SciVerse ScienceDirect
Disponible en www.sciencedirect.com

Revista Iberoamericana de Automática e Informática industrial 10 (2013) 186–196

www.elsevier.es/RIAI

Selección del periodo para la minimización del hiperperiodo

Vicent Brocal[a,*], Patricia Balbastre[b]

[a] *Fent Innovative Software Solutions*
[b] *Institut d'Automàtica i Informàtica Industrial – Universitat Politècnica de València*

Resumen

En este artículo presentamos un nuevo modelo de tareas donde el periodo de una tarea no es un valor fijo sino que, de acuerdo con un interpretación más amplia, el periodo puede ser elegido dentro de un intervalo de periodos aceptables. El objetivo principal es dotar al modelo de flexibilidad suficiente para que sea posible una reducción drástica del hiperperiodo del conjunto de tareas. El modelo está enfocado a sistemas de planificación cíclica, donde el ciclo del plan está determinado por el hiperperiodo. Sin embargo, la propuesta también es aplicable a la generación de cargas sintéticas para simulaciones, donde la reducción del hiperperiodo tiene beneficios en términos de complejidad y duración de la simulación. Debido a que el hiperperiodo crece exponencialmente con el número de tareas y con el valor de sus periodos, el análisis de los sistemas se vuelve intratable si el hiperperiodo excede unos límites razonables.

A su vez, se propone un algoritmo que permite el cálculo del hiperperiodo de acuerdo con el modelo de tareas presentado. Este algoritmo es capaz de calcular el hiperperiodo mínimo incluso para conjuntos de tareas grandes, donde la enumeración exhaustiva no es factible.

Palabras Clave: Tiempo-real, Modelo, Algoritmos de planificación

1. Introducción

El modelo de tareas periódicas es la base de la teoría de planificación de tiempo-real. Este modelo ha demostrado ser práctico y riguroso: permite representar adecuadamente el comportamiento temporal del mundo real, así como servir de modelo formal sobre el que es posible realizar análisis teóricos. En este modelo un sistema de describe mediante un conjunto de tareas periódicas, que en esencia se caracterizan mediante dos parámetros: C_i y P_i; el peor caso del tiempo de ejecución —WCET por sus siglas en inglés, *worst case execution time*— y el periodo, respectivamente. En Liu and J.W.Layland. (1973) se establecen las bases para el análisis de planificabilidad de sistemas de tiempo real basados en prioridades, que usen el modelo de tareas periódicas.

Generalmente, se atribuye el éxito del modelo de tareas periódico al hecho de que permite representar adecuadamente lo requisitos temporales de aplicaciones muy diversas. El modelo de tareas básico asume que el WCET se puede obtener mediante la inspección del código de la aplicación, o mediante la medida de los tiempos de ejecución de esta, cuando se dan las condiciones más desfavorables. En lo que respecta el periodo, este parámetro se deriva de las restricciones físicas de los dispositivos externos que forman parte del sistema y de cuyo control son responsables las tareas. Casi toda la teoría de análisis de planificabilidad existente está basada en que, tanto el periodo como el peor caso de tiempo de cómputo, son valores fijos y conocidos. Este supuesto permite simplificar enormemente el análisis, pero a cambio limita la aplicabilidad del modelo periódico y por tanto los resultados que se puedan derivar del él.

Relajar las restricciones del modelo estándar de tares, definido en base a periodos y plazos —*deadline*— fijos, permite obtener un mayor uso de los recursos, mejor rendimiento de los procesos de control, ahorro de energía, y puede permitir la adaptación a entornos dinámicos, situaciones de sobrecarga, etc.

En aplicaciones con un comportamiento más dinámico, como puedan ser los sistema multimedia, donde los periodos pueden no ser tan rígidos como en aplicaciones de control industrial, sigue siendo necesario ejecutar tareas periódicamente. Es más, incluso en las aplicaciones de control industrial se pueden obtener beneficios derivados de la ejecución de las tareas a distinta frecuencia en función de las condiciones de operación. En este sentido, la aproximación habitual consiste en adaptar, en tiempo de ejecución, los periodos de tal manera que se optimice el rendimiento global del sistema (Cervin and Eker (2000); Martí et al. (2001)). Por ejemplo, en Buttazzo et al. (1998) se

*Autor en correspondencia
Correos electrónicos: vbrocal@fentiss.com (Vicent Brocal), patricia@disca.upv.es (Patricia Balbastre)

http://dx.doi.org/10.1016/j.riai.2013.03.009

propone un modelo de tareas *elástico*, en el que los periodos de las tareas se presentan como *muelles* de tal manera que la utilización de las tareas varia con la modificación de los periodos dentro de un rango predefinido durante el diseño del sistema.

En numerosos trabajos se han propuesto modificaciones al modelo de tareas periódicas con la intención de aumentar la flexibilidad de los plazos, la de los periodos, incluir la posibilidad de que existan retardos de inicio —*offsets*—, de que se permita definir relaciones de precedencia entre tareas, que considere la problemática derivada del acceso a recursos compartidos, etc.

En este trabajo retomamos la idea de la variación de los periodos con un objetivo fundamentalmente distinto: la reducción del hiperperiodo del conjunto de tareas. Generalmente, el hiperperiodo, representado como H, se calcula como el mínimo común múltiplo (M.C.M.) de los periodos de las tareas.

En Leung and Merrill (1980) se demuestra que para un plan generado por una política de planificación expulsiva, esto es un sistema de tiempo real periódico, el patrón de activaciones de la tareas se repite dentro de un intervalo que coincide con el hiperperiodo H. El primer test de planificabilidad, que en su momento se propuso como parte del mismo trabajo, consistía en comprobar que el sistema es planificable dentro del intervalo $[0, H[$. Posteriormente, Baruah et al. (1990a) y Ripoll et al. (1996) propusieron mejoras a este test en aras de obtener un intervalo más preciso para evaluar la planificabilidad de sistemas basados en un política EDF —*Earliest Deadline First*.

Para sistemas asíncronos, donde las tareas pueden tener un retardo de inicio —offset—, se ha propuesto el intervalo de análisis de la planificabilidad $[0, 2H + \phi)$, donde ϕ representa del *offset* mayor (Baruah et al. (1990b)).

En planificación cíclica, el hiperperiodo es conocido habitualmente como *major frame* (MAF). Este parámetro se complementa con el *minor frame*, que define el intervalo mínimo dentro del cual un proceso se puede sincronizar. El *minor frame* se calcula como un submúltiplo del MAF. Cuando se pretende diseñar un sistema mediante esta aproximación, se debe verificar que sus restricciones temporales se verifican dentro del MAF. Una vez construido el plan, es posible guardarlo en una estructura fija que se consultará en tiempo de ejecución para determinar cuál es la tarea que debe ejecutarse en cada instante de tiempo. Generalmente, el tamaño del MAF repercute significativamente en el tamaño de esta estructura y por tanto en la cantidad de memoria empleada. Por ambas razones, es deseable disponer de MAFs tan pequeños como sea posible.

Por otra parte, en sistemas donde se establecen restricciones de precedencia entre las tareas, dichas restricciones se definen como grafos dirigidos. Cuando se pretende usar una aproximación heurística para la obtención de una solución al problema de planificación, la reducción del tamaño del hiperperiodo juega un papel fundamental en la reducción del numero de operaciones sobre el grafo de dependencias (Kermia et al. (2006)).

La evaluación del sistema de tiempo real mediante simulación también se puede beneficiar en gran medida de un hiperperiodo reducido: en el caso de que se empleen generadores de carga aleatoria, el problema estriba en encontrar una asignación de periodos que generen un hiperperiodo tratable computacionalmente. Por un lado, es necesario que el hiperperiodo sea representable dentro de la resolución limitada de los computadores, por otro, a mayor hiperperiodo mayor longitud de simulación, siendo deseable que esta esté dentro de un límite práctico.

Cuando los periodos de las tareas son altamente primos entre si, es decir tienen pocos divisores comunes, el hiperperiodo crece notablemente. De hecho, Macq and Goossens (2001) demostraron que el hiperperiodo crece exponencialmente de acuerdo con el periodo más grande del conjunto de tareas y con el número de tareas.

Por todas las razones mencionadas anteriormente, es deseable disponer de hiperperiodos tan cortos como sea posible. La técnica más habitualmente empleada con este propósito es la selección de periodos harmónicos, que posean divisores tan grandes como sea posible. Por ejemplo, en aplicaciones de radar se definen periodos harmónicos que permiten una sobrecarga menor para el análisis de planificabilidad y que, por desgracia, suponen un uso muy poco eficiente del tiempo de cómputo disponible. Para superar este inconveniente, en Shih et al. (2003) se propone un algoritmo que a partir de los periodos de las tareas genera un conjunto de periodos sintéticos. El problema es que en ocasiones no es posible modificar los periodos de las tareas para que sean harmónicos entre sí, debido a la naturaleza diferente de las tareas que hace que puedan ejecutarse en bases de tiempo distintas, haciendo imposible la modificación.

En Xu (2010) se presenta una propuesta de reducción del hiperperiodo mediante la reducción automática de los periodos de las tareas. Por ser el trabajo más parecido a nuestra propuesta, se describirá con más detalle en la sección 5.

1.1. Ejemplo de motivación

El propósito de este ejemplo es ilustrar la reducción que se puede conseguir en el hiperperiodo con una modificación mínima de los periodos iniciales de las tareas.

Supongamos un conjunto de tres tareas con periodos 20, 28 y 93 respectivamente. Si el hiperperiodo se calcula como el mínimo común múltiplo de los tres valores, obtenemos que $H = \text{MCM}(20, 28, 93) = 13020$. Por simplicidad, este ejemplo se circunscribe a la modificación del periodo de la última tarea. En el caso de que, dentro de las restricciones impuestas por el sistema, sea posible definir valores aceptables para el periodo dentro de, digamos, el intervalo $[90, 95]$, entonces eligiendo 90 como el periodo final, el hiperperiodo resultante $H = \text{MCM}(20, 28, 90) = 1260$ experimenta una reducción significativa. En este caso se ha elegido el periodo (90) que, además, produce el hiperperiodo mínimo.

En este trabajo proponemos la definición de los periodos de las tareas no como un valor fijo sino como un rango aceptable de valores, con la intención de que una elección adecuada de los periodos finales produzca el mínimo hiperperiodo posible.

En este sentido, el siguiente paso natural es extender la definición de rangos de periodos para todas la tareas. Supongamos que se han definido los rangos [17,21], [25,30] y [90,95], respectivamente. ¿Cual sería en este caso la selección de periodos que minimiza el hiperperiodo? Una primera aproximación podría consistir en la enumeración de todos los posibles hiperperiodos para, posteriormente, seleccionar el mínimo de entre todos ellos.

Una propuesta de algoritmo que implemente esta aproximación puede verse en el código 1. Para conjuntos de tareas relativamente grandes esta aproximación es inviable, ya que el algoritmo tiene un coste asintótico exponencial con el numero de tareas y el tamaño del mayor intervalo.

```
1  H_min:integer:=integer'last;
2  L,L':integer;
3  function ComputeMCM(L,i) is
4     if (i>n) then
5        if (L<H_min) then
6           H_min:=L;
7        end if;
8        return;
9     end if;
10    for x in l_i..u_i loop
11       L'=MCM(L,x);
12       ComputeMCM(L',i+1)
13    end loop;
14 end ComputeMCM;
15
16 function MinHyperExhaustive(τ) is
17    ComputeMCM(1,1);
18 end MinHyperExhaustive;
```

Codigo 1: Exhaustive search algorithm

1.2. Contribuciones y organización

En este artículo proponemos una extensión del modelo de tareas básico, donde los periodos de las tareas no se definen como un valor fijo, sino como un rango de valores aceptables. Además presentamos un algoritmo eficiente para la obtención del hiperperiodo mínimo a partir de estos rangos. El resultado del algoritmo es un conjunto de tareas donde las tareas tienen un periodo fijo —modelo de tareas clásico—, y que a su vez producen el hiperperiodo mínimo.

El algoritmo propuesto puede usarse como pre-proceso en aplicaciones donde es necesario disponer de un hiperperiodo reducido.

El resto del artículo está organizado de la siguiente manera: en la sección 2 se presenta el modelo de tareas extendido y la definición formal del problema de minimización del hiperperiodo; a continuación, en la sección 3, se describe el algoritmo propuesto así como consideraciones sobre su coste temporal; en la sección 4 se ilustra el funcionamiento del algoritmo mediante un ejemplo; más adelante, en la sección 5 se establece una comparativa con un trabajo anterior y en la sección 6 se presentan resultados experimentales sobre el rendimiento del algoritmo; finalmente en la sección 8 se resumen las conclusiones.

2. Modelo de sistema

Sea $\tau = \{\tau_1, ..., \tau_n\}$ un sistema de tiempo real periódico formado por n tareas. El periodo de cada tarea $\tau_i \in \tau$ se define como un rango de enteros ($T_i = [l_i, u_i]$). El rango no significa que durante la ejecución la tarea pueda adoptar cualquier periodo dentro de ese rango. De hecho el periodo no es variable sino fijo, pero se elige en tiempo de diseño dentro del rango de valores definidos para cada tarea. Se contempla la posibilidad de que $l_i = u_i$ y, por tanto, existan tareas dentro del conjunto τ que tengan un periodo fijo. El algoritmo propuesto en este trabajo es capaz de obtener una solución también en estos casos. También se supone que el tiempo se divide en un número contable infinito de "slots" iguales de 1 unidad. Es decir, suponemos que los periodos son valores enteros.

Definición 1. *Sea* $\mathbb{H} = \{MCM(t_1, \cdots, t_n)\}$*, donde* $t_i \in [l_i, u_i]$.

$\mathbb{H}$ *es el conjunto de valores enteros correspondientes a todos los posibles MCM de todas las posibles combinaciones de periodos de tareas dentro de su rango.*

De aquí en adelante, sea $h \in \mathbb{H}$.

2.1. Planteamiento del problema

El problema a resolver consiste en encontrar $H_{min} = \text{mín}(\mathbb{H})$, es decir, el mínimo del conjunto $\mathbb{H}$. Este valor puede obtenerse calculando el conjunto $\mathbb{H}$, ordenándolo y seleccionando el primer elemento. El código 1 detalla el pseudocódigo que implementa el algoritmo de construcción del conjunto $\mathbb{H}$ y obtiene el mínimo H_{min}.

El problema que surge con esta técnica reside en la talla del conjunto $\mathbb{H}$, la cual es $|\mathbb{H}| = \prod_{i=1}^{n} (u_i - l_i + 1)$. Para rangos no demasiado grandes, el productorio anterior puede llegar a ser un valor muy grande, convirtiendo el problema en intratable.

2.2. Complejidad del problema

El coste de encontrar H_{min} está acotado por $O(O(MCM) \cdot |\mathbb{H}|)$, es decir, el coste de calcular el MCM de los n enteros multiplicado por la talla del conjunto $\mathbb{H}$.

El problema de calcular el M.C.M. se reduce al problema de calcular el máximo común divisor (M.C.D.). En este sentido, el problema de calcular el M.C.D. es análogo al problema de la factorización de enteros, para el cual no se conoce ningún algoritmo eficiente. Además, se desconoce a que clase de complejidad pertenece el problema de factorización de enteros, pero se sospecha que se encuentra fuera de las tres clases de complejidad P-NP (P, NP Completo, co-NP Completo).

En nuestro caso, el problema de encontrar el mínimo hiperperiodo de un conjunto de rangos de periodos puede reducirse al problema de la factorización de enteros. Veámoslo con un ejemplo: supongamos dos tareas $\{\tau_1, \tau_2\}$ con rangos de periodos: $t_1 \in [x, x]$ y $t_2 \in [2, \sqrt{x}]$. Supongamos que existe t_2 tal que $x = at_2$, siendo $a \in \mathbb{N}$. Entonces, $H_{min} = x$ y el problema de encontrar H_{min} se ha reducido al problema de la factorización del entero x.

Como se ha comentado anteriormente, la complejidad de la factorización de enteros no es completamente conocida. La cota más aproximada hasta el momento conocida para b bits es (Crandall and Pomerance (2005)):

$$O\left(exp\left(\left(\frac{64}{9}b\right)^{\frac{1}{3}} log(b)^{\frac{2}{3}}\right)\right)$$

Existen en la literatura trabajos que proponen algoritmos más rápidos que $O((1+\epsilon)^b)$ (tiempo subexponencial) para todo número positivo ϵ (Crandall and Pomerance (2005)).

3. Fast Hyperperiod Search

Proponemos una heurística, el algoritmo *Fast Hyperperiod Search (FHS)*, que es capaz de calcular de manera eficiente el mínimo hiperperiodo de un conjunto de rangos de valores. Es importante destacar que el algoritmo calcula la solución exacta y no una aproximación. Como se muestra más adelante en la sección 6, este algoritmo es significativamente mejor en términos de coste temporal que la búsqueda exhaustiva por enumeración de los candidatos propuesta `MinHyperExhaustive` que se ha presentado en el código 1.

A continuación se presentan las definiciones, y las propiedades de los hiperperiodos, que permiten respaldar formalmente el diseño del algoritmo.

Definición 2. *Sea g el mínimo común múltiplo de los enteros $(t_1, .., t_m)$, donde t_i es un periodo válido para la tarea τ_i $1 \leq i \leq m < n$.*

$$g = MCM(t_1, \ldots, t_m) \,/\, t_i \in [l_i, u_i], 1 \leq i \leq m < n$$

Definición 3. *Sea $\mathbb{G}$ el conjunto de todos los posibles mínimo común múltiplo para m tareas, es decir, todos los posibles valores de g para m tareas.*

$$\mathbb{G} = \{MCM(t_1, \ldots, t_m) \mid \forall t_i \in [l_i, u_i], 1 \leq i \leq m < n\}$$

Inicialmente, el algoritmo pre-calcula un conjunto $\mathbb{G}$ que contiene **todos** los hiperperiodos válidos para un subconjunto de m tareas. Esto se lleva a cabo mediante la función `EnumerateMCM()`, empleada en el código del algoritmo 3. Esta función, de forma recursiva, calcula los MCM de todas las posibles combinaciones de periodos dentro de los rangos de cada tarea (ver listado2).

```
1  function EnumerateMCM(τ,L,i) is
2    if (i=m) then
3      G[iter++]=L;
4      return G;
5    end if;
6    for x in l_i..u_i loop
7      L'=MCM(L,x);
8      EnumerateMCM(L',i+1)
9    end loop;
10 end EnumerateMCM;
```

Codigo 2: Algoritmo EnumerateMCM

Dado que cada valor g es el MCM de los periodos para un subconjunto de tareas,

```
1  function FHS (τ, m)
2  begin
3    H_min := 1
4    for τ_i in τ loop
5      H_min := MCM (H_min, l_i)
6    end for
7    iter = 0;
8    G := EnumerateMCM (τ, 1, m)
9    d := 1
10   while H_min > min (d * G) loop
11     H_min := FindMCM (τ, d * G, m, H_min)
12     d := d + 1
13   end while
14   return H_min
15 end FHS
```

Codigo 3: Algoritmo FHS

Propiedad 1. *Cada $h \in \mathbb{H}$ puede expresarse en términos del MCM de g y un conjunto de periodos $t_{m+1}, \ldots, t_n$ pertenecientes a tareas que no están en el conjunto inicial de m tareas.*

$$\forall h \in \mathbb{H} \rightarrow \exists g \in \mathbb{G} \,/\, h = MCM(g, t_{m+1}, \ldots, t_n)$$

De ello se sigue que h es un múltiplo de g.

Por ejemplo, sea $\tau = \{\tau_1, \tau_2, \tau_3, \tau_4\}$. Para simplificar el ejemplo, supongamos que $\forall i\ T_i = l_i = u_i$ y que los valores concretos de periodos son $T_i = \{5, 6, 10, 13\}$. Entonces, si m=2, $\mathbb{G} = g_0$, de forma que $g_0 = MCM(T_1, T_2) = 30$. Y así,
$\mathbb{H} = MCM(MCM(T_1, T_2), T_3, T_4) = MCM(30, 10, 13) = 390$.

Definición 4. *Sea $\mathbb{D}$ el conjunto de valores por los que algún $g \in \mathbb{G}$ es múltiplo de algún $h \in \mathbb{H}$.*

$$\mathbb{D} = \{d \in \mathbb{N} \mid h = d * g\}$$

Propiedad 2. *Dado que cada h es un hiperperiodo válido, entonces es posible encontrar al menos un periodo en el rango definido para cada tarea $t_i \in [l_i, u_i]$, que es un divisor de $d * g$.*

$$\forall \tau_i \in \tau \rightarrow \exists t_i \in [l_i, u_i] \,/\, d * g \bmod t_i = 0$$

En esencia, el algoritmo trabaja enumerando incrementalmente valores para $d \in \mathbb{N}$ y comprobando que pertenecen a $\mathbb{D}$ —es decir, cumplen la condición $h = d * g$— mediante la función `FindMCM` (código 4). Además está función es capaz de encontrar el mínimo valor $d * g$ que resulta ser un hiperperiodo válido h. El algoritmo converge iterativamente al valor H_{min}, ya que va guardando el mínimo hiperperiodo, devuelto por `FindMCM`, encontrado hasta el momento. El proceso termina cuando el mínimo hiperperiodo encontrado es inferior al mínimo valor para $d * \mathbb{G}$.

A pesar de que en cada iteración se calcula el mínimo $d * g$, tal como se puede ver en la figura 1, puede suceder que alguno de estos valores no sea el hiperperiodo mínimo H_{min}. Supongamos que existe un g_0 y un g_1 que pertenecen a $\mathbb{G}$, y se verifica que

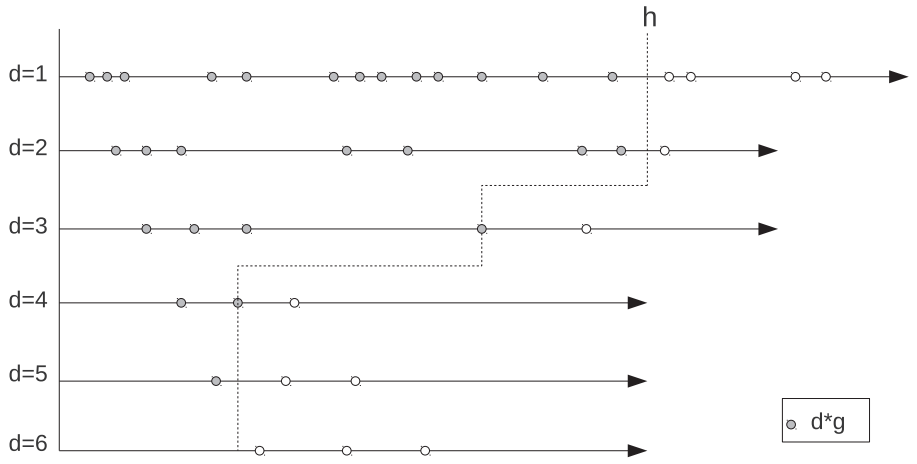

Figura 1: Ejemplo ilustrativo de funcionamiento del algoritmo

```
1  function IsMCM (g, τ_i)
2  begin
3    for t in l_i..u_i
4      if g mod t = 0 then
5        return true
6    end for
7    return false
8  end IsMCM
9
10 function FindMCM (τ, G, m, H_min)
11 begin
12   for g in G loop
13     if g ≥ H_min then
14       return H_min
15     end if
16
17     for i in (m + 1)..n loop
18       found := IsMCM (g, τ_i)
19       if not found then
20         break
21     end for
22     if found then
23       return g
24     end if
25   end for
26   return H_min
27 end FindMCM
```

Codigo 4: FindMCM algorithm

$$g_0 < g_1$$

Supongamos ahora que $d * g_1 \in \mathbb{D}$, es decir, que $d * g_1$ es un hiperperiodo válido. Para que sea mínimo entonces se tiene que satisfacer que

$$(d + 1) * g_0 \geq d * g_1,$$

El lector observará que esto puede no ser siempre cierto.

En cambio, el algoritmo propuesto es siempre capaz de encontrar el mínimo MCM para alguna de las posibles combinaciones de $[l_i, u_i]$. Para dar prueba de ello primero demostramos que el hiperperiodo obtenido es, efectivamente, múltiplo de algún periodo $t_i \in [l_i, u_i]$. A continuación, se muestra que que el valor encontrado es el mínimo MCM posible.

Por construcción, todos los valores en $\mathbb{G}$ ya son múltiplos para las m tareas seleccionadas para formar $\mathbb{G}$, sean cuales sean estas tareas. Dado que el algoritmo funciona multiplicando cada unos de los valores de $\mathbb{G}$ para encontrar el MCM, H_{min} seguirá siendo múltiplo de al menos un $g \in \mathbb{G}$. De igual manera, en cada iteración el algoritmo busca un valor $h \in d * \mathbb{G}$ que es múltiplo del resto de tareas $\tau_{n-m+1} \ldots \tau_n$ (`FindMCM`); esto quiere decir que d pertenece al conjunto $\mathbb{D}$. Por tanto, H_{min} será también múltiplo de algún $t_j \in [l_j, u_j], n - m + 1 \leq j \leq n$.

De acuerdo con la definición 4 y la propiedad 2, $H_{min} = \text{mín}\{h \mid h = d * g\} = \text{mín}\{d * g\}$. Entonces, el algoritmo es capaz de encontrar H_{min} evaluando $d * G$ cuando el conjunto $\mathbb{G}$ está ordenado y se comprueba qué números naturales pertenecen a $\mathbb{D}$.

3.1. Coste temporal de FHS y selección de m

El coste del algoritmo `FHS` depende principalmente del coste del bucle principal. Por tanto, el coste del algoritmo está acotado por

$$O_{\texttt{FHS}}(\tau, m) = d_c * O_{\texttt{FindMCM}}(\tau, m)$$

donde d_c es el número de iteraciones del bucle principal, es decir, el entero final para el cual $H_{min} = d_c * g$. El rendimiento del algoritmo es altamente dependiente de d_c.

Para cada iteración comprueba los valores en $\mathbb{G}$, contra todos los valores dentro de los rangos de los periodos, para establecer si alguno de los valores de $d * \mathbb{G}$ son un hiperperiodo válido. A partir de esto podemos establecer una cota superior para el número de iteraciones necesarias para la convergencia del algoritmo dado por la expresión:

$$d_c \leq \frac{\text{MCM}_{\,1\leq i\leq n}(l_i)}{\text{mín}(\mathbb{G})}$$

Por tanto, el bucle principal se ejecutará hasta mientras $d * \text{mín}(\mathbb{G}) \leq H_{min}$. En el peor caso, podemos asumir que no se encontrará un hiperperiodo inicial menor que el hiperperiodo inicial $H^0_{min} = \text{MCM}(l_i)$.

A su vez, $\text{mín}(\mathbb{G})$ está acotado por $\text{máx}_{1\leq j\leq m}(l_j)$, que lleva a la expresión

$$d_c \leq \frac{\text{MCM}_{\,1\leq i\leq n}(l_i)}{\text{mín}(\mathbb{G})} \leq \frac{\text{MCM}_{\,1\leq i\leq n}(l_i)}{\text{máx}_{1\leq j\leq m}(l_j)}$$

De otro lado, el coste de `FindMCM` está acotado por

$$O_{\texttt{FindMCM}}(\tau, m) = (n - m + 1) \prod_{1\leq j\leq m} (u_j - l_j) O_{\texttt{IsMCM}}(\tau, m)$$

donde $\prod_{1\leq j\leq m}(u_j - l_j)$ representa $|\mathbb{G}|$, la máxima cardinalidad del conjunto $\mathbb{G}$. A su vez, `IsMCM` está acotado por

$$O_{\texttt{IsMCM}} = \max_{m+1\leq k\leq n} (u_k - l_k)$$

lo que nos lleva la cota final para el algoritmo `FHS`

$$O_{\texttt{FHS}}(\tau, m) = \frac{\text{MCM}\,(l_i)\,\text{máx}(u_k - l_k)(n - m + 1)\prod(u_j - l_j)}{\text{máx}(l_j)}$$

para $1 \leq i \leq n$, $1 \leq j \leq m$, $m + 1 \leq k \leq n$.

De acuerdo con la cota propuesta, es recomendable incluir en el conjunto $\mathbb{G}$ el rango con el límite inferior más grande —$max(l_j)$—, ya que proporcionará un valor $min(\mathbb{G})$ más grande. Además, a pesar de que el término $\prod_{1\leq j\leq m}(u_j - l_j)$ domina el coste del algoritmo, es improbable que en la ejecución promedio tenga una impacto significativo en el tiempo de ejecución del algoritmo. Esto es debido a que `FindMCM` detiene la búsqueda del hiperperiodo para el valor actual de g cuando $d * g$ es mayor que el H_{min} actual (ver la figura 1). Es más, a pesar de que $\prod_{1\leq j\leq m}(u_j - l_j)$ se incluye para proporcionar una cota superior segura a la cardinalidad del conjunto $\mathbb{G}$, es bastante improbable que todas la posibles combinaciones de $t_i \in [l_i, u_i]$ para las m tareas sean primos entre sí, y por tanto necesitando de un hiperperiodo distinto para cada combinación.

A la hora de elegir el parámetro m, es necesario tener en cuenta que la memoria necesaria para albergar todos los posibles hiperperiodos es igual al número de elementos del vector $\mathbb{G}$ por la talla de un entero largo. De esta forma, la memoria necesaria para albergar este vector sería:

$$size(long \quad long \quad integer) * \prod_{1\leq j\leq m} (u_j - l_j + 1)$$

3.2. *Cálculo de los periodos de las tareas*

Dado que que el algoritmo `FHS` depende de la construcción del conjunto $\mathbb{G}$, no es posible obtener directamente el valor final del periodo de las tareas $T_i \in [l_i, u_i]$ para el conjunto de las m tareas iniciales. Por esta razón, a continuación se presenta en el código 5 el algoritmo necesario para calcular el periodo final para todas las tareas a partir del rango de periodos de la tarea $[l_i, u_i]$ y el hiperperiodo mínimo obtenido mediante `FHS`. Por razones de simplicidad y genericidad, en este apartado se presenta el hiperperiodo mediante el símbolo H, bien sea este el valor mínimo calculado mediante `FHS` o mediante cualquier otro método que nos proporcione bien el mínimo o bien cualquier otro valor aceptable de hiperperiodo.

```
1  function MaximisePeriods (τ, H)
2  begin
3    for τ_i in τ loop
4      for k in ⌊H/u_i⌋ .. ⌈H/l_i⌉ loop
5        if IsInteger (H/k) then
6          T_i = H/k
7          break
8        end if
9      end for
10   end for
11 end MaximisePeriods
```

Codigo 5: MaximisePeriods algorithm

Para cada tarea se busca un periodo válido T_i que divida a H, y por tanto se tiene que satisfacer $H = kT_i$ para un entero k y $l_i \leq T_i \leq u_i$, y por tanto $\frac{H}{l_i} \leq k \leq \frac{H}{u_i}$. Dado que H ya es múltiplo de algún $T_i \in [l_i, u_i]$, entonces existirá un valor entero que verifique $T_i = \frac{H}{k}$.

Debido a la naturaleza de la definición de los rangos de periodos, se puede dar el caso de que el hiperperiodo mínimo sea divisible por más de un periodo válido para una misma tarea. Tómese como ejemplo un escenario trivial con $H_{\text{min}} = 10$ y $T_1 = [5, 10]$, donde 5 y 10 son ambos divisores de 10. El algoritmo propuesto ha sido diseñado para encontrar el periodo más grande, lo que permite minimizar la utilización de la CPU para cada tarea. Sería posible modificar el algoritmo con la intención opuesta y que encontrara el mínimo periodo.

Finalmente, el coste temporal de este algoritmo está acotado por la expresión $O(n) = \sum_{0\leq i\leq n} \left\lceil \frac{H}{l_i} \right\rceil - \left\lfloor \frac{H}{u_i} \right\rfloor + 1$, dado que en el peor caso todos los periodos T_i se ajustarán a sus mínimos valores permitidos l_i, y el algoritmo se verá forzado a comprobar todos los posibles candidatos.

4. Ejemplo

A continuación ilustramos el algoritmo `FHS` presentado en la sección anterior por medio de un ejemplo. Considérese un sistema con cuatro tareas representado en la Tabla1. Para cada tarea se muestran los límites inferior y superior del rango de periodos. Para este ejemplo usaremos una asignación para el parámetro $m = 2$.

Tabla 1: Ejemplo con cuatro tareas

	l_i	u_i
τ_1	7	9
τ_2	13	14
τ_3	22	24
τ_4	35	47

Mediante búsqueda exhaustiva, es necesario calcular el MCM para un total de 702 combinaciones, para finalmente proceder a seleccionar el mínimo valor. Para este conjunto de tareas $H_{min} = 84$, que se obtiene mediante la selección de periodos $T_1 = 7$, $T_2 = 14$, $T_3 = 22$ y $T_4 = 42$.

Aplicando el algoritmo `FHS`, primero es necesario calcular el conjunto $\mathbb{G}$. Con $m = 2$, esto implica calcular todos los posibles MCM para todas las posibles combinaciones de periodos para las tareas τ_1 y τ_2:
$\mathbb{G} = (14, 56, 91, 104, 117, 126)$.
Ninguno de estos valores es múltiplo de los periodos candidatos para τ_3 y τ_4, por lo que en la siguiente iteración $\mathbb{G}$ se multiplica por $d = 2$, de lo que se obtiene
$\mathbb{G} = (28, 112, 182, 208, 234, 252)$.
En este caso, tampoco hay ningún múltiplo de los n periodos candidatos. El algoritmo continua iterando con $d = 3$, donde
$\mathbb{G} = (42, 168, 273, 312, 351, 378)$,
y encontramos el primer hiperperiodo candidato $H^2_{min} = 168$. Para $d = 4$ $d\mathbb{G} = (56, 224, 364, 416, 468, 504)$, que no contiene ningún hiperperiodo menor.

Finalmente, para $d = 6$, encontramos otro hiperperiodo $H^3_{min} = 84$, que está asociado al primer elemento de $\mathbb{G}$ ($H^3_{min} = 14 * 6 = 84$), por lo que el algoritmo no podrá encontrar ningún hiperperiodo menor. Por tanto, $H_{min} = 84$.

5. Comparativa con trabajos previos

Tal como se comenta en la sección 1, en Xu (2010) se propone un método para la reducción del hiperperiodo de un conjunto de tareas mediante la reducción automática de los periodos originales. Dada la similitud con la propuesta presentada en este trabajo se ha considerado oportuno presentar una comparación de ambos métodos.

La principal diferencia de nuestra propuesta con la presentada en Xu (2010) reside en el hecho de que en la segunda se mantiene el modelo clásico de tareas donde para cada tarea se define un valor de periodo fijo. Esta propuesta tiene como principales objetivos:

- ajustar los periodos en aras de reducir el hiperperiodo, al mismo tiempo que
- asegurar que los periodos ajustados son suficientemente cercanos a los originales para mantener una utilización de la CPU de acuerdo con los requisitos de la aplicación, al mismo tiempo que se maximizan las posibilidades de encontrar un plan donde no existan pérdidas de plazo.

Con la intención de ajustarse a las necesidades de cómputo de las aplicaciones, Xu solo considera la posibilidad de reducir el periodo de las tareas. Por tanto, se producirá un incremento en la utilización que puede llevar el conjunto de tareas de ser planificable a no planificable. Además, debido al método usado para ajustar los periodos, Xu establece una desviación máxima entre los periodos originales y los ajustados. Para limitar esta desviación, Xu se ve forzado a limitar el máximo hiperperiodo candidato así como el numero de hiperperiodos candidatos.

Por las dos razones expuestas en el párrafo anterior, este método ve limitada su capacidad para minimizar el hiperperiodo.

El siguiente ejemplo, tomado de Xu (2010), se usará para comparar el método que en él se propone con la aproximación propuesta en este trabajo.

Un sistema de comunicaciones incluye las siguientes tareas periódicas:

- tarea *CD-Audio* (τ_1) que requiere una tasa de servicio de 2,75KHz, es decir tiene un periodo $T_1 = 364$ μs;
- tarea *ISDN Channel* (τ_2) que requiere de una tasa de servicio de 1,5KHz, es decir tiene un periodo $T_2 = 667$ μs;
- tarea *Voice Channel* (τ_3) que requiere una tasa de servicio de 1,375 KHz, es decir tiene una periodo $T_3 = 727$ μs;
- tarea *KeyboardIMouse* (τ_4) que requiere servicio periódico con T_4=100.000 μs.

Para las anteriores tareas se definen los siguientes tiempos de cómputo:

$$C_1 = 240\mu s \quad C_2 = 105\mu s \quad C_3 = 115\mu s \quad C_4 = 500\mu s$$

El hiperperiodo del conjunto de tareas es

$$H = \text{MCM}\,(364, 667, 727, 100000) = 4{,}412{,}671{,}900{,}000$$

Mediante el método propuesto en Xu (2010), se obtienen los siguientes periodos ajustados:

$$T_1 = 360\mu s \quad T_2 = 660\mu s \quad T_3 = 720\mu s \quad T_4 = 92400\mu s;$$

que resultan en un hiperperiodo de 227200 s.

Para poder establecer una comparación con el método propuesto en este trabajo es necesario definir los rangos de periodos válidos para cada tarea. Se ha estimado que la condición a satisfacer para definir estos periodos es que, tal como se considera en el trabajo de Xu, se mantenga la planificabilidad del sistema. Para ello se emplea la siguiente expresión para obtener los límites inferiores de los rangos:

$$l_i = \lceil UT_i \rceil \quad \forall i = 1..n$$

donde U es la utilización del procesador para los periodos originales. De esta manera en caso de que se seleccionen los periodo mínimos para cada rango, el sistema seguirá siendo planificable.

Para el ejemplo, U=0,979946268, los rangos resultantes se muestran en la tabla 2.

Tabla 2: Rangos propuestos (FHS 1)

	l_i	u_i
τ_1	357	364
τ_2	654	667
τ_3	713	727
τ_4	97995	100000

Para estos rangos, el hiperperiodo mínimo encontrado por el algoritmo FHS es $H_{min} = 196020$

Nótese que no solo el hiperperiodo mínimo es menor que el obtenido mediante el método de Xu, sino que además, tal como se puede ver en la tabla 4, la desviación de los periodos finales y la utilización son ambos menores.

Hemos querido también incorporar la comparativa en el caso de que se permitiera incrementar el periodo de las tareas, en aras de ilustrar mejor las ventajas del algoritmo FHS. En esta segunda comparativa, se han empleado las expresiones

$$l_i = \lceil UT_i \rceil$$

$$u_i = \lfloor (2 - U)T_i \rfloor$$

para calcular los rangos de periodos. Los valores obtenidos se recogen en la tabla 3. Para esta configuración, el valor del hiperperiodo mínimo obtenido mediante el algoritmo FHS es $H_{min} = 98420$. Este valor es casi una tercera parte del obtenido mediante el método propuesto por Xu. Además, el factor de utilización resultante es muy cercano al original (tabla 4).

La tabla 4 presenta un resumen de los resultados para que sea más sencilla su comparación. En las filas rotuladas como *FHS 1* y *FHS 2* se muestran los resultados obtenidos de aplicar el algoritmo FHS a los rangos de las tablas 2 y 3. En la columna *Desviación media* se muestra la desviación media de los periodos ajustados respecto a los periodos iniciales; en la columna *U* se muestra la utilización del conjunto de tareas.

Como se puede ver, la propuesta basada en rangos de periodos consigue una reducción importante en el hiperperiodo incluso cuando solo se admite la reducción – y no aumento – de los periodos. De igual manera, tanto la desviación respecto a los periodos originales como la utilización, son menores.

A pesar de que la aproximación *FHS2* permitiría la elección de periodos que conllevaran una reducción del ancho de banda de algunas tareas, se ha considerado interesante incluirlos para ilustrar los beneficios potenciales del modelo de tareas ampliado y su influencia en la reducción del hiperperiodo.

Tabla 3: Rangos propuestos (FHS 2)

	l_i	u_i
τ_1	356	372
τ_2	653	681
τ_3	712	742
τ_4	97994	102006

Tabla 4: Comparación de propuestas

	Periodos ajustados	Hiperperiodo	U	Desviación media
Conjunto tareas inicial	T_1=364 T_2=667 T_3=727 T_4=100000	$4,413 * 10^9$	0,98	-
Xu	T_1=360 T_2=660 T_3=720 T_4=92400	277200	0,99	2,68 %
FHS 1	T_1=363 T_2=660 T_3=726 T_4=98010	196020	0,98	0,87 %
FHS 2	T_1=363 T_2=665 T_3=740 T_4=98420	98420	0,98	0,98 %

6. Simulaciones

En esta sección presentamos una serie de simulaciones que nos han permitido evaluar el rendimiento del algoritmo FHS. Los resultados obtenidos se compararán con la exploración exhaustiva propuesta en el Código 1 en términos de tiempo de ejecución. También se comparará la capacidad del algoritmo FHS para minimizar el hiperperiodo frente a la selección aleatoria de los periodos. Por último, un último conjunto de simulaciones comparará el algoritmo propuesto con el presentado por Xu (2010).

Se ha generado una carga aleatoria a partir de los siguientes parámetros:

- Parámetros de entrada:

  n: Numero de tareas en el conjunto

  maxperiod: Máximo periodo permitido

  minsize: Minima diferencia entre l_i y u_i.

  maxsize: Máxima diferencia entre l_i y u_i.

  arraysize: cardinalidad de $\mathbb{G}$.

- Los parámetros del conjunto de tareas se calculan a partir de los parámetros anteriores mediante las siguientes reglas:

  - El tamaño del rango de periodos para cada tarea (v_i) se selecciona aleatoriamente dentro del intervalo $[minsize, maxsize]$.
  - u_i se ha seleccionado aleatoriamente dentro del intervalo $[maxperiod - v_i, maxperiod]$.
  - El extremo inferior para el rango de periodos de cada tarea (l_i) se determina mediante la expresión: $l_i = u_i - v_i$

- Por razones prácticas, el valor de m se obtiene a partir de la máxima longitud (*arraysize*) del vector que guarda el conjunto $\mathbb{G}$. m se ha calculado como el máximo valor que verifica la siguiente restricción: $arraysize \geq \prod_{i=0}^{m}(u_i - l_i + 1)$

- Finalmente, los parámetros de entrada se han fijado con los siguientes valores:
  - n de 5 a 50.
  - $maxperiod = 100$.
  - *arraysize* tomando los valores 1000, 1500, 2000, 3000, 5000, 10000, 30000, 50000, 70000, 90000.
  - *minsize* de 5 a 30 en incrementos de 5 unidades.
  - *maxsize* de 1 a 11 en incrementos de 2 unidades.
  - Para cada valor de n se han generado 10000 conjuntos de tareas, resultando un total de $2,25 * 10^7$ simulaciones.

La figura 2 muestra la comparación de tiempos de ejecución para el algoritmo propuesto frente a los obtenidos para el algoritmo de búsqueda exhaustiva. Como se puede observar, la exploración exhaustiva presenta un comportamiento aceptable cuando el conjunto de tareas (n) es pequeño. Según n aumenta el coste temporal crece exponencialmente, y pronto el algoritmo requiere tanto tiempo de cálculo que no es posible obtener ningún resultado válido. En cambio, el algoritmo `FHS` es mucho menos sensible a este parámetro y sigue siendo capaz de encontrar un resultado rápidamente incluso para conjuntos con $n = 50$.

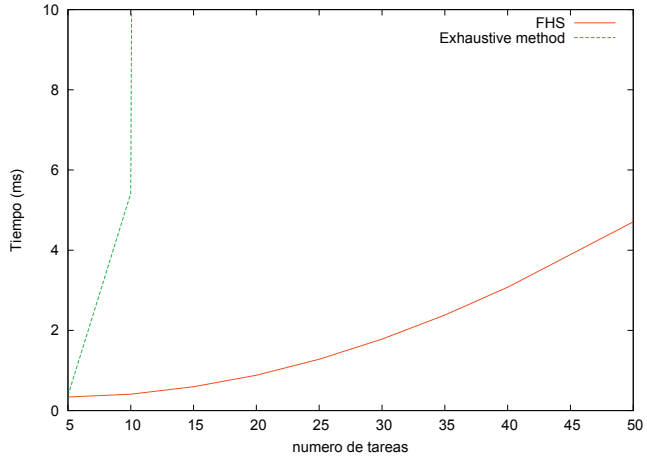

Figura 2: Comparación de los tiempo de ejecución.

En cuanto a la capacidad para reducir el hiperperiodo, la figura 3 muestra los valores de hiperperiodo obtenidos aplicando el algoritmo `FHS` frente a los resultantes de generar aleatoriamente los periodos para un conjunto de n tareas y posteriormente calcular su mínimo común múltiplo. Se ha empleado una escala logarítmica para la representación del tamaño del hiperperiodo debido a las grandes diferencias obtenidas entre los dos métodos. En cualquier caso, el tamaño del hiperperiodo alcanza rápidamente valores intratables computacionalmente hablando (para $n = 15$, $H \sim 10^{16}$).

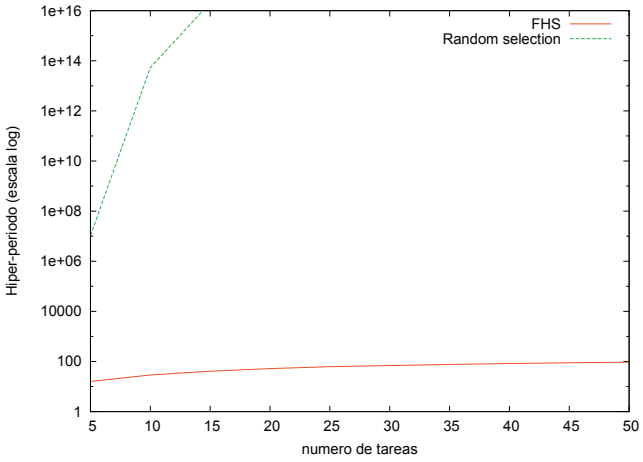

Figura 3: Reducción del hiperperiodo conseguida mediante FHS

Con la intención de extender los anteriores resultados, se ha aplicado un análisis similar con la propuesta de Xu. Para conjuntos de tareas de distinto tamaño, el periodo para cada una de ellas se ha seleccionado aleatoriamente entre dos valores $t_i^o \in [t_i^{min}, t_i^{max}]$.

Se han empleado los siguientes valores:

n Numero de tareas. Se han usado los siguientes valores: 5, 10, 15, 20, 40 y 80.

t_i^{max} Periodo máximo. Se han elegido los siguiente valores: 100, 500, 1000, 5000, 10000, 50000, 90000

δ El periodo mínimo se ha calculado como una fracción del periodo máximo ($t_i^{min} = \delta t_i^{max}$). Se han seleccionado los valores 0.1, 0.2, 0.4, 0.6 y 0.8 como valores de δ.

Para cada configuración derivada de la combinación de los anteriores valores, se han generado 10 conjuntos distintos que han servido para alimentar tanto al algoritmo de Xu como al FHS; más adelante se detallan la premisas y el procedimiento para la conversión de los periodos a los rangos esperados por el FHS.

El valor máximo de los periodos máximos elegidos viene dado por las limitaciones impuestas por el algoritmo de Xu: para limitar el hiperperiodo es necesario limitar las potencias de números primos que forman el hiperperiodo máximo admitido, y por tanto si se quiere acotar la diferencia entre periodo original y periodo ajustado es necesario limitar también los periodos de la tareas.

El trabajo de Xu asevera que para un valor de hiperperiodo máximo acotado por $H_{max} = 2^5 * 3^3 * 5^2 * 7^1 * 11^1$ se puede garantizar que la fracción entre periodo ajustado t_i' y periodo original t_i^o no supera el 90 %, es decir $0,9 \leq t_i'/t_i^o \leq 1$, cuando $t_i^o \leq 2^4 * 3^1 * 5^2 * 7^1 * 11^1$

Dado que este caso no disponemos de tiempo de cómputo de las tareas —es deseable mantener la simplicidad del análisis cuando no se tiene la certeza de que nuevos parámetros aporten datos significativos—, el cálculo de los rangos de entrada para la aproximaciones FHS se basa en la diferencia máxima entre periodos original y ajustados que el algoritmo de Xu puede garantizar (90 %):

$$l_i = t_i^o * 0{,}9$$
$$u_i = t_i^o$$

Finalmente, el valor para el parámetro m de FHS se ha calculado automáticamente para limitar el consumo de memoria del algoritmo a un valor concreto (en este caso 1MB). A continuación de la generación de los rangos, y antes de lanzar el algoritmo FHS, los rangos se ordenan por tamaño creciente y se calcula el numero de posible combinaciones de $t_i \in [l_i, u_i]$ donde $i \leq m$ para valores crecientes de m, suponiendo que eso dará lugar a diferentes valores del MCM , hasta que se alcanza el límite de memoria prefijado.

En la gráficas presentadas a continuación, la propuesta de Xu se ha rotulado como LRRP. La evaluación de las dos propuestas se ha realizado desde dos vertientes: hiperperiodo calculado y tiempo de cómputo. En todos los casos los análisis siempre se establecen frente al numero de tareas.

En el primer caso, en la figura 4 se puede ver cómo el algoritmo FHS es capaz de obtener, en término medio, hiperperiodos significativamente menores. Se presentan tanto los resultados para cada una de las pruebas realizadas, como la media de los hiperperiodos obtenidos.

Se puede ver que incluso con pocas tareas en algunos casos el LRRP es incapaz de ajustar hiperperiodos pequeños, mientras que para FHS el hiperperiodo calculado crece con el numero de tareas. Según este valor, se puede ver que en algunos casos el valor calculado por el FHS se aproxima a los máximos proporcionados por LRRP. A pesar de esto, el algoritmo FHS proporcionará valores para el hiperperiodo significativamente menores, en término medio. En todo caso, FHS es capaz de encontrar el hiperperiodo mínimo y por tanto para cada conjunto de tareas siempre obtiene un hiperperiodo igual o menor que el calculado por LRRP.

Finalmente, comentar que el hecho de que en la gráfica se presenten tan pocos puntos para LRRP se debe a la manera en que trabaja el algoritmo: como hiperperiodos solo son seleccionables el subconjunto formado por todas las posibles combinaciones de números primos hasta 11 con un exponente hasta 5, 3, 2, 1 y 1, respectivamente (2^5, 3^3, 5^2, 7^1 y 11^1).

En lo que respecta al tiempo de cómputo, figura 5, estos se presentan en escala logarítmica. Se puede observar que el algoritmo FHS requiere tiempos de cómputo notablemente superiores. Sin embargo, dos circunstancias hacen que nuestro análisis de estos datos sea positivo: por una parte la evolución de los tiempos de cómputo medios son lineales, con lo que se puede concluir que en gran medida el tiempo que requerido por el algoritmo depende del número de tareas; por otro lado, incluso con valores máximos para el número de tareas y tamaño de los periodos los tiempos obtenidos están en el rango de los minutos.

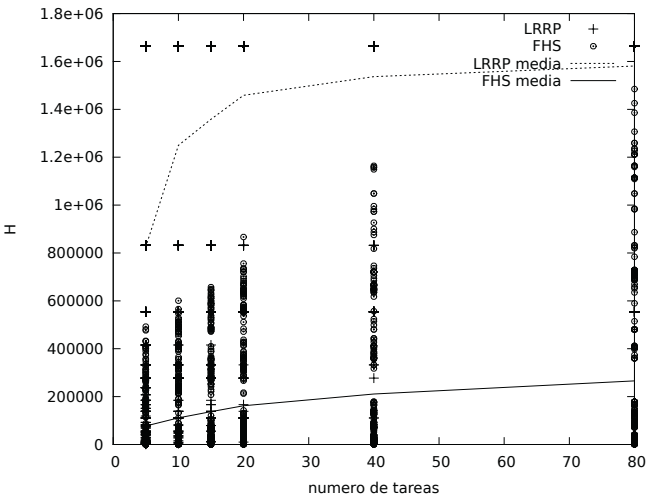

Figura 4: Ajuste del hiperperiodo para FHS y LRRP

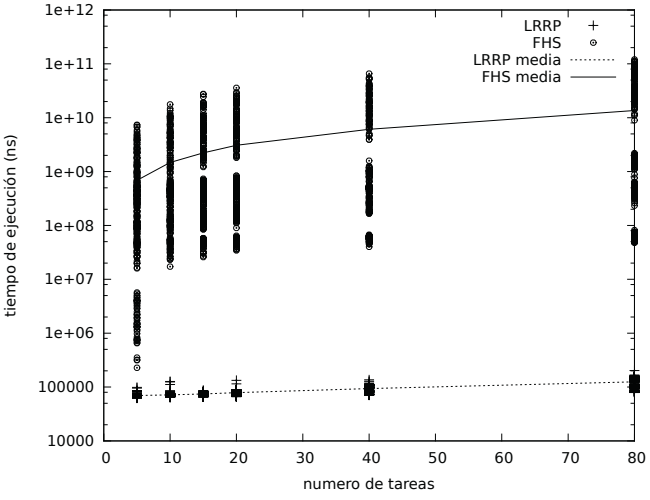

Figura 5: Ajuste del hiperperiodo para FHS y LRRP

En este sentido, para el caso medio, los tiempo de respuesta del algoritmo es del orden de segundos para conjuntos 80 tareas.

7. Aplicación de los resultados

El algoritmo descrito en este trabajo ha sido implementado en la herramienta de panificación Xoncrete (Brocal et al. (2010)). Xoncrete es una herramienta que realiza análisis de planificabilidad para sistema particionados. Lejos de ser una herramienta de propósito general, Xoncrete ha sido diseñada para dar soporte al modelo de sistema descrito por ARINC 653. Además permite analizar el impacto de otras políticas de planificación como EDF para los sistema modelados. Aparte de las capacidades de análisis de planificabilidad, Xoncrete proporciona una interfaz gráfica que le permite al usuario editar de una manera amigable el modelo sobre el que se realizará este análisis. Xoncrete también es capaz de capturar los datos necesarios para generar los ficheros de configuración de XtratuM.

XtratuM es un hipervisor de tipo 1 para sistemas críticos de tiempo real. Este hipervisor, de código abierto, está disponible para procesadores LEON2 y está siendo adaptado a procesadores LEON3, tanto con soporte para MMU como sin este soporte, en el marco del proyecto de la ESA AO5829 "Securely Partitioning Spacecraft Computing Resources".

8. Conclusiones

Hemos presentado un algoritmo que permite obtener el mínimo hiperperiodo para un conjunto de tareas, donde para cada tarea es posible definir un intervalo de periodos válidos.

Se ha mostrado que el algoritmo emplea una búsqueda heurística que permite obtener obtener dicho hiperperiodo. Del mismo modo, se han proporcionado resultados que respaldan que la propuesta tiene ventajas significativas frente a una búsqueda exhaustiva, así como frente al cálculo del hiperperiodo como el *MCM* de periodos fijos para un conjunto de tareas.

En nuestra opinión, la reducción del hiperperiodo tiene beneficios significativos en numerosas aplicaciones. Nuestra intención es continuar explorando la interpretación de periodo propuesta en este trabajo, así como otras nuevas, además de intentar proponer algoritmos alternativos que mejoren los resultados aquí consignados.

English Summary

Abstract

In this paper a new task model with periods defined as ranges is proposed with the main goal of drastically reducing the hyperperiod of the task set. The model is focused to be applied in cyclic scheduling, where the length of the major cycle of the plan is determined by the hyperperiod. But it also can be applied in synthetic task sets generation, where having a small hyperperiod reduces complexity and simulation time. As the hyperperiod grows exponentially with the number of tasks and their periods, system analysis may become unaffordable if the hyperperiod exceeds reasonable bounds.

A new algorithm, which allow us to calculate the minimum hyperperiod of such a set of tasks, is presented. This algorithm calculates the minimum value even with a large number of tasks, where exhaustive search becomes intractable.
Keywords: Real-time, Model, Scheduling algorithms

Referencias

Baruah, S., Mok, A., Rosier, L., 1990a. Preemptively scheduling hard real-time sporadic tasks on one processor. In: IEEE Real-Time Systems Symposium. pp. 182–190.

Baruah, S., Rosier, L., Howell, R., 1990b. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. Journal of Real-Time Systems 2.

Brocal, V., Masmano, M., Ripoll, I., Crespo, A., Balbastre, P., 2010. Xoncrete: a scheduling tool for partitioned real-time systems. In: Embedded Real-Time Software and Systems.

Buttazzo, G., Lipari, G., Abeni, L., December 1998. Elastic task model for adaptive rate control. In: IEEE Real-Time Systems Symposium. pp. 286–295.

Cervin, A., Eker, J., 2000. Feedback scheduling of control tasks. In: Proceedings of the 39th IEEE Conference on Decision and Control.

Crandall, R., Pomerance, C. B., 2005. Prime numbers: a computational perspective. Springer.

Kermia, O., Cucu, L., Sorel, Y., 2006. Non-preemptive multiprocessor static scheduling for systems with precedence and strict periodicity constraints. In: Proceedings of the 10th International Workshop On Project Management and Scheduling.

Leung, J., Merrill, R., 1980. A note on the preemptive scheduling of periodic, real-time tasks. Information Processing Letters 18, 115–118.

Liu, C., J.W.Layland., 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. JACM 23, 46–68.

Macq, C., Goossens, J., 2001. Limitation of the hyper-period in real-time periodic task set generation. In: Proceedings of the 9th international conference on real-time systems. pp. 133–148, iSBN 2-87717-078-0.

Martí, P., Fuertes, J. M., Fohler, G., 2001. Jitter compensation for real-time control systems. In: IEEE Real-Time Systems Symposium.

Ripoll, I., Crespo, A., Mok, A., 1996. Improvement in feasibility testing for real-time tasks. Journal of Real-Time Systems 11, 19–40.

Shih, C.-S., Gopalakrishnan, S., Ganti, P., Caccamo, M., Sha, L., dec. 2003. Scheduling real-time dwells using tasks with synthetic periods. pp. 210 – 219.

Xu, J., july 2010. A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems. In: Mechatronics and Embedded Systems and Applications (MESA), 2010 IEEE/ASME International Conference on. pp. 288 –294.