En este trabajo, desarrollan dos métodos de *SS* en Prioridades Fijas (*PF*) en los que se calcula el *slack* en tiempo de ejecución con el menor costo computacional posible. Este slack, calculado en tiempo de ejecución permitirá reducir el consumo de energía mediante la utilización conjunta de técnicas *DVS*, como así también, la atención de requerimientos no críticos sin afectar las aplicaciones de tiempo real.

**Palabras Claves**: Slack Stealing, DVS, Economía de Energía

## 1 Introducción a los *STR*

Los *STR* se pueden encontrar en casi todas las ramas de la ingeniería abarcando inclusive desde hace algunos años, áreas de sistemas electrónicos de consumo de mediana y baja complejidad.

Se puede definir a un *STR* como aquel en el que los resultados de una tarea son aceptables si se obtienen antes de un determinado tiempo denominado vencimiento. Para su caracterización es necesario integrar parámetros temporales al el resto de los requerimientos físicos y funcionales del sistema. Es aceptada entonces la definición de Stankovic[1] que dice que: *en los STR los resultados no sólo deben ser lógicamente correctos si no que además su aceptabilidad depende del tiempo en el cual son producidos.*

Los *STR* se pueden clasificar en tres tipos dependiendo de la aceptabilidad de los resultados respecto de sus vencimientos: Si no se admiten pérdidas en los vencimientos, se los denominan *duros* o *críticos* (*hard*). Si es admisible la pérdida de algunos vencimientos según alguna ley de aceptabilidad se los denominan *blandos o no-críticos* (*soft*). Estos últimos utilizan criterios basados en la calidad de servicio u otros de optimalidad global para caracterizar la tolerancia del sistema a las pérdidas de vencimientos. Es por ello que dentro de los *STR* blando podemos encontrar una amplio espectro de definiciones que van desde sistemas muy próximos a los de *STR* duro a sistemas con muy bajo nivel de predictabilidad. A partir de estos extremos es posible encontrar algunas tipificaciones como las de los sistemas *firmes* (*firm*) que admiten una determinada cantidad de pérdidas distribuidas según una ley prefijada por las características de la aplicación.

La definición de los *STR duros* parte de la premisa de que, la pérdida de un vencimiento en una tarea, puede tener consecuencias catastróficas. En los sistemas en los cuales la vida humana es participe, por ejemplo la avionica, se le debe garantizar que todas las tareas terminen antes de su vencimiento mediante un *test de diagramabilidad* o *planificabilidad*. Sí el test es exitoso, se dice que el sistema es *factible* o *planificables*.

En 1973, C. L. Liu y James W. Layland [2] realizan el primer aporte significativo para determinar la diagramabilidad de un *STR-duro*. Este trabajo se ha convertido en un trabajo liminar de la disciplina. Debido a que propone un marco formal para el análisis de factibilidad de un sistema multitarea - monoprocesador con tareas independientes.

Existen diferentes mecanismos para lograr planificar un sistema multitarea – monoprocesador que van desde los sistemas de asignación de recursos estáticos (ejecutivos cíclicos) a sistemas completamente dinámicos tanto en la asignación de recursos como en la constitución del sistemas. Para estos últimos y todas las variantes intermedias, resulta conveniente la utilización de criterios basados en la aplicación de *disciplinas de prioridades* que establezcan una relación lineal de orden sobre el conjunto de tareas de manera que el *planificador* pueda determinar que tarea ($\tau_i$) tiene derecho de utilización del recurso en cada instante de activación.

En lo que sigue se utilizará un modelo de tareas donde las mismas se consideran periódicas, independientes y apropiables. Una tarea periódica es aquella en la que su arribo se produce cada $T_i$ unidades de tiempo; independiente, cuando su ejecución no depende temporal ni lógicamente de la ejecución de otra tarea del sistema, y se dice que una tarea es apropiable, cuando el *planificador* puede suspender su ejecución y desalojar el recurso en cualquier momento.

Luego, una tarea $\tau_i$ se encontrará completamente caracterizada con la terna ($C_i$, $T_i$, $D_i$) donde $C_i$ representa su tiempo de ejecución; $T_i$ su el periodo y $D_i$ su vencimiento. Por lo tanto, un conjunto $\mathbf{S}(n)$ de $n$ tareas se encuentra especificado por $S(n)=\{(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)\}$.

Existen en la actualidad tres *disciplinas de prioridades* básicas para la diagramación de tareas: Ejecutivos Cíclicos, Prioridades Fijas (Fixed Priorities) en sus variaciones Rate Monotonic [2] (*RM*) y Deadline Monotonic [3] (*DM*) y disciplinas de prioridades dinámicas como Earliest Deadline First [2] (*EDF*). Además de ellas, existen combinaciones de las mismas como el método Dual Priority [4].

El trabajo se organiza de la siguiente manera: en la sección 2 se describen los trabajos previos y se introduce al método, en la sección 3 se describe el cálculo, en la sección 4 se presentan los métodos propuestos y finalmente en la sección 5 se desarrollan las conclusiones.

## 2 Trabajos Previos

La reciente generalización de las aplicaciones que requieren la utilización de técnicas de tiempo real, hace necesario contemplar la integración de requerimientos heterogéneos que van desde tareas de tiempo real duro hasta tareas aperiódicas sin restricciones temporales. A estos requerimientos, se le debe adicionar aquellos específicos de la aplicación del sistemas como ser por ejemplo las necesidades de ahorro de energía [5-7] en sistemas móviles.

Sí el *STR* no requiere una utilización del 100% del procesador para su correcta ejecución se dice que el mismo es *no-saturado* y, en el tiempo que permanece ocioso se podrán ejecutar tareas *no-críticas*. El método más sencillo para lograr el aprovechamiento de este tiempo ocioso se obtiene permitiendo la ejecución del subsistema no-crítico en los intervalos naturales de inactividad del subsistema de TR (servicio en *background*). Sin embargo, en algunos casos es necesario optimizar el servicio ofrecido al subsistema *no-crítico*, de manera de por ejemplo, minimizar su tiempo de respuesta. Esta mejora en la calidad en el servicio ofrecido solo es posible si se garantiza la planificación de las tareas *duras*, conjuntamente con el objetivo de lograr la calidad de servicio deseada para las tareas no-críticas. Para realizar esto, se utilizan técnicas denominadas *slack-stealing* que permiten adelantar los intervalos de ejecución del subsistema no-crítico sin afectar al subsistema crítico. Es por ello que desde la década de los 80, una amplia variedad de trabajos proponen mecanismos para obtener en tiempo de ejecución o fuera de él, el slack disponible en el sistema. Algunos trabajos a su vez, lo calculan de manera exacta y otros de manera aproximada.

Entre los métodos propuestos, se pueden diferenciar claramente dos tipos: La utilización de servidores como el *polling Server,* en el que se ejecuta una tarea dura como servidor de las tareas no-duras.. Generalmente esta tarea es de máxima prioridad y la capacidad de los servidores se determina fuera de línea. Mejoras de este tipo de servidores fueron hechas con *Priority Exchange* y *Deferrable Server* en [8],

con *Extended Priority Exchange* en [9], con *Sporadic Server* en [10] y con *Total Bandwidth Server* en [11]. Por otro lado, otra metodología para lograr adelantar el slack de la ejecución del *STR* fue presentada en [12] en un método denominado *Slack Stealing* (*SS*), modificado por los mismos autores en [13] para que resulte apto para su aplicación en tiempo de ejecución y no en tiempo de compilación como el primero. Un inconveniente en estos dos trabajos, es que necesitan arreglos de memorias del orden del *hiperperíodo* del sistema, por lo cual solo pueden ser aplicados en sistemas que su *hiperperíodo* sea pequeño. Por otra parte en [14, 15] se introduce una mejora notable en su aplicabilidad ya que no sólo es posible su utilización en tiempo de ejecución sino que se reduce notablemente el gasto en memoria que solo es del orden del número de tareas. En el trabajo de Tia [16] se presenta un nuevo algoritmo de *SS*, pero al igual que los trabajos de [12, 13] necesita grandes cantidades de memoria. La importante contribución de éste radica sin embargo en dos teoremas que prueban que un algoritmo en línea no puede aplicar un criterio de optimización uniforme y unánime para el conjunto de tareas no-críticas, que minimice el tiempo de respuesta de cada requerimiento, ni tampoco el promedio de respuesta de todos los requerimientos.

Desde el punto de vista de la ingeniería del sistema, el mayor problema de los métodos de *SS* es que poseen una carga computacional elevada $O(m.n^2)$, por lo cual para algunas aplicaciones resultan prohibitivos.

### 2.1 Introducción a los métodos existentes.

En [2] Liu & Layland demuestran que la disciplina de *Rate Monotonic* (*RM*) es óptima entre las disciplinas de prioridades fijas, para un conjunto de tareas independientes y apropiativas con $D_i=T_i$. En Leung y Whitehead [3] se demuestra que si el ordenamiento de las tareas es del tipo *Deadline Monotonic* (*DM*, $D_i \leq T_i$), la disciplina es también óptima. En adelante estas disciplinas se usarán para diagramar el conjunto de tareas duras. En el análisis efectuado en [2] bajo *RM*, se determina una cota, la cual es una condición necesaria pero no es suficiente para garantizar la diagramabilidad del *STR*. E. Bini en [17] proponen una nueva cota de diagramabilidad que mejora la anterior, pero también es una condición necesaria pero no suficiente. El análisis de diagramabilidad exacto se desarrolló en [18-21], mediante un cálculo iterativo con una complejidad $O(m.n^2)$ donde $m=T_n$ o $m=De_n$ para la disciplina de períodos monotónicos crecientes. Se demostró que $S(n)$ es *diagramable* por *PF* sss:

$$\forall i \in (1,2,....n) \qquad T_i \geq D_i \geq menor\ t \mid t = W_i(t) \tag{1}$$

donde $W_i(t)$ denota la función trabajo del sistema $\mathbf{S}(i)$ en el instante *t.* La misma representa el tiempo de ejecución requerido por el sistema en el intervalo [1, *t*]. En general, para $\mathbf{S}(n)$, se define como:

$$W_n(t) = \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil \tag{2}$$

En $t = M$, donde *M* denota el *hiperperíodo*, la función trabajo es

$$W_n(M) = \sum_{i=1}^{n} C_i \left\lceil \frac{M}{T_i} \right\rceil = \sum_{i=1}^{n} C_i \frac{M}{T_i} \tag{3}$$

y representa, por lo tanto, el tiempo necesario para ejecutar todos los requerimientos del sistema duro en el intervalo [1, *M*].

Luego la diferencia entre el tiempo disponible y el utilizado hasta el hiperperíodo representa el *slack* total en el *hiperperíodo*:

$$ST = M - W(M) \tag{4}$$

En lo que sigue, el sistema se considera: *apropiativo* (las tareas pueden ser desalojadas del procesador pero no autosuspenderse), tareas independientes y el *peor estado de carga* o *instante crítico*, ocurre cuando todas las tareas duras reclaman su ejecución en $t$ =1.

Para adelantar el slack producido naturalmente como en los tiempos ociosos, se requiere conocer en cada instante en que medida puede ser retrazando la ejecución del sistema de tiempo real sin afectar sus restricciones temporales. Con esta consideración, denominaremos $SD_i(t)$ al *slack disponible* por $\tau_i$ en el instante *t* siendo éste, el máximo retrazo que tolera $\tau_i$ sin violar su vencimiento. Para un instante dado, el

máximo slack disponible por el sistema, sin violar sus restricciones temporales resulta del mínimo $SD_i$ ($t$) y será notado $SD$ ($t$).

$$SD(t) = \min_{i=1}^{n} SD_i(t) \tag{5}$$

En [22] se cuantificó el número máximo de inversiones de prioridad que tolera una tarea sin violar sus restricciones temporales a partir del *instante crítico*. Este cálculo es muy pesimista debido que esta calculado a partir del peor estado de carga cuando en la practica la probabilidad de su ocurrencia es baja. Además, el cálculo se realiza suponiendo que todas las tareas tienen su peor tiempo de ejecución, lo cual hace más pesimista el análisis. Si bien, aún cuando pesimistas, estas condiciones resultan insoslayables si el *STR* es *duro*, el análisis realizado en [22] no analiza cuanto tiempo puede ser retrazada una tarea en cualquier instante del hiperperíodo diferente al crítico sin perder sus vencimientos. Este análisis resulta importante a fin de obtener un mejor aprovechamiento del tiempo disponible.

Ejemplo:

Se toma el siguiente sistema de tiempo real.

| ***N°*** | $T_i$ | $C_i$ | $D_i$ |
|---|---|---|---|
| 1 | 3 | 1 | 3 |
| 2 | 4 | 1 | 4 |
| 3 | 6 | 1 | 6 |

Tabla 1.

En la figura 2 vemos la ejecución de las tareas en el hiperperíodo.

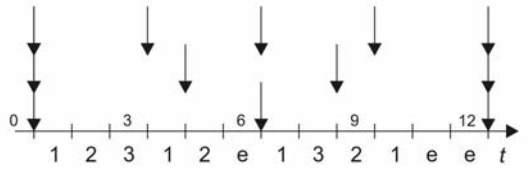


Fig. 1.

Del análisis iterativo aplicado en [12-15, 22] se determina que en $t$=1, partiendo del *instante crítico*, las tareas tienen un slack de 2, 1 y 1 respectivamente. Esto implica que el orden de ejecución para las ranuras 1, 2 y 3 podría haber sido 2, 3, 1; 3, 2, 1 cualquier otra combinación que no provoque mayor retraso que el calculado. Incluso, es tolerable que la primer unidad de tiempo no sea utilizada por el sistema siendo aún factible la secuencia *e*, 2, 1, 3. Sin embargo, un cálculo más exacto permitiría mostrar que cuando la tarea 2 termina su primera ejecución, $SD_2$ toma el valor 2 en lugar de 1 como en el instante crítico. Algo similar ocurre para el resto del sistema con lo que, resulta tolerable, entre otras, a partir del instante 5 la secuencia: e, e, 2, e, 1, 3. Se puede calcular fácilmente por inspección de la tabla 1 los $SD_i(t)$ en cada instante. A continuación se presenta en la tabla 2, el *slack* en cada instante para cada tarea.

| ***t*** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $SD_1$ | **2** | 4 | 3 | **2** | 4 | 3 | **2** | 4 | 3 | **2** | 4 | 3 |
| $SD_2$ | **1** | 1 | 3 | 2 | **2** | 4 | 3 | 3 | **2** | 3 | 3 | 2 |
| $SD_3$ | **1** | 1 | 1 | 3 | 3 | 3 | **2** | 2 | 3 | 3 | 3 | 2 |
| ***SD*** | **1** | 1 | 1 | 2 | 2 | 3 | 2 | 2 | 2 | 2 | 3 | 2 |

Table 2.

## 2.2 Detección de Singularidades.

En [22] se han presentado dos métodos basados en la detección de singularidades con el fin de optimizar la diagramación de tareas aperiódicas en un entorno de tiempo real duro. En este contexto se entiende como singularidad de nivel *i* al instante en el que todas las tareas con prioridad [1, *i*] han satisfecho todos sus requerimientos pendientes. Nótese que en un sistema basado en prioridades fijas, la finalización

de la ejecución de una tarea de prioridad $i$ en un instante $t$ implica que en $t$-1, no existen requerimientos pendientes del subsistema de mayor prioridad. Además, $s_i$ es una singularidad del subsistema [1, $i$] independientemente de las nuevas activaciones que pudiesen ocurrir en $t=s_i$ y de la ejecución pendiente del subsistema de prioridades menores a $i$.

La detección de singularidades es presentada en [22] con el fin de permitir la inicialización de un sistema de contadores con valores precalculados para el instante crítico y que son utilizados para controlar el número de inversiones de prioridad que puede soportar una tarea. Es claro que estos métodos resultan de una situación de compromiso entre el desempeño y la complejidad computacional y esta simplificación impone un cálculo pesimista, pudiendo fácilmente encontrar ejemplos donde el número de inversiones admisible es muy superior a esta cota. Sin embargo, un cálculo más preciso del slack disponible impondría, la necesidad de recalcular el máximo número de inversiones de prioridad en las condiciones de carga presentes en el sistema luego de cada singularidad.

### 2.3 Contadores

Para reducir la carga computacional del método en línea, se utilizará como en [14, 15, 22] un método de contadores. De no ser así se debería recalcular el slack de todas las tareas cuando una determinada tarea termina, incrementando notablemente la carga computacional y volviendo de imposible implementación en línea. Por lo tanto, cada tarea, tiene asociado un contador que llevar de manera precisa que cantidad de *slack* posee. A continuación se repasa las condiciones que rigen a estos contadores.

I. Cuando se ejecuta una tarea de menor prioridad, todos los contadores de las tareas de mayor prioridad deberán disminuir sus contadores el tiempo ejecutado.
II. Cuando se ejecuta un requerimiento no-crítico o el recurso queda ocioso, todos los contadores de todas las tareas deberán disminuir el tiempo consumido.
III. La recarga de un contador determinado solo se realiza cuando se termina de ejecutar completamente su tarea.
IV. Si una tarea termina antes de su WCET, el *tiempo ganado*, que es la diferencia entre el WCET y el tiempo consumido, se suma a todos los contadores de prioridad menor.

Se observa que por ningún motivo alguno de estos contadores podrá ser menor a cero. Sí esto sucediese significaría que el sistema ha dejado de ser planificable.

## 3 Cálculo del slack disponible

En este trabajo se presentan dos algoritmos que permite calcular el *slack* en cada instante tanto en forma exacta como aproximada con una carga computacional reducida. En lo que sigue, los algoritmos son descriptos y analizado su desempeño.

Para calcular el slack disponible de cada tarea comenzaremos, como en [18-20], en aplicar un método iterativo de cálculo.

En lo que sigue se utilizará la siguiente notación:

$\tau_i$ : tarea $i$

$t_{i,h}$ : tiempo de arribo de $\tau_i$, en instanciación $h$

$te_{i,h}$: instante de finalización de la ejecución de $\tau_i$ para la instanciación $h$.

$D_{i,h}$ : vencimiento de $\tau_i$ correspondiente a la instanciación $h$. Donde $D_{i,h} = t_{i,h}+D_i$

$t_a$: instante donde se calcula el slack.

$SD_i(t_a)$: Slack de $\tau_i$ en el tiempo $t_a$.

$C_{j,h}(t_a)$: Tiempo de ejecución de la tarea $j$ hasta el instante $t_a$ para la instanciación $h$

***Lema 1:***

*Dado un sistema $S(n)$ diagramable de tareas apropiativas e independientes, el máximo slack disponible para $\tau_i$ en el instante $t_a < t$ está dado por*

$$SD_i(t_a) = \max_{[0,D_i]} k_i \text{ que verifica}$$
$$D_{i,h} \geq \wedge t \mid t-(t_a-1) = k_i + \sum_{j=1}^{i}\left[ C_j\left(\left\lceil \frac{t}{T_j} \right\rceil - \left\lfloor \frac{t_a-1}{T_j} \right\rfloor\right) - C_{j,h}(t_a) \right] \quad (6)$$

**Prueba:**

En [16] se ha demostrado que dado un sistema $S(n)$ diagramable, la función trabajo del subsistema $S(1, i)$ en un intervalo $[t_a , t]$ está dada por:

$$W(t_a,t) = \sum_{j=1}^{i}\left[ C_j\left(\left\lceil \frac{t}{T_j} \right\rceil - \left\lfloor \frac{t_a-1}{T_j} \right\rfloor\right) - C_{j,h}(t_a) \right] \quad (7)$$

Nótese que para $t_a$=1, la ecuación (7) coincide con la ecuación (2).

Luego, de la ecuación (7), para que el sistema resulte diagramable en el intervalo $\left[t_a, D_{i,h}\right]$ deberá existir suficiente tiempo disponible para ejecutar la demanda pendiente de $S(i)$ en $t_a$ más la generada en dicho intervalo con lo que

$$D_{i,h} \geq \wedge t \mid t-(t_a-1) = W(t_a,t) \quad (8)$$

Luego, el máximo slack disponible en $[t_a, t]$, estará dado por:

$$SD_i(t_a) = \max_{[0,D_i]} k \text{ que verifica}$$
$$D_{i,h} \geq \wedge t \mid t-(t_a-1) = k + W(t_a,t) \quad \square \quad (9)$$

## 4 Reducción de carga computacional

Como se ha mencionado, la complejidad computacional del método es $m*n^2$ donde $n$ es el número de tareas y $m$ representa el número de iteraciones necesarias para obtener los valores máximos de $SD_i(t_a)$, estando acotado por $De_n.(\boldsymbol{M}\text{-}W(\boldsymbol{M})+1)$.

El cálculo de la expresión requiere una inspección exhaustiva sobre $t$ en el intervalo $[0, D_i]$ con lo que el número de operaciones necesario para su determinación dependerá de la definición de la unidad de tiempo del sistema y resultará directamente proporcional a la precisión deseada. En principio, ésta podría resultar tan grande como lo permita la representación numérica utilizada. Este problema se agrava cuando el sistema de tiempo real es implementado sobre una plataforma *DVS*. En ellos, se intenta sacar provecho del slack disponible, para producir ahorro de energía mediante la reducción de la frecuencia de operación de la CPU. En este caso, las diferentes frecuencias de operación pueden resultar no armónicas con lo que, para mantener la precisión del cálculo es necesario definir unidades de tiempo muy pequeñas. Debe notarse que la integración del error en el tiempo, produce un menor aprovechamiento energético.

Obsérvese que, para fines prácticos, mientras la especificación del tiempo de ejecución es variable con la frecuencia de operación del sistema, los períodos y vencimientos permanecen invariantes ya que dependen de la aplicación y no de la velocidad del procesador. Por ejemplo, una tarea de tiempo de ejecución de 1ms a 1 GHz tendrá un tiempo de ejecución de 1,66ms a 0,6 GHz.

Para calcular la ecuación (6) con tiempos de ejecución fraccionarios, se puede implementar un método de búsqueda binaria, una vez que se obtiene el valor entero del $k_i$ para que obtenga la solución en los límites de la precisión del sistema ($\varepsilon$). Por lo tanto, para obtener el valor de $k_i$ con un error menor al error del sistema se deberá incrementar marcadamente el número de iteraciones.

Debido a que el costo computacional es alto y a fin de permitir la utilización del método en aplicaciones reales, se tomarán algunas consideraciones tendientes a reducir este costo, además de minimizar el error introducido por la cuantización en los parámetros temporales.

Existen dos formas de reducir la carga computacional. Por un lado se intenta que el cálculo comience con un $t$ lo mas cercano posible al $t$ que satisface (6) para un $SD_i$ dado y por otro lado, obtener una expresión que permita reducir dicho intervalo a una sucesión de valores con una baja cardinalidad.

***Lema 2**:*

*Una vez finalizada la ejecución de una $\tau_i$ en $te_{i,h}$, el máximo slack disponible $SD_i$ se encontrará próximo al arribo de una tarea de prioridad mayor o igual que i en el intervalo $[t_{i,h+1}, D_{i,h+1}]$.*

**Prueba**: Se parte de la suposición de que el máximo slack disponible $SD_i^{'}$, se puede encontrar en cualquier instante $t \in \left[t_{i,h+1},\ D_{i,h+1}\right]$ con $t \neq \alpha T_j; \alpha \in \aleph$ y $1 \leq j \leq i$. Si esto se cumple, en dicho intervalo existiría un tiempo entre la finalización de $\tau_i$ en $te_{i,h+1}$ y el arribo de alguna tarea de prioridad mayor o igual a la de $\tau_i$ que permanecería ocioso. Luego, resulta inmediato que el valor $SD_i^{'}$ no es el máximo retardo admisible por $\tau_i$, lo que contradice la suposición inicial.□

Del *Lema 1*, si el sistema es diagramable entonces, para $t_a = te_{i,h}$, existe $t = t^* > t_a$ que satisface la ec.(6) con $k \geq 0$ y por lo tanto:

$$k_i = t^* - (te_{i,h} - 1) - \sum_{j=1}^{i} \left[ C_j \left( \left\lceil \frac{t^*}{T_j} \right\rceil - \left\lfloor \frac{te_{i,h} - 1}{T_j} \right\rfloor \right) - C_j(te_{i,h}) \right] \tag{10}$$

Si $\tau_i$ finaliza su ejecución en $te_{i,h}$, en $te_{i,h}$-1 no existen requerimientos pendientes del subsistema $S(i\text{-}1)$ con lo que la función trabajo del subsistema $S(i)$ en $te_{i,h}$ está dada por:

$$W(te_{i,h}) = \sum_{j=1}^{i} C_j \left\lfloor \frac{te_{i,h} - 1}{T_j} \right\rfloor + C_j(te_{i,h}) \tag{11}$$

Luego la ec. (9) puede escribirse como

$$k_i = t^* - \sum_{j=1}^{i} C_j \left\lceil \frac{t^*}{T_j} \right\rceil + W(te_{i,h}) - (te_{i,h} - 1) \tag{12}$$

La ec. (12) es continua y no derivable en los instantes $t^* = \alpha T_j$, con $\alpha \in \aleph$, $1 \leq j \leq n$ y creciente en $t^* \neq \alpha T_j$. Estos puntos corresponden a los instantes de activación de las tareas del subsistema $S(i)$ por lo tanto, a fin de reducir el intervalo de inspección para el cálculo de $SD_i(te_{i,h})$ basta con obtener el máximo $k_j$, $1 \leq j \leq i$, en $t^* = \alpha T_j$ con $t_{i,h+1} < t^* \leq D_{i,h+1}$.

Para: $t^* = \alpha T_j, \quad \alpha \in \aleph$

$$\mathbb{k}_i = \left\{ \bigcup_{j=1}^{i} \left( \bigcup_{t^* = t_{i,h+1}}^{D_{i,h+1}} \left( t^* - \sum_{j=1}^{i} C_j \left\lceil \frac{t^*}{T_j} \right\rceil + W(te_{i,h}) - (te_{i,h} - 1) \right) \right) \right\} \tag{13}$$

Luego

$$SD_i(te_{i,h}) = \max\{\mathbb{k}_i\} \tag{14}$$

La sucesión $\mathbb{k}_i$ que se obtiene con la ecuación (13) posee un número de elementos que resulta función de la relación entre los períodos de $\tau_i$ y del subsistema $S(i\text{-}1)$ ya que representa el número de arribos de las tareas del subsistema $S(i)$ en el intervalo $t_{i,h+1} < t^* \leq D_{i,h+1}$.

$$\left|\mathbb{k}_i\right| = \sum_{j=1}^{i} \left\lceil \frac{D_{i,h+1}}{T_j} \right\rceil - \left\lfloor \frac{t_{i,h+1}}{T_j} \right\rfloor$$

en el peor caso (instante crítico) y para $i=n$

$$\left|\mathbb{k}_n\right| = \sum_{j=1}^{n} \left\lceil \frac{D_n}{T_j} \right\rceil - \left\lfloor \frac{1}{T_j} \right\rfloor = \sum_{j=1}^{n} \left\lceil \frac{D_n}{T_j} \right\rceil \tag{15}$$

Este valor representa una cota superior en el número de computaciones que no depende de la unidad de tiempo del sistema. Si existiesen varios máximos, el tiempo donde $\tau_i$ finaliza será el de menor $t$.

### 4.1 Reducción del intervalo de inspección

En [18-20] se ha demostrado que si $te_{i,1}$ es solución de la ec. (6) para $k = 0$ en el instante crítico, en el todo intervalo $[t, t+ te_{i,1}]$, se podrá encontrar una solución a la ec. (6). En [4] Davis, utiliza esta propiedad para diseñar el método denominado *Dual Priority*. En este trabajo se retraza la ejecución de $\tau_i$, $D_i - te_{i,1}$ unidades de tiempo. Como los métodos de *SS* buscan retrazar las tareas al máximo, la solución de la ec. (6) que nos interesa encontrar estará en el intervalo más próximo al vencimiento de la tarea.

***Lema 3***:

*Siendo $te_{i,1}$ la solución de la ec.* (6) *para el instante crítico de $\tau_i$, la solución para este y cualquier otro estado de carga con el máximo $k_i$, se encontrará en el intervalo* $[D_i - te_{i,1}+ C_i, D_i]$.

**Prueba**: Si $te_{i,1}$ es solución para la ec. (6) en el instante crítico, puede retrazarse el inicio de la ejecución de dicha tarea $D_i - te_{i,1}$ unidades de tiempo (igual que en [4]). Antes de su vencimiento $\tau_i$ deberá completar su ejecución y esto estará en el intervalo $[D_i - te_{i,1}+ C_i, D_i]$. Entonces la solución a la ec. (6) con el máximo $k_i$ se encontrará en este intervalo. □

Por el *Lema 3*, el intervalo de inspección para el cálculo del slack disponible queda entonces limitado al intervalo $[D_i - t_{1,i} + C_i, D_i]$ y el número de operaciones se reduce a:

$$\left|\mathbb{k}_i\right| = \sum_{j=1}^{i} \left\lceil \frac{D_{i,h+1}}{T_j} \right\rceil - \left\lfloor \frac{t_{i,h+1} + D_i - t_{1,i} + C_i}{T_j} \right\rfloor \tag{16}$$

### 4.2 Método Aproximado

Como se puede observar de la ec. (16), dependiendo de las relaciones entre los periodos, el número de operaciones puede resultar aún elevado. Es por ello que se ofrece la siguiente conjetura la cual resulta el inicio de un futuro trabajo:

***Conjetura 1:***

*Dada una tarea $\tau_i$, el máximo valor del slack disponible se obtendrá en un instante t comprendido en el intervalo* $[D_i - \Delta, D_i] \leq [D_i - t_{1,i} + C_i, D_i]$.

La conjetura se basa en que, de que $\tau_i$ fue atrasada en su ejecución hasta el máximo posible, el intervalo entre que termina $\tau_i$ y su vencimiento, solo hay ejecución de tareas de mayor prioridad o es un intervalo nulo, es decir la tarea termina en justo antes de su vencimiento, por lo tanto el cálculo del máximo slack se encontrara, por el *Lema 2*, en los arribos de las tareas de mayor prioridad en las proximidades o en el vencimiento de $\tau_i$.

Un método aproximado, pero no exacto, sería calcular una cantidad fija de puntos que cumplan con el *Lema 2*, en las proximidades del vencimiento de $\tau_i$. Pruebas preliminares han demostrado que para diversos sistemas es una muy buena aproximación. Cabe aclarar que si todos estos puntos fueran negativos se podrá igualar el slack de $\tau_i$ a cero, debido a que el sistema es diagramable.

## 5 Conclusiones

En este trabajo se ha demostrado que es posible reducir la complejidad del cálculo del *SS*, reduciendo el intervalo de inspección necesario para el cálculo del mismo. Con estas mejoras es posible aplicar el cálculo del máximo slack en línea, por ejemplo en aplicaciones para sistemas embebidos. Para ello, se han presentado dos métodos, uno exacto y otro aproximado. Queda para trabajos futuros desarrollar una heurística que aproveche estas nuevas características tratando de reducir la complejidad computacional y contrastar la carga computacional con métodos ya creados.

## 6 Referencias


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

[2] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, pp. 46-61, 1973.

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

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

[5] S. Saewong and R. Rajkumar, "Practical Voltage-Scaling for Fixed-Priority RT-Systems," presented at 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, 2003.

[6] P. Pillai and K. G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," presented at 18th Symposium on Operating Systems Principles, Banff, Alberta, Canada, 2001.

[7] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," presented at Proc. Of the 36th Design Automation Conference, 1999.

[8] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Enviroments," presented at IEEE Real-Time Systems Symposium, 1987.

[9] B. Sprunt, J. P. Lehoczky, and L. Sha, "Exploiting Unused Periodic Time For Aperiodic Service Using The Extended Priority Exchange Algorithm," 1988.

[10] 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.

[11] G. Fohler, T. Lennvall, and G. Buttazzo, "Inproved Handling of Soft Aperiodic Tasks in Offline Scheduled Real-Time Systems using Total Bandwidth Server."

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

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

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

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

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

[17] E. Bini, G. Buttazzo, and G. Buttazzo, "A Hyperbolic Bound for the Rate Monotonic Algorithm," *IEEE Transactions on Computer*, vol. 52, pp. 933-942, 2003.

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

[19] J. P. Lechoczky, "Fixed Priority Scheduling of Periodic Task Sets With Arbitrary Deadline," *Proceedings 11Th IEEE Real-Time Systems Symposium, Lake Buena Vista, Fl, USA*, pp. 201-209, 1990.

[20] 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.
[21] 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.
[22] J. D. Orozco, R. M. Santos, J. Santos, and R. Cayssials, "Taking advantage of priority inversions to improve the processing of non-hard real-time tasks in mixed systems," presented at WIP 21st IEEE Real-Time Systems Symposium, 2000.