PLANIFICACIÓN GLOBAL EN SISTEMAS MULTIPROCESADOR DE TIEMPO REAL

Tesis propuesta para el doctorado en informática

PROGRAMA D'ARQUITECTURA I TECNOLOGIA DE COMPUTADORS

UNIVERSITAT POLITÈCNICA DE CATALUNYA

DOCTORANDO: Josep Maria Banús i Alsina

CO-DIRECTORES: Dr. Jesús José Labarta Mancho

Dr. Alejandro Arenas Moreno

Diciembre de 2007

AGRADECIMIENTOS

En primer lugar, quisiera agradecer a mis padres el haberme dado la oportunidad de estudiar y a mi querida esposa, Carme, la posibilidad de continuar haciéndolo.

También quisiera agradecer a mis tutores, Jesús Labarta y Alex Arenas, su paciencia, sabio consejo y las continuas revisiones de los artículos y de la tesis. A mis compañeros y compañeras de departamento agradecer sus continuados y tan necesarios ánimos, especialmente a Maria dels Àngels Moncusí.

Finalmente quisiera dedicar esta tesis a mis hijos, Eduard y Estel, porqué el tiempo pasa, porqué a su edad se debe aprender a perseverar y a confiar en el futuro, y para que valoren más el conocimiento y menos las apariencias y lo banal.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1

INTRODUCCIÓN

En este capítulo se proporciona una visión general de los sistemas de tiempo real, el modelado del sistema en tareas y las características generales de la planificación de las tareas en tiempo real. Una vez establecidas las bases, al final del capítulo se detalla la motivación de la tesis y los principales objetivos.

1.1 Sistemas de tiempo real

El concepto de tiempo real tiene diversas aceptaciones y es realmente difícil de resumir en una definición corta. Sus orígenes están en los dispositivos de procesamiento digital de señales y en los sistemas electrónicos de control. En estos ámbitos, un sistema de tiempo real es aquel capaz de procesar una muestra de señal antes de que se deba tomar la siguiente muestra. Es decir, existe un límite en el tiempo del orden de milisegundos para realizar una determinada tarea.

Hoy en día los computadores se encuentran instalados en cualquier tipo de dispositivo. En particular, se ha generalizado el uso de computadores en el procesado de señales y en los sistemas de control. Esto obligó a la comunidad científica a revisar los objetivos y los criterios de rendimiento de los computadores, puesto que la limitación de los tiempos de respuesta a milisegundos no se aplicaba a la informática de propósito general.

En un sistema de tiempo real con computador, la aplicación informática debe reaccionar en un tiempo preciso a eventos procedentes del entorno. En la computación de tiempo real, la corrección del sistema no solo depende del resultado lógico sino que también depende del tiempo en que este resultado es producido [Sta88]. En comparación con las aplicaciones de propósito general, en las cuales se busca un resultado, cueste el tiempo que cueste, en las de tiempo real se requiere hallar el resultado en un tiempo fijado. Si no se obtuviese el resultado a tiempo podría dar lugar a situaciones desastrosas o, en el mejor de los casos, este resultado

ya no sería relevante. En definitiva, en este tipo de sistemas van a ser determinantes las *restricciones temporales* impuestas por el entorno.

Las aplicaciones de los sistemas de tiempo real son innumerables y de muy diversa índole. Podemos tener desde el control de sistemas muy críticos, como plantas químicas y nucleares, hasta sistemas de entretenimiento, como los sistemas multimedia o los juegos por computador. En la Tabla 1 se enumeran algunas de estas aplicaciones y se agrupan según el tipo de sistema.

Tipo de Sistema	**Ejemplos**
Sistemas de Control	Sistemas de manufactura Procesos de producción Sistemas de mando y control Medios de transporte Aeronáutica
Sistemas de Transacciones	Comerció electrónico Bolsa Telefonía inalámbrica
Sistemas Multimedia	Video bajo demanda Realidad virtual Juegos

Tabla 1: Algunos ejemplos de aplicaciones de sistemas de tiempo real

Otra clasificación importante de los sistemas de tiempo real, alternativa a la utilizada en la Tabla 1, es la realizada según lo crítico que sea el sistema controlado, es decir, según si las consecuencias de un malfuncionamiento son críticas o no. En algunas aplicaciones críticas, en el hipotético caso de no cumplirse los requerimientos temporales, podría dar lugar a desastres ecológicos o hasta a la pérdida de vidas humanas. En otras aplicaciones, las pérdidas económicas derivadas del fallo de algunas transacciones podrían ser menos graves aunque no por ello dejarían de ser importantes. Finalmente, no cumplir las restricciones temporales en los sistemas multimedia podría suponer una calidad de servicio inferior que, si bien no causaría daños irreversibles, si podrían suponer perjuicios económicos para el proveedor o de satisfacción del usuario. En resumen, algunas aplicaciones en tiempo real van a ser

muy estrictas en el cumplimiento de los requerimientos temporales, mientras que otras van a presentar cierta tolerancia.

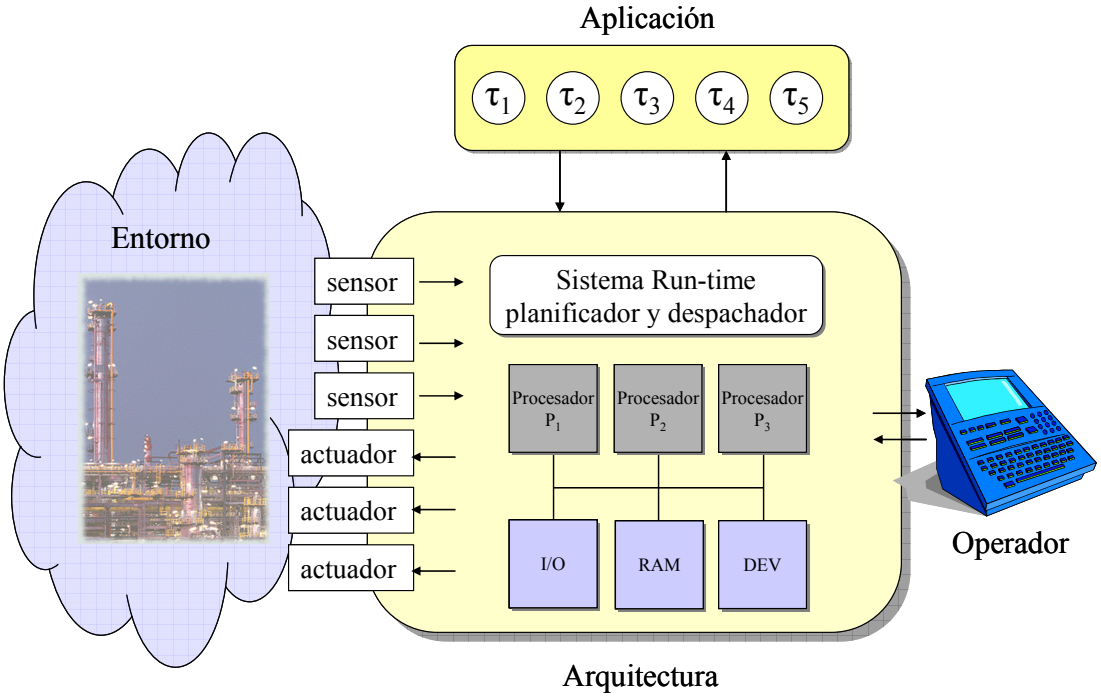


Figura 1.1: Visión simplificada de los componentes de un sistema de tiempo real

La definición de sistema de tiempo real utilizada va a repercutir en todos los aspectos del diseño de la aplicación informática, desde la metodología de la programación y su validación hasta el sistema operativo y la arquitectura del computador (véase Figura 1.1). El diseño de la aplicación, su programación y validación caen fuera del ámbito estricto de esta tesis. Sin embargo es destacable que, dada la gran complejidad de estas aplicaciones y la naturaleza del entorno controlado o modelado, estas aplicaciones se suelen descomponer en multitud de tareas, las cuáles se deben ejecutar concurrentemente, y se especifican las restricciones temporales para cada una de ellas. Los sistemas operativos convencionales no son válidos para estos entornos, entre otros motivos, porque no tienen en cuenta las restricciones temporales de las tareas y frecuentemente no tienen acotado el tiempo que su propio código consume. Así, el soporte a la ejecución concurrente de tareas con restricciones temporales y el determinismo en el propio sistema operativo van a ser factores decisivos. En lo referente a la arquitectura del computador, hay que revisar sus objetivos y los parámetros que van a medir su rendimiento, puesto que el criterio

general de obtener cada vez un mayor número de instrucciones por segundo en los sistemas de tiempo real no es el criterio primordial. Las técnicas de alto rendimiento en la computación mejoran el rendimiento promedio, pero esto no tiene necesariamente que ver con la producción de resultados correctos a tiempo. En los sistemas de tiempo real los criterios son otros, como la robusteza física en los sistemas de control en entornos agresivos o el confort personal y la calidad de servicio en los sistemas multimedia.

1.2 Modelado del sistema y definiciones

Para modelar la realidad a controlar, las aplicaciones informáticas se componen de tareas y acciones. A su vez, se especifican las restricciones temporales de estas tareas y las relaciones entre ellas.

Una *acción*, (J_i, del inglés *job*), es un conjunto de instrucciones a ejecutar con unas restricciones temporales comunes: es la mínima unidad de cómputo de un sistema de tiempo real. Una *tarea* (τ_i, *task*) es un conjunto de acciones similares que se repiten a lo largo del tiempo. La primera restricción temporal es el *tiempo de activación* (o de llegada desde el punto de vista del planificador), es el instante en que las instrucciones están disponibles para ser ejecutadas por el procesador, (r_i del inglés *release time*). La segunda restricción temporal consiste en un *plazo* de finalización (o límite de tiempo) en que las instrucciones deben haber terminado (*deadline*). Este plazo se puede especificar relativo al tiempo de activación, plazo relativo (D_i, *relative deadline*), o también se puede especificar como tiempo absoluto, plazo absoluto (*absolute deadline*, $d_i = r_i + D_i$). Además, frecuentemente es muy trascendental conocer el *peor tiempo de ejecución* de una acción en una arquitectura concreta, (C_i, *Worst-Case Execution Time*, o WCET). En un sistema de tiempo real es muy importante conocer este dato puesto que es necesario para la verificación del sistema en el peor caso y, por extensión, en todos los casos.

Según la planificación que se haga del conjunto de tareas, una tarea va a tener un *tiempo de inicio* de ejecución o de arranque (s_i, *start time*) igual o posterior al tiempo de activación ($r_i \leq s_i$). El *tiempo de ejecución real* de una tarea (E_i) va a ser menor o

igual al peor caso ($E_i \leq C_i$). De estos dos últimos tiempos depende el *tiempo de finalización* o de terminación de la ejecución de una tarea (f_i, *completion time*), pero éste también se podrá ver afectado por la ejecución de otras tareas ($s_i + E_i \leq f_i$). El *tiempo de respuesta* de una tarea (R_i, *response time*) se puede calcular de la siguiente forma: $R_i = f_i - r_i$. Finalmente, es deseable que una tarea con plazo termine antes de su limite, es decir, que cumpla su plazo ($f_i \leq d_i$ en tiempos absolutos o bien $R_i \leq D_i$ en tiempos relativos).

Un *plazo estricto* (*hard deadline*) es aquel que hay que cumplir escrupulosamente, o, en caso de no cumplirlo, el resultado puede ser un fallo fatal o con consecuencias desastrosas. Alternativamente, un *plazo laxo* o flexible (*soft deadline*) es aquel que es deseable cumplir. En este caso, las consecuencias derivadas del incumplimiento serán poco importantes e incluso el resultado podría ser útil en cierta medida. Normalmente el valor del resultado decrece conforme pasa el tiempo.

En una aplicación para controlar un sistema crítico los plazos son estrictos. Son los llamados *sistemas de tiempo real estrictos* (*hard real-time systems*). Un buen ejemplo son los sistemas de control de vuelo de los aviones. Por otro lado, los sistemas con tareas con plazos flexibles se llaman *sistemas de tiempo real laxo* o flexible (*soft real-time systems*). La flexibilidad en el incumplimiento de los plazos puede variar mucho. Así, los sistemas en los que se puede incumplir ocasionalmente alguna restricción y todavía considerar el funcionamiento como correcto son los *sistemas de tiempo real firme* (*firm real-time systems*). En estos sistemas, un resultado tardío no tiene utilidad. Por ejemplo en un sistema multimedia de video, una imagen tardía ya no tiene sentido seguir procesándola o trasmitiéndola puesto que ya debería haberse visualizado.

La mayoría de las aplicaciones de control o de tiempo real dependen de una realidad que requiere un muestreo periódico y, como consecuencia, una ejecución de tareas de forma periódica. El periodo de ejecución va a depender de la aplicación. Por ejemplo, una señal se debe muestrear al doble de su frecuencia máxima, o un sistema de video debe tratar y mostrar una imagen o *frame* cada cierto tiempo. Esta característica frecuencial nos va a propiciar una distinción entre tareas periódicas y

tareas aperiódicas. El modelado de *tareas periódicas* es de suma importancia en los sistemas de tiempo real, puesto que su caracterización es determinista y esto facilita a validación y la verificación del modelo. Al mismo tiempo, pueden suceder eventos inesperados que van a ser tratados por tareas aperiódicas.

Tareas Periódicas

Una tarea periódica es un conjunto de instrucciones cuya ejecución se va a invocar de forma periódica. Así pues, podemos decir que una tarea periódica consiste de una secuencia periódica de activaciones de acciones idénticas. La principal característica nueva de estas acciones es su *periodo* de activación (T_i). En la Figura 1.2 podemos observar tres activaciones o acciones de una misma tarea. Opcionalmente, en los sistemas en que no todas las tareas periódicas se activan simultáneamente o sistemas asíncronos, se pueden modelar estas tareas con un *desplazamiento inicial* para la primera activación (O_i, *offset*). Así pues, el modelado abstracto de una tarea periódica τ_i se especifica dando valor al conjunto de parámetros estáticos $\tau_i = \{C_i, T_i, D_i, O_i\}$ (véase Figura 1.2). De éstos se pueden derivar otras características, como por ejemplo la *utilización* (U_i) del procesador que va ha hacer la tarea (o también, *factor de utilización*). Ésta se puede calcular de la siguiente forma:

$$U_i = \frac{C_i}{T_i} \qquad \text{(ecuación 1.1)}$$

Otra característica que se puede derivar de los parámetros estáticos es la *laxitud* (L_i, *laxity*) y se define según la siguiente ecuación:

$$L_i = D_i - C_i \qquad \text{(ecuación 1.2)}$$

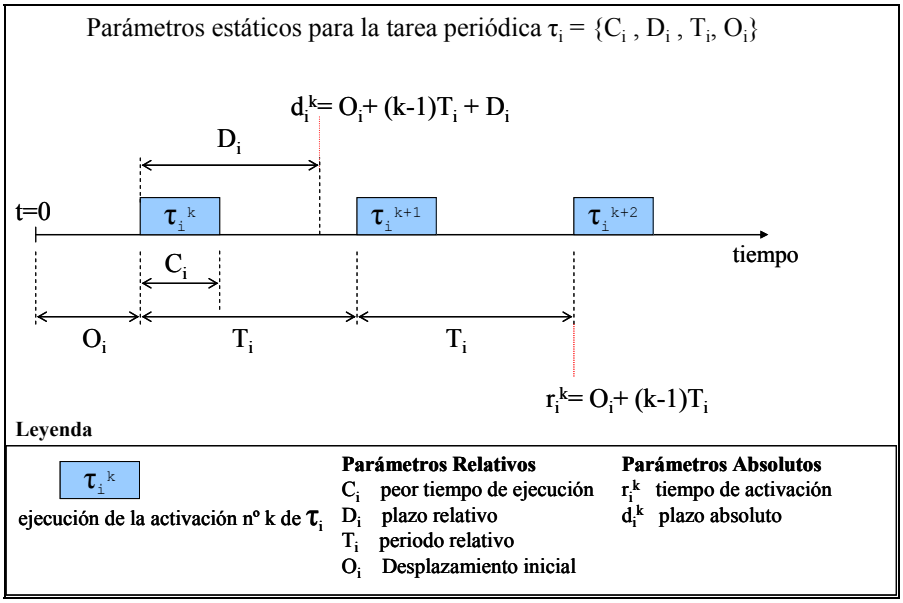


Figura 1.2: Parámetros estáticos para la tarea periódica $\tau_i = \{C_i, T_i, D_i, O_i\}$

Algunos de los parámetros dinámicos se pueden calcular a partir de los estáticos mientras que otros van a depender de la ejecución de otras tareas (véase un resumen de los parámetros dinámicos en la Figura 1.3). El tiempo de activación de la k-ésima acción (*release time*, r_i^k) de una tarea periódica se puede calcular de la siguiente forma:

$$r_i^k = O_i + (k-1)T_i \qquad \text{(ecuación 1.3)}$$

De la misma forma, el plazo absoluto para esta k-ésima acción va a ser:

$$d_i^k = Oi + (k-1)T_i + D_i = r_i^k + D_i \qquad \text{(ecuación 1.4)}$$

Los parámetros que dependen de la interferencia producida por la ejecución de otras tareas son el tiempo de inicio de ejecución (s_i^k, *start time*), el tiempo de finalización (f_i^k, *completion time*) y el tiempo de respuesta (R_i^k, *response time*) el cual se puede calcular de la siguiente forma:

$$R_i^k = f_i^k - r_i^k \qquad \text{(ecuación 1.5)}$$

Para cumplir los plazos se debe cumplir $f_i^k \leq d_i^k \; \forall \; k$ (o también $R_i^k \leq D_i \; \forall \; k$).

El *peor tiempo de respuesta de una tarea periódica* (R_i, W*orst-Case Response Time*, o WCRT) es el máximo entre los tiempos de respuesta de todas sus activaciones, es decir:

$$R_i = \max \{R_i^k\} \qquad \text{(ecuación 1.6)}$$

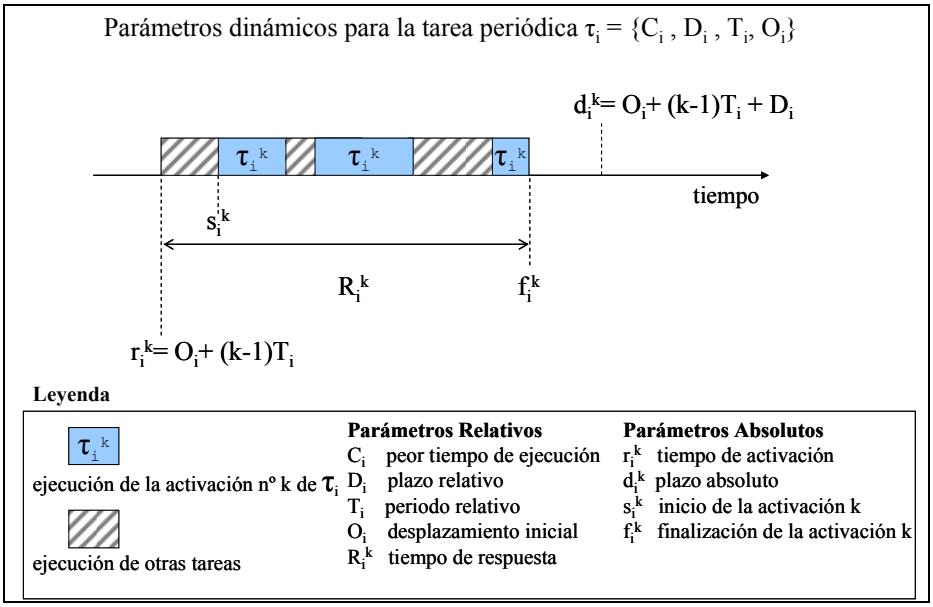


Figura 1.3: Parámetros dinámicos para la tarea periódica $\tau_i = \{C_i, T_i, D_i, O_i\}$

Un sistema con un conjunto de tareas periódicas τ tiene además algunos parámetros derivados de la caracterización de sus tareas. Los principales son el hiperperiodo, la utilización máxima y el factor de utilización máximo. Debemos hacer notar que, en un modelo estático el número de tareas (*n*) es fijo.

Un *hiperperiodo* es un periodo de tiempo que contiene todas las acciones de todas las tareas periódicas siguiendo un patrón de activaciones que se va a repetir a cada nuevo hiperperiodo. Este parámetro se calcula fácilmente hallando el mínimo común múltiplo de todos los periodos, es decir, $H(\tau) = \text{mcm}\{T_i\}$. Dada la naturaleza repetitiva de las activaciones de hiperperiodo en hiperperiodo, este parámetro va a facilitar la verificación de los sistemas cuando haya que recurrir a la simulación. Si los periodos son primos entre si el hiperperiodo puede llegar a ser extremadamente largo. Si los periodos son *armónicos*, es decir son múltiplos, el hiperperiodo coincide con el periodo mayor.

La utilización máxima del sistema (U_{per}) por parte de las tareas periódicas es

$$U_{per} = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \quad \forall \tau_i \in \tau \qquad \text{(ecuación 1.7)}$$

Para que estas tareas sean planificables como mínimo se debe cumplir que $U_{per} \leq 1$ en un monoprocesador o $U_{per} \leq m$ en un multiprocesador con m procesadores idénticos.

El factor de utilización máximo de un conjunto de tareas periódicas es

$$U_{\max} = \max_{\tau_i \in \tau} U_i = \max_{\tau_i \in \tau} \frac{C_i}{T_i} \qquad \text{(ecuación 1.8)}$$

Tareas Aperiódicas

Las *tareas aperiódicas* requieren ejecutarse en instantes de tiempo aleatorio. Sus parámetros estáticos pueden ser $H_i = \{r_i, C_i, D_i\}$ para una tarea aperiódica con plazo de finalización (*hard-aperiodic task*) o bien $S_i = \{r_i, C_i\}$ para una tarea aperiódica sin plazo de finalización (*soft-aperiodic task*). En el primer caso, es deseable (según la flexibilidad de la tarea) que se cumpla su plazo ($R_i \leq D_i$) mientras que en el segundo caso normalmente se desea que su tiempo de respuesta sea lo más inmediato posible o que en promedio sea lo menor posible. Para la generación sintética aleatoria de tareas aperiódicas se utilizará la parametrización $S_i = \{I_i, C_i\}$ donde I_i es el tiempo entre llegadas medio según una distribución exponencial.

Las *tareas esporádicas* también requieren ejecutarse en instantes de tiempo aleatorio pero, a diferencia de las tareas aperiódicas, existe una limitación nueva: el tiempo mínimo entre llegadas (I_i, *minimal interarrival time*). Sus parámetros estáticos son $Sp_i = \{r_i, C_i, D_i, I_i\}$. Estas tareas siempre cumplen que $r_i^{k+1} \geq r_i^k + I_i$.

J_i	acción (*job*), conjunto de instrucciones a ejecutar
τ_i	tarea (*task*), consta de un conjunto de acciones relacionadas
τ_i^k	k-ésima activación o *job* de la tarea τ_i
τ_i	tarea (*task*), consta de un conjunto de acciones relacionadas
d_i	plazo absoluto (*absolute deadline*) para que el trabajo J_i o de la tarea τ_i
D_i	plazo relativo (*relative deadline*)
C_i	tiempo máximo de ejecución (*Worst-Case Execution Time*, WCET)
L_i	laxitud de una tarea (D_i -C_i)
E_i	tiempo de ejecución real
T_i	periodo de activación de la tarea periódica τ_i
U_i	factor de utilización de una tarea periódica τ_i (o utilización)
s_i	tiempo de inicio de ejecución o arranque (*start time*)
f_i	tiempo de finalización o terminación (*completion time*)
τ	conjunto de tareas periódicas
O_i	desplazamiento inicial (*offset*) para la primera activación de la tarea periódica τ_i
s_i^k	tiempo de inicio de ejecución (*start time*) de la k-ésima activación de la tarea periódica τ_i
f_i^k	tiempo de finalización (*completion time*) de la k-ésima activación de la tarea periódica τ_i
R_i^k	tiempo de respuesta *(response time*) de la k-ésima activación de la tarea periódica τ_i
R_i	peor tiempo de respuesta (W*orst-Case Response Time,* WCRT) de la tarea periódica τ_i o tiempo de respuesta de una acción individual J_i
$H(\tau)$	hiperperiodo del conjunto de tareas periódicas τ
U_{per}	utilización o carga total de un conjunto de tareas periódicas
U_{max}	factor de utilización máximo de cualquier tarea periódica de un conjunto
H_i	tarea aperiódica con plazo de finalización (*hard-aperiodic task*)
S_i	tarea aperiódica sin plazo de finalización (*soft-aperiodic task*)
Sp_i	tarea esporádica
I_i	tiempo mínimo entre llegadas de tareas esporádicas (*interarrival time*)

Tabla 2: Resumen de las definiciones de parámetros de las tareas

1.3 Planificación en los sistemas de tiempo real

La *planificación* de un sistema de tiempo real consiste en realizar una reserva recursos espaciales (procesador, memoria, etc.) y recursos temporales (ranuras de tiempo) para un conjunto de tareas con restricciones temporales. Así pues, un *algoritmo de planificación* genera un *plan de ejecución* para un conjunto de tareas en un determinado sistema de ejecución. Estos algoritmos se implementan en un *planificador* (*scheduler)*, el cual se encarga de decidir en que orden han de ejecutarse las tareas. Un *plan válido* es un plan de ejecución que: i) asigna como máximo una acción a la vez a cada procesador, ii) asigna como máximo un procesador a la vez a cada acción, iii) no ejecuta ninguna acción antes de su tiempo de activación, iv) asigna a cada trabajo un tiempo de procesador igual a su tiempo de ejecución real o máximo, v) satisface todas las relaciones de precedencia y todas las restricciones en el uso de los recursos. El plan realizado por el planificador se dice que es *factible o admisible* (*feasible schedule*) si es un plan válido y asegura que todas las acciones terminan antes de su tiempo límite. Una vez realizada la selección, el despachador (*dispatcher)* se encarga de tomar las medidas necesarias para que la tarea elegida empiece o reanude su ejecución en el procesador escogido.

En la Figura 1.4 se puede observar el esquema general de la planificación de tareas de tiempo real en un único procesador. Hay que tener en cuenta que, según el tipo de planificador usado, las tareas pueden volver del estado de ejecución al estado de preparadas antes de terminar.

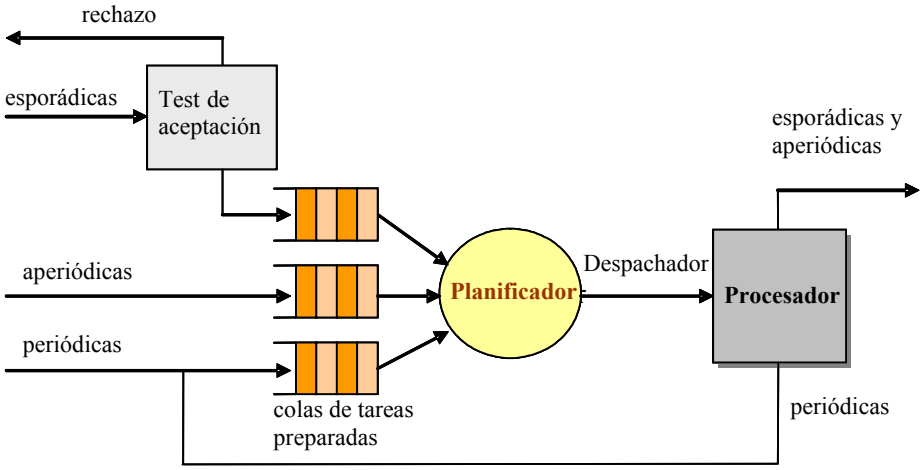


Figura 1.4: Esquema general de la planificación de tareas de tiempo real en un monoprocesador

Los *análisis de la planificabilidad (schedulability analysis)* sirven para determinar si un conjunto concreto de tareas puede ser planificado por un determinado planificador, es decir, si este planificador va a generar un plan admisible. Estos análisis frecuentemente se basan en una determinada *prueba de planificabilidad (feasibility test)*, las cuales pueden ser suficientes, necesarias o exactas. Las pruebas más simples se basan en la utilización del sistema, la cual no debe sobrepasar un determinado límite superior. Cuando el planificador utiliza un algoritmo de prioridades fijas y no existe una prueba de planificabilidad basada en la utilización o bien el límite superior de la utilización es demasiado bajo, se suele recurrir al *análisis de los tiempos de respuesta*. Para ello se calcula el peor tiempo de respuesta de cada tarea y éste tiene que ser menor o igual a su plazo para que el sistema sea planificable. Cuando el algoritmo de planificación utiliza una asignación de prioridades dinámica se suele aplicar una prueba basada en el *análisis de la demanda realizada al procesador*, la cual, para todo intervalo de tiempo, tiene que ser menor que este intervalo de tiempo. En la Tabla 6 del Capítulo 2 se resumen las principales pruebas de planificabilidad.

En general, el análisis de los tiempos de respuesta puede ser muy costoso. En algunos casos se debe recurrir a la simulación de todo un hiperperiodo. Sin embargo, en un monoprocesador el problema se puede resolver en tiempo pseudo-polinomial[1]

[1] *O(p(n,m))* para alguna función polinómica *p*, siendo *n* la longitud de la entrada y *m* su número mayor

[LeW82] si se realiza el análisis en el *instante crítico* [LiL73], momento en el que se activan todas las tareas del sistema.

Un *conjunto de tareas es planificable* si existe al menos un algoritmo de planificación que pueda generar un plan admisible. Se dice que un *sistema de tiempo real es planificable* con un determinado algoritmo de planificación si este algoritmo siempre produce planes admisibles.

Finalmente, se dice que un *algoritmo de planificación es óptimo*, por lo que a la planificación concierne, si siempre puede hallar un plan admisible cuando éste exista. Es decir, siempre que un conjunto de tareas es planificable por cualquier otro algoritmo de planificación éste algoritmo óptimo también puede hallar un plan admisible.

Los algoritmos de planificación pueden clasificarse según diversos criterios. El más común de ellos es el momento en que se generan los planes: se pueden generar de forma estática o de forma dinámica. Con la *planificación estática* el plan se genera fuera de línea (*off-line*), antes de tener las tareas disponibles e incluso antes de arrancar el sistema. Con la *planificación dinámica* el plan se va generando en vivo (*on-line*) simultáneamente a la ejecución y llegada de las tareas al sistema. En la Figura 2.1 del próximo capítulo se puede observar una clasificación de los planificadores clásicos de tiempo real para monoprocesadores.

Otra clasificación importante es según el esquema de planificación. Éste puede ser dirigido por tiempo, por turno circular o por prioridades. Los planificadores dirigidos por tiempo (*time-driven*) se ejecutan cada vez que llega una señal del reloj y siguen unas reglas establecidas fuera de línea. Los planificadores por turno circular (*round-robin*) ejecutan instrucciones de una tarea durante una ranura de tiempo y después vuelven a encolar la tarea en una cola de tareas preparadas. En la planificación con prioridades (*priority-driven*) cada tarea tiene una prioridad y se ejecuta la tarea disponible con mayor prioridad. También se llama planificación dirigida por sucesos (*event-driven*).

Otras clasificaciones de los planificadores pueden realizarse según las restricciones utilizadas. Por ejemplo, el planificador puede ser con o sin desalojo[2] (*preemptive*), con prioridades fijas o dinámicas, avaricioso (*greedy*) o justo (*fair*), con migraciones entre procesadores o sin. Normalmente se asume que un procesador solo puede ejecutar instrucciones de una tarea en un instante de tiempo determinado y que una tarea solo puede ejecutarse en un procesador en un instante de tiempo determinado. A modo orientativo, en la Tabla 3 hay un resumen comparativo de los planificadores según su tipología.

Tipo de Planificador	Ventajas	Inconvenientes
Estático (off-line)	☺ determinista ☺ comunicación efectiva entre tareas	☹ baja flexibilidad: no se adapta a los cambios ☹ inapropiado si el mínimo común múltiplo de los periodos es grande
Dinámico (on-line)	☺ alta flexibilidad: puede adaptarse a los cambios ☺ efectivo para diferentes tipos de tareas	☹ menos determinista que los estáticos ☹ más varianza ☹ comunicación más difícil entre tareas
Dirigido por tiempo	☺ muy utilizado en la planificación cíclica ☺ fácil de validar	☹ el planificador se ejecuta muy frecuentemente ☹ dificultad de mantener
Dirigido por eventos	☺ se adapta a los distintos eventos ☺ el planificador solo se ejecuta cuando hay un evento	☹ la asignación de prioridades es más compleja
Sin desalojo	☺ Exclusión mutua automática ☺ Facilita el análisis WCET	☹ Bloquea el resto de tareas hasta que finalice la actual ☹ Efecto negativo en la planificabilidad
Con desalojo	☺ aumento de la planificabilidad: las decisiones se pueden cambiar tan pronto como el estado del sistema varía	☹ Exclusión mutua con semáforos ☹ dificulta el análisis WCET ☹ costo adicional en cambios de contexto
Avaricioso	☺ facilidad de implementación ☺ soportado por la mayoría de SO	☹ las tareas de baja prioridad pueden experimentar inanición
Justo	☺ la planificabilidad se maximiza cuando los costes de los cambios de contexto son muy bajos	☹ dificultad de implementación ☹ planificabilidad pobre si el coste de los cambios de contexto es significativo
Sin migraciones	☺ no hay costes adicionales por migración	☹ menor flexibilidad: las tareas deben ejecutarse por completo en un único procesador
Con migraciones	☺ mayor flexibilidad: las tareas pueden ejecutarse en diversos proc. ☺ mayor ocupación de los proc.	☹ la planificación ha de tener en cuenta los costes de las migraciones si éstos son significativos

Tabla 3: Resumen de las características principales según los distintos tipos de planificadores

[2] Utilizaremos los términos desalojos, apropiaciones o expulsiones como sinónimos.

1.4 Motivación

Esta tesis trata de la planificación de sistemas de tiempo real utilizando sistemas multiprocesador de memoria compartida. La principal motivación para iniciar la investigación en la planificación de sistemas de tiempo real está fundamentada en el plan estratégico del departamento en el que trabajo, el *Departament d'Enginyeria Informàtica i Matemàtiques* de la *Universitat Rovira i Virgili*. Esta universidad está ubicada en un entorno geográfico donde las instalaciones industriales con entornos de tiempo real críticos abundan: refinerías de petróleo, industrias petroquímicas, diversas centrales nucleares, etc. Así, las líneas del departamento estaban principalmente enfocadas al "tiempo-real", tanto en la parte docente como en la parte investigadora.

En lo referente a la planificación de sistemas de tiempo real, la teoría para los sistemas monoprocesador ya estaba muy consolidada. Para los sistemas distribuidos había abundante bibliografía pero para los sistemas multiprocesador con memoria compartida al principio apenas se encontraban referencias. Esto era debido a los enormes problemas que conlleva y a una praxis que utilizaba el multiprocesador como si fuesen monoprocesadores independientes o como si fuese un sistema distribuido.

Recientemente las arquitecturas multiprocesador se han ido popularizando. Su uso va desde los ordenadores personales hasta grandes servidores y computadores, pasando por los sistemas empotrados. Con los límites actuales del hardware, es más fácil y con un menor coste incrementar la capacidad de cálculo con un multiprocesador que realizarlo con sistemas de un procesador de equivalente capacidad. Así, han ido apareciendo diversos chips *Multi-Core* o *Chip-level Multiprocessors* (CMP). Los fabricantes de procesadores con mayor volumen, como son Intel y AMD, primero lanzaron al mercado chips con dos procesadores (por ejemplo *AMD Athlon X2* e *Intel Core Duo*) y en la actualidad ya son frecuentes con cuatro procesadores (*Quad-Core AMD Opteron* y *Quad-Core Intel Xeon*). En la Tabla 4 se resumen los procesadores multicore actuales, donde se aprecia la gran evolución en los últimos años.

Procesador	Año	Cores	Cache Compartida	Tipo
Sun Niagara	2005	8x UltraSPARC T1	3MB L2	RISC
Sun Niagara2	2007	8x UltraSPARC T1	4MB L2	
IBM Power5	2005	dual-core	2MB L2	
Intel Pentium EE 840, 955/965	2005	dual-core (2 P4 Prescott)		
Intel Core 2 Duo	2006	dual-core	2MB L2	
Intel Xeon 5300	2007	quad-core	4MB L2 cada 2 cores	CISC
AMD Barcelona	2007	quad-core (4x NG-Opteron)	2MB L3	
Sun MAJC 5200	2000	quad-core		VLIW
Intel Montecito	2006	quad-core (2*Itanium2)		

Tabla 4: Resumen de los procesadores multicore comerciales. La mayoría puede utilizarse en sistemas multiprocesador.

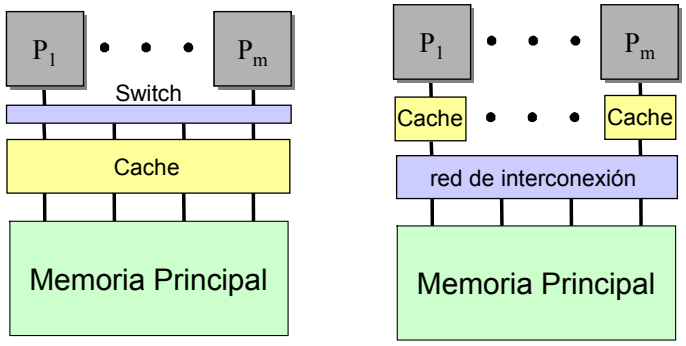
Las soluciones tradicionales de planificación de sistemas de tiempo real en multiprocesadores infrautilizaban los recursos de cálculo, pero esto nunca fue un grave problema, puesto que las aplicaciones de tiempo real tradicionales no son muy exigentes en potencia de cálculo: para ellas lo importante es obtener los resultados a tiempo y no lo es tanto el obtener en promedio un alto rendimiento. Sin embargo, el ámbito de las aplicaciones de tiempo real se está extendiendo a aplicaciones más generalistas, donde se pueden mezclar procesos de tiempo real estricto con procesos de tiempo real laxos pero con una mayor demanda de potencia de cálculo, como pueden ser aplicaciones multimedia. Así, se hacía necesario encontrar metodologías de planificación que aumentasen la utilización de la gran capacidad de cálculo que ofrecen los modernos sistemas multiprocesador.

1.5 Marco y Metodología

Para los experimentos de esta tesis se han utilizado simuladores desarrollados en C++ por el autor. Para contrastar el correcto funcionamiento de la planificación realizada, cuando fue posible se utilizó el simulador RTSIM[3] de la *Scuola Superiore Sant'Anna* [PLA01]. Aunque no se ha simulado ningún hardware específico, se

[3] http://rtsim.sssup.it/

asume que este tendrá la estructura de un sistema multiprocesador de memoria compartida con procesadores homogéneos (véase la Figura 1.5). Los costes derivados del propio hardware se consideraron incluidos en el cálculo del peor tiempo de ejecución de cada una de las tareas (WCET, C_i). En la mayoría de planificadores usados, el número de desalojos está acotado y, por lo tanto, los costes también pueden incluirse en el WCET. Las tareas consideradas en esta tesis no utilizan recursos compartidos, con lo que se simplifican los posibles problemas de coherencia de cache. No obstante, en el Capítulo 5 se podrían incluir. En este caso el acceso a los recursos compartidos se tendría en cuenta en el análisis de la planificabilidad mediante el factor de bloqueo [DaW95] y a nivel de planificador se usarían mecanismos de exclusión mutua (*Priority Ceiling Protocols*, [SRL90]). Los costes de las migraciones entre procesadores se consideraron nulos. En todo caso, los sistemas de tiempo real usualmente permiten las técnicas de replicación del código de las tareas periódicas en las memorias caches para minimizar estos costes [Bus96]. No obstante, las modernas técnicas de las memorias caches en sistemas multicore, como son la compartición de la memoria cache L2 o L3 (véase al columna *Cache Compartida* en la Tabla 4), pueden ayudar a minimizar los costes de la migración entre cores e incluso entre procesadores.



(a) Cache Compartida (b) Cache Distribuida

Figura 1.5: Estructura de un sistema multiprocesador de memoria compartida.

La carga de los experimentos se generaró sintéticamente con diversos generadores desarrollados por el autor. La entrada de estos generadores son los parámetros que especifican la caracterización del conjunto de tareas y el número de procesadores (en las tablas del Anexo B se especifican los principales parámetros de los diferentes

tipos de conjuntos de tareas utilizados). La salida de los generadores son los parámetros que especifican las características generales de un cierto número de tareas (principalmente el periodo, el plazo y el peor tiempo de ejecución). Los simuladores utilizan esta información para realizar la planificación en un modelo guiado por eventos discretos.

La parametrización utilizada se eligió a base de recopilar información de gran diversidad de artículos, según nuestra propia experiencia y según marcaban algunos *benchmarks* [KaW91]. Las principales características que influyen en la planificación son: el *period ratio*, la multiplicidad de estos y el hiperperiodo. En la Tabla 5 se muestra un resumen de estas características para los parámetros de los conjuntos de tareas detallados en el Anexo B. El *period ratio* influye en la máxima utilización del procesador alcanzable [LSD89]. Los conjuntos de tareas armónicos aumentan la utilización del procesador [LiL73] y hacen que los eventos estén sincronizados. Los hiperperiodos largos requieren simulaciones más largas y requieren mayor cantidad de memoria para algunos planificadores [LRT92,RtL94].

PCT #	**Period Ratio (T_{max}/T_{min})**	**Hiperperiodo**	**Número de periodos posibles**	**Multiplicidad de los periodos**
0	16	pequeño	13	media
1	15	mediano	39	baja
2	10	grande	52	baja
3	30	grande	76	baja
4	24,41	grande	16	alta
5	32	muy pequeño	5	muy alta
6	16	muy grande	16	muy baja

Tabla 5: Resumen las principales características de los parámetros de los conjuntos de tareas del el Anexo B. Se remarcan las principales diferencias entre parametrizaciones contiguas.

En las tablas del Anexo B se especifican al menos los periodos mínimo y máximo y el hiperperiodo. El plazo de las tareas se ha fijado igual a su periodo ($D_i=T_i$). El resto de los periodos deberán estar dentro del rango $[T_{min},T_{max}]$ y ser múltiples de los factores en que se descompone el hiperperiodo, de forma que el mínimo común múltiplo de todos los periodos será justamente el hiperperiodo. Por ejemplo, con PCT # 3 se generan periodos utilizando los factores 2^4, 3^3, 5^3 y 7^1. Con estos números primos se pueden generar 76 periodos en el rango [100,3000].

Los parámetros de los conjuntos de tareas se han elegido de manera que sean similares entre ellos excepto en una de las características, para poder experimentar los efectos de esta característica diferenciadora sobre el rendimiento. Los parámetros PCT#6 son distintos al resto y se utilizaron para poder comparar con los resultados de un artículo concreto ([AnJ00b]).

El número de tareas periódicas normalmente se fijó entre 8 y 15 tareas por el número de procesadores. El factor de utilización de una tarea (U_i) se genera aleatoriamente siguiendo una distribución normal en el rango $(0,U_{max}]$ pero se garantiza que al menos una de ellas utilice U_{max}. El peor tiempo de cálculo de cada tarea (WCET o C_i) se calcula como el producto del periodo por el factor de utilización de la tarea ($C_i=T_iU_i$). En todas las simulaciones cada tarea periódica ejecuta siempre hasta su WCET. Aunque los planificadores podrían utilizar el eventual tiempo sobrante de las tareas periódicas, el objetivo de los experimentos era ver su comportamiento en los casos límite.

Las llegadas de las peticiones aperiódicas fueron generadas siguiendo una distribución de Poisson. Normalmente los tiempos intermedios entre llegadas elegidos estaban en el mismo rango que los periodos de las tareas periódicas. Sin embargo, en algunos experimentos se explicita la utilización de tareas aperiódicas de dimensiones fijas, más pequeñas y frecuentes, por ejemplo para una carga del 25% se usó $C_{aper}=1$ y un tiempo entre llegadas de 4.

Para un conjunto de tareas se realizaba una simulación en la que se tomaba un valor de una métrica concreta. La simulación se repetía, variando la semilla inicial para la

generación aleatoria de las llegadas de tareas aperiódicas, un mínimo de 5 veces, calculando la media aritmética de los resultados. Las simulaciones se repetían hasta alcanzar un nivel de confianza del 95% utilizando la prueba estadística *T-Student*. En el caso de no alcanzar el nivel de confianza en 35 simulaciones, se continuaba el proceso pero utilizando la prueba de la *Normal*. Así, se obtenía un valor para la métrica en un conjunto de tareas. El proceso se repetía para otros conjuntos de tareas.

El número de conjuntos de tareas generados y simulados para obtener cada punto de las gráficas de los experimentos fue variable entre 1000 y 100000, dependiendo de la métrica utilizada y de los valores obtenidos. El número particular se estableció buscando un margen de error del 1%. Cuando la métrica era la media aritmética de los tiempos de respuesta medios se utilizaba la prueba estadística de la *Normal* con un nivel de confianza del 95%. Cuando la métrica era un porcentaje de éxito o un porcentaje de aceptación, se utilizaba la distribución *Binomial*.

La longitud de cada simulación se estableció entre 10 y 50 hiperperiodos, dependiendo de la parametrización, para poder comprobar bien la planificación de las tareas periódicas y para que el número de tareas aperiódicas finalizadas fuera suficientemente grande.

En cada uno de los experimentos se utiliza un PCT del Anexo B y se detalla que parámetro se varió. Generalmente se realizaron experimentos variando uno de los siguientes parámetros:

- el número de procesadores (m)
- la carga periódica (U_{per})
- el factor de carga máxima de las tareas periódicas (U_{max})
- la carga aperiódica (U_{aper})
- el tiempo de ejecución requerido por las tareas aperiódicas (C_{aper})

1.6 Objetivos

El objetivo global de esta tesis es desarrollar un método teórico de planificación de sistemas de tiempo real estricto en arquitecturas multiprocesador con memoria compartida. Este método debe garantizar los plazos estrictos y tiene que aprovechar al máximo la capacidad de proceso que proporcionan los distintos procesadores del sistema.

La planificación de sistemas monoprocesador puede ser realizada fuera de línea de forma estática, o durante la ejecución, dinámicamente. La planificación fuera de línea es el método más comúnmente utilizado en sistemas de tiempo real crítico, como por ejemplo en aviónica o automovilismo, debido a que es el método más determinista. A su vez, por el hecho de realizarse de forma estática, es el método menos flexible, puesto que difícilmente pueden realizarse cambios. En los últimos años, los sistemas de tiempo real han crecido en complejidad y funcionalidad, precisando de una mayor capacidad de proceso. Frecuentemente gran parte de esta carga adicional viene dada por tareas que no son críticas y de ejecución aleatoria. Por eso, junto con la madurez de los conocimientos, cada vez más se ha ido utilizando la planificación dinámica en entornos monoprocesador. Sin embargo, en arquitecturas multiprocesador se siguen utilizando los métodos más deterministas y menos flexibles: se reparten de forma estática en tiempo de diseño las tareas entre los procesadores para que en tiempo de ejecución se pueda tratar cada procesador como un monoprocesador. Esta metodología reduce la complejidad pero no permite aprovechar al máximo la capacidad de proceso disponible. Por ejemplo, puede darse la circunstancia de tener unos procesadores muy cargados y, al mismo tiempo, otros procesadores completamente ociosos en el sistema. Además, esta metodología difícilmente se adapta a cambios. Es por esta razón que en esta tesis proponemos llevar a la práctica métodos que hagan un uso más flexible del sistema multiprocesador.

El primer objetivo de la tesis es diseñar un método de planificación que haga posible una máxima utilización de los procesadores. Por este motivo nos hemos centrado en la planificación *on-line*, basada en prioridades, con desalojo y con migraciones.

Además, todos los capítulos de la tesis se centraran en usar la capacidad de proceso sobrante de las tareas periódicas para dar servicio a algún tipo de tareas adicionales.

El segundo objetivo es comprobar que modalidad de distribución de las tareas es la más conveniente según el tipo de tareas. Así, en el tercer capítulo se analizan diversas posibilidades de distribución de los diversos tipos de tareas entre los procesadores. En este capítulo también se establecen los algoritmos que servirán de referencia en los estudios de rendimiento de los capítulos posteriores.

El tercer objetivo de la tesis es conseguir una planificación global de tareas periódicas, de forma que esta planificación sea flexible, pero al mismo tiempo garantizando los plazos de ejecución. El planificador global ha de ser simple, no realizar complicados cálculos de forma dinámica y no debería necesitar grandes estructuras de datos, de forma que pueda utilizarse con número de procesadores elevado de forma eficiente.

Si bien en algunas aplicaciones puede ser deseable conseguir el primer objetivo, tampoco se puede pretender conseguir que las tareas periódicas utilicen totalmente todos los recursos de computación puesto que, por un lado, los sistemas multiprocesador pueden llegar a tener una gran capacidad de cálculo y, por otro lado, debemos dejar recursos libres para permitir la ejecución de las tareas no críticas, las tareas aperiódicas y las tareas asociadas a las aplicaciones de uso común.

En consecuencia, el cuarto objetivo consiste en permitir la planificación conjunta de tareas periódicas y aperiódicas. Gracias a la planificación de las tareas periódicas flexible del tercer objetivo se podrá atender a eventos no periódicos. Así, el objetivo no solo consiste en la planificación conjunta, sino que se desea obtener buenos tiempos de respuesta para las tareas aperiódicas y unos altos índices de aceptación de tareas esporádicas. De está forma, combinando la carga procedente de tareas periódicas, aperiódicas y esporádicas, se propicia la máxima utilización del multiprocesador, que es el primer objetivo.

Para cumplir con los dos últimos objetivos, en el cuarto capítulo se diseña un nuevo método de planificación global del sistema multiprocesador que obtiene suficiente

flexibilidad en las tareas periódicas para dar buen servicio a las aperiódicas pero que no cumple con el requisito se los sistemas de tiempo real estrictos de garantizar todos los plazos.

Así, en el quinto capítulo se diseña un nuevo método de planificación híbrido entre la planificación basada en monoprocesadores y la planificación global del multiprocesador que sí logra cumplir con todos los objetivos propuestos.

1.7 Contribuciones

En el capítulo 3 se ha hecho evidente la dificultat de dimensionar servidores con planificación global. Tambien se ha estudiado la distribución dinámica de las tareas aperiódicas entre los procesadores cuando los planificadores locales se basan en *Total Bandwidth* (con prioridades dinámicas) y *Slack Stealing* (con prioridades estáticas). El primero aporta poca información al distribuidor y por eso no existen heurísticas y mientras que con el segundo sí las hay, resultando la distribución *Next-Fit* la mejor estrategia, a diferencia de los sistemas distribuidos, donde se suele usar la *First-Fit*.

En el capítulo 4 se ha adaptado el algoritmo para monoprocesadores *Dual Priority* a la planificación global de los multiprocesadores. Para ello se han propuesto dos métodos no analíticos de cálculo de los peores tiempos de respuesta de las tareas periódicas y se han analizado sus ventajas e inconvenientes. También se ha analizado un método analítico propuesto pero no explotado por otro autor [And03]. En este caso, se ha mostrado como el método no está probado y también se ha mostrado su escasa aplicabilidad. El resultado ha sido un planificador global para sistemas de tiempo real no estrictos con una muy baja probabilidad de pérdida de plazos.

Las principales aportaciones se producen en el capítulo 5. En el, se diseña un método de planificación híbrido entre la planificación local y la planificación global. Este método permite la ejecución de tareas periódicas conjuntamente con tareas aperiódicas con y sin plazo. Se ha diseñado una prueba de aceptación de tareas

aperiódicas con plazo. Para cada tipo de tareas se han diseñado y analizado diversos métodos de distribución de las tareas periódicas y se ha identificado cual es el mejor en cada caso. Finalmente, se ha diseñado un método de distribución que permite prestar buen servicio a todas los tipos de tareas. Este modelo de planificación cumple con todos los objetivos propuestos para la tesis: (i) es determinista, puesto que garantiza los plazos de las tareas periódicas en tiempo de diseño; (ii) permite una alta utilización del multiprocesador con solo tareas periódicas; (iii) permite el 100% de la utilización con la ejecución conjunta de tareas periódicas y aperiódicas; (iv) las tareas aperiódicas sin plazo en promedio obtienen un buen servicio; (v) proporciona un alto porcentaje de aceptación de las tareas aperiódicas con plazo o tareas esporádicas.

Capítulo 2

ANTECEDENTES

En este capítulo se resume el estado del arte de la planificación de sistemas de tiempo real. En el primer apartado se resumirá brevemente la planificación de sistemas monoprocesador, a modo de introducción y como referencia, puesto que son la base de la planificación de multiprocesadores. A continuación, en un segundo apartado, se detallará un amplio resumen los antecedentes existentes en la teoría de la planificación de los sistemas multiprocesadores.

2.1 Resumen de la planificación de monoprocesadores

En este punto se describen someramente los principales algoritmos de planificación de un monoprocesador siguiendo las características y la clasificación establecida en el punto 1.3. A modo de resumen, en la Figura 2.1 se puede observar una clasificación de los planificadores clásicos de tiempo real para monoprocesadores.

El principal planificador fuera de línea es el *ejecutivo cíclico* el cual se basa en un plan realizado en la etapa de diseño. En éste se especifica, para cada instante, que tarea se debe ejecutar y la duración máxima de la misma. El proceso planificador simplemente se encarga de seguir el plan en tiempo de ejecución del sistema. Las ventajas principales son: que los plazos se garantizan al diseñar el sistema, que es un método sencillo y robusto aunque de bajo nivel, no hay concurrencia en la ejecución, no hace falta usar mecanismos de exclusión mutua ni de sincronización y no hace falta analizar el comportamiento temporal ya que el sistema es correcto por construcción. Sin embargo, el principal problema que presenta es que el tiempo de generación del plan crece exponencialmente con el número de tareas, dado que en el caso de periodos de las tareas armónicos es un problema NP-completo [BuW01] y en el caso más general es NP-duro [GaJ79]. Además, como principales inconvenientes cabe citar que las tareas esporádicas y aperiódicas son difíciles de tratar y es poco flexible, debe reservar para la ejecución de las tareas todo el WCET y es difícil de mantener, teniendo que rehacer todo el plan cuando se cambia la implementación de

una tarea. Por último, desde la perspectiva de la ingeniería del software, no es elegante, debido a que un programa con un tiempo de cálculo elevado normalmente tiene que ser dividido en procedimientos más pequeños, de tiempo de ejecución predefinido, lo que puede romper la estructura del código, por tanto puede ser más difícil de verificar y más propenso a errores.

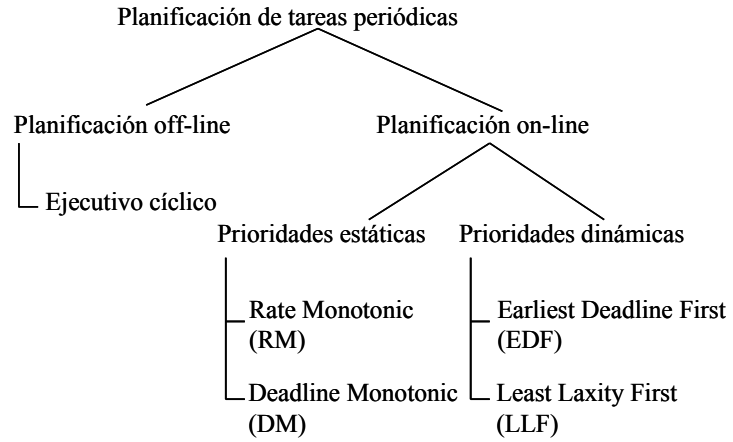


Figura 2.1: Clasificación de los planificadores clásicos de tiempo real para monoprocesadores.

Los *planificadores on-line* establecen en tiempo de ejecución del sistema que tarea se debe ejecutar en cada momento, por tanto son más flexibles al entorno de la aplicación, pudiéndose adaptar a la carga real del sistema. Los principales *planificadores on-line* con asignación estática de prioridades son los planificadores expulsivos de prioridades fijas en los que la asignación de prioridades es *Rate Monotonic* o *Deadline Monotonic*. Los principales planificadores con asignación dinámica de prioridades son el *Earliest Deadline First* y el *Least Laxity First*. Un artículo reciente de G. Buttazzo [But05] describe las ventajas, inconvenientes y falsos mitos entre ambos tipos de planificadores. A continuación se verá un breve resumen de estos planificadores.

La asignación de prioridades con un criterio *Rate Monotonic* [LiL73] asigna las prioridades a las tareas dependiendo de su periodo de activación, de manera que una tarea será más prioritaria cuanto más pequeño sea el periodo de activación. Mientras que la asignación de prioridades con criterio *Deadline Monotonic* [LeW82] asigna las prioridades en función del plazo relativo, de manera que las tareas con menor plazo relativo tendrán una prioridad mayor que aquellas tareas con mayor plazo

relativo. Se dice que ambas asignaciones con planificadores de prioridades fijas son óptimas en el sentido de la planificación, de manera que si se puede hallar una asignación de prioridades que haga que el conjunto de tareas sea planificable, usando la asignación *Rate Monotonic* (con los plazos relativo iguales a los periodos) o *Deadline Monotonic* (con los plazos relativos menores o iguales a los periodos) también lo serán. Para este tipo de planificadores de prioridades fijas existen pruebas de planificabilidad basadas en la utilización del procesador que si se superan implica que se puede garantizar la planificabilidad del sistema. La prueba de planificabilidad de C.L. Liu y J.W. Layland [LiL73] establece que la utilización máxima garantizada para *n* tareas periódicas viene dada por la siguiente ecuación:

$$U \leq n \cdot (2^{\left(\frac{1}{n}\right)} - 1) \qquad \text{(ecuación 2.1)}$$

Conforme el número de tareas del sistema va aumentando el limite tiende a "ln 2", es decir, aproximadamente el 69.3% de utilización total. Esta prueba es suficiente, pero no necesaria, lo que implica que en realidad se puede asegurar mayores utilizaciones del procesador [LSD89], dependiendo de los periodos de las tareas. Una prueba de planificabilidad necesaria y suficiente es la que está basada en el cálculo del tiempo de finalización de las tareas en el instante crítico. Si para todas ellas este tiempo es menor que su plazo temporal entonces el sistema será planificable. Para calcular el peor tiempo de finalización de las tareas exacto puede usarse la siguiente iteración lineal definida por Joseph y Pandya en [JoP86]:

$$W_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i^n}{T_j} \right\rceil \cdot C_j \qquad \text{(ecuación 2.2)}$$

donde *hp(i)* es el conjunto de tareas periódicas con una prioridad superior a la tarea τ_i, siendo un valor inicial aceptable

$$W_i^0 = C_i + \sum_{j \in hp(i)} C_j \qquad \text{(ecuación 2.3)}$$

se termina la iteración cuando

$$W^{n+1} = W^{n} \qquad \text{(ecuación 2.4)}$$

o bien cuando el valor calculado es mayor que el plazo, es decir, no cumpliría el plazo temporal

$$W^{n+1} > D_i \qquad \text{(ecuación 2.5)}$$

El planificador *Earliest Deadline First* [LiL73] asigna las prioridades a las tareas dinámicamente en función de sus plazos temporales y del estado del sistema (véase el libro de Stankovic et. al. [SSR98]). De esta manera la tarea con un plazo absoluto más inmediato será la tarea que tenga mayor prioridad. Es decir, ejecuta siempre la tarea más urgente independientemente del tiempo de cálculo de esa tarea. En el momento de una activación, deben recalcularse las prioridades para dar cabida a la nueva tarea. Si esta nueva tarea tiene su plazo temporal más inmediato pasara a ser la tarea que se ejecute, adueñándose en este momento del procesador. La prueba de planificabilidad suficiente y necesaria cuando los plazos de las tareas son iguales a sus periodos es la que viene dada por la ecuación 2.6 e indica que en EDF un conjunto de tareas es planificable siempre que la utilización del procesador sea menor o igual al 100%.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \qquad \text{(ecuación 2.6)}$$

El planificador *Least Laxity First* [Der74] asigna las prioridades dinámicamente a las tareas dependiendo del tiempo que les sobra para alcanzar el plazo temporal después de descontar el tiempo que todavía necesita para finalizar su ejecución, es decir según su laxitud ($L_i(t)= d_i - C_i(t)$). Este planificador tiene el inconveniente que puede provocar excesivos cambios de contexto en el momento en que dos tareas tengan la misma laxitud. Las tareas se van alternando el tiempo de procesador entre ellas, ya que la tarea que espera a ser ejecutada va aumentando su prioridad, mientras que la tarea que se está ejecutando la va disminuyendo. Sin embargo existe una modificación de este algoritmo que evita este problema [OhY98], permitiendo una inversión de laxitud, en caso de empate entre dos tareas se escoge la tarea con menor plazo absoluto hasta su finalización o hasta la llegada de una nueva tarea.

	$D_i = T_i$	$D_i \leq T_i$
Prioridad estática: RM y DM	$U = n*(2^{\left(\frac{1}{n}\right)} - 1)$ (ecuación 2.1)	$\forall i : R_i^n = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil \cdot C_j \leq D_i$ (ecuación 2.2)
Tipo de prueba de planificación	basada en la utilización	basada en los tiempos de respuesta
Prioridad dinámica: EDF	$U \leq 1$ (ecuación 2.6)	$\forall L : \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) \cdot C_i \leq L$
Tipo de prueba de planificación	basada en la utilización	basada en la demanda realizada al procesador

Tabla 6: Resumen las principales pruebas de planificabilidad

Todos los algoritmos explicados anteriormente son óptimos desde el punto de vista de la planificación. En su vertiente más básica, los algoritmos anteriores están basados en conjuntos de tareas periódicas independientes. Algunos de ellos se pueden extender en diversos aspectos, como por ejemplo, incorporar tareas aperiódicas, recursos compartidos. Sin embargo, la inclusión de la planificación de tareas que no sean periódicas es fundamental para esta tesis pero la inclusión de recursos compartidos va más allá de lo propuesto en esta tesis. Por ello, a continuación se resume[4] los principales algoritmos capaces de garantizar los plazos de las tareas periódicas y, al mismo tiempo, capaces de dar servicio a tareas aperiódicas.

Inclusión de tareas aperiódicas.

Para combinar la ejecución de tareas periódicas con tareas aperiódicas sin plazo la solución más fácil es tratar a estas últimas en segundo plano (*background*), es decir,

[4] Sugerimos al lector interesado el buscar más detalles en las referencias proporcionadas

ejecutarlas según orden de llegada (FIFO), cuando no haya ninguna tarea periódica pendiente de ejecución. Sin embargo esta política puede suponer unos tiempos de respuesta para las tareas aperiódicas muy prolongados. Si bien existe la posibilidad de aplicar un plazo a las tareas aperiódicas urgentes, también es un objetivo muy frecuente el tratar de minimizar el tiempo de respuesta medio de las tareas aperiódicas sin plazo. Con este objetivo en la literatura se han desarrollado multitud de algoritmos de planificación para monoprocesadores que garantizan los plazos de las tareas periódicas y, al mismo tiempo, intentan dar un buen servicio a las tareas aperiódicas. La mayoría de ellos están basados en la reserva de capacidad o ancho de banda (*bandwidth*) utilizando servidores. Para ello, añaden una o varias tareas periódicas llamadas servidores que van a dar servicio a las tareas aperiódicas, es decir, las van a ejecutar. Estos servidores serán nuevas tareas periódicas, con periodo T_s, tiempo de cómputo C_s y con una carga máxima $U_s = C_s/T_s$. Así, las pruebas de planificabilidad existentes se pueden aprovechar para calcular T_s y C_s que maximicen U_s y continuar garantizando todos los plazos de todas las tareas.

Los algoritmos que utilizan servidores de prioridades fijas más importantes son los siguientes. El servidor por encuesta (*Polling Server*) [LSS87,SSL89] cuando se activa sondea si hay tareas aperiódicas pendientes y las ejecuta. Tiene el inconveniente que si no hay tareas aperiódicas pendientes el servidor se suspende, perdiendo su capacidad. Por el contrario, el servidor diferido (*Deferrable Server*) [LSS87] mantiene su capacidad hasta el final de su periodo de manera que se pueden servir peticiones mientras quede capacidad disponible en el servidor. La capacidad se renueva con cada periodo pero sin acumularse. El servidor de intercambio de prioridades (*Priority Exchange Server*) [LSS87] también preserva su capacidad porque la intercambia por la ejecución de tareas periódicas de menor prioridad. La implementación es más compleja pero la pérdida de capacidad es menor que con el servidor diferido. En el servidor esporádico (*Sporadic Server*) [SSL89] la restauración de la capacidad no es periódica sino que solo se restaura tras haberse consumido por tareas aperiódicas. A diferencia de los anteriores servidores, éste no disminuye la utilización máxima y la sobrecarga es menor que con el intercambio de

prioridades. De hecho, el estándar POSIX (IEEE Std1003.1d D12, 1999) da soporte a este último tipo de servidores.

La extracción de holgura (SS, *Slack Stealing*) [LRT92,LRT94,RtL94] no utiliza un servidor periódico, sino que usa una tarea pasiva para servir tareas aperiódicas. Esta tarea utiliza la holgura que le proporcionan las tareas periódicas. Para ello es necesario un análisis fuera de línea para calcular la holgura de cada activación de cada tarea y guardar esta información en una tabla, que puede requerir una gran cantidad de memoria si el hiperperiodo es grande. También existe la extracción de holgura dinámica [DTB93] que no tiene el inconveniente de necesitar gran cantidad de memoria pero que requiere mayor capacidad de proceso para calcular la holgura dinámicamente. El método de la extracción de holgura mejora substancialmente el tiempo de respuesta de las tareas aperiódicas con respecto a los anteriores. Además, sus autores reclaman que estos algoritmos son óptimos en el sentido que minimizan este tiempo de respuesta entre todos los algoritmos que cumplen los plazos de las tareas periódicas (con prioridades fijas, expulsivos y tratamiento FIFO para las tareas aperiódicas), pero en [TLS96] se matiza y se proponen nuevas variantes óptimas en diversos sentidos. De hecho, en [TLS96] se establece que con prioridades fijas no existen servidores óptimos.

	Rendimiento	Complejidad computacional	Necesidades de memoria	Complejidad de implementación
Background	☹	☺	☺	☺
Polling Server	☹	☺	☺	☺
Deferrable Server	😐	☺	☺	☺
Priority Exchange	😐	😐	😐	😐
Sporadic Server	😐	😐	😐	😐
Slack Stealing	☺	☹	☹	☹

Tabla 7: Resumen de la evaluación de los servidores con prioridad fija [But00] (leyenda: ☹ malo, 😐 bueno, ☺ excelente)

Existen variantes con servidores de prioridades dinámicas, basados fundamentalmente en EDF. Éstos se caracterizan por una mejor utilización del procesador y unos

servidores con mayor capacidad que dan mejor servicio a las tareas aperiódicas. Los más importantes son: *Dynamic Priority Exchange Server*, *Improved Priority Exchange Server* y *Total Bandwidth Server* [SpB94, SpB96], *Dynamic Sporadic Server* [SSL89,SpB96] y *Earliest Deadline Late Server* [ChC89,SpB96]. Éste último es considerado como una versión dinámica de la extracción de holgura y tiene una complejidad computacional excesiva. El algoritmo *Total Bandwidth Server* (TBS) no utiliza servidor pero asigna plazos a las tareas aperiódicas de forma que se garantiza que la carga total de estas no sobrepase la capacidad de un hipotético servidor (U_s), según las siguientes ecuaciones:

$$d_k = \max(r_k, d_{k-1}) + C_k/U_s \qquad \text{(ecuación 2.7)}$$

$$U_p + U_s \leq 1 \qquad \text{(ecuación 2.8)}$$

Existe una versión de este algoritmo que optimiza la asignación de plazos (*TB Star* o TB* [BuS99]), reduciendo así el tiempo de respuesta de las tareas aperiódicas. El problema reside en que tiene una complejidad lineal creciente. A cada petición aperiódica se ejecuta un algoritmo que debe ejecutarse un número de veces que crece con la carga periódica y cuando los periodos son pequeños.

	Rendimiento	Complejidad computacional	Necesidades de memoria	Complejidad de implementación
Background	☹	☺	☺	☺
DPE	😐	😐	😐	😐
DSS	😐	😐	😐	😐
TB	😐	☺	☺	☺
EDL	☺	☹	☹	☹
IPE	☺	😐	☹	☹
TB*	☺	☹	☺	😐

Tabla 8: Resumen de la evaluación de los servidores con prioridad dinámica [But00] (leyenda: ☹ malo, 😐 bueno, ☺ excelente)

El algoritmo de planificación de prioridad dual (*Dual Priority* (DP), [BuW93,DaW95]) se basa en la misma idea del algoritmo de extracción de holgura:

no hay beneficio en terminar las tareas periódicas antes de su plazo. Calculando el peor tiempo de respuesta de las tareas periódicas se puede saber cuanto tiempo como mínimo se puede retardar su ejecución sin comprometer su plazo. Así pues, el rango de prioridades se divide en tres niveles. Las tareas periódicas empiezan con prioridades dentro del nivel bajo. Las tareas aperiódicas se ejecutan en el nivel medio, pudiendo así retardar las tareas periódicas de bajo nivel. Cuando una tarea periódica no se puede retardar más ($D_i - WCRT_i$) aumenta su prioridad al nivel alto, donde tiene el plazo garantizado. Este algoritmo es difícil de catalogar ya que, aunque utiliza prioridades fijas, la prioridad de las tareas periódicas va variando entre unos valores predeterminados dependiendo de la evolución (dinámica) de las tareas. Como puede apreciarse, al igual que SS y TB no utiliza ningún servidor. Sin embargo, a diferencia de SS no tiene necesidades de memoria. En el apartado 4.2 se detalla y analiza con mayor profundidad el algoritmo de planificación para monoprocesadores de prioridad dual puesto que es una base fundamental para la realización de esta tesis.

Una alternativa dirigida por tiempo y no por eventos es la técnica del desplazamiento de ranuras (*Slot-Shifting Server*) [Foh95,IsF00]. Fuera de línea se calcula la capacidad sobrante de las tareas periódicas en cada instante de tiempo, indicando cuanto se puede desplazar la ejecución de la siguiente tarea periódica sin que ésta deje de cumplir su plazo, y se almacena esta información en una tabla. En tiempo de ejecución, esta información es utilizada para atender tareas aperiódicas desplazando las periódicas. Aunque las tareas aperiódicas obtienen buenos tiempos de respuesta, los requisitos de memoria y la complejidad computacional son altos, puesto que se utilizan tablas de capacidad sobrante que hace falta actualizar.

En función de los objetivos planteados en el apartado 1.6, esta tesis se ha basado en planificadores dirigidos por eventos y con expulsiones puesto que proporcionan una gran flexibilidad y un alto grado de utilización del procesador. Así, principalmente se han utilizado los algoritmos *Slack Stealing, Total Bandwith y Dual Priority.* El primero, aunque puede tener unos requisitos de memoria o de cómputo muy elevados, se ha utilizado como referencia por su casi optimalidad. Los dos restantes

se han utilizado por su compromiso entre la simplicidad (con costes de implementación, memoria y computacionales bajos) y su elevado rendimiento.

2.2 Planificación de multiprocesadores

Al diseñar un planificador para un sistema de tiempo real en un monoprocesador las primeras y principales decisiones a tomar son (i) si se van a permitir las apropiaciones de cpu y (ii) si las prioridades van a ser estáticas o dinámicas. En un sistema multiprocesador además habrá que decidir (iii) si se reparten las tareas entre procesadores de forma estática (método del particionado) o si éstas se reparten de forma dinámica (planificación global) y (iv) si se van a permitir las migraciones de tareas o de parte de estas.

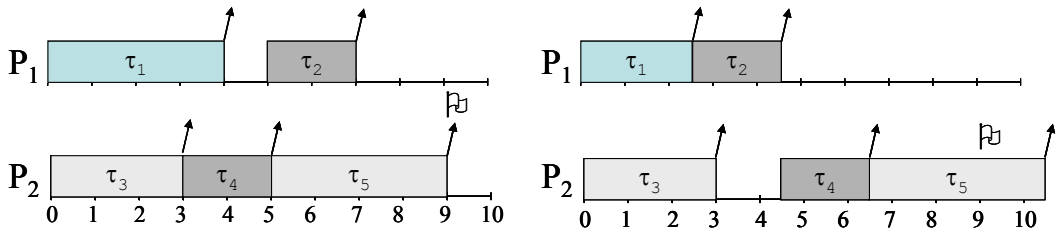
El problema de la planificación de sistemas de tiempo real con multiprocesadores es NP-completo en la mayoría de las ocasiones. Esto significa que el problema es tan 'duro' como un gran número de otros problemas que han sido ampliamente reconocidos como 'difíciles' por los expertos en algorítmica. Como consecuencia, se hace muy difícil hallar simultáneamente una solución a la distribución de las tareas entre los procesadores, asignarles unas prioridades y pasar una prueba de planificación.

Para corroborar que el problema se hace casi intratable[5] citaremos algunos casos donde se demuestra la complejidad del problema aún con hipótesis simplistas. Según Garey y Johnson [GaJ75] el problema de la planificación de un multiprocesador con solo dos procesadores, sin recursos compartidos, con tareas independientes y con cualquier tiempo de cómputo es NP-Completo. Según Ullman [Ull73] el problema de la planificación de un multiprocesador con *m* procesadores, sin recursos compartidos, con un orden parcial arbitrario y con un tiempo de cómputo de solo una unidad es NP-Completo. Incluso en el caso de los planificadores con apropiaciones, donde la flexibilidad es mayor, el problema sigue siendo complejo. Según Lawler [Law83] el problema de la planificación de un multiprocesador con *m* procesadores permitiendo las apropiaciones y minimizando el número de tareas tardías es NP-Duro.

[5] Un problema se dice que es intratable cuando "probablemente" no podrá ser resuelto por ningún algoritmo en tiempo polinómico.

Las principales causas subyacentes de la dificultad en la planificación de los sistemas multiprocesadores son: (i) las anomalías experimentadas en los multiprocesadores; (ii) la dependencia de las prioridades relativas y (iii) la dificultad de encontrar el instante crítico.

Las anomalías en la planificación de los sistemas multiprocesadores son efectos negativos que se producen de forma contra intuitiva. Así, cambios aparentemente positivos causan que algunas tareas ya no cumplan sus plazos. Según Graham [Gra76], cambiar la lista de prioridades, incrementar el número de procesadores, reducir los tiempos de ejecución o reducir las restricciones de precedencia de las tareas son factores que pueden hacer que la ejecución de algunas tareas finalice más tarde. Es decir, contrariamente a lo que se podría pensar, incrementar los recursos o reducir las demandas puede ser perjudicial, pudiendo las tareas a llegar a perder algunos plazos. A modo de ejemplo, se puede observar la Figura 2.4, donde la tarea τ_1 termina antes de lo previsto provocando así que la tarea τ_5 pierda su plazo.



(a) τ_5 termina antes de su plazo (t=9) (b) τ_1 termina antes y τ_5 pierde su plazo

Figura 2.2: Ejemplo de la Anomalía de Richardson: τ_2 y τ_4 comparten un recurso en exclusión mutua. En (a) τ_5 termina antes de su plazo. En (b) τ_1 termina antes de lo previsto y provoca que τ_2 entre en la sección crítica antes que τ_4, provocando que τ_5 pierda su plazo.

Es importante destacar que esta observación invalida la mayoría de las teorías de la planificabilidad de los monoprocesadores. Estas están basadas en los peores tiempos de ejecución y, como es de suponer, las tareas normalmente tendrán tiempos de ejecución menores al caso peor. Sin embargo, en un multiprocesador, que una tarea termine antes de lo previsto podría cambiar toda la planificación posterior si esta no es estática.

De la misma forma, el hecho de cambiar la prioridad relativa entre las tareas de mayor prioridad que una determinada tarea puede afectar a la planificabilidad de ésta. El cambio de prioridades puede alterar la asignación de tareas a procesadores, de manera que la tarea de baja prioridad podría terminar antes o después de lo previsto. Por el contrario, en un sistema monoprocesador, estas alteraciones de prioridades no afectan, pues la interferencia que sufre una tarea de menor prioridad sería la misma.

Un factor de incertidumbre adicional en la planificación de multiprocesadores es que su ganancia no es lineal con el número de procesadores. Según la Ley de Amdahl [Amd67], la ganancia o *speedup* viene limitada por la parte secuencial del programa. Las tareas periódicas de tiempo real no suelen ser paralelizables, reduciendo la posible ganancia. Además, existen otros factores a nivel de arquitectura, como pueden ser los recursos compartidos (buses, caches, etc.), que en general pueden suponer que la utilización de un multiprocesador con *m* procesadores idénticos tenga más *overheads* que un procesador *m* veces más rápido. Esta observación se utilizará en diversas fases de la tesis. En particular, esta característica dificulta enormemente que las pruebas de planificabilidad de sistemas de tiempo real para monoprocesadores se puedan aplicar a multiprocesadores.

En la planificación global de un sistema multiprocesador el instante crítico no tiene porque ocurrir cuando todas las tareas del sistema se activan simultáneamente, como ocurre con un solo procesador. Intuitivamente, puede haber instantes con cargas parecidas a la de un instante sincrónico pero ligeramente inferiores que cambien la asignación de tareas a procesadores lo suficiente para alargar la ejecución de una tarea de baja prioridad más allá de su plazo. Es importante destacar que el instante crítico es la base de muchos de los test de planificabilidad de los sistemas monoprocesadores.

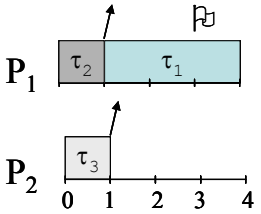
Uno de los estudios de la planificabilidad de sistemas multiprocesadores para tiempo real que ha sentado las bases es el realizado por Dertouzos y Mok [DeM89]. En él se estudia de forma teórica la planificabilidad utilizando una representación del espacio laxitud-tiempo de cálculo restante (véase Figura 2.5 en la página 56). En este artículo

se demuestra que no existen algoritmos óptimos sin un conocimiento a priori de todos y cada uno de los siguientes parámetros de las tareas: i) tiempos de inicio, ii) tiempos de ejecución y iii) los plazos de ejecución.

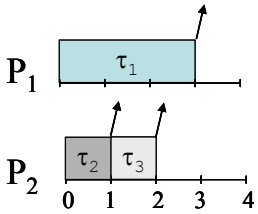
Otra conclusión importante de este trabajo es que no existen algoritmos para multiprocesadores que sean óptimos, que minimicen el número de apropiaciones y que degraden controladamente, como puede ser el caso de EDF en monoprocesadores. En particular, ni EDF ni LLF son óptimos para multiprocesadores (véanse la Figura 2.3 y la Figura 2.4). En la primera EDF no es planificable y LLF sí que lo es, mientras que en la segunda es al revés. Otro ejemplo con m procesadores y m+1 tareas con $T_i=C_i=T$ y $C_i= T/2 + \varepsilon$ (con $\varepsilon \rightarrow 0$) no es planificable con EDF ni LLF, a pesar de tener una carga aproximadamente del 50% para m grandes.

Tarea	WCET	Plazo	Laxitud
τ_1	3	3	0
τ_2	1	2	1
τ_3	1	2	1

(a) Conjunto de tareas. U= 2.0



(b) EDF: τ_1 pierde su plazo

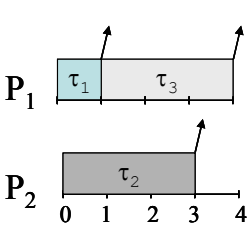


(c) LLF: acaban a tiempo

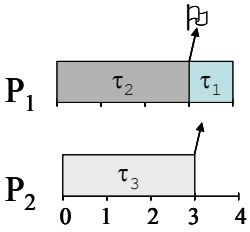
Figura 2.3: Ejemplo donde EDF no es planificable con 2 procesadores (b) mientras que LLF sí lo es (c)

Tarea	WCET	Plazo	Laxitud
τ_1	1	3	2
τ_2	3	4	1
τ_3	3	4	1

(a) Conjunto de tareas. U= 1.83



(b) EDF: acaban a tiempo



(c) LLF: τ_1 pierde su plazo

Figura 2.4: Ejemplo donde LLF no es planificable con 2 procesadores (c) mientras que EDF sí lo es (b)

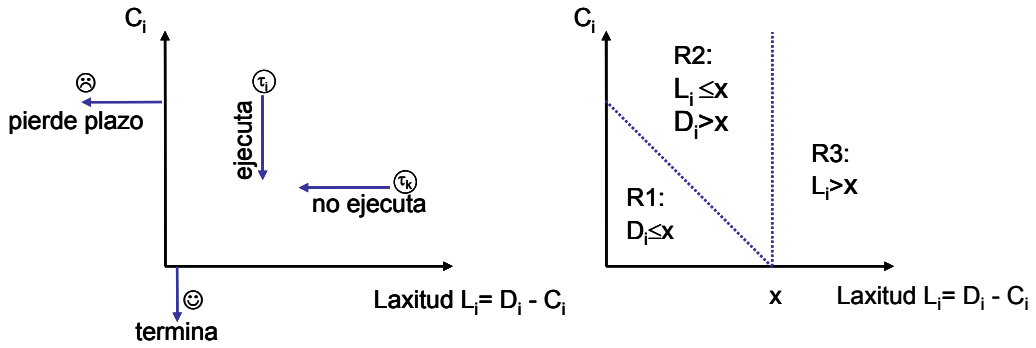


Figura 2.5: Representación del juego de la planificación usando el espacio laxitud-cálculo. En esta representación cada tarea se representa por un punto y se muestra su evolución por medio de trayectorias. En la gráfica de la izquierda se muestran las reglas: mientras una tarea no se ejecuta su trayectoria es horizontal y cuando se ejecuta es vertical. En la gráfica de la derecha se identifican las regiones que representan estados significativos de las tareas.

Utilizando la representación de la Figura 2.5 los autores hallan una condición necesaria para que un sistema sea planificable. Así pues, para todo x se debe cumplir la siguiente condición:

$$F(x,t) = m*x - \Sigma_{\tau \in R1}\, c_\tau(t) - \Sigma_{\tau \in R2}\, (x - l_\tau(t)) \geq 0 \qquad \text{(ecuación 2.9)}$$

donde

$F(x,t)$	es el cómputo sobrante en las siguientes x unidades de tiempo
$c_\tau(t)$	denota el cómputo restante en el tiempo t para la tarea τ
$l_\tau(t)$	denota la laxitud de la tarea τ en el tiempo t
R1	es la región en el espacio L-C con tareas con plazos en las próximas x unidades de tiempo (tareas críticas: tienen que ejecutar $c_\tau(t)$ unidades de tiempo urgentemente)
R2	es la región en el espacio L-C con tareas con laxitudes que no sobrepasen x unidades de tiempo y con plazos más allá de x unidades de tiempo (tareas que se deben ejecutar parcialmente: al menos $x - l_\tau(t)$)
R3	es la región en el espacio L-C con tareas con laxitudes mayores que x unidades de tiempo (no hace falta ejecutarlas en las x unidades de tiempo siguientes).

También concluyen que si esta condición se cumple cuando todas las tareas se activan al unísono, también se cumplirá si algunas se activan más tarde, pero esto no es válido para tareas periódicas. En este caso los autores hallan la siguiente condición suficiente:

Sean $T = GCD(D_1 \ldots D_m)$ y $t = GCD(T, T*C_1/D_1, \ldots T*C_m/D_m)$

Entonces el conjunto es planificable si t es un entero.

Esta condición no es genérica ni mucho menos y, por lo tanto, es de difícil utilización en aplicaciones reales.

Dada la complejidad del problema cabe tomar alguna de las siguientes líneas: (i) utilizar el multiprocesador como un conjunto de monoprocesadores (ii) diseñar algoritmos de planificación probabilísticos, que actúen con ciertos niveles de garantías; (iii) utilizar heurísticas para reducir la complejidad de la planificación.

En esta tesis abogamos por esta tercera línea y por utilizar los mecanismos que proporcionen una mayor flexibilidad para sacar el máximo partido de los recursos disponibles. Para ello hace falta fijar los criterios de diseño expuestos en el primer párrafo de esta sección.

Los planificadores que permiten apropiaciones tienen una mayor flexibilidad en la asignación de tiempo de cpu a activaciones de tareas debido a que la ejecución de estas tareas se fracciona en partes menores. Esta mayor flexibilidad puede facilitar el encontrar una solución factible cuando sin apropiaciones un conjunto de tareas no sería planificable. Esta ventaja puede desaparecer cuando hay recursos compartidos, pues el uso de un recurso compartido puede dificultar ciertas apropiaciones. El coste a pagar puede ser una mayor complejidad en el algoritmo de planificación y unos retrasos en la ejecución que deberán tenerse en cuenta en el análisis del peor tiempo de respuesta.

En esta tesis nos vamos a centrar en la planificación con apropiaciones, puesto que uno de los objetivos es el permitir ejecutar simultáneamente tareas periódicas con aperiódicas y/o esporádicas y, para obtener un buen servicio para las tareas no periódicas vamos a requerir la flexibilidad proporcionada por los planificadores con apropiaciones.

El siguiente dilema en la planificación de sistemas multiprocesadores consiste en decidir si se utiliza una planificación local o global, es decir, si la asignación de tareas a procesadores se hace de forma estática o dinámica. La asignación estática esquiva los problemas de la planificación global, pues trata el sistema como varios procesadores más o menos aislados. Así, los procesadores usados para ejecutar una tarea se establecen en tiempo de diseño, es decir, fuera de línea. En este caso existen diversas familias de soluciones disponibles: el particionado de tareas usando algoritmos de empaquetado, algoritmos de búsquedas guiadas y algoritmos de búsquedas no guiadas. Por el contrario, la planificación dinámica tiene los problemas de las anomalías, la asignación de prioridades, la asignación parcial de tareas a procesadores y finalmente, pero quizás el más importante, la determinación de la planificabilidad del sistema. En este caso, los procesadores usados para ejecutar una tarea los determina un planificador global en tiempo de ejecución.

Finalmente, otro dilema importante es el hecho de permitir a las tareas migrar de un procesador a otro. En el método del particionado esto no es posible: cada activación de una tarea debe ejecutarse en un procesador predeterminado, como si se tratase de varios monoprocesadores. En el caso de las búsquedas, tanto si son guiadas como si no, la migración dependerá de las restricciones impuestas. Finalmente, con la planificación global, las tareas pueden ejecutarse en cualquier procesador.

2.3 Asignación estática de tareas: planificación local

La asignación estática de tareas a procesadores es una simplificación del problema, reduciéndolo a una extensión de las técnicas bien conocidas de los monoprocesadores. Es uno de los métodos preferidos hasta la fecha por su fortaleza teórica.

La distribución se puede realizar utilizando algún algoritmo heurístico (planificación particionada) o bien utilizando algún algoritmo de búsqueda de una solución plausible. En este último caso la exploración del árbol de posibles soluciones se puede hacer de forma guiada o de forma aleatoria.

2.3.1 Planificación particionada

Con la planificación particionada en tiempo de diseño se organizan las tareas en grupos y cada grupo se asigna a un procesador concreto. En tiempo de ejecución existe una cola de tareas disponibles local a cada procesador, de manera que las tareas encoladas ahí solo se pueden ejecutar en el procesador predeterminado.

La principal ventaja de éste método es que se pueden aplicar las técnicas de la planificación en monoprocesadores, que están muy desarrolladas y consolidadas. Utilizando algoritmos de empaquetado se puede automatizar el reparto de las tareas entre los procesadores. De esta forma se reduce rotundamente la complejidad del sistema multiprocesador.

Sin embargo, este método tiene el inconveniente de no poder explotar bien los recursos de ejecución no utilizados: mientras un procesador puede estar muy atareado momentáneamente otro puede estar ocioso. Esto puede ser realmente exagerado si en la planificación se utilizan WCET muy pesimistas.

El problema de decidir si un conjunto de tareas es planificable en *m* procesadores con la planificación particionada es NP-completo en el sentido estricto [LeW82]. Como consecuencia, puede no haber ningún algoritmo que encuentre una solución óptima en tiempo pseudo-polinómico.

Los algoritmos de empaquetado (*bin-packing algorithms*) se basan en la idea de empaquetar objetos de dimensiones variables en cajas con el objetivo de minimizar el número de cajas usadas. En el caso de los multiprocesadores las cajas son procesadores mientras que los objetos son tareas. Así, una tarea cabe en un procesador si pasa una determinada prueba de planificabilidad junto con las tareas previamente asignadas a ese procesador.

En la Tabla 9 se resumen las características de los principales algoritmos de empaquetado estático.

Algoritmo	**Condición**	**Complejidad**	$\lim_{m_{opt} \to \infty} m/m_{opt}$	**Referencia**
RMNF	IP	O(n log n)	2.67	[DhL78]
RMFF	IP	O(n log n)	2.33	[DhL78]
RMBF	IP	O(n log n)	2.33	[OhS93]
RM-FFDU	UO	O(n log n)	5/3 = 1.67	[OhS95]
FFDUF	L&L,ecuación 2.1	O(n log n)	2.0	[DaD86a]
RMST	PO	O(n log n)	$1/(1-U_{max})$	[BLY95]
RMGT	PO, ES	O(n log n)	7/4 = 1.75	[BLY95]
RBOUND-MP	RBOUND[6]	O(n log n)	no disponible	[LMM03]

Tabla 9: Resumen de los algoritmos de reparto estático de tareas a procesadores

donde

IP (*Increasing-Period*) $\equiv$ $\frac{C_n}{T_n} \leq 2\left(1+\frac{U}{n-1}\right)^{-(n-1)} - 1$

UO (*Utilization-Oriented*) $\equiv$ $\frac{C_n}{T_n} \leq \frac{2}{\prod_{i=1}^{n-1}(1+U_i)} - 1$

PO (*Period-Oriented*) [4]

ES (*Exact Schedulability,* [LSD89]) [4]

Uno de los mejores algoritmos de empaquetado de tareas periódicas con plazos es el *Rate-Monotonic-First-Fit* (RMFF) [DhL78]. Se utiliza para distribuir tareas periódicas independientes que serán ejecutadas localmente con prioridad *Rate-Monotonic* (RM). Ordena las tareas por orden creciente de periodos y va asignando cada tarea al primer procesador, siguiendo siempre el mismo orden, que cumpla el test de utilización máxima en RM (ecuación 2.1). Conforme son necesarios más

[6] Estas condicones no se pueden expresar en una ecuación simple. El lector interesado debe recurrir a la referencia. Además, estas condiciones no se han utilizado en esta tesis.

procesadores se van añadiendo. El límite para la utilización garantizada con *m* procesadores es:

$$m(2^{1/2} -1) \leq U_{RMFF} \leq (m+1) / (1+2^{1/(m+1)}) \qquad \text{[OhB98]}$$

Es remarcable que este límite teórico solo garantiza entre un 41,42% y un 50% de utilización del multiprocesador (véase la Figura 2.6). Esto es más grave si tenemos en cuenta que para el cálculo de este límite se utiliza el WCET, cuando en la realidad la utilización será inferior. Sin embargo, en la planificación de monoprocesadores es fácil obtener utilizaciones del 88% [LSD89]. Además, si el algoritmo de planificación puede atender a otras necesidades de ejecución que no sean de tiempo real entonces el procesador puede llegar a ser utilizado al 100%.

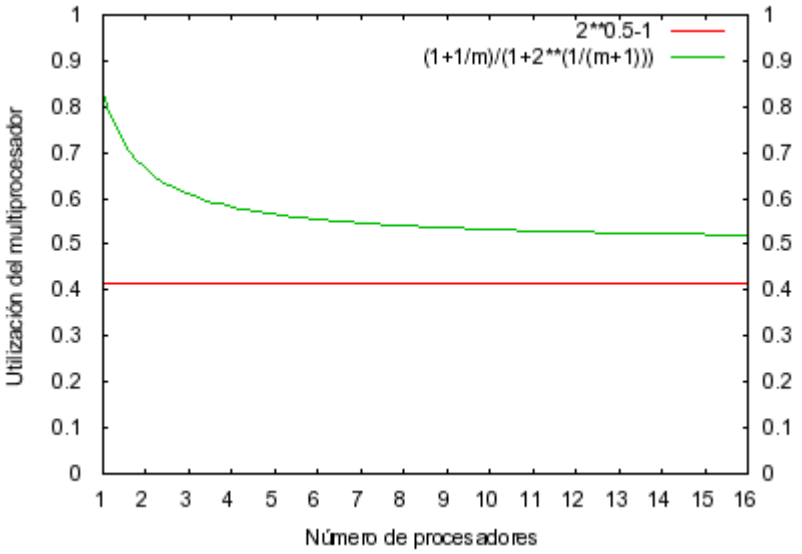


Figura 2.6: Límites de utilización para RMFF según [OhB98] divididos por *m*. En verde se representa el límite superior y en rojo el inferior.

En [SVC98] se propone una idea interesante, aunque costosa, que consiste en utilizar una combinación de varias de las heurísticas existentes para aumentar el grado de utilización del sistema. Se proponen dos métodos, SFS-DM y SFS-EDF, basados en la búsqueda secuencial hacia delante entre un conjunto determinado de ocho algoritmos de empaquetado.

2.3.2 Búsquedas guiadas.

En la familia de algoritmos de búsquedas guiadas el método más utilizado es el *Branch-and-Bound*. Estos algoritmos utilizan un árbol para representar todas las posibles asignaciones de tareas a procesadores. Un nodo en el árbol de búsqueda es una asignación parcial o completa de tareas a procesadores. En el vértice raíz no hay ninguna tarea asignada, mientras que en las hojas todas las tareas están asignadas. En cada nivel del árbol de búsqueda hay un conjunto de tareas listas para planificar. Los nodos hijos se generan asignando una de las tareas todavía no asignadas a un procesador y planificándola. Para verificar si la ramificación hija puede llevar a una solución viable se utiliza un test de planificación. El objetivo de la búsqueda es encontrar una hoja que optimice un cierto coste.

El problema de estos métodos es su complejidad: $O(n!\ m^n)$, donde n es el número de tareas y m el número de procesadores. Para reducirla existen diversas heurísticas. Se puede utilizar la poda (no explorar ramas que solo llevarían a soluciones peores), explotar simetrías en conjuntos de tareas y procesadores, e incluso se puede explorar solo ciertos subconjuntos en busca de una solución sub-óptima.

Algunos ejemplos de los criterios de optimización de esta clase de algoritmos son los siguientes. En [PSA97] se minimiza el máximo tiempo de respuesta normalizado en sistemas distribuidos. En [HoS94] se maximiza la probabilidad que se satisfagan todos los plazos aún en el caso de fallos en alguno de sus componentes. En [Xu93] se minimiza la máxima tardanza de las tareas. En [Ram95] se hace una optimización local minimizando la comunicación y minimizando el tiempo de finalizado de las tareas.

2.3.3 Búsquedas no guiadas.

Las búsquedas no guiadas empiezan la búsqueda a partir de una asignación inicial de tareas a procesadores. En cada iteración del algoritmo se examinan diversas alternativas de cambiar la asignación actual de forma más o menos aleatoria. Un test eficiente de viabilidad de la solución es utilizado para verificar si una alternativa es válida.

En [TBW92] Tindell, Burns y Wellings aplicaron la técnica del *Simulated Annealing* a multiprocesadores en sistemas de tiempo real. Para generar las distintas alternativas se mueve una tarea aleatoria a un procesador aleatorio. La función de energía consiste en la suma ponderada de valores como el número de tareas que no cumplen sus plazos, el número de procesadores con una utilización de memoria demasiado alta, la utilización total del sistema de comunicaciones, etc. Un ejemplo más reciente de utilización de la técnica s*imulated annealing* es [DiS00] donde el criterio es minimizar la máxima varianza del tiempo de respuesta de las tareas periódicas.

Otro tipo de algoritmos en esta categoría son los *Genetic Algorithms*. Basados en la teoría de la evolución en la que sobrevive el más fuerte, las soluciones a un problema se llaman individuos, los cuales forman una población. Un par de individuos pueden engendrar un hijo. Este nuevo hijo obtiene genes (tareas) de ambos padres, genes que pueden mutar (tareas que son movidas de un procesador a otro).

2.4 Asignación dinámica de tareas: planificación global

La planificación global de un sistema procesador tiene el objetivo primordial de obtener un mayor rendimiento de la capacidad de proceso del sistema. Esto se consigue gracias a tener una mayor distribución temporal de la carga entre los procesadores: a diferencia del sistema del particionado, con la planificación global se evita tener un procesador muy ocupado, con una cola de tareas pendientes larga, mientras que otros procesadores están ociosos. De esta forma se consigue adelantar trabajo, manteniendo así el sistema más disponible para atender nuevas demandas.

Al igual que en el caso de los monoprocesadores, la asignación de prioridades puede ser estática (FPS) o dinámica (DPS). Sin embargo, FPS pierde su gran ventaja, el determinismo, ya que aunque las prioridades sean fijas si sobreviene cualquier evento no tenido en cuenta, como puede ser una simple llegada de una tarea aperiódica, puede cambiar toda la asignación de tareas periódicas a procesadores, llegando incluso a comprometer la planificabilidad del sistema. Por otro lado, la asignación dinámica de prioridades busca una mayor flexibilidad, acorde con el objetivo de la planificación global.

En los próximos sub-apartados se resumen los principales métodos de planificación globales. Al final del apartado, en la página 74, la Tabla 10 sumariza sus principales características.

2.4.1 Planificadores con asignación estática de prioridades

La asignación de prioridades fija ordena según algún criterio el conjunto de todas las tareas del sistema y les asigna una prioridad decreciente. Esta prioridad es global y se mantiene durante toda la ejecución de cada una de las activaciones de una tarea. Como se verá en el próximo subapartado, estos métodos pueden tener problemas con la distribución de la carga en determinados casos, cuando hay tareas 'grandes'[7] y a estas se les asigna una prioridad errónea. Así, la mayoría de algoritmos de esta familia serán variantes del *Rate-Monotonic* con excepciones para el caso de tareas 'grandes'.

[7] Cada algoritmo definirá a partir de que factor de utilización una tarea se considera como grande.

2.4.1.1 Rate-Monotonic Global (GRM)

Este algoritmo consiste en ordenar todas las tareas del sistema de menor a mayor periodo y asignarles una prioridad decreciente. En tiempo de ejecución las tareas activas se encolarán en una única cola global ordenada según la prioridad asignada y el planificador global las irá extrayendo conforme haya procesadores libres. Desafortunadamente este esquema tiene un límite de utilización igual a cero. Esto se puede observar en el siguiente ejemplo [Dha77, DL78], donde se produce el llamado **Efecto Dhall** (Figura 2.7). Se planifica un conjunto de m+1 tareas en m procesadores con GRM. Las m primeras tareas tienen un periodo T_i=1 y un peor cómputo C_i=2ε (siendo ε pequeño), mientras que la última tiene un periodo T_{m+1}=1+ε y un C_{m+1}=1. Si todas las tareas se activan cuando t=0, entonces se ejecutan las m primeras, puesto que tienen un periodo menor. La última tarea terminaría en t=2ε+1, perdiendo su plazo (T_{m+1}= D_{m+1}=1+ε). Si $m\to\infty$ y $\varepsilon\to 0$ la utilización del sistema tiende a cero ($U(\tau)=m2\varepsilon/1 + 1/(1+\varepsilon)$ si $\varepsilon\to 0$ entonces $U(\tau)=1$ y $U=1/m$) y ¡no es planificable!. Obsérvese que este ejemplo también se aplica a DM y a EDF globales. En general, todos los algoritmos que no tienen en cuenta la carga máxima de las tareas individuales pueden sufrir el efecto Dhall.

Tarea	Periodo	Plazo	WCET
τ_i, i ≤m	1	1	2 ε
τ_{m+1}	1+ ε	1+ ε	1

$$\sum_{i=1}^{m+1} U_i = m\frac{2\varepsilon}{1}+\frac{1}{1+\varepsilon} \longrightarrow 1 \;\; si \;\; \varepsilon \longrightarrow 0$$

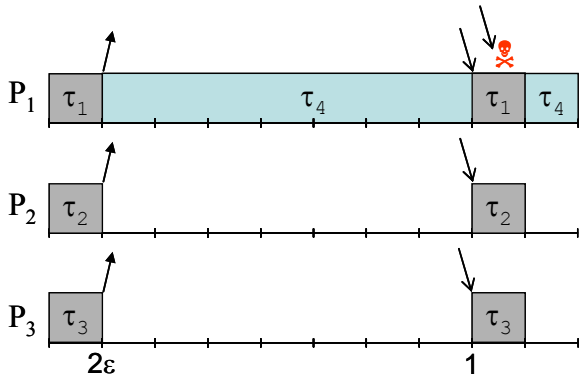


Figura 2.7: Efecto Dhall: un conjunto de tareas con una utilización muy baja no es planificable con RM, DM ni EDF en un multiprocesador.

El ejemplo anterior es un caso extremo: m tareas muy pequeñas y una de muy grande con un periodo ligeramente mayor. En la práctica un planificador global puede planificar conjuntos de tareas con mayores utilizaciones. Lo que muestra el efecto Dhall es que en multiprocesadores no se puede utilizar llana y simplemente un test de planificabilidad basado en la carga del sistema, como se hace en RM, DM y EDF. En

estos casos hay que recurrir a la simulación de todo un hiperperiodo para comprobar si el sistema es planificable.

En el artículo [LMM98] se establece la siguiente condición de planificabilidad. Si un conjunto de *n* tareas periódicas con el plazo igual al periodo puede planificarse con GRM en *m* procesadores, entonces una nueva tarea puede planificarse si se cumple la ecuación

$$C_{n+1} \leq \left(mT_{n+1} - \sum_{j=1}^{n} \left(\left\lfloor \frac{T_{n+1}}{T_j} \right\rfloor + 2 \right) C_j \right) \Big/ m \qquad \text{(ecuación 2.10)}$$

Según esta ecuación, el tiempo de cálculo de la nueva tarea debe ser menor que el tiempo disponible menos el tiempo requerido por las tareas de mayor prioridad. Ahora bien, este tiempo disponible se puede repartir entre los procesadores de una forma u otra. La idea es que el peor caso se da cuando el tiempo disponible se solapa completamente, ya que una tarea no puede ejecutarse en dos procesadores simultáneamente. Aplicando esta prueba de planificabilidad al ejemplo de la Figura 2.7 se rehusaría el conjunto de tareas, puesto que $C_{m+1}=1 > (m(1+\varepsilon)–m(1+2)2\varepsilon)/m = 1+\varepsilon - 6\varepsilon = 1-5\varepsilon$. Esta condición es suficiente pero no necesaria. En el ejemplo aceptaría $C_{m+1} \leq 1-5\varepsilon$ y sabemos que se puede planificar hasta $C_{m+1} \leq 1-2\varepsilon$.

2.4.1.2 AdaptiveTkC

Anderson y Jonsson en [AnJ00] proponen un algoritmo de planificación global con una asignación de prioridades distinta a GRM. Este método evita el efecto Dhall teniendo en cuenta la laxitud de las tareas ($T_i - C_i$) y el número de procesadores (*m*). Así se pueden tomar en consideración las tareas 'grandes'. En este método las prioridades se asignan según $T_i - k\, C_i$ creciente, con

$$k = \frac{1}{2} \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{m} \qquad \text{(ecuación 2.11)}$$

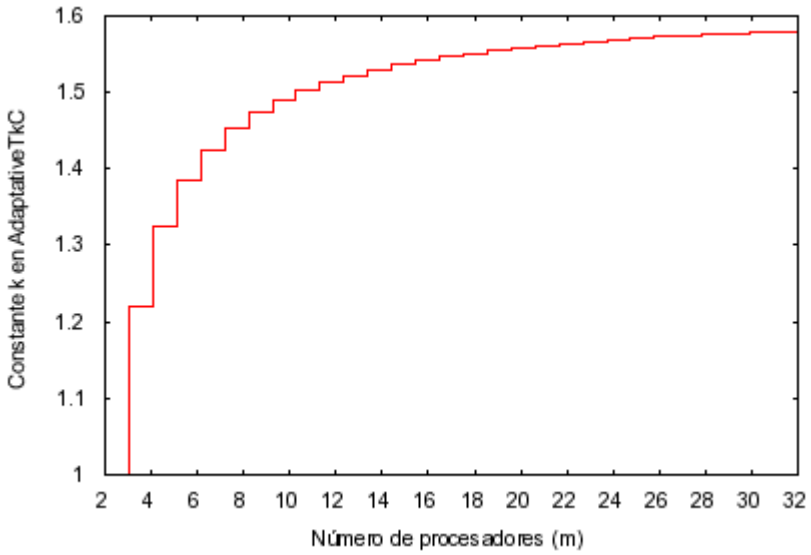


Figura 2.8: Constante *k* en AdaptiveTkC (ecuación 2.11) cuando varía el número de procesadores

Para sistemas con un único procesador, *k* es cero y sería equivalente a RM. Para dos procesadores k=1 y la prioridad es inversamente proporcional a la laxitud. Para más procesadores se da un poco más de importancia al cómputo, es decir, a la carga de las tareas. Cuando $m \to \infty$ entonces $k \to 1/2\ (1+\sqrt{5}) \approx 1{,}618$ (véase la Figura 2.8). Esto supone que el método es aplicable hasta tareas tan grandes como $U_{max} \leq 1/1{,}618 = 61{,}8\%$.

Obsérvese que este método no se ve afectado por el efecto Dhall: la tarea τ_4 tendría la mayor prioridad ($T_m - k\ C_m = 1+\varepsilon - k1$) que es menor que la del resto de tareas ($T_i - k\ C_i = 1 - k2\varepsilon$) puesto que $\varepsilon < k$ ($\varepsilon \to 0$ y $k \in [1..1{,}618]$). Sin embargo, sus autores indican que se debe recurrir a la simulación para verificar si un conjunto de tareas concreto es planificable cuando su carga es superior al 38%.

Existen métodos de asignación de prioridades estáticas que pueden utilizar test de planificabilidad basados en la utilización. Veamos algunos casos.

2.4.1.3 Algoritmo RM-US(m/(3m-2))

El algoritmo el *Rate-Monotonic Utilization-Separation* [ABJ01], utiliza GRM parcialmente. Da una prioridad máxima a las tareas con gran carga, evitando el efecto Dhall, mientras que al resto de tareas las trata con RM. Garantiza todos los

plazos para cualquier sistema de tareas τ que cumpla la condición suficiente: $U(\tau) \leq m^2/(3m-2)$.

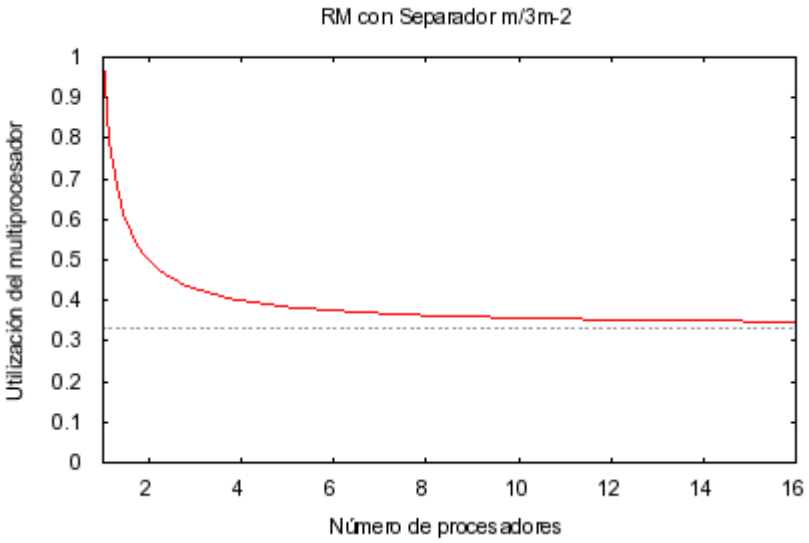


Figura 2.9: Límite teórico de la utilización del procesador con RM-US(m/(3m-2)). El separador de RM-US tiende al límite 1/3 cuando el número de procesadores crece. Para cuatro procesadores la carga máxima de una tarea es del 40%.

Se dice que este algoritmo tiene un separador m/(3m-2). Así, a una tarea τ_i se le asigna una prioridad GRM si $U_i \leq m/(3m-2)$. Para las tareas que no lo cumplan, se les asigna la prioridad máxima, desempatando igualdades de forma arbitraria. Así consigue que las tareas 'grandes' (aproximadamente $U_i > 1/3$) cumplan con su plazo, evitando el efecto Dhall. Sin embargo, con esta metodología de planificación se consiguen utilizaciones del sistema muy bajas. El porcentaje de utilización del sistema multiprocesador es la utilización total dividida por el número de procesadores, *m*, en este caso U=m/(3m-2) (véase la Figura 2.9). El límite cuando $m \to \infty$ es 1/3. Esto implica que este método garantiza plazos solo para utilizaciones excesivamente bajas.

2.4.2 Planificadores con asignación dinámica de prioridades

Los algoritmos de asignación de prioridades dinámicos para multiprocesadores se basan en el algoritmo EDF, el cual, como ya hemos mencionado, no es óptimo para estos sistemas. A pesar de ello, recientemente han aparecido numerosos estudios con EDF global que utilizan una prueba de planificabilidad basada en la carga.

2.4.2.1 Algoritmo EDF-US(m/(2m-1))

El algoritmo el *Earliest Deadline First Utilization-Separation* [SrB02] da una prioridad máxima (negativa, desempatando igualdades de forma arbitraria) a las tareas con gran carga, evitando el efecto Dhall, y las planifica junto al resto de tareas con EDF. Se dice que este algoritmo tiene un separador m/(2m-1). Garantiza todos los plazos para cualquier sistema de tareas τ que cumpla la condición suficiente: $U(\tau) \leq m^2/(2m-1)$. El porcentaje de utilización del sistema multiprocesador es U=m/(2m-1) y su límite cuando $m \to \infty$ es 1/2. Esto implica que este método garantiza los plazos para conjuntos de tareas que solo utilizan el 50% de la capacidad del multiprocesador.

2.4.2.2 El test GFB (Goossens, Funk y Baruah)

El test propuesto en [GFB02] establece que un conjunto de tareas τ es planificable con EDF en un multiprocesador con *m* procesadores de capacidad unitaria si

$$\sum_{\tau_i \in \tau} \lambda_i \leq m\,(1-\lambda_{\max}) + \lambda_{\max} \qquad \text{(ecuación 2.12)}$$

$$\text{con} \qquad \lambda_{\max} = \max_{\tau_i \in \tau} \{\lambda_i\}, \qquad \lambda_i = \frac{C_i}{D_i}$$

Obsérvese que este test rechazaría el ejemplo de la Figura 2.7, donde $\lambda_{max}= 1/(1+\varepsilon) \approx 1$. Sin embargo, para λ_{max} más comunes se obtienen cargas totales razonables. Por ejemplo, $\lambda_{max} \leq 0{,}31$ se obtienen cargas de hasta ln(2) indistintamente del número de procesadores (véase Figura 2.10, donde se representa U_{max} aislada en la ecuación 2.12 después de fijar una carga *m* ln(2)).

En este test el límite del porcentaje de utilización total del sistema cuando $m \to \infty$ es $1-\lambda_{max}$. Es decir, si tenemos conjuntos de tareas con tan solo una tarea grande, el porcentaje de utilización que permite este test puede ser muy bajo. Sin embargo, para conjuntos de tareas pequeñas el test es muy efectivo.

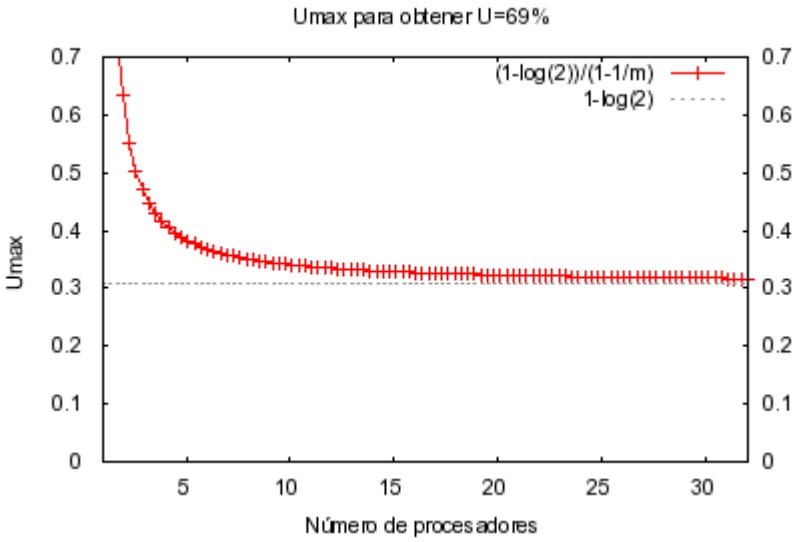


Figura 2.10: λ_{max} o U_{max} para obtener utilizaciones totales parecidas a las de RM en monoprocesadores (Ln(2)).

2.4.2.3 El test Baker

El test propuesto en [Bak03] establece que un conjunto de tareas τ compuesto de *n* tareas es planificable con EDF en un multiprocesador con *m* procesadores de capacidad unitaria si

$$\forall \tau_k : \sum_{i=1}^{n} \min\{1, \beta_i\} \leq m\,(1-\lambda_k) + \lambda_k \qquad \text{(ecuación 2.13)}$$

$$\text{con} \quad \beta_i \begin{cases} U_i\left(1+\dfrac{T_i - D_i}{D_k}\right) & si\ \lambda_k \geq U_i \\ U_i\left(1+\dfrac{T_i - D_i}{D_k}\right) + \dfrac{C_i - \lambda_k T_i}{D_k} & si\ \lambda_k < U_i \end{cases}$$

Según [BCL05] este test es algo más efectivo que el anterior cuando hay alguna tarea grande pero va peor cuando todas son pequeñas. En todo caso, ninguno de los dos domina al otro. Además, ambos tienen una tasa de conjuntos de tareas planificables muy baja cuando existen una o varias tareas pesadas (Ui>0,5). En este caso proponen una mejora pero, según los propios autores, todavía queda un gran margen para la mejoría.

2.4.2.4 Algoritmo EDF$^{(k)}$

Este algoritmo fue propuesto por Goossens, Funk y Baruah en [GFB03]. Para evitar el efecto Dhall este algoritmo ordena decrecientemente las tareas según su factor de utilización y asigna la prioridad más alta a las primeras k-1 tareas y para el resto utiliza EDF. En el ejemplo de la Figura 2.7 se puede fijar k=2, ejecutando primero la tarea m+1 en un procesador y el resto, con una carga muy pequeña, en los m-1 procesadores restantes. En realidad eso da lugar a una familia de algoritmos según el valor asignado al parámetro k. Un caso particular es el algoritmo PriD, el cual detallamos en el próximo apartado.

2.4.2.5 Algoritmo PriD

Este algoritmo es una variación del algoritmo EDF$^{(k)}$ y también se publicó en [GFB03]. Ordena las tareas por utilización decreciente, calcula el valor de k que minimiza el número de procesadores ($k_{min(\tau)}$ y $m_{min}(\tau)$ respectivamente) y planifica con el algoritmo EDF($k_{min(\tau)}$).

$$m_{\min}(\tau) \overset{def}{=} \min_{k=1}^{n} \left\{ (k-1) + \left\lceil \frac{U(\tau^{(k+1)})}{1-u_k} \right\rceil \right\} \quad \text{(ecuación 2.14)}$$

2.4.2.6 Planificación Proportionate-Fair

La planificación *Proportionate-Fair* o *pfair* [BCP96] se basa en la idea de ejecutar las tareas de forma proporcional y justa usando fracciones de tiempo o *quantums*. El tiempo de ejecución dedicado a una determinada tarea durante un periodo de tiempo es proporcional al peso de esta tarea (C_i/T_i). Es una planificación con desalojo, pero el desalojo solo se puede dar en determinados instantes de tiempo, al final de un quantum. Esto significa que el tiempo dedicado a una tarea un intervalo de tiempo [0,t) es aproximadamente proporcional a su peso ($t(C_i/T_i)$), con un error máximo del orden de un quantum. El algoritmo minimizará el máximo error cometido (*lag*), expresado según la siguiente fórmula:

$$lag(\tau_i,t) = t(C_i/T_i) - dedicado(\tau_i,t) \quad \text{(ecuación 2.15)}$$

Si la tarea τ_i se ejecuta, $lag(\tau_i,t+1)$ se decrementa en $1-(C_i/T_i)$, mientras que si no se ejecuta se incrementa en (C_i/T_i).

Se dice que un algoritmo es *pfair* si y solo si para toda τ_i y para todo t se cumple que

$$-1 < lag(\tau_i,t) < 1 \qquad \text{(ecuación 2.16)}$$

Los algoritmos *pfair* utilizan prioridades dinámicas para minimizar en todo momento este error, ejecutando las *m* tareas de mayor prioridad. El siguiente esquema describe el algoritmo PF [BCP96] y es *pfair*:

- Ejecutar todas las tarea urgentes, es decir las tareas τ_i tales que $lag(\tau_i,t) >0$ y $lag(\tau_i,t+1) \geq 0$ si la tarea se ejecutase.
- No ejecutar las tareas *tnegru*, es decir, aquellas tareas τ_i tales que $lag(\tau_i,t) <0$ y $lag(\tau_i,t+1) \leq 0$ si la tarea no se ejecutase.
- Para el resto de tareas, ejecutar la tarea que tiene el menor $t'>t$ tal que $lag(\tau_i,t') >0$.

La condición para que un conjunto de tareas sea planificable con este algoritmo es que la carga no supere la capacidad de proceso del multiprocesador. En otras palabras, es un algoritmo óptimo. El mayor inconveniente es un elevado número de apropiaciones si el quantum es pequeño o bien un error mayor y una mala utilización de la cpu cuando el quantum es grande. Además, el algoritmo para recalcular las prioridades, a pesar de ser ejecutado en tiempo polinómico, debe ejecutarse a cada quantum.

Dentro de esta categoría existen diversos algoritmos, diferenciados básicamente por la forma en que se desempata la igualdad de prioridades. El algoritmo PD [BGP95] parte las tareas en subtareas, cada una de las cuales tiene un *pseudo-release-time* y un *pseudo-deadline*, los cuales delimitan ventanas de ejecución, y utiliza diversos parámetros para desempatar la igualdad de prioridades (por ejemplo, los *pseudos-deadlines*). Existen extensiones para soportar tareas esporádicas ([AnS00] y [SrA02])

y diseños con servidores para tareas aperiódicas con plazo [SHA02], todos ellos óptimos.

Los principales inconvenientes de este tipo de planificadores son: (i) requieren la sincronización a nivel de quantum; (ii) parte del quantum puede quedar desaprovechada porque las tareas terminan antes; (iii) producen un alto número de cambios de contexto; (iv) pueden producir un gran número de migraciones y (v) las decisiones de planificación se agrupan al inicio del quantum.

2.4.2.7 Multiprocessor Total Bandwidth

En el artículo [BaL04a] se describe un método *Multiprocessor Total Bandwidth* (M-TBS) basado en EDF para multiprocesador para utilizar el tiempo remanente de la ejecución de tareas periódicas de tiempo real estricto para ejecutar otras tareas aperiódicas. Para ello asignan plazos a las tareas aperiódicas para poderlas ejecutar con EDF según la siguiente fórmula:

$$\max\left(D_{i-1}, \frac{mE_i + \sum_{\tau_i \in \tau} P_i^{(\mathrm{ub})} U_i(1-U_i) + E_R(A_i)}{m - U_{\mathrm{sum}}(\tau)} + P_{\max}(\tau)\right) \qquad \text{(ecuación 2.17)}$$

Tiene la ventaja de incorporar tareas aperiódicas con EDF en una plataforma multiprocesador pero tiene el inconveniente que su ejecución es equivalente a una ejecución en segundo plano. La diferencia estriba en que esta fórmula proporciona un método de control de admisión para las tareas aperiódicas con plazo o las tareas esporádicas.

2.4.2.8 Multiprocessor Constant Bandwidth Servers

En [BGL02] se describe una implementación de servidores de ancho de banda constantes para multiprocesadores que basan el dimensionado de los servidores (para tareas periódicas) en la ecuación 2.14 ([GFB03]) y que obtiene un nivel de utilización de los procesadores equivalente a [SrB02], es decir, cercana al 50%. En [BaL04b] se extendió el modelo para ejecutar tareas aperiódicas basándose en la ecuación 2.17 ([Bal04a]) mejorándola en el sentido que asigna plazos más cortos a las tareas aperiódicas pero con un coste computacional grande.

Asignación de prioridades	Planificador	Método de validación	Límite de Carga	Efecto Dhall
Estática	GRM	Simulación	0	sí
	AdaptiveTkC	Simulación	Variable	Evita, no siempre
	RM-US(m/3m-2)	Test	1/3	Evita porque solo cargas bajas
Dinámica	EDF-US(m/2m-1)	Test	1/2	Evita porque solo cargas bajas
	GFB	Test	$1-\lambda_{max}$	Evita si cargas bajas
	Baker	n Tests	n/a	n/a
	$EDF^{(k)}$		n/a	Evita, según k
	PriD		n/a	Evita
	M-TBS	Test GFB	$1-\lambda_{max}$	Evita si cargas bajas
	M-CBS	Test GFB	$1-\lambda_{max}$	Evita si cargas bajas
	Pfair	$U \leq 1$	100%	Evita

Tabla 10: Resumen de los principales algoritmos de planificación globales para multiprocesadores en sistemas de tiempo real.

Es importante remarcar que existen conjuntos de tareas que no pasan las pruebas de planificabilidad anteriores y, sin embargo, son planificables. Para comprobar la planificabilidad en estos casos existen varias posibilidades. Cuando la carga periódica es estática se puede realizar una simulación de un hiperperiodo para comprobar si se cumplen los plazos. Cuando la carga es dinámica se puede recurrir a la elaboración de un cronograma de factibilidad en la llegada de cada nueva tarea periódica. Por ejemplo, este es el caso de los algoritmos *Latest Start Time* y *Deadline Ordering Over Multiprocesadors* [SVC96].

2.5 Conclusiones

La planificación de multiprocesadores tradicionalmente se ha venido realizando según el modelo particionado, con una planificación para monoprocesadores. El principal motivo es el determinismo. Si bien los algoritmos de empaquetado tienen unos límites en la utilización de los procesadores, esta capacidad de proceso sobrante puede ser fácilmente utilizada por los planificadores para servir eventos aperiódicos (por ejemplo con *Deferrable Server*, *Slack Stealing*, etc.). A pesar de esto, detectamos en la literatura una falta de estudios que discriminen que tipo de distribución de tareas aperiódicas es más conveniente.

Algunos estudios se han dedicado a comparar la planificación particionada con la planificación global (por ejemplo [LMM98] y [AnJ00b]). A partir de ellos, la teoría de los planificadores globales ha experimentado un notable avance en los últimos años ([AnJ00], [ABJ01], [SrB02], [GFB02], [Bak03], [GFB03], [BaL04a], [BaL04b], [BCL05], [Bak05]). Esta teoría ha mostrado que los límites de la utilización del sistema multiprocesador con tareas periódicas son muy bajos, principalmente debido a una serie de anomalías detectadas. Aún así, no se ha hallado bibliografía que permita la planificación global conjunta de tareas periódicas y aperiódicas, con la que se podría utilizar la capacidad de proceso sobrante del sistema multiprocesador.

En conclusión, dados estos antecedentes, en esta tesis se pretende sobrepasar estas limitaciones de utilización de los procesadores con las tareas periódicas y, en el caso de quedar capacidad sobrante, permitir la ejecución juntamente con tareas aperiódicas para conseguir explotar toda la capacidad de proceso. Así en el próximo capítulo se exploran diversas posibilidades, sobretodo la planificación particionada de las tareas periódicas con la planificación dinámica de las tareas aperiódicas, mientras que en el resto de capítulos se explora la planificación global de tareas periódicas y aperiódicas.

Capítulo 3

DISTRIBUCIÓN DE LAS TAREAS

En este capítulo se describen varios estudios realizados antes de profundizar en la planificación global de multiprocesadores. Estos estudios sirvieron para sentar las bases, establecer una metodología y para disponer de unos puntos de referencia para las comparativas en los experimentos de rendimiento de los capítulos posteriores.

3.1 Introducción

En el capítulo anterior se ha resumido el estado del arte de la planificación de sistemas de tiempo real con multiprocesadores y se ha observado que la literatura se ha centrado principalmente en la distribución de tareas periódicas pero hay pocas referencias que traten el problema de la planificación conjunta con tareas aperiódicas y de su distribución. Así, como primera parte de la investigación realizamos unos estudios con algoritmos de planificación sencillos con el objetivo de comprobar si la ejecución de las tareas de forma dinámica, es decir, sin una asignación estática previa entre los procesadores, puede ser ventajosa para el servicio a las tareas aperiódicas. Cabe remarcar que, en esta fase, no se pretendía obtener resultados óptimos ni tener pruebas de planificabilidad exactas, aunque sí razonables o aproximadas.

En el primer apartado analizamos las distintas posibilidades de distribución de tareas periódicas y aperiódicas ya sea de forma estática o dinámica. En el segundo apartado consideramos con mayor profundidad la distribución dinámica de las tareas aperiódicas con el objetivo de establecer una referencia comparativa en los capítulos siguientes. Finalmente, en el último apartado se resumen las conclusiones resultantes de este capítulo. Como resultado de este capítulo se publicaron los artículos [BML99] y [BAL02] para cada apartado respectivamente.

3.2 Las distintas posibilidades de distribución de tareas on-line

En este estudio vamos a distinguir entre dos tipos de tareas: las periódicas y las aperiódicas sin plazo. La distribución de las tareas entre los distintos procesadores podrá efectuarse de forma estática[8] en tiempo de diseño o bien de forma dinámica en tiempo de ejecución. Con la distribución dinámica una activación de una tarea no quedará asignada permanentemente a un procesador, sino que podrá ser expulsada de uno y pasar a ejecutarse en otro más adelante, cuando haya alguno disponible. Esto da lugar a cuatro posibles combinaciones de planificadores:

1. PF/AF: tareas periódicas y aperiódicas fijas[9], es decir, todas las tareas distribuidas estáticamente en tiempo de diseño entre los procesadores,

2. PF/AD: tareas periódicas fijas y las aperiódicas dinámicas,

3. PD/AF: tareas periódicas dinámicas y las aperiódicas fijas,

4. PD/AD: tareas periódicas dinámicas y aperiódicas dinámicas, es decir, cualquier tarea podrá ser ejecutada en cualquier procesador.

A priori se esperaban buenos tiempos de respuesta medios para las tareas aperiódicas en el segundo caso (PF/AD) pero sobretodo en el cuarto (PD/AD, donde todas las tareas son dinámicas) puesto que las tareas deberían tener mayores opciones de disponer de tiempo de proceso en algún procesador. El caso más desfavorable se esperaba que fuese el primero (PF/AF, todas fijas) por su falta de flexibilidad o de adaptación a la demanda de servicio. Por completitud también se ha añadió el tercer caso (PD/AF), donde las aperiódicas deben ejecutarse en un procesador concreto mientras que las periódicas lo pueden hacer en cualquiera. Este caso puede parecer que no tenga demasiado sentido, desde el punto de vista del servicio ágil a las tareas aperiódicas, pero lo puede tener en casos específicos, como por ejemplo cuando las tareas aperiódicas estén relacionadas con un procesador o un hardware concreto.

[8] Utilizaremos los terminos “distribución estática” y “distribución fija” indistintamente.
[9] En tiempo de diseño se especificará en que procesador debe ejecutarse cada tarea aperiódica.

En el momento en que se realizó este estudio, los autores no conocíamos algoritmos capaces de tratar las tareas periódicas de forma dinámica, así que elegimos hacer algunas adaptaciones a algoritmos para monoprocesadores existentes. En concreto, se utilizo un planificador *Rate-Monotonic* con servidores para las tareas aperiódicas. En los capítulos posteriores de esta tesis se estudiará más a fondo la planificación dinámica, con nuevas propuestas y los nuevos algoritmos que han ido apareciendo en la literatura a lo largo de los últimos años.

3.2.1 Planificación con servidores

El algoritmo de planificación elegido se basa en el algoritmo clásico para monoprocesadores: el *Rate Monotonic Scheduling Algorithm* (RMA) [LiL73], donde las tareas periódicas tienen una prioridad inversamente proporcional a su periodo. Este algoritmo es sencillo desde el punto de vista de planificación: solo hace falta asignar las prioridades y ejecutar siempre la tarea con mayor prioridad. La ejecución de tareas aperiódicas la realizamos mediante servidores, los cuales son tratados como una tarea periódica más y, por lo tanto, se tienen en cuenta en la prueba de planificabilidad del sistema. El dimensionado de estos servidores puede ser una parte compleja del diseño, puesto que los métodos existentes en el momento solo eran válidos para sistemas monoprocesador. Así que los servidores utilizados fueron una variante del *Polling Server* (PS) [LSS87,SSL89], unos de los más simples. Los esquemas correspondientes a cada tipo de distribución se muestran en la Figura 3.1. Una posible alternativa interesante fue la de usar servidores con mayores prestaciones, capaces de preservar su capacidad, como por ejemplo el *Deferrable Server* [LSS87,SLS95], el *Priority Exchange* [LSS87] o el *Sporadic Server* [SSL89], pero surgieron serios interrogantes. Por un lado la complejidad de la implementación es mayor y podía aumentar considerablemente en su uso con multiprocesadores. Por otro lado, el dimensionado de los servidores garantizaba la planificabilidad en monoprocesadores pero no en multiprocesadores. Además, si bien estos servidores son capaces de preservar su capacidad, la capacidad máxima que se les puede asignar es inferior a la del PS. Finalmente, existen algoritmos de planificación de tareas aperiódicas que no se basan en servidores que ofrecen mayores prestaciones y que, a priori, su adaptación a la planificación global de multiprocesadores se suponía más

adecuada. Estas alternativas serán analizadas en el apartado 3.3 y en los capítulos posteriores. En consecuencia, se desestimó el utilizar estos servidores más complejos.

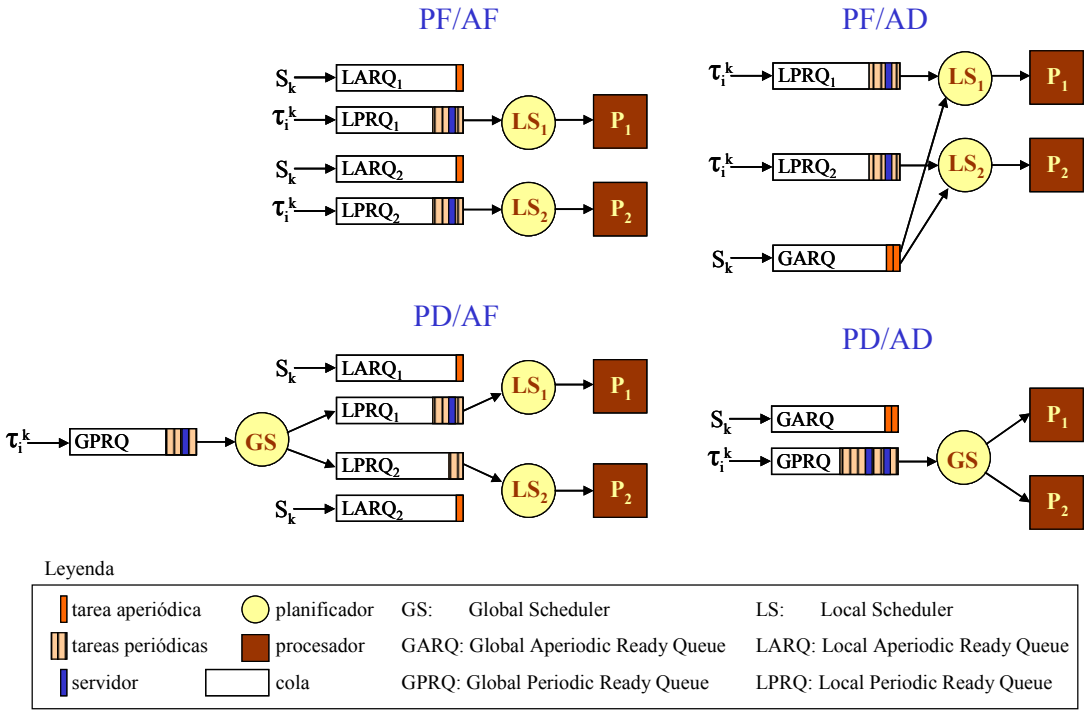


Figura 3.1: Esquemas generales de los planificadores y las colas necesarias para cada una de las cuatro modalidades de distribución analizadas.

Diseño de los servidores

Tal como ya hemos comentado los servidores utilizados son una variante del *Polling Server* (PS) [LSS87,SSL89]. En tiempo de diseño, se dimensionan tantos servidores como procesadores tenemos, calculando su periodo, su capacidad de ejecución y un desplazamiento inicial para evitar el solapamiento de sus activaciones. En tiempo de ejecución, al igual que PS, nuestros servidores no pueden conservar su capacidad: si en el momento de su activación no llega ninguna tarea aperiódica, el servidor sirve a la tarea periódica de mayor prioridad, perdiendo el servidor parte de su capacidad de servicio. A diferencia del PS, si se activa una tarea aperiódica cuando el servidor está sirviendo una tarea periódica, éste pasa a servir la tarea aperiódica y así no pierden su capacidad por completo. Adicionalmente, si por el dimensionado de los servidores en el momento de requerir un servicio a una tarea aperiódica no hay capacidad disponible, estas tareas también pueden ejecutarse en segundo plano.

En tiempo de diseño se debe dimensionar la capacidad de cada servidor. Como la teoría de la planificación global tiene limites de utilización muy bajos (véase el apartado 2.4) no deja margen para añadir los servidores y hemos recurrido a la teoría de la planificación local o particionada. Así, debemos distinguir los casos en que las tareas periódicas se distribuyen de forma estática o de forma dinámica.

Dimensionado de servidores: asignación estática de tareas periódicas (PF/*)

Como ya hemos indicado, el dimensionado de los servidores siempre es una tarea compleja. En los sistemas monoprocesador los recursos disponibles para las tareas aperiódicas son muy limitados, pero en los multiprocesadores las oportunidades de servirlas crecen. Así pues, en principio no pretendimos hacer un dimensionado de los servidores óptimo pero sí uno que nos valiese para hacer el estudio. Para ello, añadimos tantos servidores como procesadores. Como utilizamos RMA, para que los servidores tuviesen una prioridad alta se les asigno un periodo igual al mínimo del conjunto de tareas periódicas, o sea, el más frecuente. La capacidad del servidor se eligió de forma que la carga del procesador, incluida la del servidor, fuese menor que el máximo teórico para la planificación con RMA [LiL73] (ecuación 2.1). Aunque este dimensionado no será óptimo si que será fácil de calcular usando la siguiente ecuación para cada servidor de cada procesador:

$$U_{server} = \frac{C_{server}}{T_{\min}} \leq (n+1)\cdot\left(2^{\frac{1}{n+1}} - 1\right) - U_{per} \qquad \text{(ecuación 3.1)}$$

donde

U_{server}	la capacidad del servidor de aperiódicas en un procesador
T_{min}	el mínimo de los periodos de las tareas periódicas del procesador
n	el número de tareas periódicas asignadas al procesador
U_{per}	la utilización de las tareas periódicas asignadas al procesador

Es necesario destacar que para valores grandes de n la carga máxima teórica no puede pasar del ln(2), o sea del 69,3 %. Esto nos restringe a diseñar experimentos y simulaciones con una carga periódica inferior al 69%, para poder dimensionar un servidor con un mínimo de capacidad. Por ejemplo, un procesador con n=8 tareas

periódicas entonces el límite teórico de la utilización total es 72%. Así, si la utilización de estas tareas es U_{per}=65% entonces la parte derecha de la ecuación 3.1 fija el límite de la utilización del servidor $U_{server} \leq 7\%$. Si el periodo mínimo de las tareas asignadas al procesador es 100, entonces C_{server}=7 y T_{server}= 100.

Dimensionado de servidores: asignación dinámica de tareas periódicas (PD/*)

En los casos en que la ejecución de las tareas periódicas puede ser en cualquier procesador (PD/AF y PD/AD) las tareas periódicas no están asignadas a ningún procesador y deben ser consideradas de forma global. En este caso también utilizamos tantos servidores como procesadores pero su dimensionado se realizo de forma ligeramente distinta: se considera como si tuviéramos un único procesador pero *m* veces más potente, siendo *m* el número de procesadores, y modificando la forma de obtener la capacidad de cada servidor (U_{server}) de la siguiente forma:

$$U_{server} = \frac{C_{server}}{T_{\min}} \leq \frac{1}{m}\left((n+m)\cdot\left(2^{\frac{1}{n+m}} - 1\right) - U_{per} \right) \qquad \text{(ecuación 3.2)}$$

donde

U_{server}	la capacidad de cada uno de los servidores de aperiódicas
T_{min}	el mínimo de los periodos de **todas** las tareas periódicas
n	el número **total** de tareas periódicas del sistema
m	el número de procesadores (y el número de servidores)
U_{per}	la utilización máxima de **todas** las tareas periódicas del sistema

Un sistema multiprocesador con *m* procesadores no equivale a un sistema monoprocesador *m* veces más rápido, sino que usualmente es más lento[10]. Según esto la prueba de planificabilidad no es válida y, por lo tanto, el dimensionado de los servidores podría llegar a hacer el sistema no planificable, es decir, que los plazos de las tareas periódicas no están garantizados. Además, la comparativa no seria justa puesto que la capacidad total ofrecida por los servidores con un dimensionado global (ecuación 3.2) es inferior a la ofrecida si las tareas periódicas son tenidas en cuenta separadamente en cada procesador (ecuación 3.1). Esto es debido a que en la

[10] Lley de Amdahl: véase la observación en el segundo párrafo de la página 54

ecuación 3.2 el periodo de los servidores será menor o igual que en el caso anterior, lo cual puede ser una ventaja por tener mayor prioridad y frecuencia, pero también un inconveniente a la hora de elegir la capacidad de cálculo para que nos aproximemos al límite teórico, puesto que hemos utilizado números enteros. Por otro lado, el número total de tareas es mayor y por lo tanto el límite máximo teórico será menor.

Así, para que la comparativa fuese más equitativa, para el estudio PD/* finalmente utilizamos el siguiente método de dimensionado. Repartir "virtualmente" las *n* tareas periódicas entre los *m* procesadores y utilizamos la siguiente ecuación para dimensionar los servidores:

$$U_{server} = \frac{C_{server}}{T_{\min}} \leq \left(\frac{n}{m} + 1\right) \cdot \left(2^{\frac{1}{\frac{n}{m}+1}} - 1\right) - \frac{U_{per}}{m} \qquad \text{(ecuación 3.3)}$$

De nuevo, esta ecuación se basa en una prueba de planificabilidad para monoprocesadores y no hay garantías de cumplir los plazos de las tareas periódicas en una asignación dinámica en un multiprocesador. A pesar de esto, el método puede ser útil para poder realizar un estudio inicial. Por un lado, sabemos que empíricamente se ha corroborado que con RMA se pueden conseguir frecuentemente cargas planificables hasta el 88% en monoprocesadores [LSD89]. Por otro lado, como en todos los experimentos realizados para este estudio la carga periódica no ha superado el 70%, realmente en ningún caso se perdió ningún plazo.

Sabiendo que las cargas planificables pueden en realidad ser superiores, alternativamente hemos realizado un segundo método de dimensionado de servidores según el Algoritmo 3-1. A este método le llamamos dimensionado máximo porque utiliza la prueba de planificación exacta de Joseph y Pandya [JoP86], que tiene en cuenta las tareas periódicas concretas, mientras que el dimensionado anterior produce unes servidores mínimos ya que solo tiene en cuenta el número de tareas.

El Algoritmo 3-1 intenta dimensionar un servidor que ocupe toda la capacidad sobrante de las tareas periódicas pero mientras no se pase la prueba de planificabilidad se va reduciendo la capacidad del servidor. Así, al final se obtiene un servidor que no compromete los plazos de las demás tareas periódicas.

```
Algoritmo Dimensionar_Servidores_Maximos
1: para cada procesador p
2:    Userver_inicial= Carga_máxima_permitida – Uper/m
3:    Tserver_p= Tmin_p
4:    Cserver_p= Tserver_p x Userver_inicial
5:    mientras no test_planificabilidad_J&P(p) ; ver ecuación 2.2
6:          Cserver_p= Cserver_p – 1
7:    fin_mientras
8: fin_para
```

Algoritmo 3-1: Dimensionado de los servidores máximos

Con una carga máxima permitida del 100% este algoritmo obtiene un dimensionado máximo de los servidores de cada procesador según las tareas con las que va a interaccionar. Esto nos va a garantizar los plazos de las tareas periódicas, pero solo con las asignaciones estáticas (PF/*). Con las asignaciones dinámicas es probable que se incumplan plazos, pero, ¿cuantos? ¿cuan frecuentemente?. Para responder a estas preguntas se ha realizado el primer experimento del siguiente apartado y, en caso necesario, poder afinar el dimensionado de los servidores partiendo de un `Carga_maxima_permitida` inicial inferior.

3.2.1.1 Evaluación experimental del rendimiento

En este apartado se describen diversos experimentos de simulación de los planificadores utilizando los servidores descritos en el apartado anterior (los denominados Smin-PF/* usan la ecuación 3.1, los Smin-PD/* usan la ecuación 3.3 y los denominados Smax-* usan el Algoritmo 3-1) y utilizando los diferentes tipos de distribución de las tareas (-F ≡ PF/AF, -P ≡ PD/AF, -A ≡ PF/AD y -D ≡ PD/AD). El objetivo es comparar el rendimiento obtenido con los diversos servidores y ver que

tipo de distribución es la que proporciona el menor tiempo de respuesta a las tareas aperiódicas.

Para los experimentos de este apartado, si no se especifica lo contrario, se ha utilizado la siguiente parametrización. Se han generado al azar conjuntos de tareas periódicas para 4 procesadores según los parámetros de la tabla PCT # 1 (Anexo B)[11]. La carga periódica máxima por defecto se ha establecido en U_{per}=65%. Esta carga está casi en el límite teórico y los servidores quedan con poca capacidad. Para los sistemas con asignación dinámica se ha tenido que limitar el número total de tareas periódicas a menos de 32, para tener margen para los servidores mínimos (ecuación 3.3). Esta restricción en el número de tareas es aceptable en el marco de este estudio. En los experimentos que no especifican el factor máximo de utilización éste se ha fijado U_{max}=15%.

La métrica de rendimiento utilizada es la media aritmética de los tiempos de respuesta medios de las tareas aperiódicas (*T. Respuesta Aperiódicas,* TRA)[12]. Para poder comparar, independientemente de los tiempos de cálculo de las tareas aperiódicas el tiempo de respuesta medio de éstas se representa normalizado con respecto al peor tiempo de cálculo (R_i/C_i). Así, con esta métrica lo deseable es TRA→1. En cada simulación se ha medido el tiempo de respuesta medio normalizado de 100000 tareas aperiódicas. Para cada conjunto de tareas se han ido lanzando simulaciones hasta que el valor obtenido estaba dentro de un intervalo de confianza del 95% con una probabilidad del 5%. Para cada punto de las gráficas se han generado 1000 conjuntos de tareas sintéticos y se ha calculado la media aritmética de las muestras obtenidas.

En el primer experimento se determinará el valor inicial para la carga máxima permitida de los servidores máximos. En los subsiguientes experimentos se variarán las condiciones de las cargas para comprobar el TRA obtenido en un amplio rango de posibles aplicaciones: en el segundo se varía la carga periódica, en el tercer la carga aperiódica y en el cuarto el factor de utilización máximo de las tareas periódicas.

[11] Véase el apartado 1.5 en la página 34 para más detalles de la metodología
[12] Para abreviar, en las gráficas se denomina *T. Respuesta Aperiódicas* y en el texto *TRA*

Experimento 1

El objetivo de este primer experimento es identificar un valor inicial para `Carga_maxima_permitida` en el Algoritmo 3-1 que no dimensione los servidores de forma que provoque incumplimientos de plazos en las distribuciones PD/*. La carga periódica se ha fijado al 65% y la carga aperiódica al 20%. La carga máxima de una tarea periódica (U_{max}) se ha variado entre 15%, 20%, 25%, 30% y 35%. De esta forma se han probado tareas periódicas ligeras y tareas más pesadas para aumentar la probabilidad de pérdidas de plazos. La métrica utilizada es el porcentaje de pérdidas de plazos y se calcula dividiendo el número de conjuntos de tareas en las que alguna activación de una tarea periódica ha perdido su plazo entre el total de conjuntos generados.

En la Figura 3.2-(a) podemos observar que para `Carga_maxima_permitida= 85%` el porcentaje de conjuntos de tareas que incumplen algún plazo con una distribución totalmente dinámica es casi nulo. Solo para U_{max}=35% el porcentaje de pérdidas es del orden del 5%. Este porcentaje del 85% permitiría servir a las tareas aperiódicas ((U_{server} = 85% - 65% = 20% equivale a U_{aper}) sin recurrir a su ejecución en segundo plano. En la Figura 3.2-(b) vemos que cuando las tareas aperiódicas permanecen estáticas en un procesador el porcentaje de pérdidas es incluso menor. En este caso la carga máxima permitida podría ser del 88%. Cabe destacar que, cuando la carga máxima permitida es cercana al 100%, a pesar que las tareas periódicas serían planificables con una distribución estática, puesto que pasan J&P, con una distribución dinámica el porcentaje de conjuntos de tareas con perdidas de plazos es del 80% en un caso y del 70% en el otro (Figura 3.2 (a) y (b)).

En conclusión, no se puede utilizar los servidores dimensionados al máximo con distribuciones dinámicas. Un límite superior al valor máximo razonable para el experimento actual es del 85%.

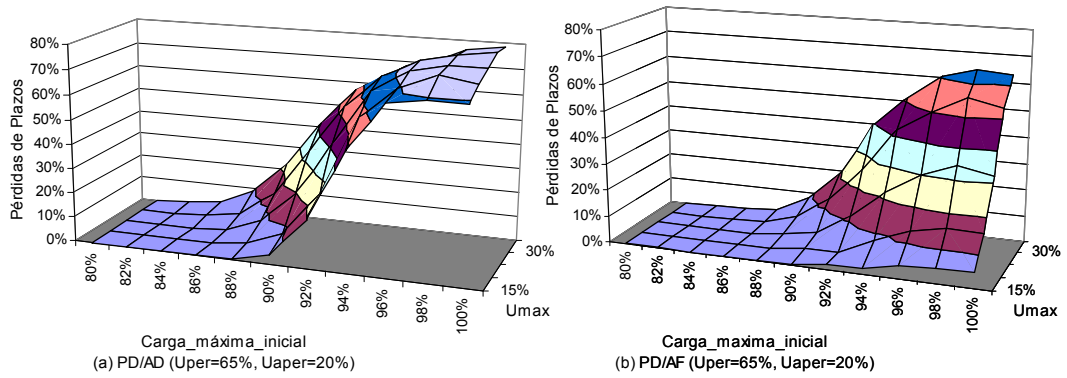


Figura 3.2: Porcentaje de conjuntos de tareas que provocan una pérdida de plazos si se usa el Algoritmo 3-1 para dimensionar los servidores, en función de la carga máxima por procesador permitida (eje x, con $U_{server} = x - U_{per}$) y el U_{max} del conjunto de tareas.

Así pues, a partir de los resultados de este primer experimento, para los experimentos que siguen hemos utilizado dos dimensionados de servidores: el dimensionado mínimo, usando la ecuación 3.3 (denominados Smin-*) y el dimensionado máximo, usando el Algoritmo 3-1 con la carga máxima permitida del 85% (denominados Smax-*).

Experimento 2

El objetivo de este experimento es observar el comportamiento de los distintos servidores cuando la carga periódica aumenta. Así, la carga periódica se ha variado desde el 45% hasta el 65% por procesador. La carga de les tareas aperiódicas se ha mantenido fija al 20% por procesador, pero la caracterización del tiempo entre llegadas (en el rango de los periodos) y el tiempo de cálculo se ha ido variando de forma aleatoria, siguiendo una distribución exponencial.

Los resultados obtenidos quedan reflejados en las Figura 3.3 (a) y (b). En ellas se puede observar que resulta bastante negativo tener las tareas aperiódicas fijas. Smin-F obtiene los tiempos de respuesta entre un 80% y un 90% peores que Smin-A mientras que Smax-F es entre un 76% y un 113% peor que Smax-A. Algo mejores son Smin-P y Smax-P: Smin-P es entre un 60% y un 67% peor que Smin-A y Smax-P es entre un 55% y un 79% peor que Smax-A. La flexibilidad de migrar tareas periódicas mejora un poco el tiempo de respuesta de las aperiódicas pero no lo suficiente. Las mejores distribuciones son las que nos esperábamos a priori, pero es

sorprendente como la mejor es PF/AD (Smin-A y Smax-A). La diferencias entre Smin-A/Smin-D o Smax-A/Smax-D están alrededor del 4%. No son muy significativas, pero siempre son favorables a Smin-A y a Smax-A.

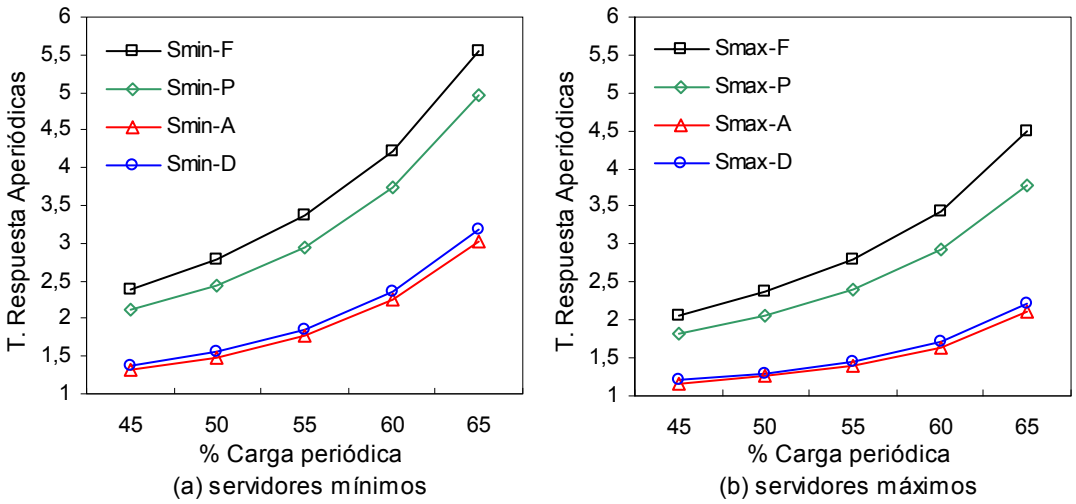


Figura 3.3: Comparativa de los tiempos de repuesta medios de las tareas aperiódicas obtenidos con las distintas posibilidades de asignación de tareas cuando aumenta la carga periódica y con una carga aperiódica del 20%

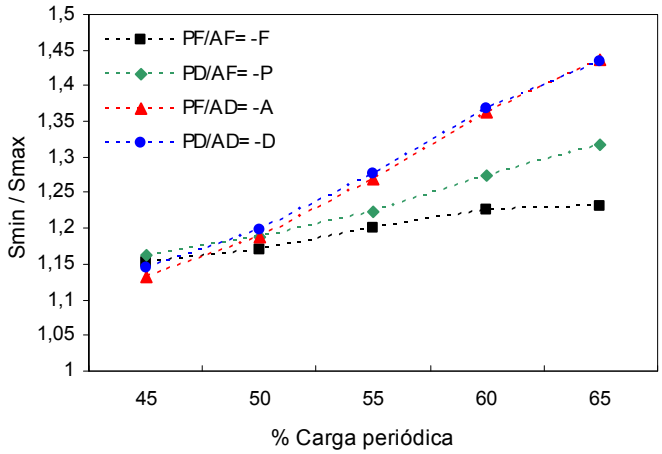


Figura 3.4: Comparativa de los tiempos de repuesta obtenidos con los servidores mínimos versus los servidores máximos (Smin/Smax) con las distintas posibilidades de asignación de tareas

En la Figura 3.4 se representa los TRA obtenidos con los servidores mínimos divididos por los obtenidos con los servidores máximos en los mismos conjuntos de tareas. En ella podemos observar que, en este experimento, los resultados con los servidores máximos son entre un 13% y un 44% mejores que los mínimos. Con la distribución totalmente estática la mejora es la menos notable (alrededor de un 20%) mientras que las distribuciones dinámicas son las que mejoran más. La mejoría es

más notable cuando la carga periódica es mayor. Es decir, cuando hay mayor flexibilidad se obtiene mejor partido de los recursos existentes y esto se nota más cuando los recursos son escasos.

En conclusión, la distribución estática de las tareas aperiódicas (AF) obtiene los peores resultados, llegando a doblar el mejor resultado cuando la carga periódica es más alta. La distribución PF/AD es la mejor pero con PD/AD se obtienen unos resultados similares. Con servidores de mayor capacidad se obtiene una mayor ganancia con las distribuciones dinámicas que con las estáticas, sobretodo cuando la carga periódica crece.

Experimento 3

El objetivo de este tercer experimento es observar el comportamiento de los distintos servidores cuando la carga aperiódica varía desde utilizaciones muy bajas hasta utilizaciones considerables. En la Figura 3.5 mostramos los resultados, obtenidos con la misma parametrización, pero manteniendo la carga periódica relativamente alta del 60% y variando la carga aperiódica desde el 5% hasta el 25% para ver como afecta este aumento de carga. La intersección con el experimento anterior esta cuando x=20 (U_{per}=60% y U_{aper}=20%). Las conclusiones obtenidas en el experimento anterior son extrapolables a este. Además, ahora podemos ver que las tendencias se mantienen cuando la carga aperiódica crece y las diferencias entre las distintas políticas de distribución y los dimensionados de los servidores se mantienen estables. La única diferencia destacable que hemos observado es que la diferencia entre los servidores mínimos y máximos se reduce para U_{aper}=25% y nos hace pensar que si esta carga aumentara más entonces la diferencia se reduciría más. Esto puede ser debido a que la carga es demasiado alta y, en consecuencia, las tareas aperiódicas se ejecutan en su mayor parte en segundo plano, ya que la capacidad de los servidores es pequeña.

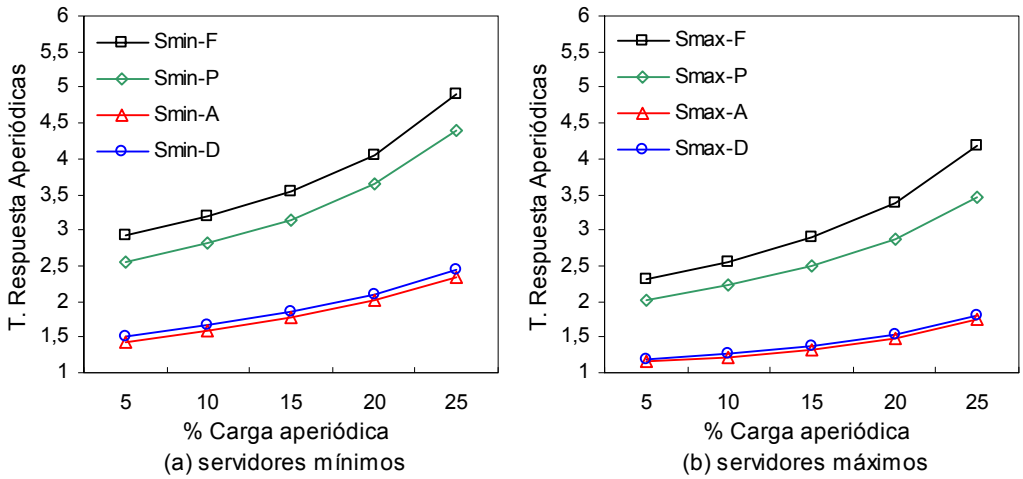


Figura 3.5: Comparativa de los tiempos de repuesta obtenidos con las distintas posibilidades de asignación de tareas cuando aumenta la carga aperiódica y una carga periódica del 60%

Experimento 4

El objetivo de este experimento es observar como afecta al tiempo de respuesta de las tareas aperiódicas el hecho de tener tareas periódicas mayores. Así, para este experimento usamos los mismos parámetros que para el segundo pero U_{max} se ha fijado al 30%. En la Figura 3.6 se muestra la división de los TRA obtenidos con U_{max}=30% entre los obtenidos con U_{max}=15%.

En la Figura 3.6-(a) podemos observar como el hecho de tener tareas periódicas más pesadas afecta negativamente a la distribuciones estáticas de tareas periódicas. Smin-F y Smin-P se sitúan por encima de la igualdad, empeorando entre el 2% y el 5%. En cambio con la distribuciones dinámicas los resultados son entre un 5% y un 8% mejores con U_{max} mayor. Esto es debido a que las tareas aperiódicas que se encuentran con una tarea periódica pesada la pueden eludir migrando a otro procesador. Cuando la carga periódica crece este efecto es más notable, es decir, obtenemos un tiempo de respuesta menor con tareas periódicas mayores (para Umax=30%). Esto significa que, en este caso, los tiempos de respuesta de las aperiódicas no crecen tanto cuando la carga periódica crece como lo hacía en los experimentos anteriores y esto puede ser debido a que el resto de tareas periódicas son menores (en promedio 5% vs. 8%, puesto que $(U_{per}-U_{max})/(n-1)$= (65%-30%)/7=5% y (65%-15%)/7=8%).

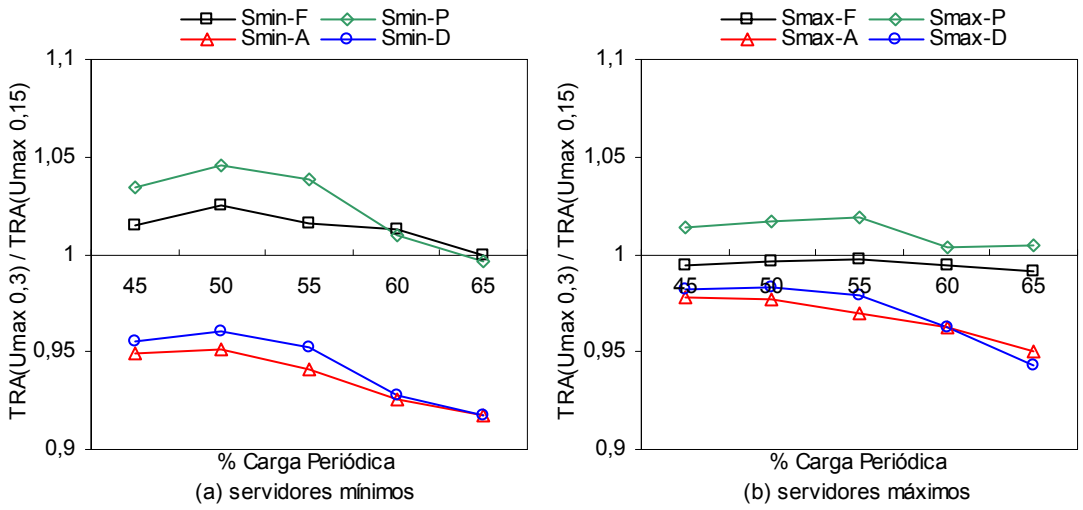


Figura 3.6: Comparativa de los resultados obtenidos con tareas periódicas con U_{max}=30% versus U_{max}=15%

Con los servidores mayores (véase la Figura 3.6-(b)) las diferencias entre las distintas distribuciones se reducen pero las tendencias son similares. Esto se debe a que al tener los servidores mayor capacidad hay más recursos para ejecutar tareas aperiódicas en cada procesador y se hace menos necesario migrar.

En conclusión, el hecho de tener tareas periódicas mayores, que a priori se espera que dificulte el servicio a las tareas aperiódicas, para las distribuciones dinámicas de tareas aperiódicas incluso llega a ser beneficioso.

3.2.2 Comparación con otros planificadores

Para completar el apartado 3.2 se ha comparado el rendimiento de los servidores diseñados versus dos algoritmos de referencia para monoprocesadores: *Slack Stealing* y *Total Bandwidth*. El primero tiene un rendimiento óptimo y el segundo tiene un buen rendimiento con unos menores costes. Desafortunadamente, no admiten la distribución dinámica de tareas periódicas y por eso solo experimentamos los casos PF/AF y PF/AD. El primer caso es trivial, puesto que se trata de varios monoprocesadores, pero para el segundo se ha tenido que diseñar un método de distribución de las tareas aperiódicas.

El esquema de planificación PF/AD está detallado en la Figura 3.7. Utilizamos planificadores locales (*Local Scheduler*, LS) en cada procesador, de forma que cada

uno de ellos planificará las tareas periódicas estáticas de su procesador junto con las tareas aperiódicas sin plazo que le asigne el planificador global, tratando de minimizar su tiempo de respuesta medio. Este planificador global (*Global Scheduler*, GS) conocerá el estado de todos los planificadores locales y distribuirá las tareas aperiódicas de acuerdo con la capacidad disponible en cada procesador para ejecutar tareas aperiódicas. Así, cuando una nueva tarea aperiódica sin plazo es activada esta se encola en una cola de tareas disponibles global (*Global Aperiodic Ready Queue*, GARQ). El planificador global toma tareas aperiódicas de esta cola, intenta asignarlas a un procesador p concreto y para ello las encola en la cola de disponibles local pertinente (*Local Ready Queue*, LRQp). La asignación va a depender de la capacidad disponible en cada procesador y pueden surgir diversas heurísticas de selección. En el caso de que no haya ningún procesador disponible las tareas aperiódicas permanecerán en la cola GARQ hasta que haya suficiente capacidad en algún procesador.

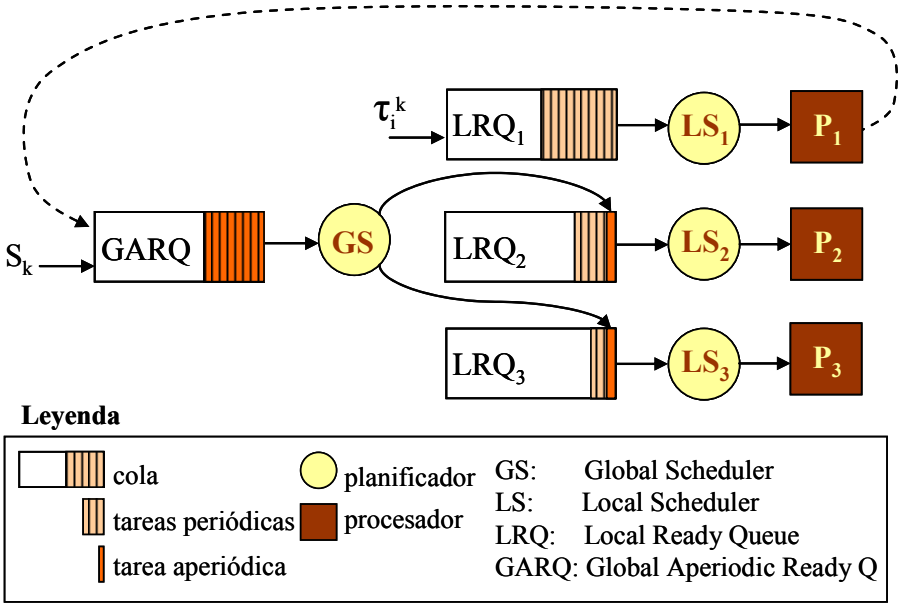


Figura 3.7: Esquema general de la combinación de planificadores locales y un planificador global. Las tareas aperiódicas sin plazo se encolan en una cola global y el GS se encarga de distribuirlas entre las colas locales asignándoles una determinada prioridad.

3.2.2.1 Planificador global de tareas aperiódicas Slack Stealing+ First-Fit (SS+FF)

En este apartado el planificador local (LS) del esquema de la Figura 3.7 está basado en el algoritmo de extracción de holgura o *Slack Stealing* (SS) con prioridades estáticas [LRT92,LRT94, RtL94]. Se ha elegido este algoritmo porque es óptimo en monoprocesador, en el sentido que minimiza el tiempo de respuesta de las tareas aperiódicas entre todos los algoritmos que cumplen los plazos de las tareas periódicas (con prioridades fijas, expulsivos y tratamiento FIFO para las tareas aperiódicas). Para cada procesador, se realiza un análisis fuera de línea para calcular la holgura de cada activación de cada tarea periódica y se guardar esta información en una tabla.

Cada planificador local (LS) planificará las tareas periódicas estáticas de su procesador junto con las tareas aperiódicas sin plazo que le asigne el planificador. Las tareas aperiódicas tendrán la prioridad más alta para así minimizar su tiempo de respuesta medio. El conjunto de tareas de cada procesador será planificable puesto que de otra forma el planificador global no le hubiese asignado ninguna tarea aperiódica.

Utilizamos un planificador global (GS) que distribuye las tareas aperiódicas entre los diferentes LS en función del estado de cada uno de ellos. Así, GS consultará las tablas de holgura calculadas en tiempo de diseño (A_{pi}) y conocerá el estado de todos los LS (la holgura ya consumida por tareas aperiódicas $A_{pi}(t)$ y la inactividad $I_{pi}(t)$ en cada nivel de prioridad i) para así poder distribuir las tareas aperiódicas de acuerdo con la holgura existente en cada procesador (véase la ecuación 3.4). Cuando una nueva tarea aperiódica sin plazo es activada esta se encola en GARQ. El planificador global toma tareas aperiódicas de esta cola, busca el primer procesador que no tenga ninguna tarea aperiódica asignada y con suficiente holgura para ejecutarla y le asigna la nueva tarea. En caso de no hallar tal procesador, asigna la tarea al que tenga mayor holgura y, cuando se agote la holgura, la tarea aperiódica es devuelta al GS para buscar una nueva asignación.

La holgura disponible en los procesadores puede ser suficiente para terminar la tarea, puede ser insuficiente e incluso puede ser excesiva. De igual forma podría darse el caso de haber diversos procesadores con holgura. Así, la asignación va a depender de la holgura disponible en cada procesador y de una de las diversas heurísticas de selección de procesador posibles. En el apartado 3.3.1 se detallan y se analizan las principales heurísticas. En el caso de que no haya ningún procesador disponible las tareas aperiódicas permanecerán en la cola GARQ hasta que haya suficiente holgura en algún procesador. La holgura disponible se revisa conforme vayan terminando las tareas periódicas.

Puede darse el caso en que un procesador tenga suficiente holgura como para que se le asignen más de una tarea aperiódica. En este caso es necesario llevar la cuenta de la holgura ya asignada anteriormente. Esta información la guardamos en un vector llamado V_p y el cálculo de la holgura disponible ahora se va a realizar según la siguiente ecuación:

$$S_{pk}(t) = \min_{0 \le i \le n}(A_{pi}(\gamma_{pi}(t)+1) - A_{pi}(t) - I_{pi}(t)) - V_p(t) \qquad \text{(ecuación 3.4)}$$

donde

- $S_{pk}(t)$ es la holgura disponible en el procesador *p* en el nivel *k* y el instante *t*
- *n* es el número de tareas periódicas asignadas al procesador *p*
- $A_{pi}(t)$ es la holgura calculada según [LRT94] en tiempo de diseño en el procesador *p* en nivel *i* y el instante *t*
- $\gamma_{pi}(t)$ es el número de activaciones de la tarea periódica de nivel *i* del procesador *p* terminadas en el instante *t*
- $A_{pi}(t)$ es la holgura ya consumida en el procesador *p* en nivel *i* durante [0,t]
- $I_{pi}(t)$ es el tiempo de inactividad del procesador *p* a nivel $_i$ durante [0,t]
- $V_p(t)$ es la holgura ya asignada y no consumida en el procesador *p* en el instante *t*

La asignación simultánea de varias tareas aperiódicas a un solo procesador podría llegar a ser ineficiente, puesto que la cola local LRQ_p podría llegar a contener muchas tareas aperiódicas. Como ejemplo, supongamos que en un instante *t* el procesador P_1 dispone de mucha holgura mientras que el procesador P_2 no la tiene. Durante un periodo de tiempo posterior a *t* todas las tareas aperiódicas que lleguen se

van a encolar en LRQ_1. Después de esto, una tarea periódica que termina en P_2 puede generar holgura en este procesador, pero al no llegan más tareas aperiódicas, se desperdiciaría esta holgura. Y, peor todavía, puede suceder que la holgura de P_1 este agotada para un periodo de tiempo largo, dado que sus tareas periódicas se han retrasado al máximo para servir las aperiódicas. El resultado final es un incremento en el tiempo de respuesta medio del conjunto de tareas aperiódicas. Cabe notar que esto puede suceder incluso en el caso que el procesador P_2 tuviese alguna holgura, aunque menor que la de P_1 ($S_2(t) << S_1(t)$). Como conclusión, parece mejor que las tareas aperiódicas esperen en la cola global que no en las colas locales y que el planificador global no asigne simultáneamente más de una tarea aperiódica a cada procesador.

En cada procesador se despachan las tareas aperiódicas si hay holgura disponible. Si ésta no es suficiente para terminar la tarea entonces hay que programar un evento que señale el final de la holgura para retornar la tarea aperiódica a la cola de disponibles, a la espera de nueva holgura. El retorno se puede hacer a la cola local LRQ_p del mismo procesador o a la cola global GARQ. Hacerlo en la cola local puede ser contraproducente, puesto que en ese instante podría haber otros procesadores que tuviesen holgura suficiente o incluso que estuviesen libres. Además, también es posible que otros procesadores ejecutasen tareas aperiódicas que hubiesen llegado más recientemente, rompiendo el servicio en orden FIFO. Así pues, es más razonable re-encolar las tareas en la cola GARQ y mantener esta cola ordenada por tiempo de llegada. En el Algoritmo 3-2 mostramos el pseudo-código resultante.

En el artículo [SVC99], coetáneo a nuestro estudio [BML99], se utiliza un esquema parecido pero con planificadores locales EDF. Para ello utilizan dos métodos. El primero consiste en asignar toda la ejecución de una tarea aperiódica al procesador que le proporcione el mejor tiempo de respuesta, para ello se debe calcular el tiempo de respuesta exacto para cada procesador, con la sobrecarga que conlleva. La segunda permite ejecutar parcialmente una tarea aperiódica en varios procesadores. Este último es equivalente al esquema propuesto en la Figura 3.7. La diferencia estriba en que en [SVC99] se asigna la tarea aperiódica al procesador que tenga el

primer hueco libre. Para hallarlo se debe hacer una búsqueda hacia adelante en el tiempo calculando cuando se producirá el primer hueco en cada procesador, usando el algoritmo de *Slack Seatling* dinámico para EDF. En nuestro caso únicamente se consulta una tabla estática para saber la holgura actual. Estos costes inferiores están más acorde con los objetivos de esta tesis. Una posibilidad alternativa utilizando el planificador local EDF con unos costes muy inferiores es el *Total Bandwidth* que se analiza en el siguiente apartado.

```
Algoritmo SS_PF/AD_ Global_Scheduler
1: suspender hasta
2:  -1a) una nueva tarea aperiódica Sk se active
3:  -1b) o Sk ejecutándose se quede sin holgura asignada
4:  -1c) o se genere nueva holgura en un procesador que no disponía
5: si (1a o 1b) entonces encolar Sk en la cola GARQ
6: mientras ( no vacía GARQ ) hacer
7:    Si = extraer la primera tarea aperiódica de la cola GARQ
8:    p= seleccionar el primer procesador con holgura disponible
9:    si existe tal procesador p entonces
10:         encolar Si en la cola LRQp
11:   sino
12:         encolar Si en la cola GARQ
13:         ir al paso 1
14:   fin_si
15: fin_mientras
```

Algoritmo 3-2: Pseudo-código del planificador global con SS+FF para PF/AD

3.2.2.2 Planificador global de tareas aperiódicas Total Bandwidth - Shortest Deadline (TB-SD)

Una alternativa interesante es utilizar para los planificadores locales el algoritmo *Total Bandwidth* (TB) con prioridades dinámicas [SpB94, SpB96], el cual puede tener un rendimiento algo inferior a SS, al utilizar menos cantidad de información,

pero tiene menos requisitos de memoria y de cómputo, ya que no calcula las holguras y tan solo asigna un plazo 'ficticio' a las tareas aperiódicas sin plazo.

El esquema general es el mismo que el de la Figura 3.7 , solo varía el planificador global, el cual esta descrito en el Algoritmo 3-3. Los planificadores locales en este caso son planificadores EDF. A diferencia de SS+FF, en este caso no hace falta calcular nada en tiempo de diseño y no hace falta guardar nada. Los planificadores locales no han de compartir información de su estado con el planificador global. Tan solo el planificador global ha de guarda en un vector los últimos plazos asignados a tareas aperiódicas en cada procesador (last_dl[p], líneas 6 y 12). Las tareas aperiódicas, permanecen en la cola GRAQ poco tiempo (a la espera de ser tratadas por el GS) y una vez asignadas a un procesador ya no migrarán a ningún otro procesador. Esto es así porque no disponemos de suficiente información para saber cuando va a ejecutarse ni cuando va a terminar.

```
Algoritmo TB_SD_Global_Scheduler
1: suspender hasta que una nueva tarea aperiódica se active
2: mientras ( no vacía GRAQ )
3:     S_k = extraer primero de GRAQ
4:     min_dl= 0
5:     para cada procesador p
6:           d_k = max(r_k, last_dl[p]) + ⌈C_k/U_s⌉    ; ver ecuación 2.7
7:           si (min_dl =0) o (d_k < min_dl) entonces
8:                 min_dl= d_k
9:                 p_seleccionadado= p
10:          fin_si
11:    fin_para
12:    last_dl[p_seleccionado]= min_dl
13:    asignar a S_k el plazo min_dl
14:    encolar S_k en la cola LRQ_p_seleccionado
15: fin_mientras
16: volver al paso 1
```

Algoritmo 3-3: Pseudo-código del planificador global *Total Bandwidth-Shortest Deadline* (TB-SD)

3.2.2.3 Evaluación experimental del rendimiento

En primer lugar se ha repetido el segundo experimento pero comparando los resultados con los planificadores SS+FF y TB-SD. Con la finalidad de mantener la coherencia de los nombres de los algoritmos de este apartado, utilizamos las siguientes equivalencias de nombres: TB-F ≡ TB locales, TB-A ≡ TB-SD, SS-F ≡ SS locales y SS-A ≡ SS+FF. Para mayor completitud también se ha incluido la planificación RM local en segundo plano Bkg-F y Bkg-A. Estos ejecutan las tareas aperiódicas solo cuando no hay tareas periódicas locales disponibles. El primero toma las tareas aperiódicas de una cola local mientras que el segundo las toma de una cola global.

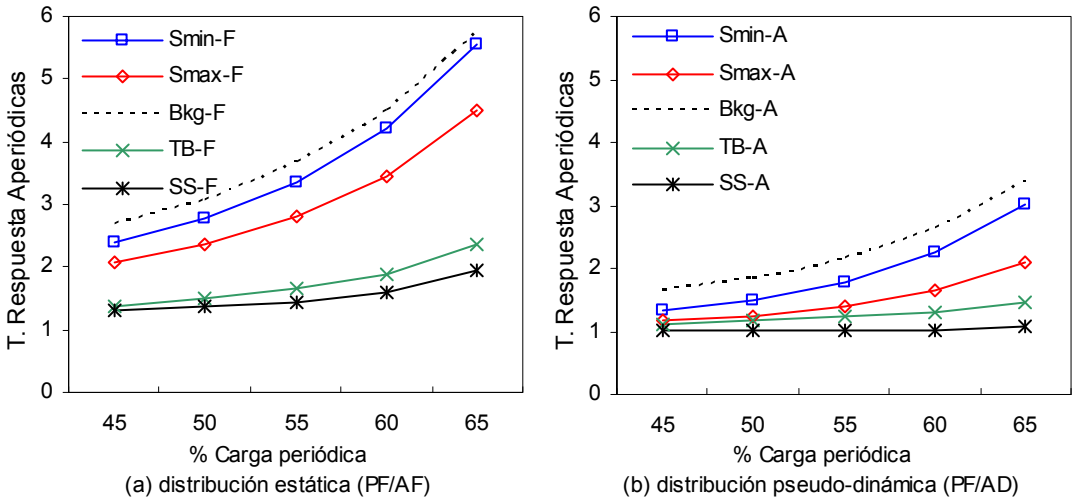


Figura 3.8: Comparativa de los resultados obtenidos con servidores versus otros tipos de planificadores (m=4, T∈ [100..1500], Uaper= 20%, Umax= 15%)

En la Figura 3.8 se muestran los resultados obtenidos en el segundo experimento junto con los obtenidos con los últimos planificadores. Si comparamos la gráfica (a) con la (b) observamos que la distribución PF/AD es mejor que la distribución estática con todos los algoritmos de planificación. Como era de esperar, los planificadores SS+FF y TB-SD obtienen los mejores resultados. La mayor diferencia se obtiene con la distribución estática. Estos planificadores nuevos no dependen de servidores ni de su dimensionado y pueden obtener muy buenos resultados incluso para cargas periódicas altas, sobretodo con la distribución dinámica de las tareas aperiódicas por la flexibilidad que les confiere. De los planificadores anteriores, Smax es el que

obtiene resultados más próximos a los nuevos, pero para cargas periódicas altas llega a doblar el tiempo de repuesta del mejor. Por su parte, con Smin se obtienen resultados algo mejores que el servicio en segundo plano: un promedio del 8,5% mejor con la distribución PF/AF y un promedio del 16,6% mejor con la distribución PF/AD.

Como experimento final, se ha variado el número de procesadores para comprobar como escala el comportamiento de los diversos planificadores. Los resultados se muestran en la Figura 3.9. En las gráficas (a) y (c), cuando las tareas aperiódicas se distribuyen estáticamente, se puede observar que el TRA apenas varía cuando aumenta el número de procesadores.

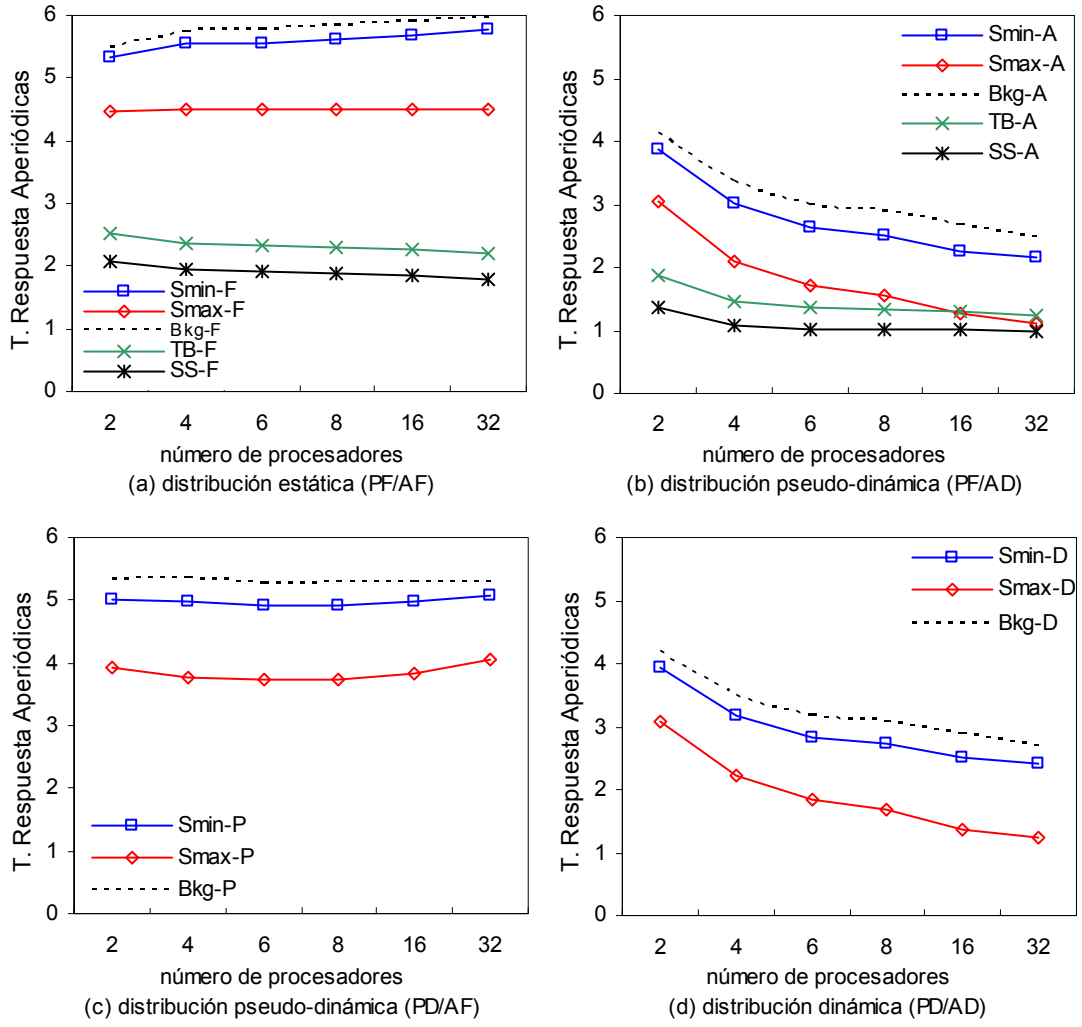


Figura 3.9: Comparativa de los resultados obtenidos con servidores versus otros tipos de planificadores cuando varía el número de procesadores ($T \in [100..1500]$, U_{per}=65%, U_{aper}=20%, U_{max}=0,15)

En las gráficas (b) y (d), el TRA mejora cuando aumenta el número de procesadores, excepto para TB-A, que se mantiene. Smax-A se acerca al óptimo con 32 procesadores, pero la tendencia indica que el resto también llegarían a serlo si el número de procesadores fuese suficientemente grande. El problema de los servidores es que cuando una tarea aperiódica se activa necesita encontrar un servidor activo. Cuando el número de procesadores aumenta también aumenta el número de servidores y, por lo tanto, aumenta su presencia activa a lo largo del tiempo.

3.2.3 Conclusiones

En el primer experimento hemos visto como un dimensionado maximalista de los servidores puede incumplir un gran número de plazos cuando las tareas periódicas se distribuyen de forma dinámica. Hemos determinado que para la parametrización utilizada el valor máximo actual de la carga incluida el servidor es del 85%. De hecho, utilizando este máximo, en el resto de los experimentos no se ha detectado ningún incumplimiento de plazo. En el segundo y tercer experimento hemos observado que la distribución PF/AD es la mejor, pero con PD/AD se obtienen unos resultados similares, mientras que la distribución estática de las tareas aperiódicas obtiene los peores resultados, llegando a doblar el mejor resultado cuando la carga periódica es más alta. Con un mejor dimensionado de los servidores se obtiene una mayor ganancia con las distribuciones dinámicas que con las estáticas, sobretodo cuando la carga periódica o aperiódica es alta. En el cuarto experimento hemos observado que las distribuciones dinámicas de tareas aperiódicas soportan mejor el hecho desfavorable de tener tareas periódicas mayores. En el quinto experimento hemos confirmado que los algoritmos no basados en servidores TB y SS obtienen todavía mejores resultados, pues no dependen del dimensionado de los servidores. Ambos obtienen mejores resultados con PF/AD, cuando disponen de mayor flexibilidad. Cuando el número de procesadores crece los servidores máximos proporcionan un servicio mejor al de TB y cercano al de SS. El rendimiento de TB y SS es bastante constante, pero se observa un ligero empeoramiento cuando la carga es más alta. Esto hace necesario realizar más experimentos, pero sin la planificación con servidores, puesto que con cargas altas no se podrían dimensionar los servidores.

En resumen, hemos constatado que la mejor política de distribución con los métodos de simulación simulados es la clásica PF/AD. A pesar que hemos utilizado una planificación con servidores de un diseño simplista, PD/AD obtiene unos resultados similares. En el próximo apartado de este capítulo profundizaremos más en la planificación PF/AD con *Slack Stealing* para obtener una buena referencia para los capítulos siguientes, puesto que hemos comprobado que es el algoritmo que ha obtenido mejores resultados y hemos intuido que, en este caso concreto, existen diversas heurísticas para mejorar su rendimiento cuando la carga sea mayor.

3.3 Distribución on-line de las tareas aperiódicas sin plazo

En los sistemas distribuidos la migración de tareas es más costosa que en los multiprocesadores. Por eso se suelen realizar distribuciones de grupos de tareas aperiódicas en lugar de asignaciones individualizadas y se utiliza el paso de mensajes para el intercambio de la información de estado y la comunicación entre planificadores. Normalmente los métodos propuestos solo buscan un nodo candidato a servir las tareas nuevas pero no tratan la asignación cuando existen varios de estos nodos disponibles ([RSZ89],[Foh95]). En este apartado nos centramos en los sistemas multiprocesador de memoria compartida, donde no es necesario mandar mensajes para la comunicación debido a que va a existir un planificador global que usará la información del estado de los planificadores locales localizada en la memoria compartida. Además, la migración es menos cara debido a que el código de las tareas e incluso sus datos no se van a mover de dicha memoria compartida. Esto marca una gran diferencia con los sistemas distribuidos y abre nuevas posibilidades, ya que asignar tareas a un procesador u otro tendrá los mismos costes pero encontrar el mejor procesador al que asignar una tarea no es trivial. Esto es así porque las decisiones de planificación tomadas en vivo pueden modificar substancialmente el comportamiento y la respuesta ante futuras peticiones.

En el apartado anterior vimos como la distribución estática de tareas periódicas combinada con la distribución dinámica de tareas aperiódicas (PF/AD) proporciona los mejores tiempos de respuesta medios de las tareas aperiódicas. La planificación de las tareas periódicas distribuidas estáticamente se puede realizar con un planificador para monoprocesadores. La distribución de las tareas aperiódicas sin plazo la realizar un planificador global, teniendo en cuenta la información del estado proporcionada por los planificadores locales de cada procesador. Cuanto más significativa sea esta información de estado mejor podrán ser las decisiones del planificador global y, por lo tanto, las tareas aperiódicas recibirán un mejor servicio pero, seguramente los costes de mantener y compartir esta información serán mayores.

Como ya hemos comentado, el tipo de planificador local utilizado puede variar. Una vez fijado el LS, hay que adaptar el GS a la información que proporciona los LS. En el apartado 3.2.2.2 utilizamos como LS la planificación *Total Bandwidth*, pero estos LS proporcionan poca información al GS y éste no tiene mucho margen de maniobra. En cambio, el planificador *Slack Stealing* sí proporciona información muy detallada y en este caso el GS puede usar diversas heurísticas de distribución, las cuales detallamos en el siguiente apartado.

3.3.1 Heurísticas en la elección del procesador con holgura

En la implementación del *Slack Stealing* (SS) para multiprocesadores es necesario seleccionar un procesador entre los que disponen de holgura (línea 8 del Algoritmo 3-2). Según las circunstancias la elección será fácil pero en otras habrá que utilizar alguna heurística de desempate.

Supongamos una tarea aperiódica S_k que requiere una ejecución C_k en el instante *t* y que hay dos procesadores, P_1 y P_2, con holguras $h_1(t)$ y $h_2(t)$ respectivamente. Si $h_1(t) < C_k < h_2(t)$ entonces es necesario elegir P_2 porque es el único que puede ejecutar S_k íntegramente. En el supuesto que si $h_1(t) < h_2(t) < C_k$, a priori la mejor opción parece ser P_2 porque éste puede ejecutar una mayor porción de S_k. De otra forma, si ambos procesadores dispusiesen de suficiente holgura ($C_k \le h_1(t)$ y $C_k \le h_2(t)$) entonces no estaría claro cual elegir. Elegir el de mayor holgura no tiene porque ser la mejor opción, puesto que puede restringir futuras peticiones. Así pues, vamos a analizar las siguientes estrategias:

- *First-Fit-Allocation* (FF): elegir el primer procesador hallado con suficiente holgura
- *Next-Fit-Allocation* (NF): elegir el primer procesador hallado con suficiente holgura, empezando a buscar desde el siguiente procesador a la última elección
- *Best-Fit-Allocation* (BF): elegir el procesador con menor pero suficiente holgura
- *Worst-Fit-Allocation* (WF): elegir el procesador con mayor holgura.

En todos los casos, cuando no hay ningún procesador con suficiente holgura, se elige el que tenga mayor holgura (rompiendo empates arbitrariamente).

Las dos primeras estrategias tienen la ventaja que detienen la búsqueda cuando encuentran un procesador con suficiente holgura, mientras que las dos últimas deben recorrer todos los procesadores. Esto implica que FF y NF escalan mejor.

Antes de comparar las diferencias de eficiencia en términos de los tiempos de respuesta medios de las tareas aperiódicas primero analizaremos un ejemplo que muestra como pueden darse tales diferencias. En la Figura 3.10 representamos la evolución temporal de la holgura en cada procesador usando FF y la evolución con NF. En estas figuras podemos ver como la base de los picos, que representa el tiempo que la holgura esta disponible, es más ancha con NF para todos los procesadores que con FF. Esto significa que NF está distribuyendo mejor las tareas aperiódicas de forma que la holgura de los procesadores se mantiene y así su eficiencia será mejor que con FF.

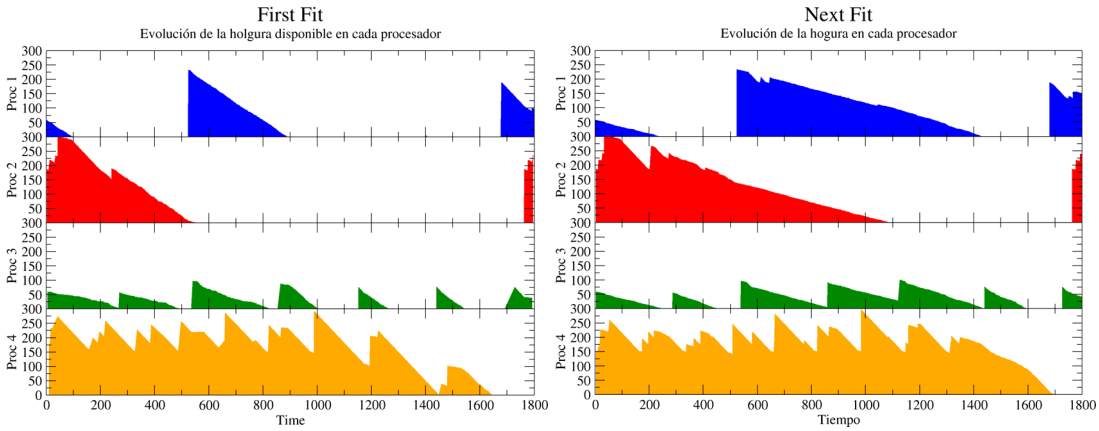


Figura 3.10: Ejemplo de la evolución de la holgura disponible en cada procesador usando *Slack Stealing* con FF y NF

El procesador P_4 dispone de una gran holgura que se renueva de forma periódica. FF solo usa esta holgura de forma eventual, cuando el resto de los procesadores ya no disponen de holgura puesto que siempre sigue el mismo orden secuencial de consulta de la holgura. Por el contrario, NF si la usa pues la consulta de la holgura es del tipo circular (*Round Robin*), aumentando así la disponibilidad de la holgura en los otros procesadores. Curiosamente, al principio con NF el procesador P_4 tiene un poco menos de holgura que con FF (puesto que éste no la usa) pero al final esta holgura

está disponible un poco más de tiempo (t=1640 vs. t=1700). Para finalizar, observamos que con FF cerca de un 30% del tiempo tiene un único procesador (P_4) con holgura disponible, mientras que con NF esto solo ocurre un 5% del tiempo (y solo al final), aumentando la posibilidad de servir tareas aperiódicas. Así pues, podemos predecir que el la distribución de tareas aperiódicas con NF obtendrá mejores tiempos de respuesta que con FF.

3.3.2 Evaluación experimental del rendimiento

En esta sección mostramos los resultados de las simulaciones obtenidos con el esquema de planificación global de la Figura 3.7 para las cuatro estrategias de asignación. El objetivo es estudiar el tiempo de respuesta medio de las tareas aperiódicas usando conjuntos de tareas generados sintéticamente y caracterizados con diversidad de parámetros para cubrir un amplio espectro de posibles aplicaciones. En el primer experimento se identificará el rango de distribuciones de cargas que genere mayores diferencias entre las distintas estrategias de distribución. En el segundo y tercer experimento se ensayarán distintos rangos de periodos. En el cuarto experimento se usarán periodos más armónicos. Finalmente, en el quinto experimento se evaluarán conjuntos de tareas con puntos de ruptura diferentes.

Descripción de los parámetros de los conjuntos de tareas.

Hemos fijado el número de procesadores en m=4. En los experimentos de la Figura 3.9 se probo SS+FF hasta con 32 procesadores y su rendimiento se mostró estable, independientemente del m. Además, se realizaron pruebas preliminares con hasta 8 procesadores y las cuatro heurísticas. Cuatro procesadores mostraron ser suficientes para evidenciar diferencias de rendimiento y para permitir simular una gran cantidad de experimentos en un tiempo razonable. Por otra parte, la configuración con cuatro procesadores es muy común en sistemas de memoria compartida y sistemas multicore.

Los conjuntos de tareas periódicas se han generado con las cargas equilibradas por procesador utilizando los parámetros del PCT # 2 del Anexo B. Los periodos de las tareas periódicas están en el rango [100..1000] y el hiperperiodo es 378000. Estos

parámetros producen periodos no armónicos. El número de tareas periódicas en cada procesador ha sido fijado a 15 y se ha establecido el factor de utilización máximo ($U_{max}=C_i/T_i$) al 20%. El plazo de las tareas se ha fijado igual a su periodo ($D_i=T_i$). Esta carga periódica es lo suficientemente baja para generar conjuntos planificables y lo suficientemente alta para experimentar algunas dificultades en servir las peticiones aperiódicas. Todos los conjuntos de tareas generados son planificables y tienen una utilización de ruptura[13] mayor que 75%. De esta forma, la carga periódica de los experimentos ha podido ser incrementada gradualmente hasta el 75% aumentando proporcionalmente todas los WCET de las tareas periódicas.

En todas las simulaciones todas las tareas (periódicas y aperiódicas) ejecuta siempre hasta su WCET. Aunque el SSA podría utilizar el eventual tiempo sobrante de las tareas periódicas, el objetivo de estos experimentos es ver su comportamiento en los casos límite.

A menos que se indique explícitamente, la carga aperiódica esta fijada en el 25%. Así, la carga máxima total por procesador es del 98%. Hemos utilizado unos requisitos de tiempo de cómputo para las tareas aperiódicas en el rango [1..25]. Los tiempos intermedios entre llegadas han de estar en el rango [4..100] para conseguir la carga aperiódica promedio del 25%. Las llegadas de las peticiones aperiódicas fueron generadas siguiendo una distribución de Poisson. Los tiempos intermedios entre llegadas han de estar en el rango [4..100] para conseguir la carga aperiódica promedio del 25%. Si utilizásemos un servidor esporádico con un periodo igual al periodo más pequeño, el tamaño máximo del servidor sería de 30 unidades de tiempo (periodo mínimo x(utilización máxima de ruptura - carga periódica) =100x(100%-70%)). Por lo tanto, se han generando cargas aperiódicas exigentes pero factibles.

La métrica de rendimiento utilizada es la media aritmética de los tiempos de respuesta normalizados medios de las tareas aperiódicas (*T. Respuesta Aperiódicas, TRA*)[14]. Cada punto de las gráficas se ha obtenido como la media aritmética del resultado obtenido en la simulación de 100 conjuntos de tareas distintos.

[13] Véase *Breakdown Utilization* en [LSD89]
[14] Para abreviar, en las gráficas se denomina *T. Respuesta Aperiódicas* y en el texto *TRA*

Para cada conjunto de tareas se ha repetido la simulación hasta que hemos alcanzado un nivel de confianza del 95% que el valor medido estaba dentro del intervalo del 5% del valor real. El único parámetro que variamos en estas simulaciones es la semilla inicial, para la generación de la distribución de las llegadas aperiódicas.

En los experimentos, se han utilizado conjuntos de periodos armónicos y conjuntos no armónicos. Los conjuntos de tareas armónicos son representativos de algunos sistemas de tiempo real. Generalmente resultan en una utilización más alta del procesador [LiL73] y han sido de uso general en los *benchmarks* de la literatura [KaW91]. Por otro lado, los conjuntos de tareas no armónicos dan lugar a hiperperiodos muy grandes y pueden causar problemas a algoritmos como SSA por tener requisitos de memoria muy elevados. También hemos comparado conjuntos de tareas con diferentes ratio de periodos (el cociente entre el periodo más grande y el más pequeño) de manera que los rangos de periodos generados fueron [100..1000] y [100..3000] (10x y 30x respectivamente).

Finalmente, hemos generado conjuntos de tareas con una utilización de ruptura alta y otros con baja para ver si esta característica de las tareas afecta distintamente al rendimiento.

Experimento 1

El primer experimento se ha diseñado para localizar el rango de cargas donde las diferencias entre las estrategias de asignación son más relevantes. El resto de experimentos se realizarán teniendo en cuenta estos rangos de cargas. Los resultados se muestran en la Figura 3.11 y la Figura 3.12.

En el Figura 3.11 la carga aperiódica se ha fijado aproximadamente al 15%, y la carga total varía entre el 85% y el 99%. Como se puede observar, las diferencias entre las estrategias de asignación se presentan cuando el sistema esta muy cargado, principalmente para cargas periódicas superiores al 80% (95% la total). Aunque no se puede apreciar a simple vista, la diferencia llega a ser del 20% cuando la carga es máxima. La mayor diferencia en rendimiento se obtiene para C_{aper} grandes. Hemos obtenido resultados similares pero más evidentes con cargas aperiódicas que

constituyen un porcentaje mayor de la carga total. Los resultados están detallados en la Figura 3.12.

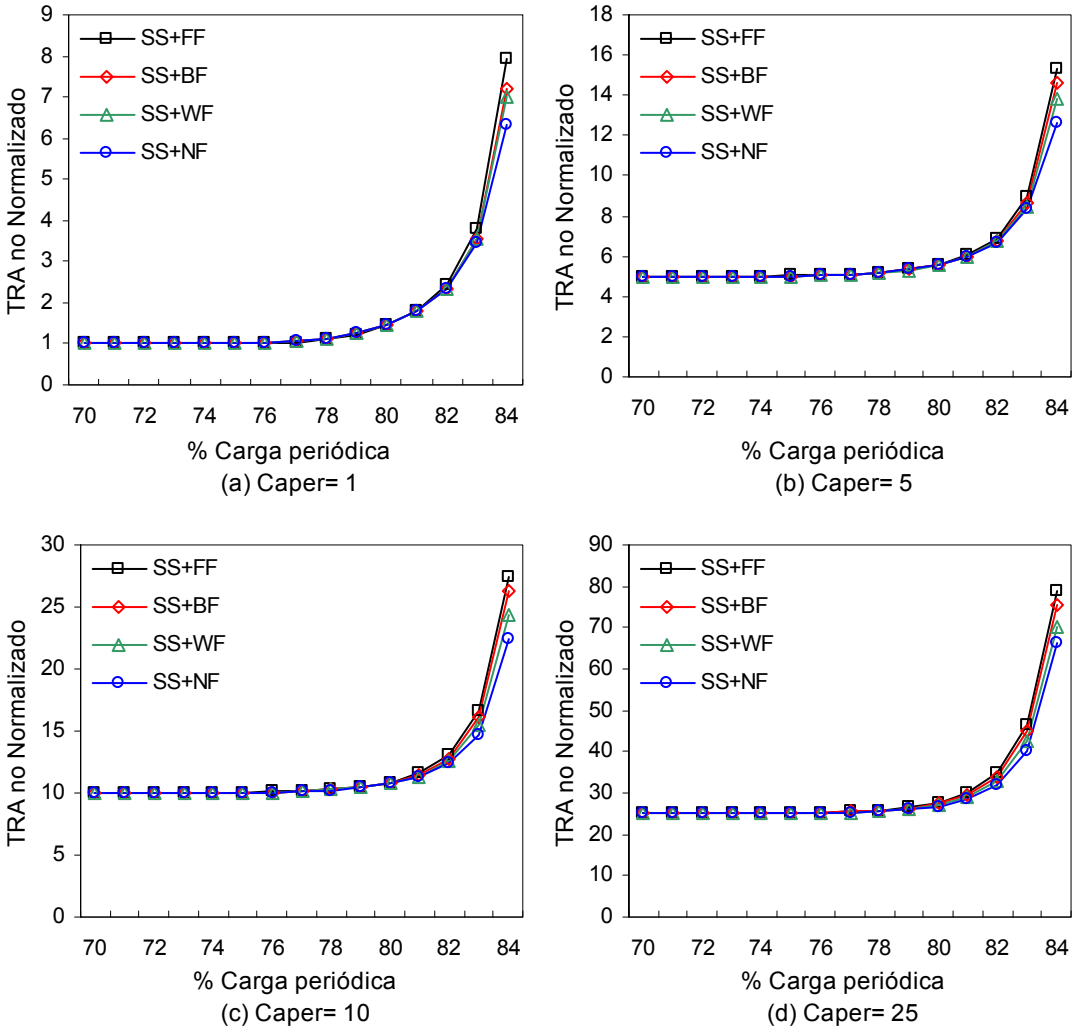


Figura 3.11: Experimento 1 (m=4, T∈ [100..1000], U_{aper}= 15%, U_{max}= 10%)

En el Figura 3.12 la carga aperiódica se ha fijado aproximadamente al 33,33%, y la carga total varía entre el 83,33% y el 97,33%. El factor de utilización máximo (U_{max}) ahora puede ser mayor, de forma que hay una tarea grande y el resto de tareas son pequeñas. En este caso, las diferencias superiores al 20% empiezan a partir de una carga periódica superior al 60% (cerca del 95% del total). A diferencia del caso anterior, la mayor diferencia en rendimiento se obtiene para C_{aper} pequeños. Esto puede ser debido al U_{max} mayor, puesto que una tarea periódica grande de alta prioridad puede retardar bastante a las aperiódicas.

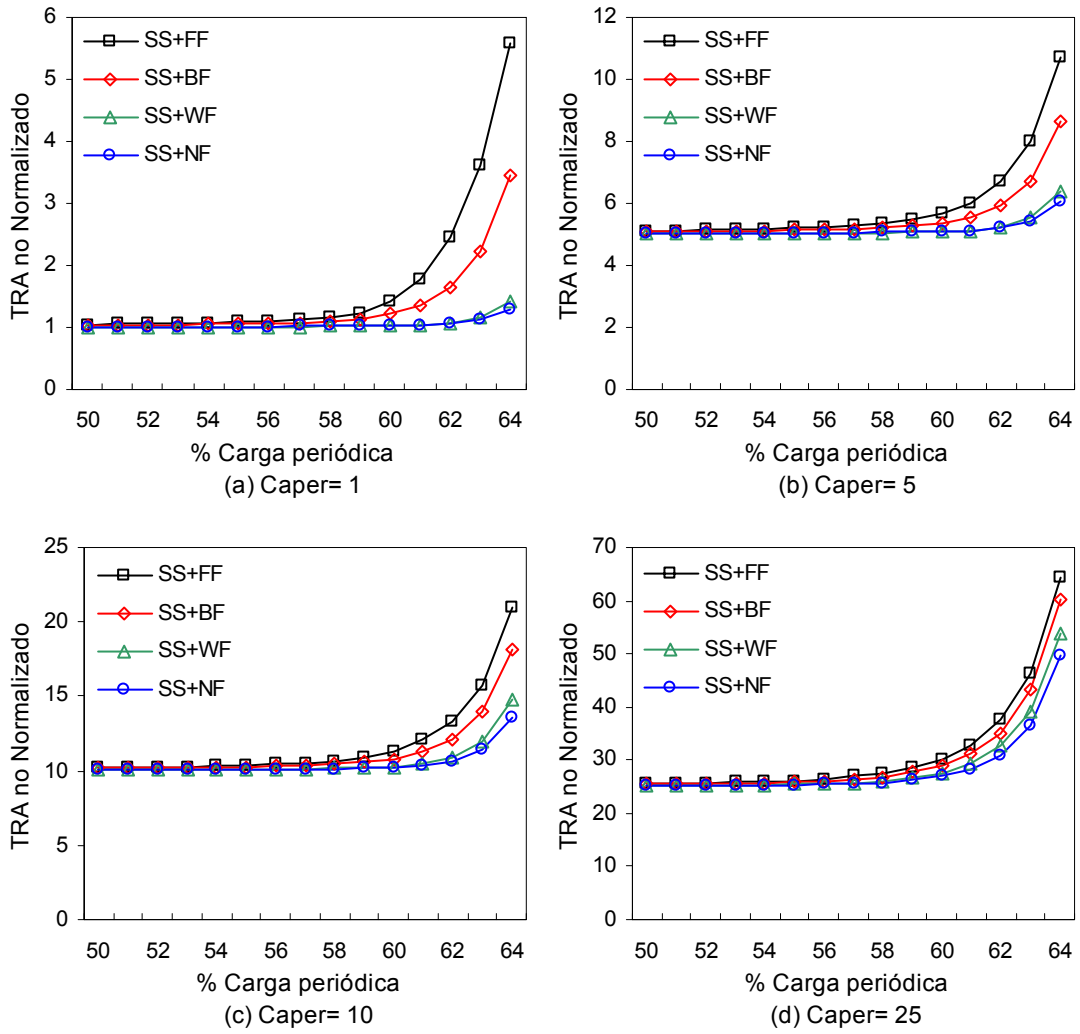


Figura 3.12: Experimento 1 (m=4, T∈ [100..1000], U_{aper}= 33,3%, U_{max}= 20%)

En estas simulaciones el método de asignación peor es siempre FF. Por otro lado, WF y NF se comportan de forma similar. Sin embargo, cuando C_{aper} es grande, NF se comporta algo mejor. Por ejemplo, en la Figura 3.12-(d) WF llega a ser un 8% peor. También simulamos cargas aperiódicas del 50% variando la carga periódica desde 33% a 47% y las diferencias también aparecen a partir de cargas muy altas (a partir del 90%) y son mayores para C_{aper} pequeños.

Vistos los resultados de este experimento, hemos diseñado los siguientes experimentos con una carga periódica mínima del 70% y una carga aperiódica del 25%, resultando la carga total entre el 95% y el 98%. El parámetro U_{max} se ha fijado al 20%, que representa un término medio entre los dos casos probados en el primer

experimento. Es decir, hay una tarea grande pero el resto no son muy pequeñas, puesto que hay suficiente carga para repartir.

Experimento 2

Los resultados del segundo experimento se representan en la Figura 3.13.

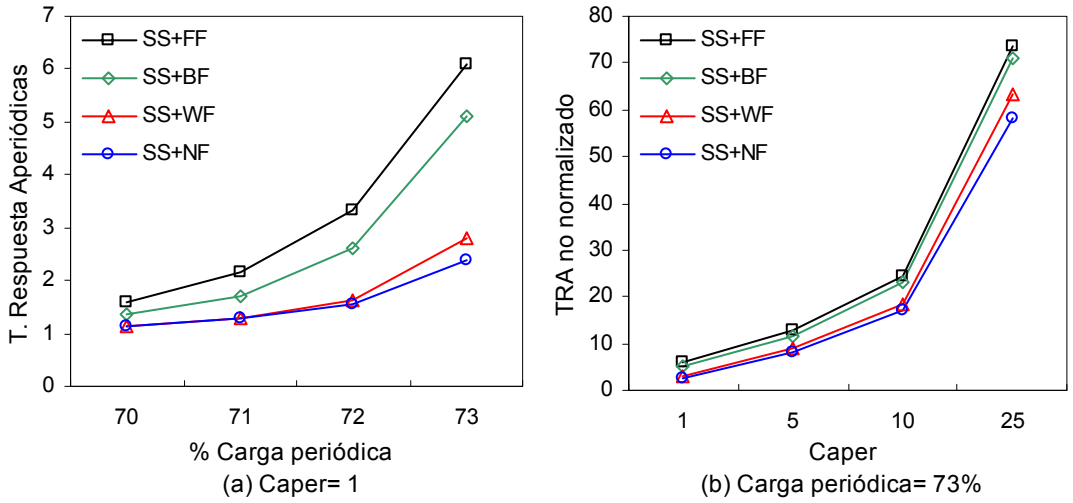


Figura 3.13: Experimento 2 (m=4, T∈ [100..1000], U_{aper}= 25%, U_{max}= 20%)

En el Figura 3.13-(a) se muestra el promedio del tiempo de respuesta medio aperiódico frente la carga periódica que va desde el 70% hasta el 73% con una carga aperiódica del 25%. El tiempo de cálculo de tareas aperiódicas se ha fijado a 1 unidad de tiempo. De esta forma vamos a tener muchas tareas aperiódicas pequeñas (C_{aper}=1, tiempo medio entre llegadas=4). Los resultados muestran que el mejor método de asignación en este caso es el NF, seguido de cerca por el WF, mientras que los peores son BF y FF, en este orden. El incremento de FF respecto a NF es del 158% cuando la carga total de cada procesador es la máxima (98%). La variancia del tiempo de respuesta medio conseguida con NF es en promedio del 12%, mientras que con el resto de estrategias varía entre el 20% y el 35%.

En el Figura 3.13-(b) analizamos el caso específico de la carga máxima (el 73% de la carga periódica) cuando el tiempo de cálculo de las tareas aperiódicas varía de 1 a 25 unidades de tiempo. De esta forma experimentamos como afecta a su tiempo de respuesta medio las distintas caracterizaciones del tamaño de las tareas aperiódicas. Los resultados muestran que las diferencias entre las distintas distribuciones se

mantienen cuando C_{aper} aumenta. En la Figura 3.13-(b) la métrica utilizada es la media aritmética de los tiempos de respuesta medios de las tareas aperiódicas. Sin embargo otra métrica a considerar son los tiempos relativos de espera, los cuales obtenemos de la siguiente forma: $(TRA - C_{aper})/C_{aper}$). Así, los tiempos relativos de espera de las tareas aperiódicas con NF y WF también se mantienen, mientras que con BF y FF decrementan substancialmente. Usando FF el tiempo de espera con C_{aper}=1 es 5,1 y con C_{aper}=25 éste es 1,94, es decir, se ha reducido un 38%. Similarmente, para BF el tiempo de espera pasa de 4,1 a 1,84 con una reducción del 45%. Esto significa que estas dos estrategias de distribución son más apropiadas para demandas aperiódicas grandes. Como estas dos estrategias concentran la asignación de tareas en algunos procesadores (en los primeros para FF y en los de menor holgura para BF) cuando las tareas son pequeñas se forman largas colas y esto aumenta el tiempo de espera, en cambio cuando las tareas son grandes su concentración es menor y el tiempo de espera se reduce.

Experimento 3

El tercer experimento muestra el comportamiento cuando los periodos tienen una gama más amplia que en el segundo experimento (PCT # 3 del Anexo B.). En este caso los nuevos periodos están dentro del rango [100..3000]. Los resultados se representan en la Figura 3.14.

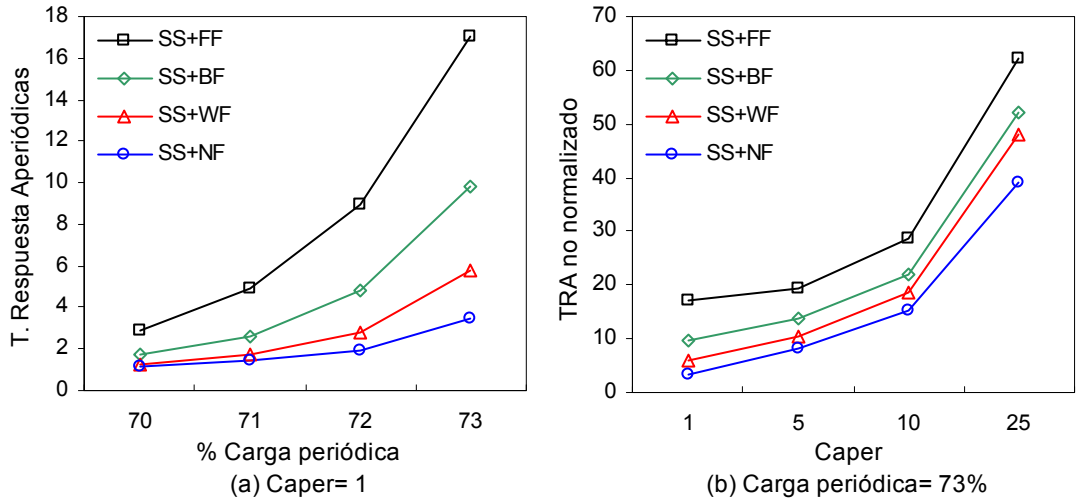


Figura 3.14: Experimento 3 (m=4, T∈ [100..3000], U_{aper}= 25%, U_{max}= 20%)

Los resultados son similares a los obtenidos en el segundo experimento pero los tiempos de respuesta aperiódicos se incrementan de forma proporcional al periodo máximo para FF, es decir casi tres veces los valores obtenidos en el segundo experimento cuando C_{aper}=1 (Figura 3.14-(a) vs. Figura 3.13-(a)). Esto es debido a las tareas aperiódicas son muy pequeñas y en cambio las tareas periódicas, con periodos mayores que en el experimento anterior y manteniendo el resto de características, tienen unos cómputos mayores, forzando a las aperiódicas a esperas más prolongadas. Este efecto negativo se nota menos en el resto de estrategias. La estrategia más robusta a la extensión del rango de periodos es el NF, donde el incremento máximo es del 48%, porque distribuye más las tareas, manteniendo colas más cortas.

Observando la Figura 3.14-(a), vemos que es notable la diferencia entre NF y FF. NF representa una mejora del 80% del tiempo de respuesta medio aperiódico. Incluso con WF, la estrategia más cercana al NF, la diferencia es aproximadamente del 40%.

En la Figura 3.14-(b) analizamos el caso específico de la carga máxima cuando el tiempo de cálculo de las tareas aperiódicas varía de 1 a 25 unidades de tiempo. En esta gráfica se puede apreciar que las diferencias entre las cuatro estrategias de la asignación se mantienen. Sin embargo, estas diferencias son más considerables que en el experimento anterior (Figura 3.13-(b)). El tiempo de respuesta medio aperiódico aumenta cuando crecen los cómputos aperiódicos pero en realidad los tiempos de espera relativos disminuyen entre un 10%-20%. De hecho, para un requisito aperiódico mayor o igual a 5 los tiempos de espera relativos convergen en un rango estrecho para las cuatro estrategias. Esto significa que los tiempos de espera son similares para las cuatro estrategias cuando aumenta la demanda de ejecución de las tareas aperiódicas. En particular, cuando esta demanda es similar o superior a las holguras disponibles, cada método de distribución se comporta de forma parecida, puesto que todos ellos han de comprobar la holgura de cada procesador y probablemente todos elijan el mismo. Por el contrario, las diferencias más significativas aparecen cuando la holgura es mayor.

Si volvemos a comparar con el experimento anterior (ahora con la Figura 3.13-(a)) vemos que los incrementos son menores que con C_{aper}=1 (los promedios de los incrementos son 59%, 20%, 30% y 2% para cada distribución respectivamente). En realidad, para C_{aper}=25, FF reduce un 16% el TRA y NF reduce un 33%. Esto es debido a que las colas de aperiódicas locales son menores por falta de holgura en la asignación y a que el tiempo entre llegadas de las tareas aperiódicas está más acorde con los periodos de las tareas periódicas, es decir, con la generación de holgura. Una vez más NF distribuye mejor y es la estrategia más robusta a la variación del rango de periodos.

Experimento 4

En el cuarto experimento deseamos estudiar el efecto de la multiplicidad de periodos en el rendimiento del sistema. Hemos utilizado los parámetros la PCT # 4 del Anexo B. El hiperperiodo utilizado es 400000 y los periodos de las tareas están en el rango [128..3125], generando periodos más armónicos que en el tercer experimento (múltiples de potencias de 2 y de 5). A pesar de esta diferencia en los periodos, su rango y el hiperperiodo son equiparables a los de los experimentos segundo y tercero.

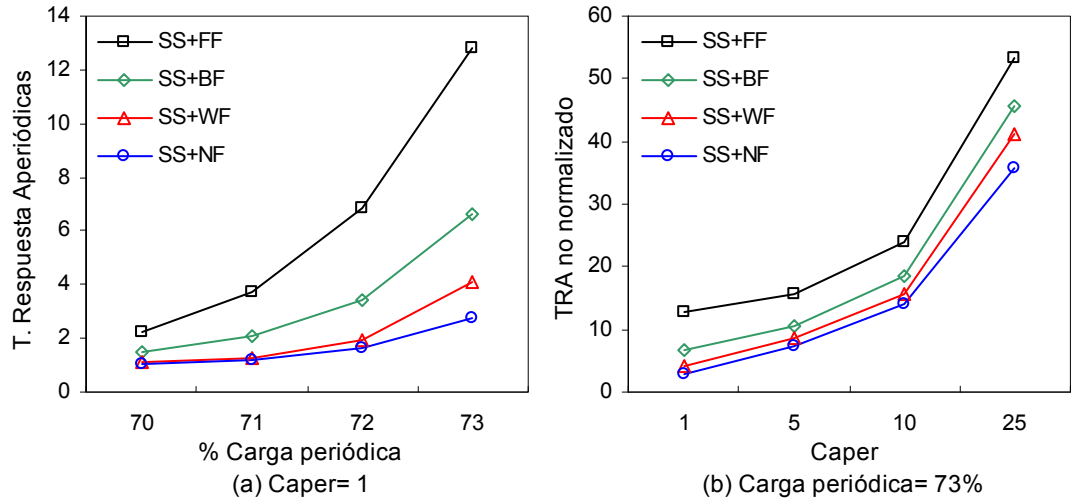


Figura 3.15: Experimento 4 (m=4, T∈ [128..3125], U_{aper}= 25%, U_{max}= 20%)

Los resultados obtenidos son similares a los alcanzados en los experimentos anteriores, pero los TRA son mejores que en el tercer experimento (varían entre 1 y 13 cuando C_{aper}=1 mientras que en el tercero variaban entre 1 y 17). Comparado los

resultados obtenidos en los dos últimos experimentos contra el segundo experimento, el cual utiliza una gama más corta de periodos, hemos observado que la diferencia relativa entre WF y NF se ha hecho mayor (WF es ahora menos eficiente) mientras que la diferencia relativa entre BF y NF se ha reducido (BF es ahora más eficiente).

Experimento 5

Finalmente, en el quinto experimento investigamos el efecto la utilización de ruptura por procesador sobre la diferencia de rendimiento entre las cuatro estrategias. Los parámetros de este experimento son iguales que los usados en el segundo experimento, pero hemos seleccionado de entre los conjuntos de tareas generados aquellos con la misma utilización de ruptura. Esta restricción reduce el número posible de experimentos drásticamente. Así pues para este experimento solo hemos generado 10 conjuntos de tareas para cada punto del eje de las abscisas. Se han simulado dos escenarios. En uno todos los procesadores que tenían una alta utilización de ruptura (el 95%) y en el otro todos los procesadores que tenían una utilización de ruptura más baja (el 85%).

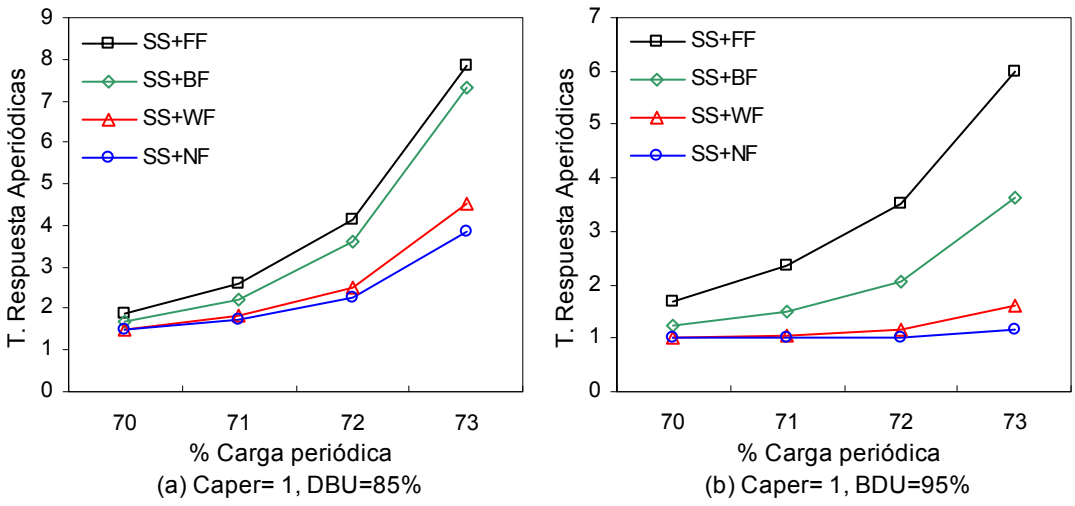


Figura 3.16: Experimento 5 (m=4, T∈ [100..1000], U_{aper}= 25%, U_{max}= 20%)

Cuando la utilización de ruptura es el 85% (Figura 3.16-(a)), el TRA es un 30% peor para la carga periódica más baja que el de la Figura 3.13-(a) y va creciendo, pues la carga periódica aumenta. Cuando la utilización de ruptura es más alta (el 95% por procesador, Figura 3.16-(b)), el tiempo TRA mejora con respecto al caso anterior. Este resultado se asemeja a los comportamientos que se pueden obtener en un

sistema monoprocesador. En estas circunstancias, el método de la asignación FF no es capaz de aprovecharse de la utilización de ruptura más alta, al contrario que NF que obtiene un incremento substancial de rendimiento, reaccionando a las demandas aperiódicas casi inmediatamente incluso para un sistema cargado casi al 100%.

Conclusiones

En un amplio abanico de caracterizaciones de los conjuntos de tareas hemos mostrado que NF es la mejor opción con SS para reducir el tiempo de repuesta medio de las tareas aperiódicas, si bien las diferencias entre las estrategias de asignación se presentan cuando la carga es muy alta. En general, este tiempo de respuesta es el doble del tiempo de cómputo de las tareas aperiódicas. Opinamos que este resultado es perfectamente aceptable para un gran número de aplicaciones. Los resultados contrastan con la asignación más utilizada en los sistemas distribuidos, donde se utiliza una estrategia de asignación al procesador con una mayor capacidad sobrante [RSZ89].

La distribución NF además de obtener TRA menores es la que tienen menor variancia y la que soporta mejor las distintas variaciones de las características de las tareas sintéticas. La distribución WF, que también balancea la carga, obtiene unos resultados similares, aunque algo peores y empeora más cuando los periodos son más armónicos. Las distribuciones que concentran la carga aperiódica, FF y BF, generan colas de tareas aperiódicas más largas en algunos procesadores y esto hace que su TRA sea entre 3 y 6 veces peor. Además, esto es más evidente cuando las tareas aperiódicas son más pequeñas.

Dados estos resultados, en los siguientes capítulos utilizaremos el método SS+NF como referencia en las comparaciones de rendimiento.

3.4 Conclusiones

En este capítulo se ha estudiado la ejecución conjunta de tareas periódicas y aperiódicas sin plazo en un sistema multiprocesador de memoria compartida. En la primera parte se han utilizado servidores de tareas aperiódicas y en la segunda se ha analizado la distribución de tareas aperiódicas usando planificadores locales no basados en servidores.

En el apartado 3.2 se han analizado diversas modalidades de distribución en función del tipo de tarea. Dado que en la literatura existen pocos métodos para planificar de forma global tareas periódicas y aperiódicas, se ha estudiado la posibilidad de usar servidores de tareas aperiódicas con planificación global. Los experimentos han hecho evidente la dificultad de dimensionar estos servidores, tanto desde el punto de vista de las garantías a los plazos como desde el punto de vista del servicio a las tareas aperiódicas.

Los experimentos muestran que la distribución estática de las tareas aperiódicas obtiene los peores resultados, llegando a doblar el mejor resultado cuando la carga periódica es más alta y que las distribuciones dinámicas de tareas aperiódicas soportan mejor el tener tareas periódicas mayores. Con un mejor dimensionado de los servidores hemos obtenido una mayor ganancia con las distribuciones dinámicas que con las estáticas, sobretodo cuando la carga crece.

En los apartados 3.2.2 y 3.3.1 se ha adaptado los algoritmos de planificación *Total Bandwidth* y *Slack Stealing* a sistemas multiprocesadores para la planificación de tareas aperiódicas distribuidas dinámicamente. Éstos planificadores han obtenido los mejores resultados con PF/AD, cuando disponen de mayor flexibilidad. Su rendimiento es muy constante, con un ligero empeoramiento cuando la carga crece. Sin embargo, con estos algoritmos no se han podido probar las distribuciones PD/AF y PD/AD, puesto que son planificadores locales. Con la planificación *Slack Stealing* se ha detectado la posibilidad de utilizar diversas heurísticas de distribución de las tareas aperiódicas. Así, en la parte final de este capítulo se han explorado diversas heurísticas de la planificación PF/AD con *Slack Stealing*. Al no usar servidores se

han podido utilizar cargas mayores. Los experimentos muestran que, independientemente de las características de las tareas, la distribución *Next-Fit* es la mejor opción para reducir el tiempo de repuesta medio de las tareas aperiódicas. Por otro lado, las distribuciones que concentran la carga aperiódica, FF y BF, obtienen unos resultados inaceptablemente peores. Estos resultados contrastan con la asignación más utilizada en los sistemas distribuidos, donde se utiliza una estrategia de asignación al procesador con una mayor capacidad sobrante [RSZ89].

En resumen, hemos constatado que la mejor política de distribución es la clásica PF/AD. A pesar que hemos utilizado una planificación con servidores de un diseño simplista, PD/AD obtiene unos resultados similares.

Concluimos que con otros dimensionados de los servidores, o con otro tipo de planificación, la tendencia se podría invertir. Con el dimensionado de servidores máximos los resultados han mejorado y esto nos indica que cuanta mayor flexibilidad tengamos, más rápidamente podremos atender y finalizar las tareas aperiódicas. Lamentablemente, los servidores máximos no pueden usar toda la capacidad del multiprocesador y no dan garantías en el cumplimento de los plazos. Además, la teoría de la planificación global no es suficientemente madura como para permitir el dimensionado de servidores con mayor capacidad. Asimismo, como los algoritmos no basados en servidores (TB y SS) obtienen mejores resultados, todo nos indica que la planificación con servidores no será la más adecuada.

Así pues, en los próximos capítulos realizaremos otros algoritmos de planificación dinámicos que permitan mejorar el rendimiento y que hagan PD/AD la mejor opción. Dados los resultados excelentes obtenidos con SS+NF y PF/AD utilizaremos éste método como referencia en los futuros experimentos de rendimiento.

Capítulo 4

PLANIFICACIÓN GLOBAL CON DUAL PRIORITY

En la primera parte de este capítulo se detalla y se analiza el algoritmo de planificación *Dual Priority* (DP) para monoprocesadores. En la segunda y principal parte se detalla su adaptación a la planificación global de sistemas multiprocesador.

4.1 Introducción

El algoritmo de planificación *Dual Priority* [BuW93,DaW95], a pesar no haber sido utilizado en demasía, tiene un fascinante equilibrio entre sencillez y potencia. Se basa en un estudio en tiempo de diseño de las tareas periódicas. En este estudio se obtiene una característica nueva para cada tarea periódica que indica cuanto se puede retardar una de estas tareas sin comprometer su plazo de finalización. Es esta nueva característica la que se utiliza en tiempo de ejecución para atender a otros tipos de tareas a base de retardar, cuando se pueda, las periódicas. Todo ello con un planificador que requiere unos bajos costes en tiempo de cómputo y en uso de memoria.

El desarrollo de los próximos apartados es como sigue. En primer lugar se detalla y analiza el algoritmo para monoprocesadores y se proponen una serie de mejoras. En segundo lugar se describen los problemas en la adaptación a multiprocesadores y se detalla una solución de compromiso. Finalmente se analiza el comportamiento del algoritmo propuesto para multiprocesadores con finalidades distintas: el servicio a otras tareas periódicas que a priori no serian planificables, el servicio a tareas aperiódicas sin plazo y la combinación de todos los casos. Partes de este capítulo se publicaron en los artículos [MBL97], [BML99] y [BAL03].

4.2 Algoritmo Dual Priority para Monoprocesadores

En este apartado se detalla y se analiza el algoritmo de planificación para monoprocesadores conocido como *Dual Priority* (DP). Como veremos más adelante en este apartado, este algoritmo tiene muy buenas cualidades y será adaptado a los sistemas multiprocesador con diversas versiones en el próximo apartado y en el próximo capítulo.

El algoritmo DP fue formulado inicialmente por Burns y Wellings en [BuW93] para poder aumentar en determinados casos el porcentaje de utilización del procesador con tareas periódicas. Posteriormente Davis y Wellings lo modificaron en [DaW95] para poder dar servicio a tareas aperiódicas sin plazo y lo extendieron para soportar plazos arbitrarios, variaciones en el tiempo de activación y recursos compartidos. Sin embargo, un estudio más detallado del rendimiento y de su adaptación a la diversidad de conjuntos de tareas era necesario. Así pues, en este apartado detallamos el algoritmo, analizamos sus características y su rendimiento. A pesar de todo, este método de planificación no es siempre tan bueno como sería deseable, así que en el Anexo A[15] proponemos diversas mejoras.[16]

4.2.1.1 El algoritmo DP básico

En este algoritmo las tareas periódicas tienen unas prioridades básicas asignadas de acuerdo a un orden fijo. Además, cada tarea periódica tiene una prioridad diferente. Por el contrario, las tareas aperiódicas sin plazo tendrán una prioridad fija, que puede ser la misma en todas ellas.

En este algoritmo existen dos niveles de prioridad para las tareas periódicas: el nivel de prioridades bajas (*Low Priority Level*, **LPL**) y el nivel de prioridades altas (*High Priority Level*, **HPL**). De esta manera, cada tarea periódica tiene asignados de forma estática una prioridad en cada uno de los dos niveles de prioridades. Estas prioridades se calculan en tiempo de diseño de la forma que se detallará más adelante. Las prioridades iniciales serán alteradas temporalmente por el planificador,

[15] Se decidió ponerlas en el anexo porqué es planificación en monoprocesador.
[16] En adelante llamamos DP básico al algoritmo que no incorpora las mejoras detalladas en Anexo A.

pudiendo una tarea periódica ser promocionada[17] de una prioridad baja a una prioridad superior. Así, en tiempo de ejecución, una tarea periódica empieza con su prioridad más baja hasta que llega su instante de **promoción,** cuando cambia a su prioridad alta. Este instante de promoción está basado en el cálculo del tiempo peor de respuesta obtenido por las tareas periódicas. Su cálculo se realiza en tiempo de diseño del sistema y también sirve como prueba de planificabilidad (véase el Algoritmo 4-1). Para ello se utiliza el método definido por Joseph y Pandya en [JoP86] (véase la ecuación 2.2). Entonces, el peor tiempo de respuesta es $R_i = R_i^0 = W_i^{n+1}$, si la iteración converge ($W_i^{n+1} = W_i^n$) y se cumple el plazo ($W_i^{n+1} \leq D_i$).

La *promoción relativa* de las tareas periódicas se computa de la siguiente forma:

$$Y_i = D_i - R_i \qquad \text{(ecuación 4.1)}$$

Y para la k-ésima activación, la *promoción absoluta* es:

$$Y_i^k = k\,T_i + D_i - R_i \;=\; k\,T_i + Y_i \qquad \text{(ecuación 4.2)}$$

Es decir, una tarea τ_i activada en el tiempo kT_i puede retardar su ejecución a lo sumo $D_i - R_i$ si quiere cumplir con su límite de ejecución. Véase la Figura 4.1 a modo de resumen ilustrativo.

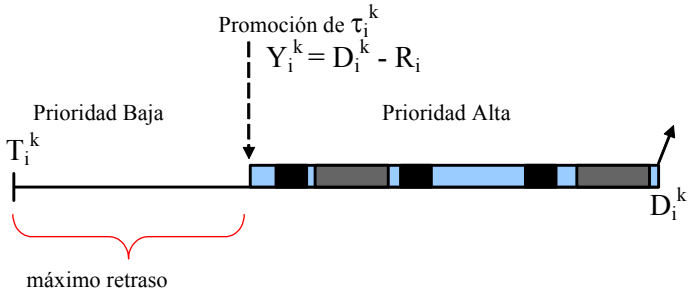


Figura 4.1: Máximo retraso de una tarea periódica con DP. El color más claro indica la ejecución de la tarea τ_i^k y los más oscuros la interferencia de las tareas más prioritarias.

[17] Dudamos entre utilizar los terminología *promocionar/promocionado* y *promover/promovido*. Los primeros términos se utilizan más en entornos comerciales. Los segundos parecen más apropiados y, además, indican movimiento. A pesar de ello, nos sentimos más cómodos con los primeros, sobre todo con el término "instante de promoción". Según el diccionario de la lengua castellana de la RAE:
Promocionar: Elevar o hacer valer artículos comerciales, cualidades, personas, etc.
Promover: 1. Iniciar o impulsar una cosa o un proceso, procurando su logro. 2. Levantar o elevar a alguien a una dignidad o empleo superior al que tenía. 3. Tomar la iniciativa para la realización o el logro de algo.

El rango y orden de las prioridades de bajo nivel no es de vital importancia y puede ser variado a conveniencia. En particular, cualquier servicio que se quiera prestar, como por ejemplo a tareas aperiódicas, podrá tener las prioridades más altas dentro de este nivel bajo. En el caso de las prioridades de nivel alto existen dos restricciones: por un lado, tienen que ser superiores a las del nivel de baja prioridad y por otro, deben tener el mismo orden relativo usado para el cálculo de las promociones, o sea, el orden utilizado en la ecuación 2.2. De esta forma, una tarea periódica puede ser retardada por las tareas aperiódicas, pero una vez promocionada a lo sumo recibirá las interferencias de las tareas periódicas de mayor prioridad previstas de antemano y así podrá terminar antes del vencimiento de su plazo.

```
Algoritmo Calculo_de_las_Promociones(τ) retorna booleano
1: ordenar las tareas periódicas
2: asignar las prioridades de acuerdo a este orden
3: para cada tarea periódica τi ∈ τ hacer
4:     Ri = el peor tiempo de respuesta de τi  ;según la ecuación 2.2
5:     si Ri > Di entonces
6:       retorna falso    ; τ no es factible con el orden establecido
7:     fin_si
8:     Yi = Di − Ri       ; anotar la nueva característica (promoción)
9:   fin_para
10: retorna cierto
```

Algoritmo 4-1: Pseudo-código del cálculo de las promociones en tiempo de diseño

En tiempo de diseño se debe utilizar el Algoritmo 4-1 para calcular las promociones de las tareas periódicas y se anotan estos valores como una característica más de las tareas.

En tiempo de ejecución el planificador es guiado por eventos utilizando un temporizador para la promoción de cada tarea periódica. Cuando en un instante t se produce cualquier evento relacionado con la planificación entonces se activa el planificador, el cual aplica el Algoritmo 4-2.

Algoritmo Planificador_DP()

Caso que la interrupción es debida a:

a) La activación de alguna tarea periódica: Mover todas las tareas con tiempo de activación menor o igual a t del estado letárgico al de activación. Para cada una de estas tareas, si Y_i=0 la tarea es encolada en la cola de preparadas de alta prioridad (HPRQ). Cuando Y_i>0 se usa la cola de preparadas de baja prioridad (LPRQ) y su promoción se calcula como $Y_i^k = t + Y_i$, programando un temporizador para este tiempo.

b) Un evento externo: mover la correspondiente tarea aperiódica del estado letárgico al estado activo y se encola en la cola de baja prioridad (LPRQ).

c) Una promoción: Todas las tareas periódicas en la cola LPRQ con $Y_i^k \leq t$ son promocionadas, es decir, se desencolan de LPRQ, se les aumenta su nivel de prioridad de LPL a HPL y se encolan en HPRQ.

Elegir la primera tarea encolada en la cola de mayor prioridad no vacía, si la hubiere (es decir, consultar primero HPRQ y luego LPRQ)

Si no el procesador está ocioso o la tarea elegida tiene una prioridad mayor que la tarea ejecutándose entonces mandar ejecutar la tarea elegida.

Esperar una nueva interrupción.

Algoritmo 4-2: Descripción del planificador DP en tiempo de ejecución

4.2.1.2 Análisis del algoritmo DP básico

Tareas periódicas.

Todas las activaciones de las tareas periódicas cumplen sus plazos. Si el conjunto de tareas periódicas es planificable bajo un algoritmo con expulsiones convencional éste conjunto también cumplirá sus plazos con el algoritmo DP. La prueba es que toda tarea periódica es encolada en HPRQ un tiempo R_i antes de su plazo absoluto D_i^k.

Una vez en la cola HPRQ una tarea tan solo puede ser retardada por tareas periódicas más prioritarias. Entonces, a lo sumo terminará en un tiempo R_i después de su promoción, es decir, antes de D_i^k.

Tareas aperiódicas sin plazo.

Si no hay tareas periódicas promocionadas entonces las tareas aperiódicas reciben un servicio inmediato, según un orden de llegada entre ellas,

Cuando haya tareas periódicas promocionadas puede parecer que el servicio a las tareas aperiódicas será malo, puesto que las promociones son calculadas de forma pesimista (el peor tiempo de respuesta, cuando todas las tareas se activan simultáneamente y se ejecutan en su peor caso), pero esto no sucede en la mayoría de activaciones de las tareas periódicas. En realidad, la k-ésima activación de una tarea cualquiera va a recibir de las tareas más prioritarias una interferencia menor a la calculada para el peor caso (puesto que $R_i^n \leq R_i$) y, por lo tanto, después de su promoción va a terminar bastante antes de su plazo, habilitando así el servicio a tareas aperiódicas si ya no quedarán más tareas periódicas en la HPQ (véase la Figura 4.2).

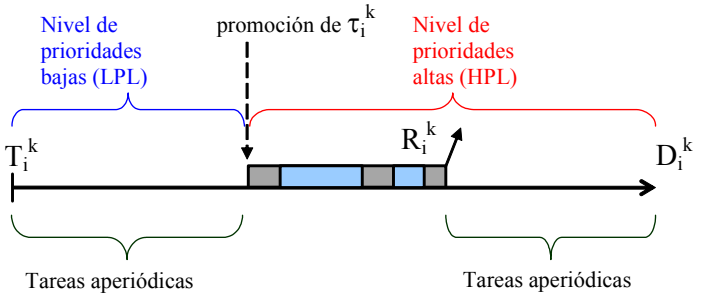


Figura 4.2: Una activación de una tarea periódica termina antes del máximo previsto, pudiéndose ejecutar tareas aperiódicas de nuevo

Solo si existen periodos de tiempo largos con tareas periódicas promocionadas, las tareas aperiódicas tendrán que esperar mucho tiempo y, por lo tanto, tendrán un tiempo de respuesta alto. Esto va a suceder en los siguientes casos:

a) cuando haya tareas periódicas muy grandes, sobretodo si son de baja prioridad, puesto que se ejecutan promocionadas durante mucho tiempo,

b) cuando el desfase entre las promociones absolutas sea siempre prácticamente nulo, puesto que las tareas periódicas reciben siempre la máxima interferencia de las más prioritarias,
c) cuando las promociones se dispersen de forma que las ejecuciones de sus tareas sean prácticamente secuenciales, puesto que cuando finaliza una se promociona la siguiente.

En estos casos son muy particulares. El primer caso podría mirar de solucionarse cambiando el orden de prioridades inicial. El resto de los casos depende de la sincronización de los periodos y no se puede hacer nada[18] si la aplicación no permite modificarlos.

Análisis de costes.

Puesto que este algoritmo no requiere conocimiento alguno sobre la ejecución de las tareas pasadas ni de las siguientes tareas, no necesita hacer complicados cálculos dinámicos ni requiere de tablas grandes para almacenar información, como en el caso de algoritmos como el *Slack Stealing* dinámico o estático respectivamente. Esta característica lo hace especialmente útil en aquellos conjuntos de tareas con hiperperiodos grandes. Así, los costes en tiempo de ejecución del algoritmo de planificación son del orden *O(log n)* para insertar los eventos de las promociones, además de algunas apropiaciones de cpu extras cuando no haya ninguna tarea aperiódica. En el otro extremo, cuando siempre hay tareas aperiódicas, las tareas periódicas se retrasan al máximo y, en este caso las apropiaciones son inevitables con cualquier algoritmo.

Propiedades.

Este algoritmo tiene las siguientes propiedades:

- garantiza todos los plazos de las tareas periódicas,
- logra un buen tiempo de respuesta medio para las tareas aperiódicas,

[18] Sin embargo, en un sistema multiprocesador sí se puede actuar, distribuyendo las tareas periódicas entre los procesadores de forma adecuada, como se verá en el próximo capítulo.

- requiere poco tiempo de proceso y poca memoria en tiempo de ejecución,
- puede usar el tiempo sobrante de las tareas periódicas, cuando éstas finalizan antes de su peor tiempo de ejecución,
- puede usar el 100% de la capacidad de proceso,
- se recupera de una forma rápida y controlada de las situaciones de sobrecarga (es decir, si alguna tarea periódica pierde su plazo, ésta será las de menor prioridad a la que ha provocado la sobrecarga).

Rendimiento.

Para comprobar el servicio que se puede prestar a las tareas aperiódicas con DP se realizaron extensos experimentos. Sin embargo, dado que el ámbito de la tesis es la planificación de multiprocesador, en este punto solo detallamos los resultados más significativos (véase la Figura 4.3).

Para la generación de los conjuntos de tareas se utilizaron los conjuntos de parámetros de la tablas PCT # 1 y PCT # 5 (Anexo B). Con el primero se generan periodos no armónicos mientras que con el segundo se generan periodos armónicos (todos son potencia de 2). La métrica utilizada es la media aritmética de los tiempos de respuesta medios de las tareas aperiódicas (*T. Respuesta Aperiódicas,* TRA). Los resultados se contrastan con los planificadores *Slack Stealing* (SS), un algoritmo óptimo entre los de prioridades fijas, y *Total Bandwidth* (TB), un algoritmo de prioridades dinámicas y con costes muy bajos. Los costes de memoria y de proceso de DP son equiparables a los de TB.

En la Figura 4.3-(a) se puede observar como DP obtiene los mejores resultados. El TRA que obtiene se ha reducido entre un 5% y un 20% respecto a SS. Sin embargo, en la Figura 4.3-(b), con periodos armónicos, los resultados se invierten y DP llega a incrementar el TRA en un 18% respecto a SS. En este caso TB mejora alrededor de un 1%, SS mejora un 7% en promedio. En cambio DP empeora un promedio del 10%. Esto es debido a que los periodos armónicos llevan a la des-sincronización de las promociones, resultando que siempre hay tareas promocionadas y evitando servir tareas aperiódicas. Los resultados en las gráficas inferiores, cuando la carga

aperiódica se mantiene al 25% y la carga periódica crece, confirman las observaciones anteriores pero con unas diferencias algo menores. En todo caso, el TRA obtenido con DP siempre es mejor que el obtenido con TB, el algoritmo comparable a nivel de costes.

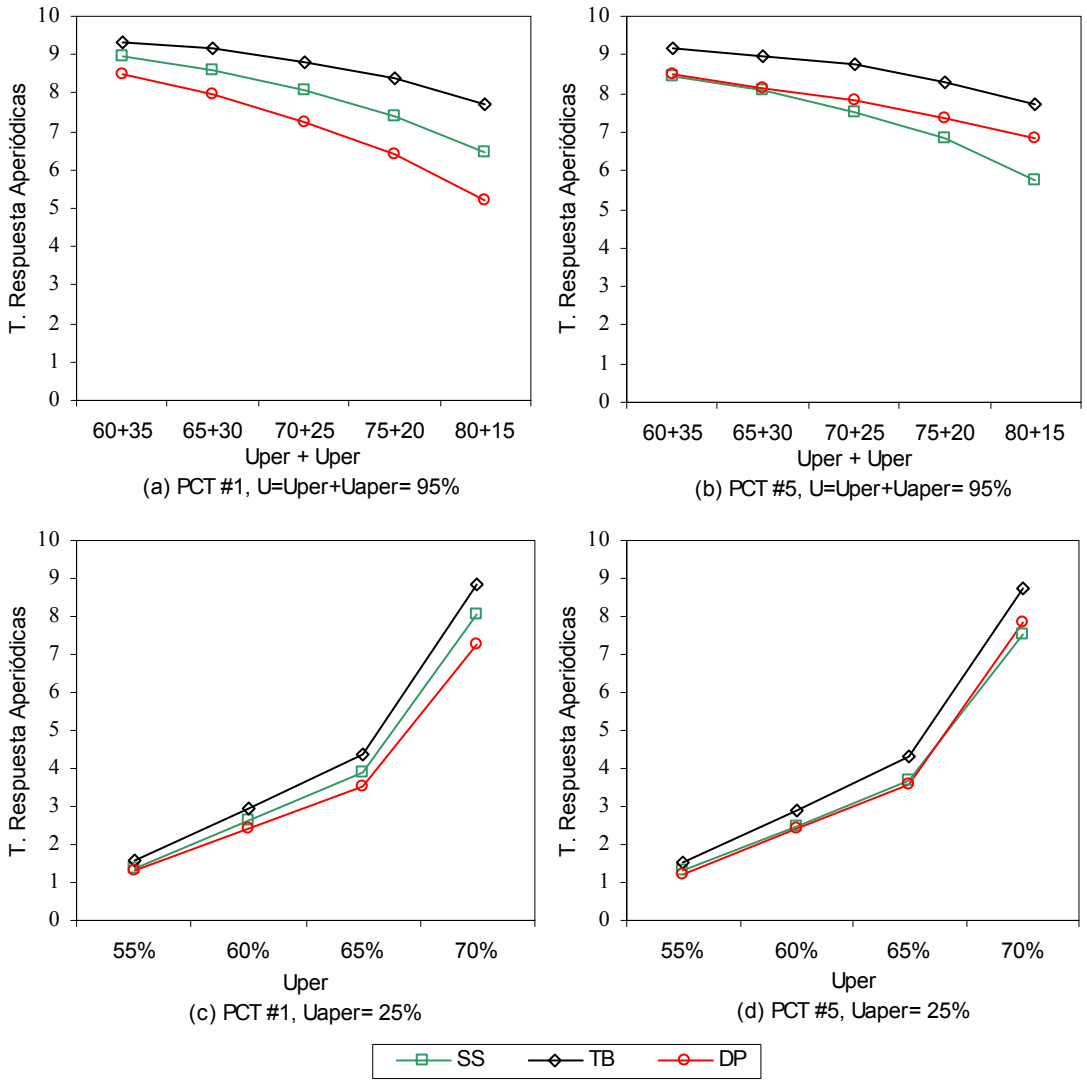


Figura 4.3: Experimentos con monoprocesadores. En las gráficas superiores se varió la distribución de la carga entre tareas periódicas y aperiódicas, manteniendo una utilización total del 95%. En las gráficas inferiores se varió la carga periódica, manteniendo la carga aperiódica al 25%.

En conclusión, la característica más destacable del algoritmo DP es que, aún tratándose de un sistema con prioridades fijas en tiempo de diseño, éste tiene parte de la flexibilidad de un sistema con prioridades dinámicas. El hecho de ser un sistema

con prioridades fijas hace asequible su validación formal. La variación de las prioridades de las tareas periódicas en dos niveles de prioridades confiere la suficiente flexibilidad para adaptarse a circunstancias imprevistas de antemano, como son el servicio a tareas aperiódicas o las situaciones de sobrecarga. En concreto, el tiempo de respuesta de las tareas aperiódicas es equiparable al de *Slack Stealing* y es mejor que el de *Total Bandwidth*, es decir el mejor rendimiento a un coste bajo. Sin embargo, se ha detectado una caída en el rendimiento en los conjuntos de tareas con periodos armónicos. En el Anexo A se detallan una serie de mejoras al algoritmo para aumentar su rendimiento. Estas mejoras permiten aumentar el rendimiento con los conjuntos de tareas armónicos pero añaden complejidad a la implementación. Por todo ello, nuestra valoración del algoritmo DP se resume en la Tabla 11 para poderlo comparar visualmente con los mejores algoritmos del capítulo de la introducción.

	Rendimiento	Complejidad computacional	Necesidades de memoria	Complejidad de implementación
Slack Stealing (Tabla 7, p.49)	☺	☹	☹	☹
Total Bandwidth (Tabla 8, p.50)	😐	☺	☺	☺
Dual Priority básico	☺/☺	☺	☺	☺
Dual Priority mejorado	☺	😐	😐/☺	😐/☺

Tabla 11: Nuestra valoración de las cualidades del algoritmo *Dual Priority* (leyenda: ☹ malo, 😐 bueno, ☺ muy bueno, ☺ excelente)

Dada la valoración tan positiva del algoritmo *Dual Priority* básico, en el siguiente apartado se propone su adaptación de la versión básica como planificador global en los sistemas multiprocesador. De las mejoras propuestas en solo se utilizarán aquellas que tengan un coste escalable con el número de procesadores. No obstante el efecto negativo de los periodos armónicos se tendrá en cuenta en los próximos experimentos.

4.3 Algoritmo Dual Priority para Multiprocesadores

La adaptación del DP para utilizarlo como planificador global en un sistema multiprocesador no es directa y, según nuestro conocimiento, no ha sido explorada hasta el momento. Recordemos que el algoritmo DP se fundamenta en los tiempos de promoción de las tareas periódicas y que este tiempo depende del peor tiempo de respuesta experimentado por dichas tareas. En el próximo apartado detallaremos las dificultades encontradas, para en el siguiente apartado detallar la solución propuesta.

4.3.1 Cálculo de los peores tiempos de respuesta

Para adaptar el algoritmo DP a multiprocesadores es necesario encontrar una forma de calcular el peor tiempo de respuesta para todas las tareas periódicas utilizando un planificador global. Recordemos que el algoritmo DP se fundamenta en los tiempos de promoción de las tareas periódicas y que este tiempo depende del peor tiempo de respuesta experimentado por dichas tareas. Sería óptimo hallar una ecuación equivalente a la ecuación 2.2, la cual solo es válida para monoprocesadores. Desafortunadamente esto no nos ha sido posible.

En primer lugar, la ecuación 2.2 no se puede generalizar al caso de los multiprocesadores porque tener m procesadores idénticos no equivale a tener un procesador m veces más rápido. Además, en nuestro marco de trabajo, una tarea periódica no se puede subdividir en subtareas para ejecutarse en paralelo en varios procesadores. En segundo lugar, las aproximaciones para hallar un límite superior son demasiado pesimistas dando lugar a promociones demasiado inmediatas e invalidando el método como prueba de planificabilidad. En tercer lugar, el método de la simulación para hallar los peores tiempos de respuesta no es aplicable, porque supone que no se va a cambiar para nada la planificación y en realidad es lo que se pretende hacer con DP.

A continuación analizamos tres aproximaciones al cálculo de los peores tiempos de respuesta. El primer método es analítico, el segundo es algorítmico y el tercero és simulado. Este último es el que a priori se creía más exacto y por eso se ha utilizado como referencia para los otros dos. Para mas detalles véan el apartado 4.3.1.3.

4.3.1.1 Ecuación para hallar un límite superior al WCRT

Björn Anderson en [AnJ00,And03] propone una forma recurrente para aproximar el cálculo del peor tiempo de respuesta de las tareas periódicas ejecutadas globalmente en un multiprocesador (véase Ecuación 4.1), aunque no llega a analizar su rendimiento y su aplicabilidad. Si T_j y C_j son el periodo y el peor tiempo de ejecución respectivamente de la tarea τ_j, *hp(i)* es el conjunto de tareas de mayor prioridad que τ_i y *m* es el número de procesadores, entonces calculan R_i^{n+1} , el peor tiempo de respuesta para τ_i en el paso n+1 en función del hallado en el enésimo paso. La iteración se detiene cuando se estabiliza ($R_i^{n+1}=R_i^n$), o cuando la tarea perdería su plazo ($R_i^{n+1}>D_i$). Si el resultado se estabiliza y es planificable entonces $WCRT_i= R_i^{n+1}$. Es decir, la forma de hallarlos es parecida a la de la ecuación 2.2 pero se supone que la velocidad de proceso es *m* veces superior.

$$R_i^{n+1} = C_i + \frac{1}{m} \sum_{j \in hp(i)} \left(\left\lfloor \frac{R_i^n}{T_j} \right\rfloor C_j + 2\,C_j \right)$$

Ecuación 4.1: Cálculo analítico del límite superior del peor tiempo de respuesta (WCRT) de una tarea periódica en un sistema multiprocesador con planificador global.

Esta ecuación tiene en cuenta el cómputo de la tarea τ_i más la ejecución de la interferencia de las tareas de mayor prioridad en paralelo en *m* procesadores. Se basa en la idea que el peor de los casos se experimenta cuando la interferencia de las tareas de mayor prioridad se solapa o, dicho de otra forma, cuando el tiempo disponible para que se ejecute se solapa, ya que la tarea no se puede ejecutar en paralelo en varios procesadores. Así, a interferencia máxima en la Ecuación 4.1 es la misma que en el caso de la ecuación 2.2 más una tarea extra de cada una de las de prioridad superior. Como sabemos la ganancia o el *speedup* con *m* procesadores normalmente es inferior a m[19], luego la interferencia real será algo mayor y de ahí que se añadan estas tareas extra. Es decir, en realidad el multiprocesador no tendría tiempo de terminar todas las tareas de mayor prioridad y R^n sería mayor, pudiendo llegar a incluir una nueva activación de cada tarea de mayor prioridad. De esta forma se calcula un límite superior al peor tiempo de respuesta muy pesimista, lo cual no lo hace apropiado para calcular las promociones en un DP global.

[19] Lley de Amdahl: véase la observación en el segundo párrafo de la página 54

Para comprobar cuán pesimista es el resultado de utilizar demuestra la Ecuación 4.1 se han realizado una serie de experimentos de manera que se han generado conjuntos de tareas sintéticos usando los parámetros de la Tabla 15 en el Anexo B, se han computado los WCRT usando la Ecuación 4.1 y se han contrastado con los WCRT medidos utilizando una simulación de GRM a lo largo de un hiperperiodo. En la Figura 4.4 se puede observar como el promedio de WCRT obtenidos usando la Ecuación 4.1 dobla sobradamente el resultado obtenido vía la simulación de GRM. De hecho, la predicción es peor cuando la carga es más baja y cuando existen tareas pesadas, llegando a ser en este punto muy pesimista. De hecho, si se utilizase esta ecuación como prueba de planificabilidad, el resultado sería que en la mayoría de ocasiones, al ser tan pesimista, daría una respuesta negativa cuando en realidad el conjunto de tareas podría ser planificable.

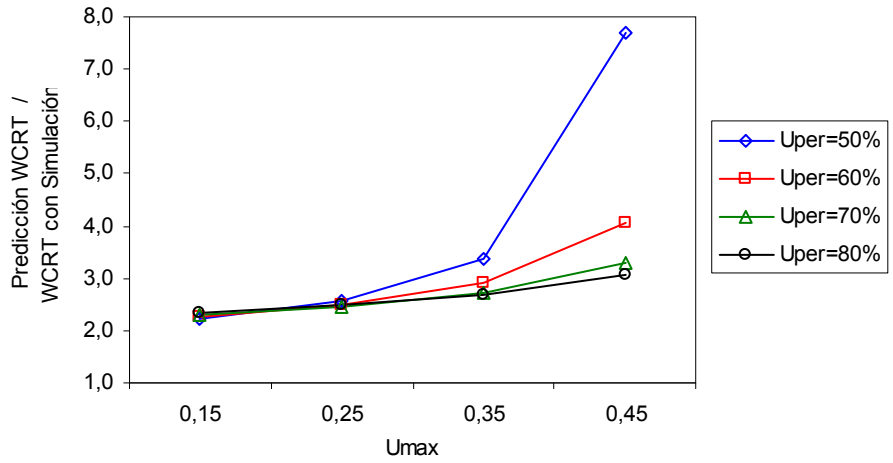


Figura 4.4: Comparación de la predicción del cómputo del WCRT usando la Ecuación 4.1 versus el resultado obtenido vía la simulación de GRM

Además de ser una forma muy pesimista de calcular los WCRT tenemos dudas sobre la validez de la Ecuación 4.1. Sus autores basaron su desarrollo en un teorema de [LMM98] (véase la ecuación 2.10 en la página 66) pero en realidad la demostración de la ecuación 2.10 no demuestra la Ecuación 4.1. El siguiente contra-ejemplo demuestra que existen casos en los que se cumple la Ecuación 4.1 y no se cumple la ecuación 2.10. Supongamos que tenemos *m* tareas de prioridad alta con C_j=3 i T_j=11 más una tarea extra con C_{m+1}=4 i T_{m+1}=12. Entonces existe un R_{m+1}=10 que satisface la Ecuación 4.1 ($4 \leq 10 - 1/m(m(0+2)3) = 10–6$). En cambio, aplicando la ecuación 2.10 obtenemos que $C_{m+1} \leq 3$ ($12 - 1/m(m(1+2)3) = 12–9$). En realidad ambas

ecuaciones establecen un límite para C_i pero la ecuación 2.10 es más restrictiva, porque tiene en cuenta toda la interferencia de las tareas más prioritarias a lo largo de un periodo (T_i), en cambio en la Ecuación 4.1 se tiene en cuenta la interferencia hasta que R_i se estabiliza (y $R_i \leq T_i$). En definitiva, la Ecuación 4.1 no está demostrada y la ecuación 2.10, que si lo está, no se puede usar porque es todavía más pesimista.

Dada la dificultad de encontrar una fórmula para calcular los peores tiempos de respuesta uno se podría plantear recurrir a la simulación para hallarlos. Sin embargo, como veremos en el apartado 4.3.1.3, este método también es deficiente en los sistemas de tiempo real estrictos. Los planificadores globales para multiprocesadores existentes actualmente (véase el apartado 2.2) tienen unas limitaciones muy importantes.

En primer lugar, no hay ningún algoritmo óptimo, es decir, no todos los algoritmos pueden planificar todos los conjuntos de tareas planificables por algún otro planificador. Esto dificultaría el elegir uno de ellos. En segundo lugar, los algoritmos con prueba de planificabilidad basada en la simulación no son válidos para calcular el peor tiempo de respuesta, pues requieren que el orden y el procesador de ejecución sean fijos y no permiten introducir ninguna variación. Como ya hemos mencionado en el apartado 2.2, la ejecución en sistemas multiprocesadores puede experimentar ciertas anomalías. En concreto, es bien conocida la anomalía de Richardson [Bur91], la cual establece que si se reduce el tiempo de ejecución de alguna tarea esto puede cambiar el orden y lugar de ejecución de las tareas restantes y puede dar lugar a una ejecución total más larga. En otras palabras, si algunas tareas no se ejecutan hasta su WCRT, cosa a priori bastante razonable, pueden variar toda la planificación incluso dando lugar a tiempos de respuesta perores a lo previsto inicialmente en la simulación. En tercer lugar, los algoritmos con prueba de planificabilidad basada en la utilización tienen unos límites de utilización muy bajos. Por ejemplo, el planificador de prioridades estáticas RM-US(m/(3m-2)) [ABJ01] tiene un límite del 33% de carga periódica y el planificador de prioridades dinámicas de GFB [GFB02] con una tarea pesada (U_{max}=0,5) el límite es del 50%. Para superar estos límites se requiere de una simulación a lo largo de un hiperperiodo del conjunto de tareas para

comprobar si los plazos se cumplen, pero entonces el método también queda invalidado como en el caso de los algoritmos con prueba de planificabilidad basada en la simulación por los mismos motivos.

En esta tesis hemos utilizado dos aproximaciones para calcular estos peores tiempos de respuesta: (i) hemos desarrollado un algoritmo para calcular en tiempo de diseño un límite superior al peor tiempo de respuesta para cada tarea periódica; (ii) hemos recurrido a la simulación para encontrarlos. A continuación haremos el estudio de ambos métodos y compararemos sus ventajas e inconvenientes.

4.3.1.2 Algoritmo para aproximar los WCRT

En este apartado se describe un algoritmo para aproximar el cálculo de los peores tiempos de respuesta de las tareas periódicas y se analiza su rendimiento. El Algoritmo 4-3 aproxima estos WCRT de forma que intenta no ser muy pesimista, como ocurría en el aparado anterior con el uso de la Ecuación 4.1 que ha resultado poco efectiva, puesto que rechazaba demasiados conjuntos de tareas y no podía ser utilizada para el cálculo de las promociones de DP porque daría lugar a promociones casi inmediatas que proporcionarían muy poca flexibilidad a DP.

```
Algoritmo aproximar_wcrt() retorna booleano
1: ordenar tareas por periodo creciente
2: ordenar tareas de igual periodo por calculo creciente
3: para cada tarea τi
4:    Ri= Ci
5:    si ( i > m )
6:          Ri= min{m_max_{j∈HP(i)}{Rj}}
7:          Ri= Ri + ⌈Ci*(1+Σ_{τj∈HP(i)}Uj)/m)⌉
8:    fin_si
9:    si ( Ri > Di ) entonces
10:         retorna test_fallido
11: fin_para
12: retorna test_correcto
```

Algoritmo 4-3: Algoritmo para aproximar el peor tiempo de respuesta ($WCRT_i$, R_i) de cada una de las tareas periódicas del sistema

Las *m* tareas más prioritarias no tienen interferencia de ninguna tarea y por lo tanto la respuesta no depende de ninguna otra tarea (líneas 4 y 5). Para el resto de tareas, la aproximación se hace de la siguiente manera. El peor tiempo de inicio de la ejecución de una tarea τ_i dependerá de los *m* peores (máximos) tiempos de respuesta de las tareas más prioritarias que τ_i (línea 6). En concreto, τ_i podrá empezar cuando termine la que tenga el menor tiempo de respuesta (de los *m* peores). Una vez τ_i empiece a ejecutarse no se puede saber a ciencia cierta si se producirá una expulsión ni cuando ésta se produciría. De hecho, al ser τ_i la tarea menos prioritaria ejecutándose, con la llegada de cualquier tarea más prioritaria se produciría su expulsión y no la de una tarea en otro procesador, como ocurría con el cálculo de los WCRT anteriores. Dicho de otra forma, no podemos saber la interferencia que sufrirá por parte de tareas más prioritarias. Lo único que podemos intuir es que cuanto mayor sea C_i mayor probabilidad de recibir más interferencia habrá. Es decir, la interferencia será proporcional a C_i. Así pues, en la línea 7, aproximamos el trabajo a ejecutar por el multiprocesador como la carga producida por las tareas más prioritarias mientras dure la ejecución de la tarea ($C_i \Sigma_{\tau j \in HP(i)} U_j$) de forma concurrente en los *m* procesadores junto con la propia tarea (($C_i + C_i \Sigma_{\tau j \in HP(i)} U_j$)/m). Sabemos que el *speedup* va a ser menor que *m* pero también sabemos que la carga de las tareas más prioritarias normalmente será inferior (excepto en el caso de la sincronización de las tareas, por ejemplo en el hiperperiodo). Además, en la línea 6 ya se ha contemplado una gran parte de la interferencia de las tareas más prioritarias. Así pues, esperamos que ambos términos se compensen en cierta medida y que la aproximación sea suficientemente buena. En el límite, cuando $m \to \infty$ y la carga periódica del sistema es grande, $C_i/m \to 0$ y $(\Sigma_{\tau j \in HP(i)} U_j)/m \to 1$, luego $R_i = \min\{m\_max_{j \in HP(i)}\{R_j\}\} + C_i$, que podría ser una alternativa más optimista para línea 7.

Para comprobar si la aproximación de este algoritmo es buena se han realizado dos experimentos generando conjuntos de tareas sintéticos usando los parámetros de la Tabla 15 (Anexo B), se han computado los WCRT usando el Algoritmo 4-3 y se han contrastado con los WCRT medidos utilizando la simulación *Global Rate Monotonic*

a lo largo de un hiperperiodo. Los resultados se muestran de la Figura 4.5 a la Figura 4.8.

Experimento 1

El objetivo del primer experimento es comprobar las diferencias entre los dos métodos de calcular los WCRT cuando la carga periódica varía. Para ello se han generado desde conjuntos de tareas poco cargados (50%) a muy cargados (80%) y en cada caso se ha probado desde tener tareas pequeñas (U_{max}=0,15) hasta tener tareas bastante grandes (U_{max}=0,45) puesto que esto puede influir mucho en los WCRT.

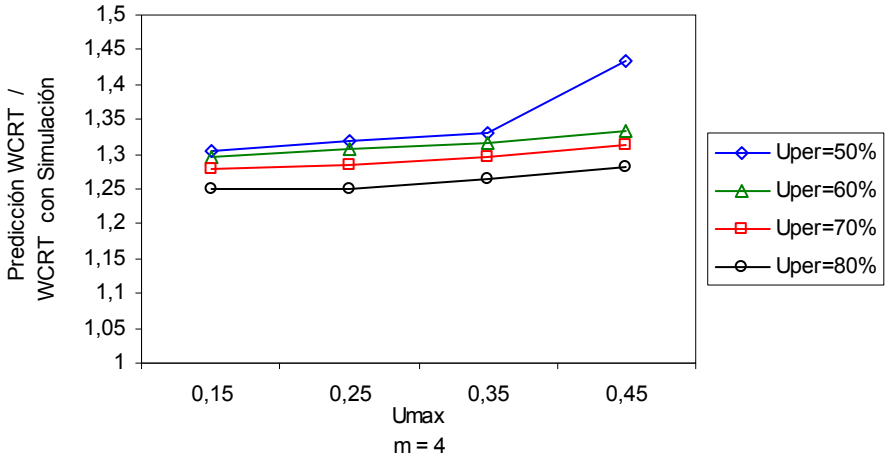


Figura 4.5: Comparación de la predicción del cómputo del WCRT usando el Algoritmo 4-3 versus el resultado obtenido vía la simulación GRM cuando varía la carga periódica entre 50% y 80% y el factor máximo de utilización entre 15% y 45%.

En la Figura 4.5 se puede observar el promedio de WCRT obtenidos usando el Algoritmo 4-3 divididos por los obtenidos con la simulación. Como puede observarse el error relativo de las medias es mucho menor que en el caso anterior, cuando usábamos la Ecuación 4.1: experimenta un incremento del 30% mientras que antes doblaba sobradamente. Además el error es bastante estable, independientemente de la carga del sistema y de la carga máxima de las tareas (U_{max}). De hecho, cuanto mayor es la carga, menor es el error. Esto nos permitiría usar este método como prueba de planificabilidad aproximada, puesto que no es tan pesimista como la Ecuación 4.1. Efectivamente, en la Figura 4.6-(a) se puede observar como el porcentaje de conjuntos de tareas rechazados que obtenemos es alto

solo para conjuntos de tareas con U_{max} muy grandes y cargas periódicas del sistema muy altas, siendo la prueba de planificabilidad útil en el resto de los casos.

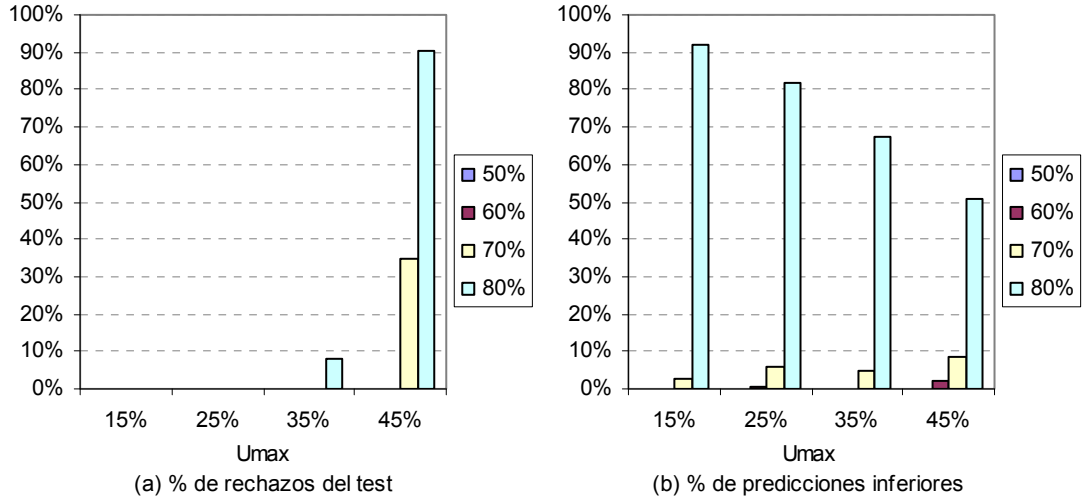


Figura 4.6: Comportamiento del Algoritmo 4-3 cuando varía la carga periódica entre 50% y 80% y el factor máximo de utilización entre 15% y 45%. (a) Porcentaje de conjuntos de tareas rechazados. (b) Porcentaje de conjuntos de tareas en los que se hace una o más predicciones inferiores al WCRT hallado con la simulación GRM.

Cabe remarcar que el cómputo de los R_i depende de los R_j de las tareas más prioritarias y, por lo tanto, existe un efecto de error acumulativo. Las predicciones para las tareas más prioritarias suelen ser muy acertadas. Para conseguir errores relativos no excesivos sería necesario que el algoritmo no obtuviese errores grandes en las tareas de prioridad media, para evitar la progresiva acumulación de error en las tareas de prioridad baja. Luego, si un algoritmo intentase ajustar el error de exceso en las tareas de prioridad alta-media podría darse el caso de cometer un error por defecto en las tareas de baja prioridad. En definitiva, este método no es del todo fiable, puesto que, como el algoritmo es aproximado y no se pretendía que fuese muy pesimista, en algunas ocasiones la predicción del WCRT es inferior a la hallada con el método de la simulación. En consecuencia, puede darse el caso en que la prueba de planificabilidad sea satisfactoria y, sin embargo, durante la ejecución alguna tarea puede incumplir su plazo.

Dado el diseño pesimista del algoritmo, los WCRT calculados en promedio son peores que los obtenidos con la simulación, pero como no es un método exacto también puede suceder que se obtengan algunos WCRT menores: es lo que llamamos

predicciones inferiores. En la Figura 4.6-(b) se puede constatar que, para los conjuntos de tareas generados aleatoriamente, solo cuando la carga del sistema es del 80% el porcentaje de predicciones inferiores al WCRT real, o sea erróneas, es alto. Cuando la carga es del 70%, este porcentaje se mantiene por debajo del 10%. Para el resto de los casos la probabilidad de cometer un error es ínfima. Esto, junto con el hecho que usualmente sean las tareas de menor prioridad las que obtengan una predicción inferior (y consecuentemente las que podrían llegar a perder alguno de sus plazos), hace que el método pueda ser aplicable en sistemas de tiempo real no estricto.

Experimento 2

En este experimento se ha evaluado el comportamiento del algoritmo cuando el número de procesadores varía. Para ello se ha fijado una carga periódica relativamente alta (70%) y también se ha variado el factor de utilización U_{max} desde tareas pequeñas a tareas grandes. En la Figura 4.7 se puede observar como las predicciones son mejores cuando el número de procesadores aumenta. También es destacable que el error en las predicciones se mantiene constante cuando varía la carga máxima de las tareas individualmente (U_{max}).

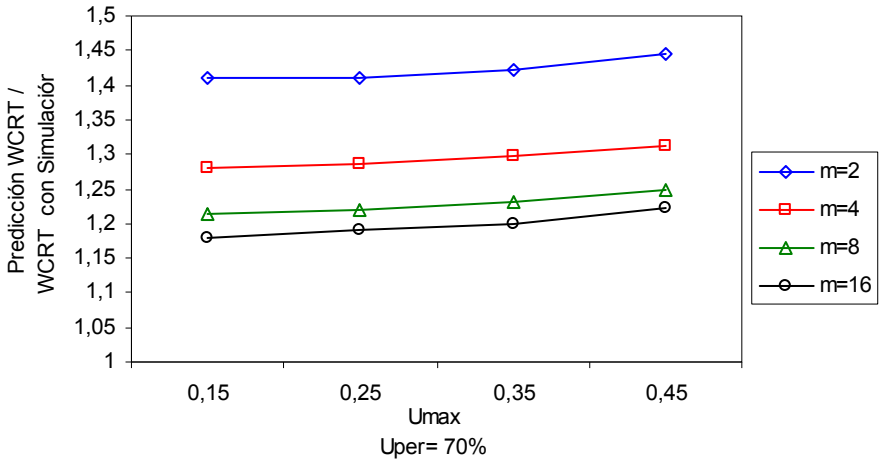


Figura 4.7: Comparación de la predicción del WCRT usando el Algoritmo 4-3 versus el resultado obtenido vía la simulación GRM cuando varía el número de procesadores.

En la Figura 4.8-(a) se puede observar como el rechazo de conjuntos de tareas solo es significativo cuando existe alguna tarea grande. Incluso en estas circunstancias el porcentaje de rechazos disminuye cuando el número de procesadores aumenta. Sin

embargo en la Figura 4.8-(b) podemos apreciar que el porcentaje de predicciones inferiores con más procesadores aumenta cuando U_{max} se hace menor (y, por lo tanto, la carga de todas las tareas se iguala).

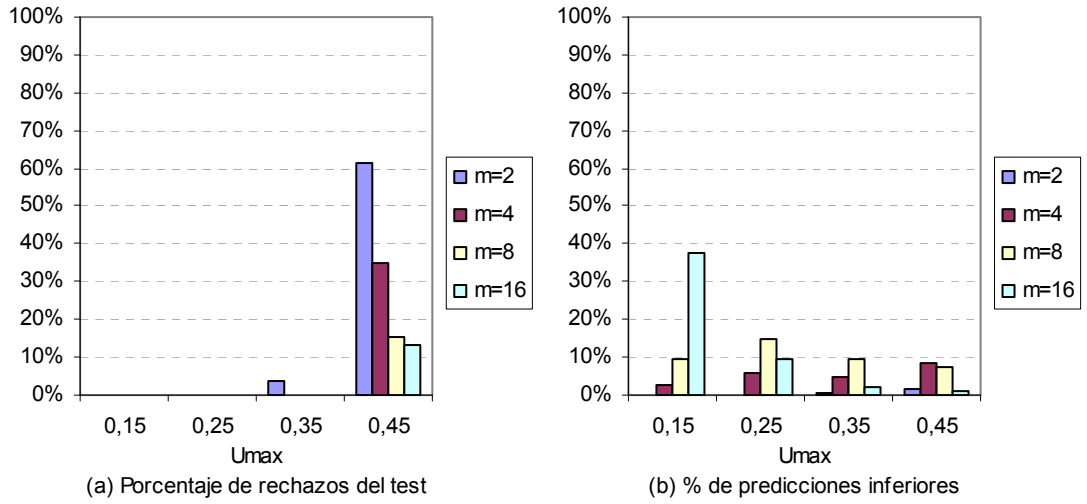


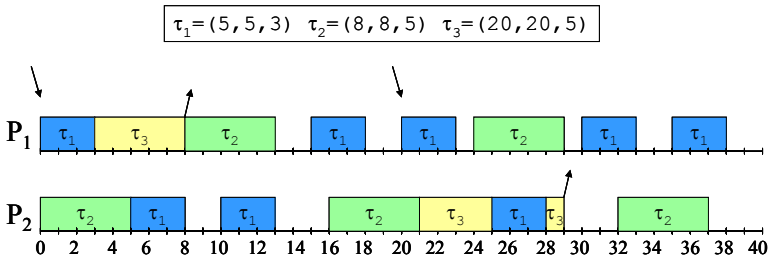
Figura 4.8: Comportamiento del Algoritmo 4-3 con una carga periódica del 70% cuando varía el número de procesadores. (a) Porcentaje de conjuntos de tareas rechazados. (b) Porcentaje de conjuntos de tareas en los que se hace una o más predicciones inferiores al WCRT hallado con la simulación GRM.

Es interesante remarcar que el porcentaje de rechazos y el porcentaje de predicciones inferiores son dos métricas antagonistas. Sabemos que la interferencia recibida de las tareas más prioritarias en realidad puede ser muy inferior a $C_i \Sigma_{\tau j \in HP(i)} U_j/m$. Esto es debido al hecho de que las tareas periódicas no se pueden dividir en subtareas más pequeñas ni paralelizarlas. Así, por ejemplo, si tuviésemos *m*-1 tareas con carga grande tendríamos que la interferencia computada sería grande, cuando en realidad la interferencia podría ser nula, puesto que las grandes solo se ejecutarían en los otros procesadores. Así, variando la forma de computar la interferencia, se pueden diseñar variaciones de este algoritmo que se aproximen mejor y rechacen menos conjuntos de tareas pero entonces aumentaría el porcentaje de predicciones inferiores, y, por lo tanto erróneas. También se pueden diseñar variaciones de este algoritmo que obtengan un porcentaje de predicciones inferiores muy pequeño pero entonces las predicciones irían en alza y el porcentaje de rechazos de conjuntos de tareas sería mayor, invalidándolo como prueba de planificabilidad. En estos dos experimentos se ha comprobado que el algoritmo propuesto mantiene un buen equilibrio entre ambas métricas.

4.3.1.3 Simulación para hallar los WCRT

En este caso, en tiempo de diseño se realiza una simulación de la ejecución de las tareas periódicas a lo largo de un hiperperiodo usando un planificador global, como por ejemplo GRM[20], y haciendo que cada activación requiera ejecutarse el tiempo máximo. De esta forma se miden todos los tiempos de respuesta de las tareas periódicas y se puede calcular su máximo.

Es importante destacar que en el caso de multiprocesadores, a diferencia de los monoprocesadores, el instante crítico no tiene porque ser el instante inicial, cuando se activan todas las tareas simultáneamente. Obsérvese que la ecuación 2.2 se basa en este principio y por lo tanto queda invalidada una metodología que derive de dicha ecuación. También, por este motivo, es necesario que la simulación para hallar los WCRT no finalice al terminar la ejecución de la primera activación de cada tarea periódica, sino que se debe simular a lo largo de todo un hiperperiodo. La Figura 4.9 ilustra este fenómeno:



Rate Monotonic: $R_3^1 = 8$, $R_3^2 = 9$

Figura 4.9: Ejemplo donde el WCRT no se produce cuando t=0 con un planificador RM global. Las flechas descendentes marcan la activación de la tarea 3 y las ascendentes su finalización.

En la Figura 4.9 se muestra la planificación del conjunto de tareas del ejemplo usando un GRM en dos procesadores. Se puede observar que la primera activación de la tarea τ_3 finaliza en el instante t=8, con un tiempo de respuesta de 8 unidades de tiempo, mientras que su segunda activación (en el instante t=20) termina en el instante t=29, resultando un tiempo de respuesta de 9 unidades, una unidad superior al obtenido en el instante supuestamente crítico. Esto es debido al hecho de que esta segunda activación recibe la interferencia de una activación de τ_2 que ya existía antes

[20] Se debe utilizar el mismo método que se usará en tiempo de ejecución.

de la segunda activación de τ_3 (es decir, esta segunda activación de τ_3 recibe la interferencia de 2 activaciones de τ_2 cuando la primera solo recibía de una).

A pesar de simular a lo largo de todo un hiperperiodo, este método también ha resultado deficiente. En algunos casos se han dado tiempos de respuesta peores que los WCRT hallados en la simulación cuando se han introducido algunos desfases debidos al servicio a otros tipos de tareas (por ejemplo, tareas aperiódicas). Veámoslo con el ejemplo de la Figura 4.10.

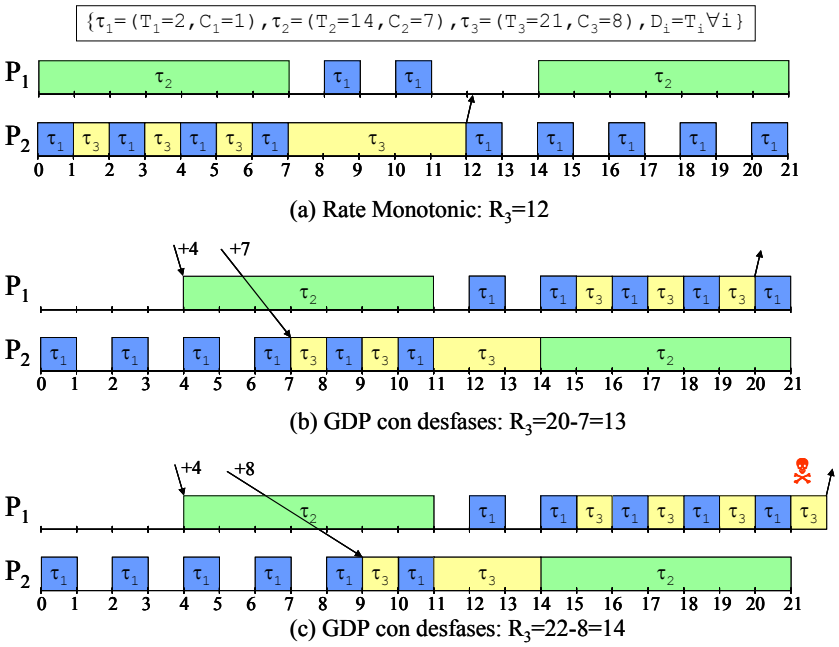


Figura 4.10: (a) hallar WCRT con la simulación GRM (ej. R_3=12), (b) usando estos valores para la ejecución con GDP y experimentar un WCRT mayor (ej. R_3=13), (c) usar estos valores para la ejecución con GDP y no terminar a tiempo (ej. R_3=22 > D_3=21).

En la Figura 4.10-(a) se detalla la simulación de la ejecución con GRM para medir los WCRT. Para la tarea τ_3 el resultado es R_3=12. Esto permitirá retrasar su promoción en 9 unidades (P_3=21-12). En la Figura 4.10-(b) se detalla la ejecución con GRM pero las tareas τ2 y τ3 se les ha aplicado un desfase de 4 y 7 unidades de tiempo respectivamente (que podrían ser debidos al servicio a otras tareas). En este caso el tiempo de respuesta que experimenta la tarea τ_3 es R_3= 20 –7 = 13, una unidad superior a lo previsto. Pero aún es peor en el caso de la Figura 4.10-(c) donde τ_3 es retrasada 8 unidades (recuérdese que el máximo teórico era P_3=9) y finalizando en el instante t=22 (con R_3=14) sobrepasando así su límite (D_3=21).

Podemos establecer que ninguna de las siguientes tres premisas válidas en monoprocesadores para DP se cumple en multiprocesadores:

1) *Ninguna tarea aperiódica se ejecuta mientras haya alguna tarea periódica promocionada*. Esto no se cumple en multiprocesadores debido justamente a su capacidad de procesamiento paralelo real.

2) *Retrasar la ejecución de una tarea periódica hasta su promoción no es ningún problema, puesto que este cambio en los órdenes de ejecución nunca podrá ser peor que en el instante crítico*. Esto no es válido porque el número de desplazamientos y combinaciones posibles en un multiprocesador es intratable en el cálculo del WCRT en tiempo de diseño.

3) *El WCRT de cada tarea periódica incluye la interferencia producida por todas las tareas de mayor prioridad*. Esto no es válido en multiprocesadores porque la ejecución en paralelo de tareas más prioritarias oculta su presencia y no se tienen en cuenta en el cálculo del WCRT. En tiempo de ejecución, al atrasar o avanzar la finalización de algunas tareas puede cambiar la asignación de tareas a procesadores y hacer que una o algunas tareas que no teníamos contempladas interfieran.

En resumen, este método para hallar los WCRT no es exacto y no se puede utilizar para calcular las promociones para un planificador global con DP con garantías. Quizás se podrían realizar simulaciones más largas e introduciendo desfases aleatorios para hallar unos WCRT más ajustados, pero aún así no existirían las garantías necesarias para un sistema de tiempo real estricto.

4.3.2 Global Dual Priority para STR laxos (GDP)

En el apartado anterior llegamos a la conclusión que no se pueden calcular los WCRT exactos con ninguno de los tres métodos planteados. Por lo tanto, no se pueden usar para calcular las promociones y usar GDP en un sistema de tiempo real estricto. Sin embargo, según los experimentos que hemos realizado con anterioridad, con el método de la simulación y con el algoritmo aproximado parece que el número de plazos incumplidos no es muy grande[21]. Esto significa que el método se puede aplicar a sistemas de tiempo real laxos, aquellos en los que incumplir algunos plazos no es crítico.

En este apartado proponemos usar el planificador GDP_A, el cual calcula las promociones de forma aproximada con el Algoritmo 4-4, o el planificador GDP_S, el cual calcula las promociones con una simulación de un hiperperiodo en tiempo de diseño. Ambos son susceptibles de calcular erróneamente las promociones y por eso corregirán los errores de forma dinámica en tiempo de ejecución.

Con GDP_A en tiempo de diseño se aplica el método explicado en el apartado 4.2.1.1 pero se usa el Algoritmo 4-4, derivado del Algoritmo 4-3, para calcular las promociones de las tareas periódicas. Como el algoritmo de cálculo de los WCRT es aproximado y generalmente pesimista, puede suceder que los WCRT sean demasiado grandes (en cuyo caso el Algoritmo 4-3 finalizaría y diría que el conjunto de tareas no es planificable). Ahora, con el Algoritmo 4-4 no se retorna un error y sino que se asigna una promoción inmediata (línea 10) y se continua con la siguiente tarea. Al final el algoritmo no dice si el conjunto es planificable o no.

[21] Los experimentos del presente apartado ya servirán para cuantificarlos.

```
Algoritmo Calcular_Promociones_GDP
1: ordenar_tareas_por periodo_y_calculo_crecientes();
2: para cada tarea τi
3:     Ri= Ci
4:     si ( i > m )
5:           Ri= min{m_max_{j∈HP(i)}{Rj}}
6:           Ri= Ri + ⌈Ci*(1+Σ_{τj∈HP(i)}Uj)/m)⌉
7:     fin_si
8:     si ( Ri < Di ) entonces
9:           Yi= Di - Ri
10:    sino  Yi= 0
11: fin_para
```

Algoritmo 4-4: Algoritmo para calcular los tiempos relativos de promoción de las tareas periódicas para GDP_A utilizando la aproximación al WCRT del Algoritmo 4-3

En tiempo de ejecución GDP_A y GDP_S aplican el método explicado en el apartado 4.2.1.1 pero con *m* procesadores y según la Figura 4.11.

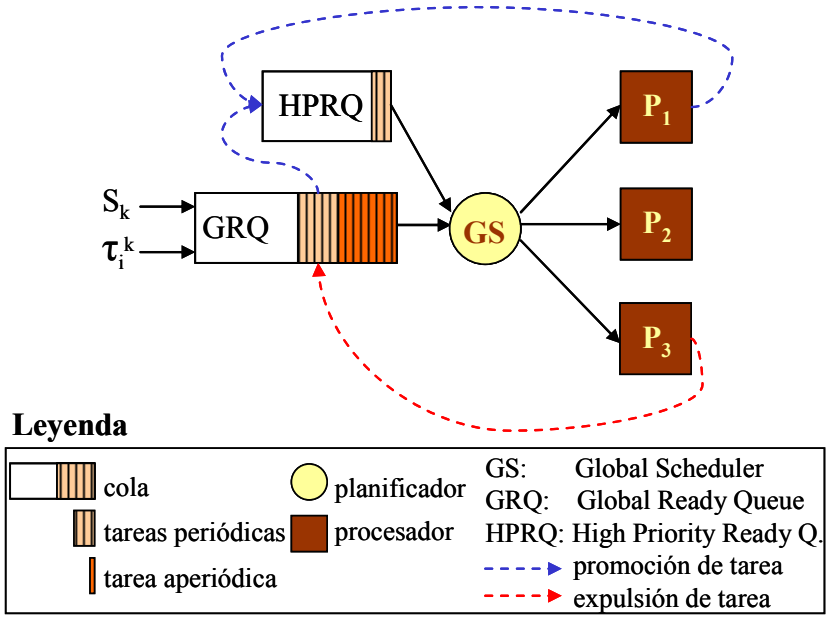


Figura 4.11 Esquema del planificador global GDP. Todas las tareas se encolan en la cola global GRQ. Las tareas periódicas promocionadas lo hacen en la cola global HPRQ. El planificador GS primero toma las tareas de HPRQ y luego las de GRQ.

Cuando cualquier tarea se activa se encola en una cola de preparados global (GRQ). En ella, las tareas periódicas tienen una prioridad inferior a las aperiódicas. El orden relativo entre estas últimas es FIFO. De esta forma se les proporcionará un buen

servicio. Cuando una tarea periódica es promocionada su prioridad aumenta de nivel y toma preferencia, puesto que pasa al una cola de preparados global de alta prioridad (HPRQ). Una vez promocionada una tarea periódica solo recibe la interferencia de otras tareas periódicas promocionadas de forma que su plazo pueda cumplirse. Sin embargo, como ahora los métodos para calcular las promociones no son exactos puede suceder que se promocionen algunas tareas demasiado tarde y que pierdan su plazo. Así, en tiempo de ejecución se debe hacer una monitorización del incumplimiento de los plazos y hacer un ajuste dinámico de los tiempos de promoción de las tareas. Cada vez que se incumpla un plazo se puede saber el tiempo de cómputo restante de la tarea y se debe adelantar los tiempos de promoción de las subsiguientes activaciones de la tarea en cuestión de la siguiente forma:

$$Y_i = \max \{ 0 , Y_i - rem_i^k \}$$

donde

Y_i tiempo de promoción (absoluto) de la tarea periódica τ_i

rem_i^k tiempo de cómputo remanente de la k-ésima activación de la tarea τ_i

Cuando el tiempo de cómputo restante es mayor que el tiempo absoluto de promoción las nuevas promociones se planifican de forma inmediata (Y_i=0). Puede suceder que esto sea suficiente para cumplir los plazos de las futuras activaciones pero también podría suceder que esta tarea no fuese planificable. No existen garantías pero es poco probable, puesto que una promoción inmediata implica una alta prioridad y dado que un sistema multiprocesador tiene una gran capacidad de absorción de la interferencia de otras tareas.

Los costes del planificador en cuanto a requerimientos de memoria son: una cola RQ para las tareas promocionadas de longitud finita y conocida (el número de tareas periódicas); una cola de longitud el número de tareas periódicas más el número de tareas aperiódicas; una cola de eventos de comprobación de los plazos y una cola de eventos promociones, ambas de longitud el número de tareas periódicas. Los costes en tiempo de ejecución son los propios de atender a los eventos, los cambios de cola de las tareas promocionadas. En todo caso, el planificador no debe hacer ningún cálculo intensivo.

4.3.3 Evaluación experimental del rendimiento

El objetivo de los siguientes experimentos es mesurar el tiempo de respuesta medio de las tareas aperiódicas obtenido con GDP_S y GDP_A y compararlo con los resultados obtenidos con los planificadores usados en los capítulos anteriores. En la Tabla 12 se resumen las principales características de los planificadores utilizados. Además, también se estudiará como afecta la interferencia de las tareas aperiódicas con los ajustes dinámicos de las promociones de las tareas periódicas y se comprobará que método para calcular las promociones es mejor, GDP_S o GDP_A.

Planificador	**Observación**	**Características**	**Más detalles**
SS+NF	*Slack Stealing* *Next Fit*	PF/AD óptimo en monoprocesadores mejor distrib. Aperiódicas usa mucha información	3.2.2.1 página 92
TB+SD	*Total Bandwidth* *Shortest Deadline*	PF/AD bueno en monoprocesadores usa muy poca información	3.2.2.2 página 95
MTB	*Multiprocessor Total Bandwidth*	PD/AD usa muy poca información	2.4.2.7 página 73
GDP_A	*Global Dual Priority* WCRT aproximados con Algoritmo	PD/AD usa información casi nula	4.3.2 página 140
GDP_S	*Global Dual Priority* WCRT hallados vía Simulación	PD/AD usa información casi nula	4.3.2 página 140

Tabla 12: Tabla resumen de los planificadores usados en las simulaciones de esta sección

Descripción de los parámetros de los conjuntos de tareas.

Para los experimentos se han utilizado conjuntos de tareas generados sintéticamente y caracterizados con unos parámetros equivalentes a los usados en el capítulo anterior. Sin embargo, se ha fijado el número de procesadores en 4 porqué en el apartado 4.3.1.2 se observo que las predicciones con un número mayor de procesadores ya son suficientemente buenas.

Los conjuntos de tareas periódicas se han generado utilizando los parámetros de la PCT # 1 del Anexo B con las cargas de los procesadores equilibradas. Los periodos de tareas periódicas están en el rango [100..1500] y el hiperperiodo es 63000. Estos parámetros producen periodos no armónicos. El número de tareas periódicas en cada procesador ha sido fijado a 8. Cuando no se estipula lo contrario, se ha establecido el factor de utilización máximo de las tareas periódicas U_{max}=20%. Previamente, en la Figura 4.6-(a) y la Figura 4.8-(a) hemos visto que U_{max} mayores provocan mayores rechazos, pero ahora esto no es determinante, porque no rechazamos sino que ajustamos las promociones. Por otro lado, U_{max} no afecta al incremento medio de las predicciones (Figura 4.5 y Figura 4.7) y con U_{max} mayores tendríamos la ventaja de reducir el número de predicciones inferiores (Figura 4.6-(b) y la Figura 4.8-(b)). La mayoría de pruebas se han realizado con una carga periódica U_{per}= 70% y una carga aperiódica U_{aper}= 25%, resultando un total del 95%. Para esta U_{per} se espera un incumplimiento de plazos muy bajo (Figura 4.6-(b)). Cargas periódicas bajas no establecerían diferencias entre los algoritmos en lo referente a la respuesta a tareas aperiódicas. Por otro lado, cargas periódicas superiores implican mejores predicciones (Figura 4.5 y Figura 4.7). De todas formas también se han realizado pruebas con U_{per}= 80% y U_{max}=20%, que fue el peor de los casos de predicciones inferiores (Figura 4.6-(b)).

La métrica de rendimiento utilizada es la media aritmética de los tiempos de respuesta medios normalizados de las tareas aperiódicas (*T. Respuesta Aperiódicas, TRA*)[22]. Cada punto de las gráficas se ha obtenido como la media aritmética del resultado obtenido en la simulación de 2000 conjuntos de tareas distintos (usados en cada algoritmo planificación). Así, hemos comprobado con el test estadístico de la distribución normal que el error cometido en cálculo de la media aritmética obtenida en las diversas simulaciones era inferior al 1%.

En la Figura 4.12 se muestran los resultados cuando la carga total es alta (95%). En ella se puede apreciar como los mejores resultados son los obtenidos con ambos GDP. La diferencia entre GDP_S y GDP_A no es estadísticamente significativa. La

[22] Para abreviar, en las gráficas *T. Respuesta Aperiódicas* y en el texto *TRA*

explicación es que, aunque las predicciones de los WCRT iniciales del GDP_A fuesen peores, con la adaptación dinámica las promociones de los dos algoritmos convergen. Cabe destacar que para U_{per} 75% y 80% (recordemos que junto a U_{max}=20% es el peor de los casos de predicciones inferiores (Figura 4.6-(b)) los resultados también son los mejores. Así, en el resto del análisis usaremos el nombre GDP para referirnos a ambos métodos indistintamente y más adelante estudiaremos las diferencias entre ellos.

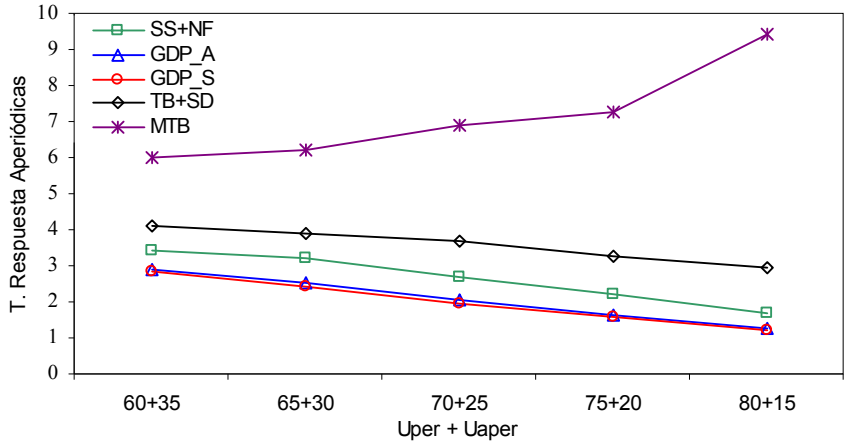


Figura 4.12: Variación de las características de la carga de las tareas periódicas y aperiódicas, manteniendo una carga total del 95%

En lo referente al resto de planificadores, MTB es el que peores resultados ofrece: el tiempo medio de respuesta de las aperiódicas es superior al doble del obtenido con GDP y empeora cuando la carga periódica crece. Sus autores ya establecían que se da un servicio equivalente al servicio en segundo plano, aunque no realizaban ningún experimento que lo constatara. SS+NF produce un incremento en el tiempo de respuesta que varía entre el 18% y el 26% con respecto al obtenido con GDP. Por su parte, TB+SD produce un incremento en el tiempo de respuesta que varía entre el 43% y el 136% con respecto al obtenido con GDP, siendo más acusada la diferencia cuando la carga periódica es mayor.

En la Figura 4.13 se han variado las características de la carga de las tareas periódicas manteniendo U_{aper}= 25%. En ambos experimentos se corrobora lo que hemos constatado en el anterior: MTB es una opción muy mala para planificar tareas aperiódicas sin plazo. En la Figura 4.13-(a) podemos apreciar que para cargas periódicas inferiores al 65% el rendimiento de GDP es similar al SS+NF, es decir,

equivalente al resultado de un planificador óptimo y muy clarividente. Para cargas altas GDP proporciona una reducción en el tiempo de respuesta de hasta un 23%. Comparando con TB+SD, incluso para cargas bajas la reducción es importante, siendo esta un 42% en el caso inferior. En la Figura 4.13-(b) podemos observar que el tiempo de respuesta se ve poco afectado por el incremento del U_{max}. Parece que cuando existen tareas grandes todos los planificadores tienen más problemas y sus resultados tienden a converger. Por ejemplo, la diferencia entre SS+NF y GDP_S se reduce de 0,8 a 0,6. Solo en este caso llega a ser algo significativa la diferencia entre GDP_A y GDP_S, llegando a experimentar un incremento del 10% para GDP_A.

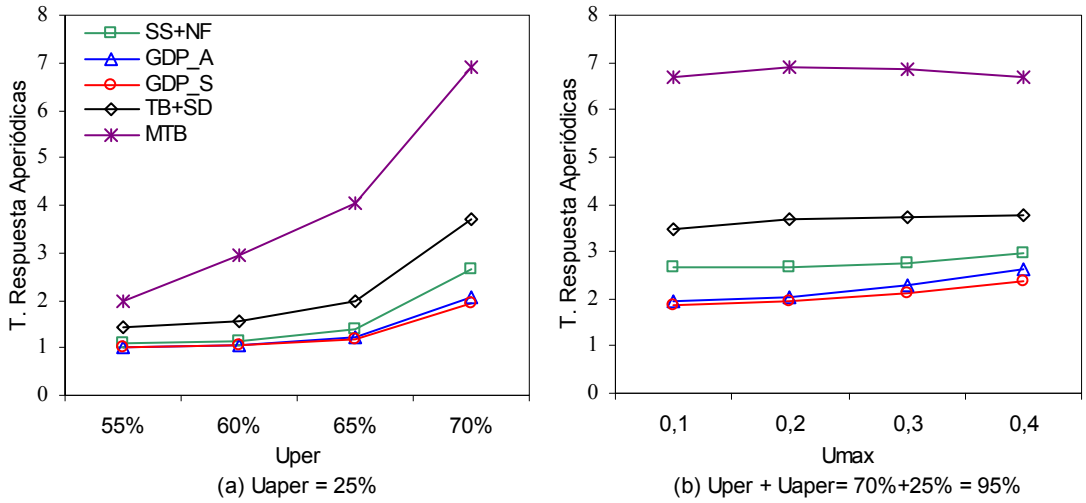


Figura 4.13: Variación de las características de la carga de las tareas periódicas manteniendo Uaper= 25%

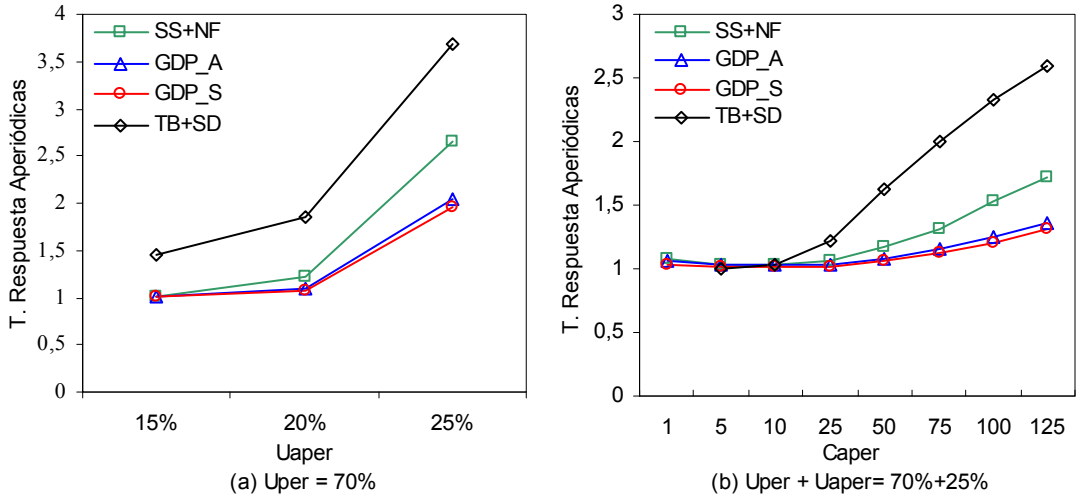


Figura 4.14: Variación de las características de la carga de las tareas aperiódicas manteniendo Uper= 70%

En la Figura 4.14 se han variado las características de la carga de las tareas aperiódicas manteniendo U_{per}= 70%. Los resultados van en la línea de los del experimento anterior, aunque matizan que la mejora de GDP con respecto a SS+NF son significativas cuando U_{aper} es grande (Figura 4.14-(a)) y cuando C_{aper} es grande (Figura 4.14-(b)).

Para los planificadores GDP hemos tomado muestras de otra métrica, el *índice de ajustes de promociones*. En las gráficas de la Figura 4.15 se muestran los resultados de esta métrica obtenidos en las simulaciones anteriores. Esta métrica se calcula contabilizando el número de veces que se ha hecho un ajuste de la promoción de cualquier tarea en todas las simulaciones, es decir, el número total de plazos incumplidos, y lo dividimos por el número de conjuntos de tareas simulados. El valor en si tiene un interés relativo, puesto que las simulaciones son largas y en ellas cada tarea se activa centenares de veces. Si tuviéramos esto en cuenta entonces las cifras obtenidas serían muy bajas. Sin embargo, si que tiene interés desde el punto de vista de la comparación entre el comportamiento de nuestras dos formas de calcular y ajustar las promociones.

Es curioso destacar que GDP_S tiene que corregir más frecuentemente las promociones que GDP_A. Esto nos indica que, debido a la inclusión de la entropía generada por las tareas aperiódicas, las tareas periódicas se han desplazado y los WCRT han ido aumentando. Es decir, el método de la simulación para hallar los WCRT en multiprocesadores no es válido, como ya intuimos con un ejemplo en el apartado 4.3.1.3, y además se equivoca más frecuente de lo que a priori pensábamos. Por su lado, GDP_A hace muchas menos correcciones de las promociones y es debido a que la aproximación del Algoritmo 4-3 es más pesimista, aunque ahora podemos decir que es más realista. Solo en el caso U_{per}=80% de la Figura 4.15-(c) GDP_A supera a GDP_S. Es un resultado esperado, puesto que en la Figura 4.6-(b) vimos que para U_{per}=80% el porcentaje de conjuntos de tareas donde el Algoritmo 4-3 hacia predicciones inferiores al obtenido con simulación para alguna tarea era muy alto.

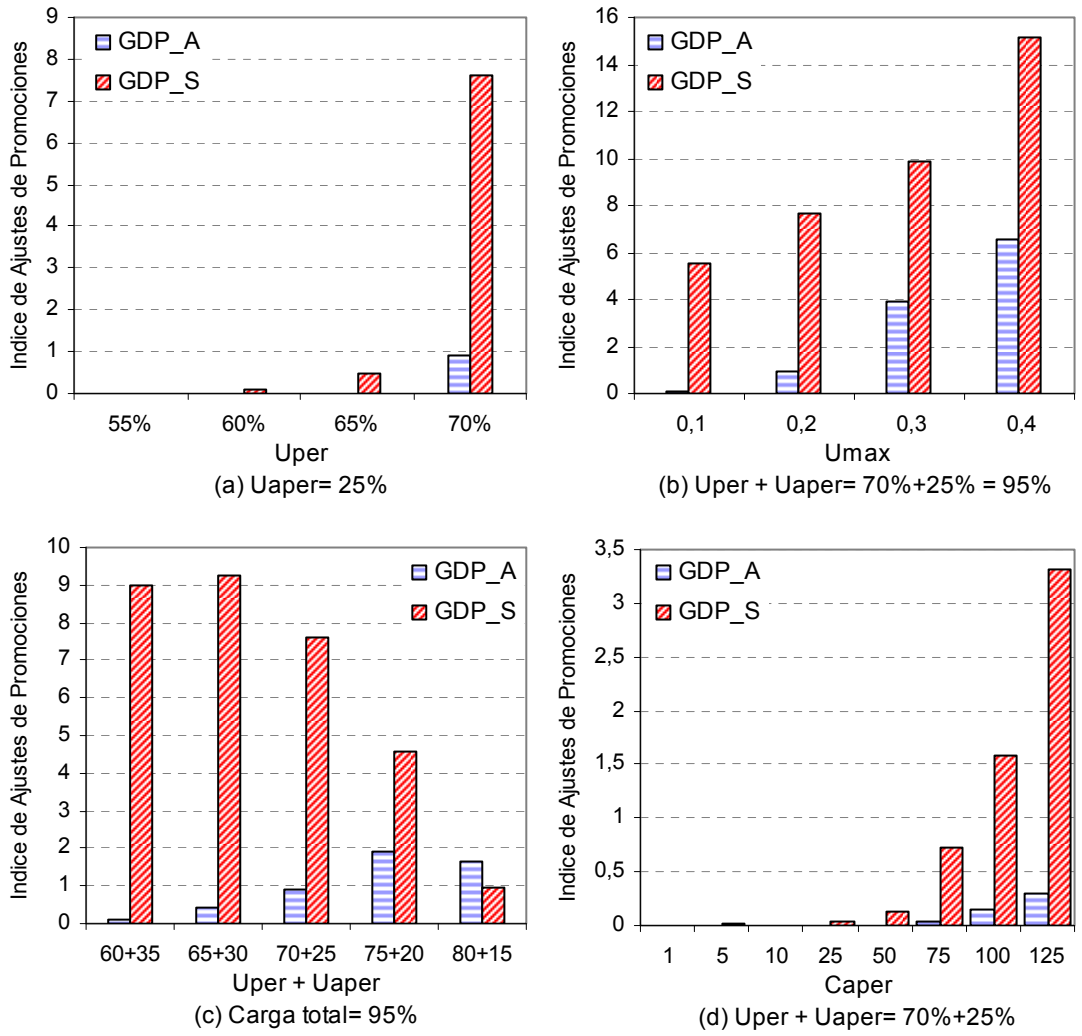


Figura 4.15: Índices de los ajustes de las promociones obtenidos para las simulaciones correspondientes a las figuras: Figura 4.13-(a), Figura 4.13-(b), Figura 4.12 y Figura 4.14-(b) respectivamente.

En los restantes experimentos se han utilizado los parámetros PCT # 4, los cuales producen conjuntos de tareas más armónicos para ver como afecta esto a nuestros planificadores. Un resumen de los resultados obtenidos se muestra en la Figura 4.16 y son parecidos a los de los experimentos anteriores. En la Figura 4.16-(b) y (d) se dividen los resultados obtenidos con los anteriores. En ellas se puede observar que ha habido un incremento que varía entre el 5% y el 17% para todos los planificadores (excepto MTB, el cual llega a incrementar un 23%). Los que menos se han incrementado son SS+NF y TB+SD. Esto significa que las diferencias entre ellos y GDP se ha reducido ligeramente, pero todavía son considerables, como se puede ver en la Figura 4.16-(a) (SS+NF incrementa entre un 11% y un 23% respecto GDP) y en

la Figura 4.16-(c) (SS+NF incrementa entre un 8% y un 26% respecto GDP). Podemos pues concluir que aumentar la multiplicidad de los periodos afecta negativamente a todos los algoritmos, el que más afectado resulta es GDP pero todavía obtiene unos resultados mejores que el resto de algoritmos.

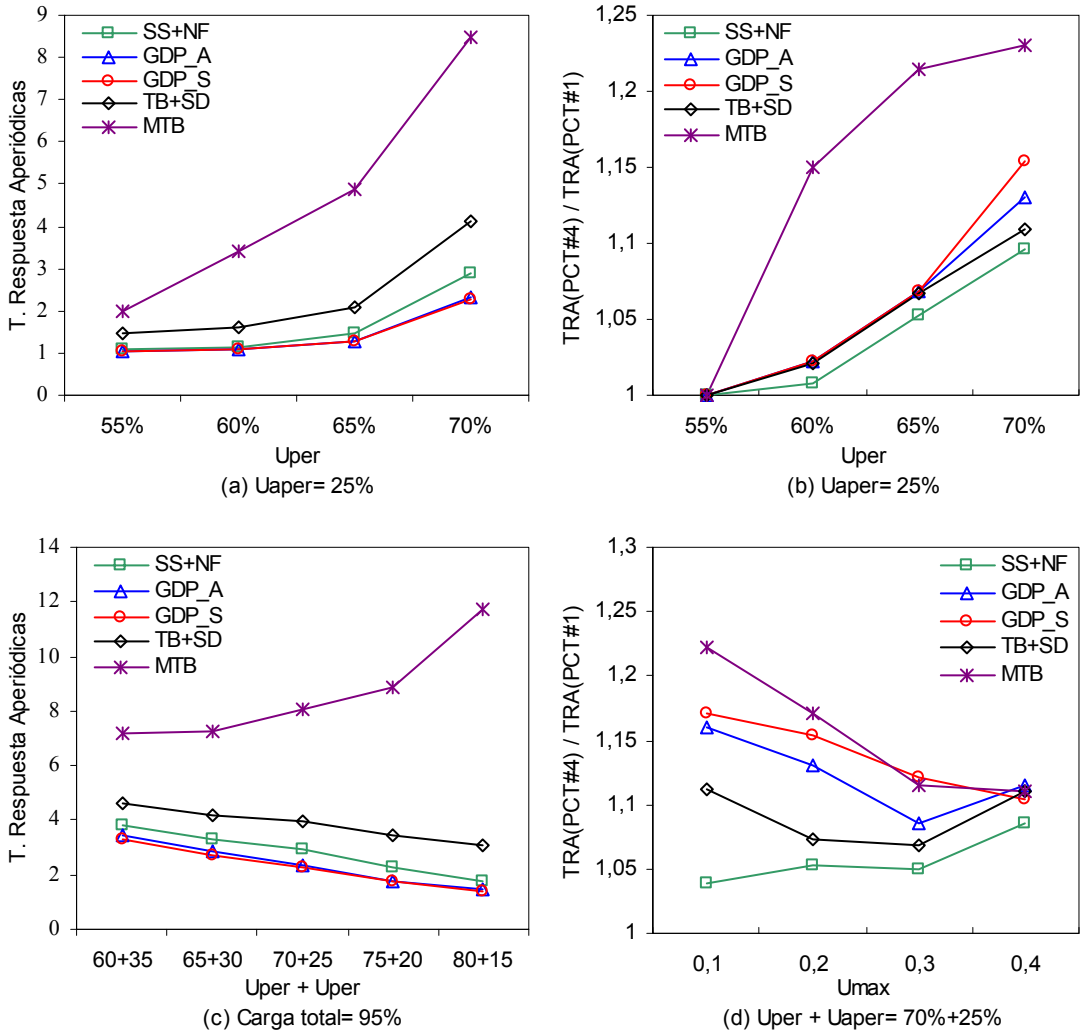


Figura 4.16: Resultados obtenidos con los parámetros PCT # 4:

a) variación de las características de la carga de las tareas periódicas manteniendo U_{aper}=25%

b) comparación de los resultados obtenidos con los parámetros PCT # 4 versus PCT # 1 cuando varía U_{per}

c) variación de las características de la carga de las tareas periódicas y aperiódicas, manteniendo una carga total del 95%

d) comparación de los resultados obtenidos con los parámetros PCT # 4 versus PCT # 1 cuando varía U_{max}

4.4 Conclusiones

En este capítulo se ha diseñado un planificador global para multiprocesadores basado en el algoritmo *Dual Priority*. Se han descrito los problemas que existen para el cálculo exacto de los peores tiempos de respuesta de las tareas periódicas (WCRT). Los métodos analíticos se han mostrado demasiado pesimistas: rehúsan demasiados conjuntos de tareas planificables y, en los casos aceptados, las promociones calculadas proporcionan poca flexibilidad. El método de la simulación tampoco proporciona la garantía requerida por los sistemas de tiempo real estrictos. Lo hemos mostrado con ejemplos y se ha verificado en los experimentos con la inclusión de tareas aperiódicas.

Se ha propuesto un algoritmo para hallar los WCRT aplicable en tiempo de diseño que se aproxima bastante a los resultados obtenidos con el método de la simulación. Estos resultados se utilizan para calcular las promociones para utilizarlas con un planificador DP global. En tiempo de ejecución el planificador comprueba que se cumplan los plazos y, si es necesario, ajusta dinámicamente las promociones. La diferencia de rendimiento entre este algoritmo y el método de la simulación no es estadísticamente significativa, resultando más conveniente en tiempo de diseño aplicar el algoritmo que tener que hacer una simulación. Además, la cantidad de plazos perdidos y las correcciones de los tiempos de promoción han sido superiores con el método de la simulación. El resultado es un planificador global aplicable a sistemas de tiempo real no estricto que, con un bajo coste, proporciona un buen servicio a las tareas aperiódicas sin plazo y que, si bien los plazos de las tareas periódicas no están garantizados, el número de plazos incumplidos es bajo y siempre en tareas poco prioritarias.

En comparación con el planificador global MTB [BaL04a], el tiempo de respuesta de las tareas aperiódicas es mucho mejor con GDP, por lo menos la mitad, llegando a ser de un orden de magnitud cuando la carga periódica es muy alta. En comparación con el planificador SS+NF, el mejor del capítulo anterior, cuando las cargas son muy altas también el tiempo de respuesta de las tareas aperiódicas es mejor con GDP, aunque en este caso las diferencias son del orden del 20% (algo inferiores si los

periodos son armónicos). Cuando el cálculo de las tareas aperiódicas es grande más evidente se hace la diferencia. Esto confirma la hipótesis que la planificación PD/AD ofrece mayor flexibilidad que PF/AD y es útil cuando las cargas son altas.

En el próximo capítulo se ha optado por un sistema híbrido entre la planificación global y la planificación local. Este método no requiere del cálculo del WCRT en multiprocesadores ya que utiliza la ecuación 2.2 para la planificación local y así se garantizan los plazos de las tareas periódicas, siendo una alternativa aplicable a sistemas de tiempo real estrictos.

Capítulo 5

PLANIFICACIÓN HÍBRIDA CON DUAL PRIORITY

En los capítulos anteriores se han ido detallando diversos problemas de la planificación conjunta de tareas heterogéneas de forma global en sistemas de tiempo real basados en una arquitectura multiprocesador. En este capítulo exploramos un método de planificación híbrido entre la planificación global y la planificación local, de forma que obtenga los beneficios de ambas planificaciones y minimice los inconvenientes de cada una.

5.1 Introducción

A diferencia de otros métodos utilizados en los capítulos anteriores, la discriminación entre planificación local y global no se hará según el tipo de tarea (como se hacia en los métodos PF/AD), sino que todos los tipos de tareas podrán ser planificadas de forma global, al menos durante parte de su ejecución. En los experimentos de los capítulos anteriores hemos comprobado que con cargas bajas casi cualquier método de planificación puede funcionar bien. Sin embargo, con cargas altas pueden surgir problemas, por ejemplo con PF/AD el tiempo de respuesta de las tareas aperiódicas sin plazo se puede incrementar y con planificación global pura los conjuntos de tareas pueden no ser planificables. Así, en estas circunstancias se hace necesario un método de planificación que sea flexible en el tratamiento de las tareas periódicas y al mismo tiempo obtenga un alto grado de planificabilidad.

Utilizaremos un único planificador, el cual será global, obviamente, pero este se comportará como un planificador local para algunas tareas periódicas con el fin de poderlas garantizar. Las tareas periódicas dividirán su ejecución en dos fases: durante la primera podrán ser planificadas de forma global; en cambio, durante la segunda fase deberán ser planificadas de forma local. La primera fase proporcionará la flexibilidad del balanceo de la carga, mientras que la segunda proporcionará la

garantía de los plazos y el aumento del nivel de utilización máximo de los procesadores.

Así, proponemos utilizar el algoritmo DP de forma mixta entre una planificación local y una planificación global. Este algoritmo de planificación ya distingue claramente la ejecución de las tareas periódicas en dos fases, antes y después de su promoción, y encaja de forma natural en una planificación dual entre la planificación global y la local. Como resultado de este capítulo se publicaron los artículos [BAL03], [BAL03b] y [BAL05].

5.2 El planificador Híbrido Dual Priority (HDP)

En el modelo híbrido, las tareas periódicas tienen dos fases de ejecución. La primera fase es dinámica y permite a las tareas periódicas ejecutarse en cualquier procesador según un nivel de prioridades bajas. Estas prioridades de nivel bajo tendrán un significado global, es decir, una tarea tendrá la misma prioridad de nivel bajo en todos los procesadores. Después de su promoción, una tarea periódica entrará en la segunda fase, la fase estática, ejecutándose en un procesador especificado en tiempo de diseño y con una prioridad fija en un nivel de prioridades alto y con significado local al procesador. Sin embargo, es posible que una activación de una tarea no entre en esta fase cuando se dé la circunstancia de que ya haya terminado su ejecución antes de su promoción.

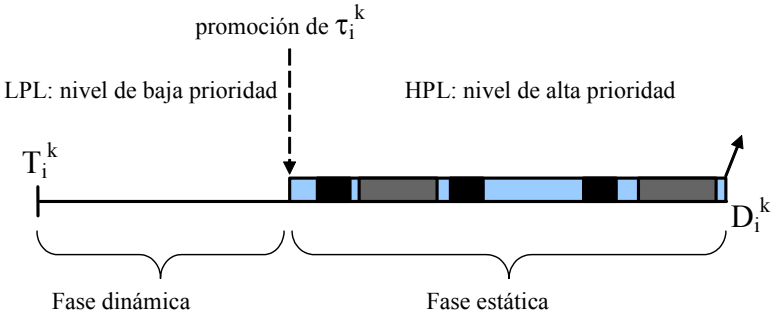


Figura 5.1: Fases de la ejecución de la k-ésima activación de la tarea τ_i: fase de ejecución en cualquier procesador (dinámica) y fase de ejecución estática en un procesador concreto después de la promoción. El color más claro indica la ejecución de la tarea τ_i^k y los más oscuros la interferencia de las tareas más prioritarias.

En tiempo de diseño, lo primero que se hace es distribuir el conjunto de tareas periódicas entre los distintos procesadores de forma que las tareas asignadas a cada

procesador puedan planificarse localmente en el procesador designado (es decir, de forma que pasen una prueba de planificabilidad para un algoritmo de planificación de monoprocesadores). Esta labor puede realizarse mediante cualquier algoritmo de empaquetado (*binpacking*) de los muchos que existen (véase el apartado 2.3.1 Planificación particionada). Las principales diferencias entre estos algoritmos surgen para cargas grandes, cuando algunos de ellos pueden fallar en la distribución si el número de procesadores es fijo. Sin embargo, recordemos que nuestro interés estriba en sistemas con conjuntos de tareas heterogéneos y, por consiguiente, normalmente las tareas periódicas dejarán un margen para la ejecución de otros tipos de tareas. Así, normalmente la carga inicial a distribuir no será un problema para el empaquetado, que podrá efectuarse de una forma u otra según el objetivo principal del servicio que queramos proporcionar. Este punto será analizado con mayor detalle en secciones posteriores.

Una vez distribuidas las tareas periódicas entre los procesadores ya se puede aplicar la ecuación 2.2 (p. 45) para hallar los WCRT que obtendría cada tarea periódica si se ejecutase exclusivamente en "su" procesador (R_i). Más tarde, estos valores serán utilizados para calcular los correspondientes tiempos de promoción en el planificador global ($Y_i = D_i - R_i$). Es importante remarcar que, a diferencia del capítulo anterior, estos cálculos son exactos y que así se garantiza que todas las tareas periódicas cumplirán con sus plazos, de la misma forma que lo hacen con un DP para monoprocesadores.

En la Figura 5.2 se muestran los elementos utilizados por el algoritmo de planificación en tiempo de ejecución. Cuando cualquier tarea llega al sistema o es activada, sea del tipo que fuere, es encolada en la *Global Ready Queue* (**GRQ**). En esta cola, las tareas periódicas tendrán el nivel de prioridades más bajas (LPL). Estas prioridades pueden ser arbitrarias, aunque normalmente se prefiere un orden *Earliest Promotion First* global (EPF, véase el Anexo A.2). Las tareas a las que deseemos dar cierta preferencia, como por ejemplo las tareas aperiódicas, tendrán la prioridad más alta dentro de esta cola (véase "tareas servidas" en la Figura 5.2). El planificador

global (**GS**) selecciona las *m* primeras tareas de esta cola para ejecutarlas en los *m* procesadores del sistema.

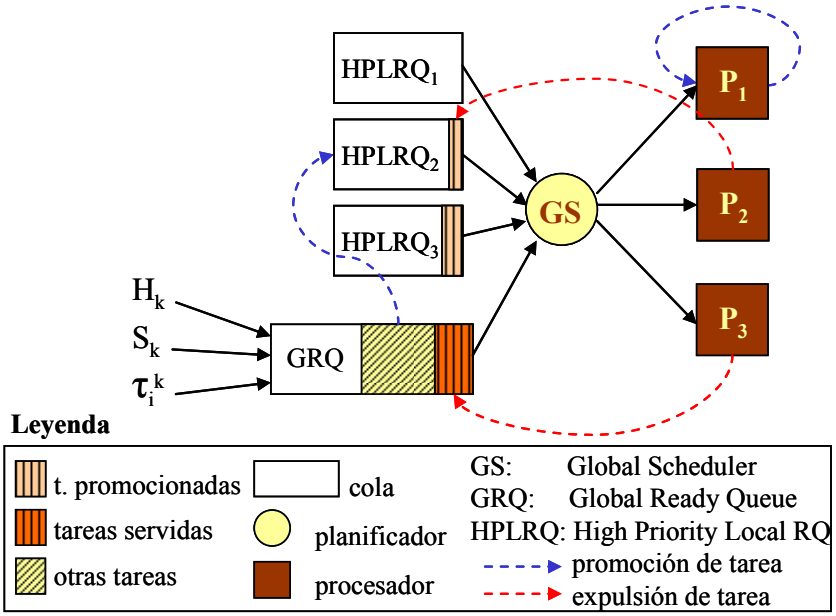


Figura 5.2: Esquema de planificación global con HDP. Para cada procesador P_i, si $HPLRQ_i$ no está vacía entonces ejecuta la primera tarea (promocionada) de P_i, sino P_i ejecutará la primera tarea esperando en GRQ.

Adicionalmente, existen *m* colas locales de alta prioridad para las tareas periódicas promocionadas (*High Priority Local Ready Queues*, **$HPLRQ_i$,** con *i* en [1..m] y siendo *m* el número de procesadores). Cuando una tarea τ_{pi} es promocionada, se le cambia el nivel de prioridad al nivel superior y se mueve desde la GRQ hacia su correspondiente $HPLRQ_p$. En caso de estar ejecutándose en su procesador, solo se le promociona la prioridad (como en el caso del procesador P_1 de la Figura 5.2). Es importante notar que una promoción implica un cambio de prioridad y esto puede resultar en una ***expulsión*** (como ocurre en el procesador P_3 de la Figura 5.2) e incluso en una migración (si la tarea promocionada no estaba ejecutándose en su procesador). Los procesadores con tareas promocionadas activas no ejecutan bajo ningún concepto tareas de la GRQ. De esta forma una tarea promocionada solo recibe la interferencia de las tareas más prioritarias ya promocionadas, contempladas de antemano en la prueba de planificabilidad realizada en tiempo de diseño, garantizándose de esta forma que cumpla con su plazo.

Con este método, mientras una tarea periódica no es promocionada se puede ejecutar en cualquier procesador (fase dinámica). Esto puede reducir el número de tareas esperando a ser ejecutadas en un procesador específico y permite balancear parte de la carga y avanzar el trabajo periódico cuando no haya otro tipo de tareas a servir. Así se aumentará la disponibilidad del sistema para futuros servicios a tareas que lo requieran, aportando flexibilidad en la fase dinámica y garantías en la fase estática.

En un caso extremo, se puede hacer un diseño en el que algún procesador no tenga ninguna tarea periódica asignada, por ejemplo en previsión de futuras ampliaciones del conjunto de tareas. Incluso, puede ser beneficioso disponer de tal procesador para ejecutar el propio planificador global. En estos casos HDP es capaz de utilizar estos procesadores de repuesto para ejecutar tareas periódicas asignadas a otros procesadores en sus fases dinámicas, balanceando así la carga, llegando incluso a conseguir que las tareas se ejecuten por completo fuera de su procesador (si esto es posible, es decir para las tareas que cumplan $C_i \leq Y_i$).

Los costes del planificador en cuanto a requerimientos de memoria son: *m* colas locales RQ de longitud finita y conocida para las tareas promocionadas; una cola global de longitud el número de tareas periódicas más el número de tareas aperiódicas; una cola de eventos de comprobación de los plazos de longitud el número de tareas periódicas y *m* colas de eventos con las promociones de una longitud acumulada igual al número de tareas periódicas. Los costes en tiempo de ejecución son los propios de atender a los eventos, los cambios de cola de las tareas promocionadas y los costes de migración de las tareas periódicas cuando son promocionadas. En todo caso, el planificador no debe hacer ningún cálculo intensivo.

El método propuesto es genérico y se puede aplicar para servir a diversos tipos de tareas, incluso de forma combinada. Para discriminar entre los distintos tipos de tareas a servir se deberá aplicar una política de prioridades apropiada en la cola GRQ. En las siguientes secciones discutiremos como dar servicio a tareas periódicas que no serían planificables (5.3), dar servicio a tareas aperiódicas sin plazo (5.4), o a tareas aperiódicas con plazo (5.5) o a una combinación de ellas (5.6).

5.3 Servicio a más tareas periódicas: aumentando el grado de planificabilidad

El método de planificación propuesto en el apartado anterior permite dar servicio a tareas adicionales a las contempladas en tiempo de diseño. En particular, en el apartado actual describimos como usar esta capacidad para dar servicio a tareas periódicas extra, es decir, tareas que no cabrían en el sistema con un algoritmo de particionado, puesto que no serían planificables en ningún procesador de forma local, aumentando así el grado de planificabilidad.

El objetivo de la propuesta original del algoritmo DP [BuW93] ya era incrementar la utilización de un monoprocesador con tareas periódicas en el caso promedio. Resumiendo, esto se lograba estableciendo tiempos de promoción a una prioridad superior *solo* a las tareas periódicas que de otra forma perderían su plazo con la asignación de prioridades RM. El resto de tareas no usaban la doble prioridad: siempre se ejecutaban en el nivel de prioridad bajo.

En el caso de los multiprocesadores los problemas de planificación en parte pueden ser solucionados adaptando la distribución estática de las tareas entre los procesadores. Si una tarea no cabe en un procesador se puede tratar de acomodar en otro. Desafortunadamente, esto no siempre es posible, en particular cuando las cargas son muy altas. Cuando la carga periódica total del conjunto de tareas de una aplicación concreta no puede ser distribuida entre los distintos procesadores usando un algoritmo de *binpacking* la estrategia que proponemos en este apartado puede ser utilizada para planificarlos.

Así pues, en este apartado describimos las modificaciones realizadas al algoritmo HDP del apartado 5.2 para que pueda dar servicio a *tareas conflictivas*, es decir, a las tareas que en tiempo de diseño no se puede encontrar ningún procesador en el que sean planificables. La idea consiste en asignar estas tareas a un procesador, aunque el resultado en un monoprocesador no fuese planificable, de forma que usaremos la capacidad de balancear la carga de HDP para liberar parte de la carga de este procesador, de forma que la tarea a priori no garantizada pueda cumplir sus plazos. Es importante destacar que incluso se puede sobrepasar la capacidad máxima de un

procesador y esto facilitará mucho el trabajo del empaquetador. El inconveniente es que no existirán garantías del cumplimiento de los plazos, aunque sí altas probabilidades de hacerlo. No obstante, en el supuesto que no haya fuentes de indeterminismo en el sistema, como podrían ser tareas aperiódicas o esporádicas, si la simulación de un hiperperiodo demuestra que las distintas activaciones de las tareas garantizadas cumplen sus plazos entonces sí se van a tener garantías.

En el método de planificación del apartado anterior proponemos realizar las siguientes acciones para afrontar el problema actual:

A1) variar la distribución de tareas periódicas entre procesadores, sin importar sobrecargarlos en estático, puesto que HDP permite cierto grado de balanceo de la carga en tiempo de ejecución,

A2) asignar promociones inmediatas a las tareas no garantizadas,

A3) variar el orden de prioridades globales de bajo nivel (LPL) para favorecer a las tareas conflictivas.

En las siguientes secciones analizamos como pueden afectar estas acciones al aumento de la utilización de los procesadores por parte de las tareas periódicas y, más adelante, en el apartado 5.3.1, se describe el algoritmo que las implementa todas. Es conveniente remarcar que con el método propuesto las tareas conflictivas no estarán garantizadas en tiempo de diseño pero las siguientes acciones están orientadas a facilitar que en tiempo de ejecución los plazos se cumplan.

Distribución de las tareas periódicas en tiempo de diseño

La distribución de tareas entre procesadores no es trivial ni tiene una solución única y ésta podría afectar en gran medida al resultado final. Asignar una tarea conflictiva a un procesador puede ser exitoso o no. En caso de fallar la asignación, la acción A1 facilita la distribución, puesto que se permite una carga superior al 100% de los procesadores. De esta forma el resultado obtenido será más independiente de la distribución, no importando tanto en que procesador ubicar una tarea conflictiva,

puesto que en tiempo de ejecución habrá cierto grado de balanceo dinámico de la carga.

El hecho que una tarea periódica esté garantizada o no puede ser más o menos importante dependiendo de la aplicación en concreto a la que pertenezca. Así, el diseñador de la aplicación debería ordenar las tareas periódicas según la importancia en el cumplimiento de sus plazos. A posteriori, se usa un algoritmo de empaquetado para distribuirlas entre los procesadores disponibles. Si no se dispone de esta información, lo más razonable es usar el algoritmo de empaquetado RM-FFDU (*Rate-Monotonic First-Fit Decreasing Utilization*, [OhS95]) para distribuir las tareas entre los procesadores. Este algoritmo es muy efectivo (véase la Tabla 9 en la página 60 y [AnJ00b]) y encaja muy bien para los objetivos de este apartado. De todas formas, en el apartado 5.3.4 se experimentarán otros métodos de distribución. El algoritmo RM-FFDU primero ordena las tareas según su utilización (U_i), de mayor a menor, y luego las va asignando a un procesador mientras le quepan, pasando al siguiente en caso contrario. De esta forma, las tareas que se distribuirán al final serán las más pequeñas que son las más fáciles de ubicar. Además la carga se va concentrando en unos procesadores, dejando otros más libres para poder albergar las siguientes tareas. En el caso de no encontrar un procesador donde se puedan ubicar, estas tareas no estarán garantizadas en tiempo de diseño pero serán las de menor carga y así HDP necesitará encontrar "huecos" menores para ejecutarlas (véase el Algoritmo 5-1 en la página 166). Si la tarea no garantizada fuese la de mayor carga en tiempo de ejecución el planificador global difícilmente podría encontrar suficiente capacidad para ejecutarla.

Asignación de promociones

Con el algoritmo DP las tareas promocionadas son las que tienen el mayor nivel de prioridad. Así, lo mejor que se puede hacer con las tareas no garantizadas es promocionarlas de inmediato para que se ejecuten en el nivel de prioridad superior, aumentando así sus posibilidades de éxito. Sin embargo, para no comprometer las tareas garantizadas, su prioridad en el nivel HPL será la inferior. Veamos un ejemplo

clásico[23] solucionado con la promoción inmediata (acción A2). Las tareas de este ejemplo generan una carga total de tan solo un 152% y, a pesar de disponer de dos procesadores[24], no son planificables con RM global. En la Figura 5.3-(a) se representa la ejecución del conjunto de tareas detallado en la leyenda usando GRM[25]. En este caso la tarea τ_3 pierde el plazo en el instante t=100 puesto que todavía le faltan 2 unidades para finalizar. Este simple ejemplo muestra como algoritmos óptimos en monoprocesadores[26] no lo son en multiprocesadores. Sin embargo la ejecución con GDP (del capítulo anterior) o con HDP y una promoción inmediata de la tarea τ_3 puede planificar este conjunto de tareas (véase la Figura 5.3-(b). Como GRM ha fallado en la ejecución de τ_3, GDP utiliza las promociones Y_1=25, Y_2=25 y Y_3=0, ejecutando primero τ_1 y τ_3, cumpliendo así todos los plazos.

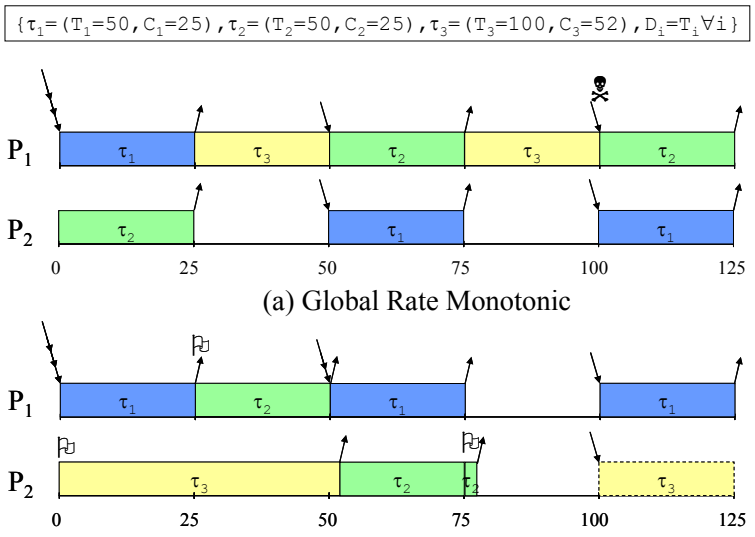


Figura 5.3: Ejemplo de como GDP puede aumentar el grado de planificabilidad en algunos casos

El ejemplo de la Figura 5.3 también se solucionaría con la distribución DP-FFDU (página 166). Este asignaría las tareas τ_3 y τ_1 al primer procesador, cargándolo al 102%, y la tarea τ_2 al segundo procesador. En tiempo de ejecución, la tarea τ_1 se

[23] Extraído de [Bur91], donde se usa para ilusatrar como conjuntos de tareas con utilizaciones no muy altas no son planificables con GRM.
[24] Supone una carga por procesador del 76%, inferior al límite de Liu &Layland, que sería 83% para 2 tareas o 78% para tres
[25] Adviértase que la planificación con EDF global sería exactamente la misma.
[26] La planificación local es posible si se asignan las tareas τ_2 y τ_3 al mismo procesador.

podrá ejecutar en el segundo procesador, obteniendo un resultado similar a si hubiésemos asignado τ_1 y τ_2 al mismo procesador.

Un caso general y más extremo es el de *m* procesadores, m+1 tareas de periodo y plazo iguales a T, *m* de ellas con cálculo $C_i = T/2 + \varepsilon$ (con $\varepsilon \to 0$) y la restante con cálculo $C_{m+1} = T/2$. En este caso[27], a pesar que la carga promedio por procesador sea aproximadamente el 50% (para *m* grandes) ni RM global ni EDF global pueden planificarlo. Peor aún, estas tareas no pueden distribuirse estáticamente entre los procesadores y aplicar un algoritmo de planificación local, puesto que habría *m*-1 procesadores con una tarea y cargados al 50% mientras que el procesador restante tendría dos tareas y quedaría con una carga de $2(T/2+\varepsilon)/T > 100\%$ o bien de $(T/2+\varepsilon + T/2)/T > 100\%$. Sin embargo, con HDP si se puede planificar este ejemplo. Hacemos la distribución estática anterior, sin importarnos que la carga de un procesador sea mayor que el 100%, y asignamos las promociones de las tareas de la siguiente forma:

$$R_i = C_i = T/2 + \varepsilon \qquad \forall\ i \le m \qquad \Rightarrow \qquad Y_i = T - (T/2 + \varepsilon) = T/2 - \varepsilon$$

$$R_{m+1} > T \qquad \Rightarrow \qquad Y_{m+1} = 0$$

Así, la tarea τ_{m+1} se ejecutará primero hasta que en el instante $t=T/2-\varepsilon$ una de las tareas más prioritarias será promocionada y se apropiará del procesador, expulsando τ_{m+1}. El cómputo pendiente de la tarea τ_{m+1} será ε y podrá ejecutarse en cualquiera de los otros procesadores pasado un tiempo 2ε (siempre y cuando se cumpla que $\varepsilon \le T/2 - \varepsilon$, o sea $\varepsilon \le T/4$), cuando finalice una de las tareas que se ha ejecutado en paralelo con ella.

La promoción inmediata de las tareas que no pasan el test de planificación puede no ser suficiente y puede ser necesario tomar medidas adicionales. Por ejemplo, puede ser insuficiente cuando haya otras tareas que ya de por si tengan una promoción inmediata (cuando $R_i = D_i$, como es el caso de la tarea τ_2 en el ejemplo de la Figura

[27] Este ejemplo es diferente del usado para el Efecto Dhall (página 65) el cual usa una tarea muy grande junto a *m* tareas muy pequeñas y en cambio en el ejemplo actual todas las tareas son parecidas.

5.3). Estos casos justifican la necesidad de tomar medidas adicionales para favorecer en tiempo de ejecución garantizar un mayor número de casos.

Orden de prioridades del nivel bajo

Además de favorecer a las tareas no garantizadas promocionándolas al nivel superior de prioridades también se las puede favorecer con una asignación apropiada de prioridades de nivel inferior (LPL).

Volviendo al ejemplo de la Figura 5.3 un algoritmo de particionado asignaría τ_1 y τ_2 al primer procesador y τ_3 al segundo, puesto que ni τ_1 ni τ_2 pueden combinarse con τ_3. El resultado sería planificable, sin recurrir a promociones inmediatas, si se asigna la máxima prioridad en el nivel bajo a la tarea τ_3. Las promociones serían Y_1=25, Y_2=0 y Y_3=48. Gracias al orden en LPL se ejecutarían primero las tareas τ_3 y τ_2, resultando una planificación equivalente a la de la Figura 5.3-(b).

En el sencillo ejemplo de la Figura 5.3 ha sido suficiente alterar el orden de LPL, pero en general será conveniente combinar las dos técnicas de manipulación de las prioridades. Así, si asignamos promociones inmediatas a las tareas periódicas no garantizadas, estas no utilizan el nivel de prioridad bajo, luego, para favorecerlas es necesario quitar la interferencia de algunas tareas de su procesador con mayor prioridad en el nivel HPL asignándoles el mayor nivel de prioridad dentro de LPL (recordemos que este es global). Así, se favorece que estas tareas de mayor prioridad se ejecuten en otros procesadores y, en caso de promocionar, reducir su interferencia con las tareas no garantizadas. Así pues, indirectamente se favorece la ejecución de las tareas no garantizadas en su procesador, aumentando así las posibilidades de cumplir con sus respectivos plazos.

5.3.1 Algoritmo usado en tiempo de diseño

El método de planificación usado en esta sección no varía respecto al del apartado 5.2. Solo hace falta cambiar la parte que se hace en tiempo de diseño: la distribución de tareas, la asignación de prioridades y el cálculo de las promociones. Las tres acciones propuestas en las secciones anteriores se combinan en el Algoritmo 5-1 en la página 166.

La distribución RM-FFDU (acción A1) hace que las tareas no garantizadas sean las menores y, cuando es posible, deja algunos procesadores más descargados para que puedan absorber las tareas de alta prioridad en LPL (acción A3).

El algoritmo primero ordena el conjunto de tareas periódicas usando el orden *rate-monotonic* y fijamos las prioridades bajas (LPL) de acuerdo a este orden (líneas 2-5). Después, usa el algoritmo *Rate-Monotonic First-Fit Decreasing Utilization* (RM-FFDU) [DaD86a] para distribuir las tareas entre los procesadores (línea 6).

En los conjuntos de tareas sobrecargados habrá tareas que no se podrán asignar a ningún procesador, puesto que, aún en el caso de caber en términos de capacidad de proceso, estas tareas no podrían ser ejecutadas antes de su plazo (recordemos que el límite superior teórico de planificabilidad con RM está aproximadamente en el 69%). Así pues, se etiqueta a estas tareas como **no_garantizadas** y se procede a asignar dichas tareas excedentes a los procesadores con menor utilización (líneas 7-14) para aumentar sus posibilidades de terminar a tiempo. Así, las tareas conflictivas más frecuentes se asignarán a los procesadores con menor carga. Obviamente los procesadores con sobrecarga no pasarán la prueba de planificabilidad. Incluso puede darse el caso extremo en que un procesador en tiempo de diseño tenga asignada una carga superior al 100%.

Una vez distribuidas todas las tareas, se asignan las prioridades altas (HPL) de acuerdo al orden RM local a cada procesador (línea 17). En el siguiente paso se calculan los tiempos de promoción usando la ecuación 2.2 y se establece una promoción inmediata para las tareas que no pasen el test, es decir las no garantizadas (líneas 19 a 22). Resumiendo:

$$\forall \tau_i \text{ tal que } W_i^{n+1} > D_i \text{ entonces } Y_i = 0 \text{ y } Y_i^k = k\, T_i$$

Para tratar de conseguir que estas tareas terminen antes de su límite se debe aliviar parte de las interferencias producidas por las tareas de prioridad superior (más frecuentes). Para ello se asignan a las tareas de alta prioridad local (o sea, las garantizadas) una prioridad global LPL mayor que al resto para darles preferencia de ejecución en otros procesadores (línea 25). Esto se consigue gracias a que se ha dejado un margen en las prioridades LPL (*offset* en la línea 4). Las tareas etiquetadas como **seleccionadas** (línea 26), serán las que en tiempo de ejecución recibirán el servicio prioritario del HDP. Curiosamente, las tareas conflictivas se van a apropiar del procesador asignado y las tareas garantizadas van a migrar a otros procesadores. Eso sí, después de su promoción, las tareas garantizadas van a tener mayor prioridad y podrán recuperar su procesador para cumplir con sus respectivos plazos. Es importante remarcar que esto es una gran diferencia respecto al algoritmo DP inicial [BuW93] (a demás de ser para multiprocesadores): las tareas conflictivas ahora solo tienen un nivel de prioridad (HPL) y el resto dos, pero son las de prioridad más alta, tanto en LPL como en HPL.

Es importante remarcar que este método no sufre el Efecto Dhall[28], el cual es el fundamento del bajo nivel de utilización de los planificadores globales. Recordemos que se basa en un conjunto de tareas con una tarea muy grande junto a *m* tareas muy pequeñas. En este caso RM-FFDU (línea 6 del Algoritmo 5-1) es capaz de distribuirlas: al principio asigna la tarea grande a un procesador y al final deja un procesador con dos tareas pequeñas. Así, las tareas asignadas a cada procesador son planificables localmente, se puede calcular sus promociones y se puede planificar con HDP.

[28] Véase la página 65

```
1: Algoritmo DP-FFDU (τ, PS)  { Task Set, Processor Set }
2:    ordenar (τ, orden RateMonotonic)
3:    para cada tarea τi ∈ τ hacer
4:          LPL (τi)= offset + i
5:    fin_para
6:    RM-FFDU (τ, PS)              { distribución parcial }
7:    mientras no vacio(τ) hacer    { distribuir el resto }
8:          seleccionar Pj ∈ PS de forma que Uj= min (Up), p=1..m
9:          elegir τk ∈ τ en orden
10:         remover τk de τ
11:         marcar τk como no_garantizada
12:         asignar τk al procesador Pj
13:         marcar Pj como sobrecargado
14:   fin_mientras
15:   para cada procesador Pi ∈ PS hacer  { asignar prios y promos }
16:         para cada tarea τij asignada a Pi hacer
17:               HPL(τij)= j        { RM/DM local }
18:         fin_para
19:         calcular_tiempos_de_promocion (Pi)
20:         si Pi está sobrecargado entonces
21:               para cada tarea τij asignada a Pi hacer
22:                     si no_garantizada(τij) entonces Yij=0
23:                     para cada tarea τik asignada a Pi Y
24:                           más frecuentes que τij hacer
25:                                 LPL (τik)= LPL (τik) – offset
26:                                 marcar τik como seleccionada
27:                     fin_para
28:               fin_para
29:         fin_si
30:   fin_para
```

Algoritmo 5-1: Algoritmo DP-FFDU: distribuye un conjunto de tareas periódicas (τ) entre un conjunto de procesadores (PS) y realiza la asignación de prioridades para utilizar HDP con procesadores sobrecargados, incrementando así la capacidad de carga periódica total del sistema. La distribución de tareas inicial utiliza el algoritmo RM-FFDU (*Rate-Monotonic First-Fit Decreasing Utilization*)

5.3.2 Evaluación experimental del rendimiento

En esta sección realizamos una evaluación del funcionamiento del algoritmo híbrido detallado en la sección anterior (HDP, aplicando el algoritmo DP-FFDU en tiempo de diseño). Para ello lo comparamos con los resultados obtenidos con planificadores locales y planificadores globales de otros autores. El objetivo es comprobar si HDP puede planificar un mayor número de conjuntos de tareas que otros planificadores globales e incluso que los locales. La metodología de la evaluación que hemos utilizado se basa en la simulación de conjuntos de tareas sintéticos generados aleatoriamente. Esto nos permite verificar el funcionamiento general y la robustez del algoritmo de planificación bajo un amplio abanico de situaciones. Como métrica de rendimiento utilizamos principalmente el porcentaje de éxito en la planificación de conjuntos de tareas (*Success Ratio, SR*). Este porcentaje de éxito es la proporción de conjuntos de tareas planificados con éxito con un algoritmo respecto al total de los conjuntos generados.

A menos que se indique lo contrario, se ha utilizado la misma caracterización de los experimentos usados en el artículo [AnJ00b] (véase el PCT # 6 del Anexo B). Generalmente el número de procesadores es m=4. El número de tareas, n, sigue una distribución uniforme con un valor previsto E[n] =8, con un mínimo de 0,5E[n] y un máximo de 1,5E[n]. Los periodos se eligen aleatoriamente de entre el conjunto {100,200,300,…, 1600}. El factor de la utilización U_i de cada tarea sigue una distribución normal con un valor previsto E[U_i]=0,5 y una desviación de estándar del stdev[U_i]=0,4 acotada a (0..1]. La única diferencia con respecto a [AnJ00b] es que nuestro generador de tareas desecha los conjuntos imposibles de planificar cuando la carga total generada resulta mayor a la capacidad disponible. De no hacerlo así, esto disminuiría el porcentaje de éxito de todos los algoritmos por igual y la evaluación relativa sería válida pero introduciría una distorsión del funcionamiento real de los algoritmos. Con E[n] =8, E[U_i]=0,5 y m=4 la carga del sistema promedio sería del 100% pero como hemos rechazado los casos que pasan del 100% en realidad la carga promedio de los experimentos ha sido U_{per}= 80%. Para la mayoría de los experimentos, para cada punto se generan 100.000 conjuntos de tareas plausibles y se computa la proporción de conjuntos que son planificables con un determinado

algoritmo. Se ha establecido un número tan grande de simulaciones porque los parámetros dan lugar a generar conjuntos de tareas muy variables y, por consiguiente, dan lugar a resultados muy variables. Como la métrica utilizada (SR) sigue una distribución binomial, este elevado número de muestras garantiza un error inferior al 1%. Además, esto ha sido posible gracias a que, al tratarse solo de tareas periódicas, las simulaciones solo debían realizarse durante un hiperperiodo.

Como referencia hemos utilizado dos versiones del algoritmo de particionado RM-FFDU [DaD86b]. Una versión utiliza la prueba de planificabilidad suficiente con una complejidad polinómica (véase la página 60) y la otra versión, RMFFDU+respan, utiliza la prueba necesaria y suficiente de planificabilidad de Joseph y Pandya [JoP86] (con complejidad pseudo-polinómica). Obviamente, ésta última requiere mayor tiempo de cómputo fuera de línea pero alcanzará un porcentaje de éxito mayor, siendo así una buena referencia. En el artículo [AnJ00b] observaron que éste era el método de particionado que alcanzaba el porcentaje de éxito mayor para la disposición experimental propuesta. Además del planificador global HDP propuesto en este apartado, también hemos simulado otros dos planificadores globales con asignación estática de prioridades: GRM y AdaptativeTkC (véase los apartados 2.4.1.1 y 2.4.1.2 respectivamente). Todos utilizan la característica 'consciencia-de-expulsiones' (*preemption-aware*) descrita en [AnJ00b], es decir, todos los despachadores consideran el estado anterior para reducir al mínimo el número de expulsiones a la hora de despachar las siguientes tareas.

En la Figura 5.4-(a) se muestran los resultados obtenidos con los diversos algoritmos de planificación cuando el número de procesadores varía. Como podemos observar, nuestro HDP es el algoritmo con mayor porcentaje de éxito, variando entre el 84% y el 94%. El segundo en rendimiento es un algoritmo de particionado, el RM-FFDU+respan. Esto es lo que a priori esperábamos, porque el porcentaje de éxito para HDP incluye todos los casos planificables con RM-FFDU más otros casos en que el *binpacking* falla y en los cuales HDP es capaz de migrar las tareas seleccionadas para acomodar las restantes.

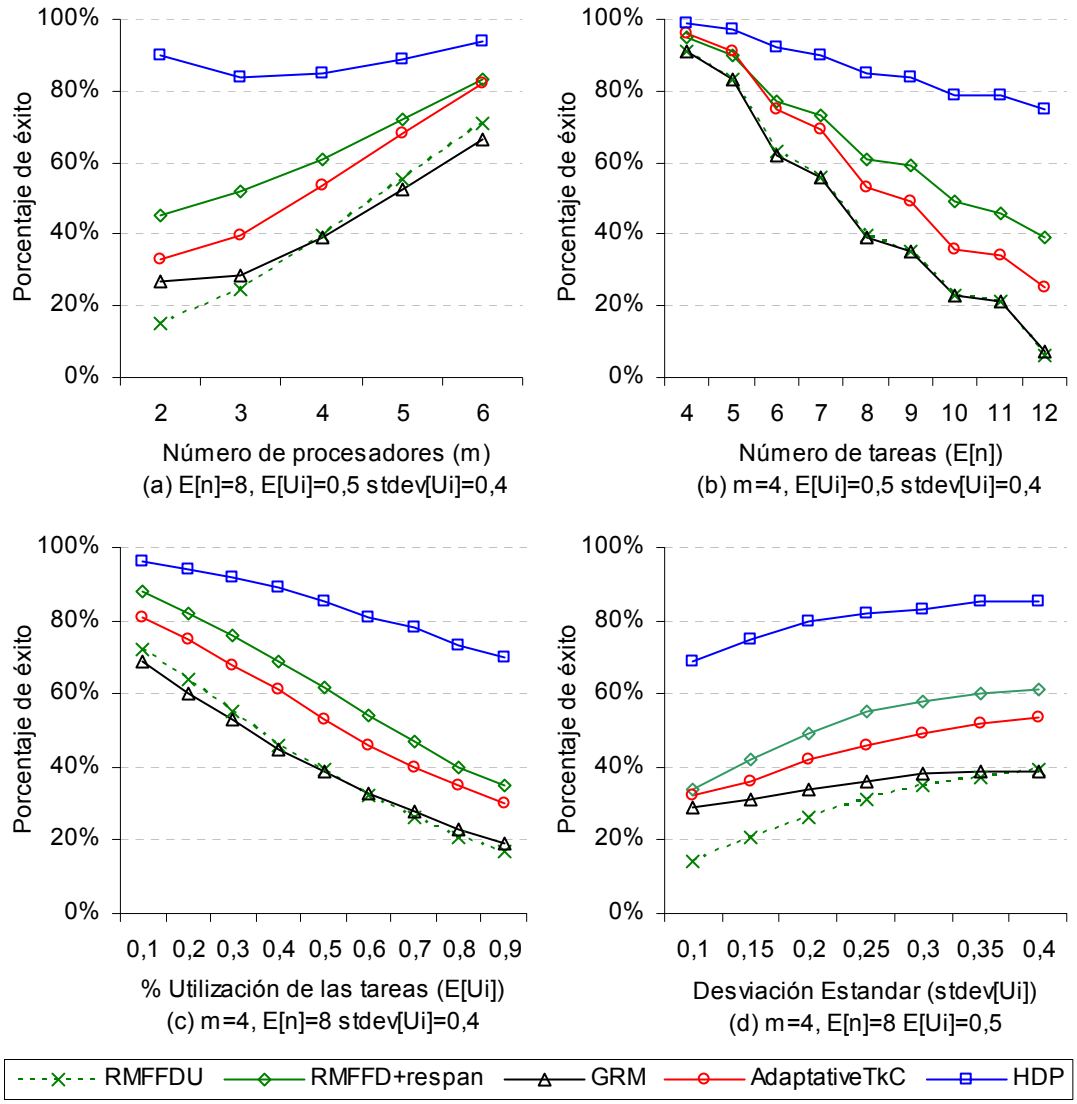


Figura 5.4: Porcentaje de éxito en función de (a) el número de procesadores (b) el número de tareas medio (c) la utilización media de las tareas y (d) la desviación estándar de la utilización de las tareas

Cuando el número de los procesadores aumenta, como la carga relativa disminuye, hay más recursos disponibles y, por lo tanto, todos los algoritmos aumentan su porcentaje de éxito. Con respecto a los otros planificadores globales, las diferencias son notorias cuando el número de procesadores es bajo: HDP triplica el rendimiento de GRM y duplica al de AdaptativeTkC. En lo referente al cambio de pendiente entre m=2 y m=3, analizando los resultados a fondo hemos concluido que el generador tiene problemas con esta parametrización (puesto que las cargas generadas son demasiado altas para ser planificables en dos procesadores) y tiende a generar conjuntos de tareas con cargas totales más bajas pero con algunas tareas grandes,

siendo algo más fáciles planificar por los planificadores globales que con m=3, aumentando sus probabilidades de éxito.

Para el resto de las simulaciones de las gráficas de la Figura 5.4 se ha fijado *m*=4 procesadores, un número medio, ni demasiado difícil para todos los algoritmos ni demasiado fácil, y se han ido variando el resto de parámetros. En la Figura 5.4-(b) los resultados son similares a los de la figura anterior. Cuando el número de tareas aumenta el número de recursos se reduce, ya que el número de procesadores se mantiene. En estas situaciones HDP se comporta mejor que el resto porque al tener dos niveles de la prioridad que rompen la ejecución de las tareas periódicas en dos fases tiene más flexibilidad para adaptarse a estas circunstancias. Cuando el número promedio de tareas es 12 su rendimiento es mayor al doble del resto.

En la Figura 5.4-(c) podemos ver que el porcentaje de éxito decrece cuando la utilización de tareas aumenta, pues es más difícil para los algoritmos el acomodarlas. De nuevo, HDP puede partir estas tareas grandes y, a pesar de la disminución, el porcentaje de éxito todavía alcanza los valores superiores al 70% para los casos con tareas muy difíciles de planificar por ser muy grandes ($E[U_i]$=90%).

En la Figura 5.4-(d) representamos el porcentaje de éxito en función de la desviación estándar del factor de utilización de las tareas (stdev[U_i]). Cuando la variación es pequeña las tareas son todas de dimensiones parecidas y bastante grandes ($E[U_i]$=50%). Esto afecta negativamente a todos los algoritmos, pero en especial a RM-FFDU, el cual tiene mayores problemas para acomodar estas tareas grandes y similares. Algo parecido ocurre con RM-FFDU+respan, pero en menor medida. Este algoritmo en el resto de las gráficas mantenía las diferencias con el resto en las situaciones difíciles, pero esta vez proporcionalmente empeora más, teniendo más dificultades para empaquetar las tareas.

En resumen, de los experimentos detallados en la Figura 5.4 podemos concluir que HDP se comporta bien en todas las circunstancias, obteniendo porcentajes de éxito superiores al 80% en la mayoría de los casos, independientemente de la caracterización de la carga. Incluso cuando existen tareas periódicas muy grandes

puede ser exitoso debido a que con el método de la prioridad dual es como si las tareas se partiesen automáticamente en dos subtareas, siendo así más fáciles de acomodar. También es interesante remarcar que es el único método de planificación global que es superior a RM-FFDU+respan, el mejor de los métodos de particionado.

La disposición inicial del experimento [AnJ00b] se diseñó para contener cargas periódicas muy diversas, puesto que se sabe que los planificadores globales pueden tener problemas incluso con algunas cargas relativamente bajas (según el Efecto Dhall [Dha77, DL78]). A la par, es comprensible que los casos más difíciles de planificar sean aquellos con cargas altas. También, en el experimento anterior hemos constatado que el hecho de no tener en cuenta la carga generada distorsiona algo los resultados (por ejemplo, en la Figura 5.4-(a) el SR aumenta con *m* ya que la carga relativa se reduce y no estrictamente debido a que *m* crece). Por lo tanto, para los experimentos de la Figura 5.5 hemos utilizado el mismo generador de tareas sintéticas y la misma parametrización que en el resto, pero hemos clasificado los conjuntos de tareas generados según la carga total resultante con el objetivo de no mezclar los conjuntos de tareas y también para poder probar conjuntos con una carga muy alta.

En la Figura 5.5-(a) podemos observar como evoluciona el SR cuando el número de procesadores crece pero la utilización del sistema se mantiene al 80%. A diferencia del experimento anterior, en general el SR no aumenta con el número de procesadores (recordemos que antes si lo hacía, ya que la carga total generada era menor y ahora es el 80% constante). Con HDP ahora el SR se mantiene al 100% y con AdaptativeTkC está alrededor del 90%. Esto puede ser debido a que para cargas altas el se genera una mayor cantidad de tareas y con U_{max} menores. RM-FFDU+respan se comporta bien, aunque parece que su rendimiento tiende a la baja cuando el número de procesadores aumenta. El otro planificador global, GRM, obtiene unos resultados muy bajos cuando *m* crece y no es aplicable para estas cargas (cuando *m* crece es más difícil cumplir $U_i \leq m/(3m-2)$, véase el apartado 2.4.1.3).

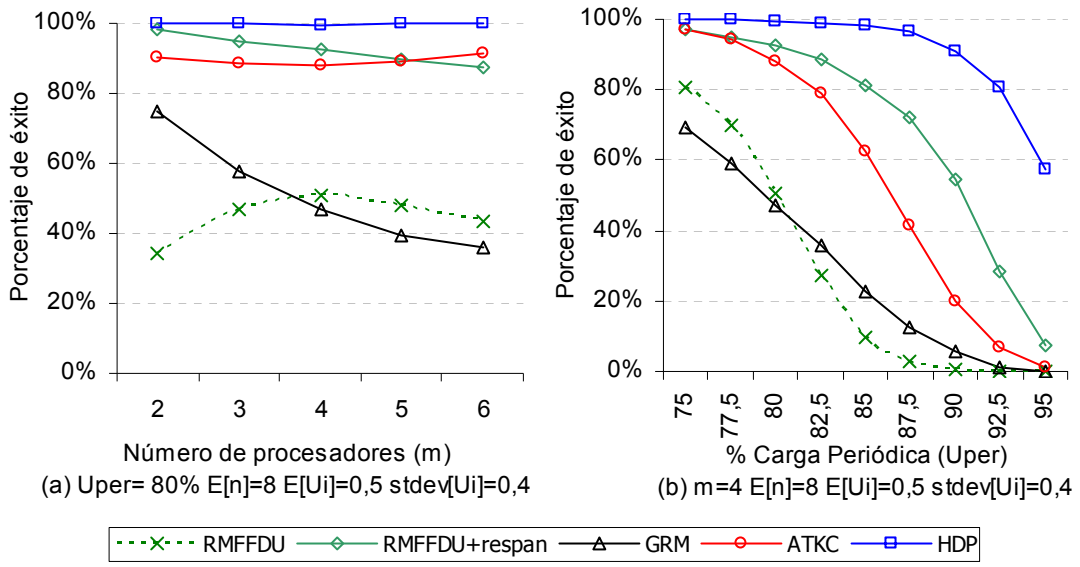


Figura 5.5: Porcentaje de conjuntos de tareas planificados con éxito cuando el número de procesadores varía (E[n]=8 E[U_i]=0,5 stdev[u]=0,4)

En la Figura 5.5-(b) nos hemos centrado en cargas altas. En primer lugar, observamos que el rango 85%-90% es el límite superior, algo parecido a lo que sucede en el dominio de los monoprocesadores [LSD89]. En segundo lugar, el algoritmo de particionado tiene una caída notoria a partir del 87,5% aproximadamente debido a la discretización de los tamaños de los compartimentos disponibles y de las tareas a empaquetar. Generalmente llega un punto en el cual las tareas tienen unos tamaños que no caben en ningún procesador. Por su parte, AdaptativeTkC en las otras gráficas estaba unos 10 puntos por debajo pero ahora, con cargas grandes, está aproximadamente a unos 20-30 puntos por debajo de RM-FFDU+respan. Finalmente, es remarcable que GRM no es factible para la mayoría de casos con cargas altas (véase el apartado 2.4.1.1). En el otro extremo, HDP puede mantener altos porcentajes de éxito hasta aproximadamente el 95%. Esto es debido a que puede acomodar tareas en procesadores de forma que se exceda su capacidad total (es decir, puede desbordar los compartimientos o *bins*). Llegado un cierto punto, la carga total es tan alta que los procesadores ociosos no pueden ayudar a los procesadores sobrecargados y el rendimiento también cae. Estos resultados corroboran las conclusiones del experimento anterior y además demuestran que las cargas alcanzables son incluso superiores al 90% pero no se puede llegar al 100% (con solo tareas periódicas).

5.3.3 Expulsiones y migraciones

En los experimentos anteriores hemos mostrado las ventajas del planificador global HDP pero este puede tener algunos costes adicionales, como por ejemplo el hecho de migrar tareas entre procesadores podría tener sus costes. Consecuentemente, para completar la comparativa entre los distintos algoritmos se han tomado en consideración las expulsiones y las migraciones de tareas.

El mecanismo de las promociones de prioridad del algoritmo HDP puede tener dos efectos adicionales no deseados. Primero, el número de expulsiones de tareas que se están ejecutando puede verse incrementado. Segundo, el número de migraciones también puede aumentar, debido a la capacidad de balanceo de la carga y debido a que las tareas promocionadas deben ejecutarse en un procesador concreto[29]. Sin embargo, gracias a que sabemos que las promociones son la causa de un gran número de expulsiones, también podemos usar un mecanismo que las reduzca. La idea consiste en usar un mecanismo de renombrado de los procesadores y así no tener que migrar una tarea promocionada, evitando así una migración (a su procesador) y una más que posible expulsión (en su procesador). De hecho, en el peor de los casos, una promoción puede provocar una migración, una expulsión y otra migración (de la tarea expulsada al procesador de donde proviene la tarea promocionada).

El mecanismo de renombrado de procesadores es un mecanismo sencillo, con costes bajos de memoria (dos vectores) y de cómputo (durante las promociones y al finalizar una tarea, con coste constante). La idea es la siguiente: cuando una tarea que se está ejecutando en un procesador (P_{run}) es promocionada y es la única de su procesador (P_{home}), esta no migrará al procesador físico correspondiente (P_{home}), sino que se produce un intercambio lógico de los nombres de los procesadores ($P_{run} \leftrightarrow P_{home}$). Los detalles se pueden encontrar en el pseudos-código del Algoritmo 5-2. El número de procesador (o nombre físico) se usa como índice del vector *actua_como* y el contenido indica el nombre lógico. El vector *quien_me_sirve* se indexa con el número de un procesador lógico y devuelve el número del procesador físico que

[29] Obsérvese que las promociones de GDP del punto 4.3.2 no obligan a migrar a ningún procesador concreto y, por lo tanto, el número de migraciones y desalojos es menor.

actúa como el procesador requerido (la información de este segundo vector es redundante, pero evita un bucle de búsqueda en el primer vector).

```
1: rutina swap_nombres(p1,p2)
2:    tmp= actua_como[p1]
3:    actua_como[p1]= actua_como[p2]
4:    actua_como[p2]= tmp
5:    quien_me_sirve[actua_como[p1]]= p1
6:    quien_me_sirve[actua_como[p2]]= p2
7: fin_rutina

1: rutina promocionar_tarea(t)
2:    t.prioridad= t.hpl
3:    si (t.status = running) entonces
4:    si (t.cpu != t.home) entonces
5:          si (ocioso(actua_como[t.home]) o
6:          prioridad(actua_como[t.home]) < t.prioridad)) entonces
7:                swap_nombres(t.cpu , t.home)
8:          sino
9:                encola_HPLRQ(t.home)
10:         fin_si
11:   sino
12:         mover_de_GRQ_a_HPLRQ(t.home)
13:   fin_si
14: fin_rutina

1: rutina finalizar_tarea(t)
2: ...
3:    si (t.cpu = t.home) entonces
4:          buscar t2 de mayor prioridad tal que t2.home = t.home
5:          si encontrada entonces
6:                swap_nombres(t.home , t2.cpu)
7:          fin_si
8:    fin_si
9: ...
10: fin_rutina
```

Algoritmo 5-2: pseudo-código del renombrado de procesadores para ahorrar expulsiones y migraciones en la versión HDP+ren

Para los experimentos de la Figura 5.6 hemos utilizado la misma parametrización que anteriormente (PCT # 6 del Anexo B) y hemos simulado la versión de HDP que incorpora el método de renombrado de procesadores (HDP+ren). En este caso hemos simulado 5.000 conjuntos de tareas para cada punto y hemos medido las densidades de expulsiones y de migraciones, calculando su media aritmética. Estas densidades se calculan dividiendo el número de las expulsiones o de las migraciones por la longitud de la simulación respectivamente (un hiperperiodo). De esta forma hemos normalizado las cifras, puesto que la longitud de la simulación puede variar mucho en función de los periodos generados[30]. También, para poder hacer una comparativa justa, hemos tomado solamente en la consideración los conjuntos de tareas que eran planificables por todos y cada uno de los algoritmos de planificación[31] utilizados.

En los experimentos de la Figura 5.6 se puede comprobar como HDP es el algoritmo que tiene mayor número de expulsiones y de migraciones. Cuando el número de procesadores aumenta (gráficas (a) y (b)) la diferencia entre HDP y el resto de planificadores es abismal, llegando a quintuplicar los costes. Este es el precio a pagar para tener un porcentaje de éxito tan alto. Sin embargo HDP+ren mantiene el porcentaje de éxito y obtiene unos resultados excelentes en lo referente a expulsiones y migraciones: es el algoritmo que experimenta menos expulsiones y el que experimenta menos migraciones (exceptuando RMFFDU, que es local y, por lo tanto, no tiene migraciones).

En la Figura 5.6-(a1) se observa que los algoritmos de particionado tienen más expulsiones que los algoritmos globales (aparte de HDP sin renombrado). Esto es razonable si tenemos en cuenta que cuando se activa una tarea más prioritaria los planificadores locales no pueden eludir la expulsión mientras que los globales se las pueden ahorrar si pueden hallar otro procesador donde ejecutarla y las probabilidades aumentan cuando la carga es baja o cuando aumenta el número de procesadores. Efectivamente podemos comprobar este efecto en la Figura 5.6-(a1). Con el aumento del número de los procesadores, como el resto de los parámetros se mantiene, la carga generada por procesador disminuye y entonces los planificadores globales

[30] El hiperperiodo máximo posible es 72.072.000
[31] Hemos eliminado RMFFDU ya que aporta poco y restringe mucho el número de casos muestreables

pueden realizar balanceo de la carga, disminuyendo así la densidad de expulsiones. Incluso teniendo en cuenta las migraciones[32] (en la Figura 5.6-(a2)) cuando el número de procesadores aumenta los costes son mayores para RMFFDU que para los globales.

En el resto de las gráficas de la Figura 5.6 nos hemos centrado en las cargas más altas. En la Figura 5.6-(b1) la carga se ha mantenido al 80% y, si bien todos los métodos han incrementado el número de expulsiones, las diferencias se mantienen. Esto es así a expensas de un mayor número de migraciones con los globales, como puede verse en la Figura 5.6-(b2), donde RMFFDU es el método con menores costes. Sin embargo, en la misma figura puede observarse que cuando el número de procesadores es grande los costes entre los métodos globales y los locales se igualan, puesto que se incrementan las migraciones pero se reducen las expulsiones.

Para elaborar la Figura 5.6-(c1) y la Figura 5.6-(c2) se han tenido que contabilizar las densidades en todos los conjuntos de tareas, no solo para aquellos que son planificables con todos los métodos (como en (a1), (a2), (b1) y (b2)), puesto que se han ensayado cargas muy altas y el porcentaje de éxito baja mucho para la mayoría de planificadores. En estas gráficas se puede apreciar como las densidades aumentan en todos los algoritmos cuando la carga crece. Para cargas mayores al 85% las densidades de RMFFDU no crecen tanto, debido a que planifica menos conjuntos de tareas. RMFFDU pasa de tener el doble de expulsiones que HDP+ren a solo un 10% más para las cargas mayores. GRM pasa de estar un 33% por encima de HDP+ren a un 12% para cargas altas. Si tenemos en cuenta las migraciones (Figura 5.6-(b2)) los planificadores globales ahora tienen unos costes mayores. Las diferencias entre ellos aproximadamente se mantienen. GRM tiene unas densidades alrededor de un 25% superiores a HDP+ren y AaptativeTkC un 10%.

[32] En multiprocesadores de memoria compartida el coste de las expulsiones puede ser similar al coste de una migración entre procesadores (principalmente debido a la recarga de la cache).

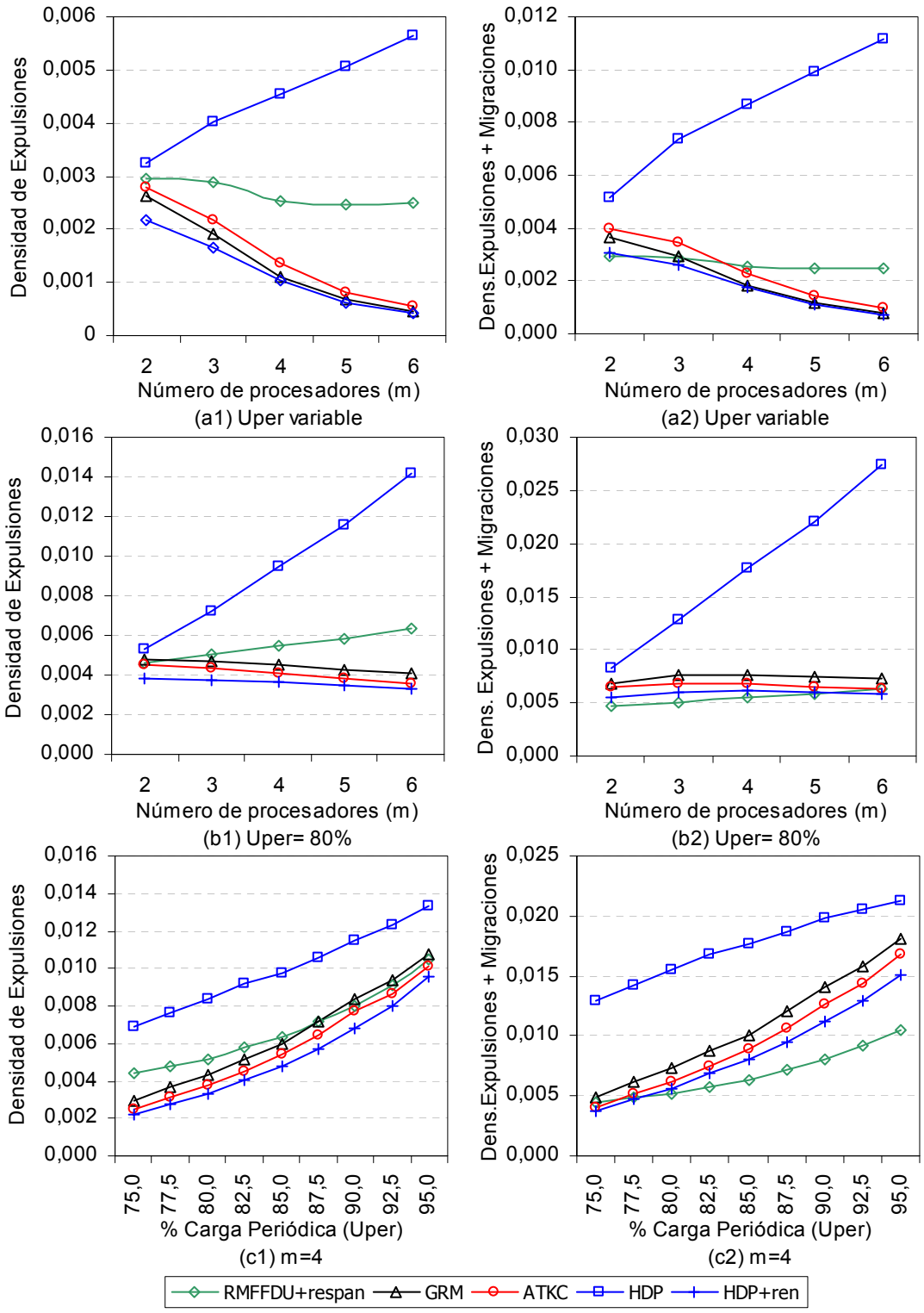


Figura 5.6: Densidad de expulsiones y de migraciones medidos en los experimentos anteriores. (a) los obtenidos para la Figura 5.4-(a); (b) los obtenidos en Figura 5.5-(a); (c) los obtenidos en Figura 5.5-(b)

5.3.4 Experimentos de distribución de la carga en tiempo de diseño

En los experimentos anteriores hemos comprobado que HDP+ren es netamente superior al resto cuando las cargas son altas (a partir de ahora simplemente le llamaremos HDP[33]). Pero ahora nos preguntamos si en realidad su efectividad viene del planificador o de la distribución utilizada. En las secciones anteriores para la distribución hemos utilizado RM-FFDU como base porque es muy efectiva en términos de utilización de los procesadores (véase la Tabla 9 en la página 60 y [AnJ00b]), pero tiene el posible inconveniente que tiende a concentrar la carga en algunos procesadores, puesto que es una política *First-Fit*. Como ya hemos mencionado, el planificador HDP se encarga de compensar esta distribución asimétrica de la carga balanceándola en tiempo de ejecución, pero esto puede ir en detrimento del servicio a otros tipos de tareas (véanse los apartados 5.4 y 5.5). Además, si en realidad es el planificador el que se encarga de paliar las limitaciones del algoritmo de empaquetado entonces podríamos utilizar una distribución menos efectiva y el planificador ya lo complementaría. En particular, nos interesa comprobar el funcionamiento del planificador cuando se utilizan otras distribuciones que balanceen más la carga en estático.

Como ya hemos indicado, HDP no requiere que las tareas asignadas a un procesador sean planificables localmente e incluso permite que la carga de un procesador supere el 100%. Debido a esta relajación en las restricciones de empaquetamiento, los algoritmos de distribución que utilizamos en esta sección no se corresponden estrictamente con ninguno de los hallados en la literatura sino que son adaptaciones de estos. Los pseudos-códigos están detallados en los Algoritmos 5-3 y sus principales características están resumidos en la Tabla 13. Estos algoritmos pueden sustituir FM-FFDU en la línea 6 del Algoritmo 5-1.

Puesto que el orden decreciente de utilizaciones se ha mostrado efectivo, hemos añadido dos variantes que balancean la carga, WFDU (*Worst-Fit Decreasing-Utilization*) y WFDU+test. El primero no comprueba la planificabilidad y el segundo

[33] El rendimiento es exactamente el mismo. La diferencia está en el número de expulsiones y migraciones.

la comprueba usando la prueba de Joseph y Pandya en [JoP86] que, aunque no es requerido siempre será un valor añadido el tener el mayor número de tareas garantizadas. Según [LDG01] la política Worst-Fit es la que tiene un límite de utilización más bajo, pero si se usa el orden decreciente de utilización de las tareas entonces el límite es el mismo para cualquier esquema de distribución con RM. Además, el hecho de utilizar la prueba de [JoP86] incrementa más la utilización de los procesadores.

También hemos probado otro algoritmo que concentra la carga, LLFF (*Least-Laxity First-Fit Balanced*, Algoritmo 5-5 en la página 198), porqué es el que utilizaremos más adelante para incluir tareas aperiódicas con plazo (véase el apartado 5.5). Finalmente hemos incluido RMRR[34] (*Rate-Monotonic Round-Robin*), el cual balancea el número de tareas y sus periodos, pero no necesariamente la carga y las demás características. Si las tareas más grandes son las de periodo mayor entonces no balanceará la carga. Como no tiene en cuenta la carga ni la prueba de planificabilidad es el que esperamos que de peores resultados, pero lo hemos incluido para ver si cualquier método sirve.

Aunque los algoritmos WFDU+test y LLFF no generan sobrecargas, el Algoritmo 5-1 a posteriori las puede generar, puesto que asigna las tareas que todavía permanecen en el conjunto de tareas τ al procesador menos cargado (líneas 7 a 14).

Algoritmo	¿Usa prueba de planificabilidad?	¿Genera sobrecargas?	¿Concentra la carga?	Orden de las tareas
RMRR	No	Sí, cualquiera	depende	Periodos crecientes
WFDU	No	Sí, el menos cargado	No	Utilización decreciente
WFDU+test	J&P (ecuación 2.2)	No	No	Utilización decreciente
DP-FFDU	J&P (ecuación 2.2)	Sí, el menos cargado	Sí	Utilización decreciente
LLFF	J&P (ecuación 2.2)	No	Sí	Laxitud creciente

Tabla 13: Resumen de las características de los algoritmos de distribución ensayados

[34] Sería equivalente a *Rate-Monotonic Next-Fit* pero sin comprobar la planificabilidad

```
1: Algoritmo RMRR (τ, PS)            { Task Set, Processor Set }
2:    ordenar (τ, orden RateMonotonic )
3:    p= primer procesador
4:    para cada tarea τi ∈ τ hacer
5:          quitar τi de τ
6:          asignar τi a p
7:          p= siguiente procesador       { round robin }
8:    fin_para

1: Algoritmo WFDU (τ, PS)            { Task Set, Processor Set }
2:    ordenar (τ, orden decreciente de factor de utilización)
3:    para cada tarea τi ∈ τ hacer
4:          p= procesador menos cargado de PS   { Worst-Fit }
5:          quitar τi de τ
6:          asignar τi a p
7:    fin_para

1: Algoritmo WFDU+test (τ, PS)       { Task Set, Processor Set }
2:    ordenar (τ, orden decreciente de factor de utilización)
3:    para cada tarea τi ∈ τ hacer
4:          encontrado= falso
5:          PSno_visitados= PS
6:          mientras no encontrado y PSno no vacio hacer
7:                p= procesador menos cargado de PSno
8:                si Test_J&P(p, τi) entonces          ; ecuación 2.2
9:                      encontrado= cierto
10:               sino
11:                     quitar p de PSno
12:         fin_mientras
13:         si encontrado entonces
14:               quitar τi de τ
15:               asignar τi a p
16:         fin_si
17:   fin_para
```

Algoritmos 5-3: Algoritmos para la distribución de tareas periódicas en tiempo de diseño alternativos a RM-FFDU para utilizarlos con HDP.

Para comprobar distintas distribuciones hemos repetido los experimentos de la Figura 5.5 y los representamos en la Figura 5.7. Las gráficas superiores se corresponden con la Figura 5.5-(a) y Figura 5.6-(b2) respectivamente, mientras que las inferiores se corresponden con la Figura 5.5-(b) y Figura 5.6-(c2) respectivamente.

En las dos gráficas de la izquierda de la Figura 5.7 podemos observar que el porcentaje de éxito es bastante independiente de la distribución que se haga en tiempo de diseño, con la excepción de RMRR, el cual reparte las tareas un poco a ciegas. En este caso el rendimiento no es aceptable cuando la carga o cuando el número de procesadores crecen, indicándonos que no vale cualquier distribución. Para el resto de algoritmos el número de procesadores no parece influir y para cargas superiores al 90% solo el algoritmo LLFF pierde un 10%. Esto nos confirma que el mérito del aumento de la planificabilidad es más del planificador y no tanto de la distribución inicial de las tareas. Por lo tanto, dado que el porcentaje de éxito es parecido en las distribuciones balanceadas pueden ser las más razonables de usar.

En las dos gráficas de la derecha de la Figura 5.7 se muestran la suma de las densidades de expulsiones y las densidades de migraciones. En este caso si que se observan algunas diferencias entre los métodos, aunque solo varían un máximo del 10%. El que tiene menores densidades es RMRR, pero los valores no son muy significativos, puesto que solo se han medido los pocos conjuntos de tareas en que tiene éxito. De entre los que tienen mayor éxito, DP-FFDU es el que tiene unos costes en términos de expulsiones y migraciones menores. WFDU incrementa entre un 1% y un 10%. LLFF incrementa un 9% constante. Finalmente WFDU+test es muy parecido a DP-FFDU.

Como conclusión final y a la espera de otros criterios aportados en los próximos apartados, podemos concluir que DP-FFDU y WFDU+test son las mejores opciones para la distribución de las tareas periódicas entre procesadores en tiempo de diseño para el planificador HDP.

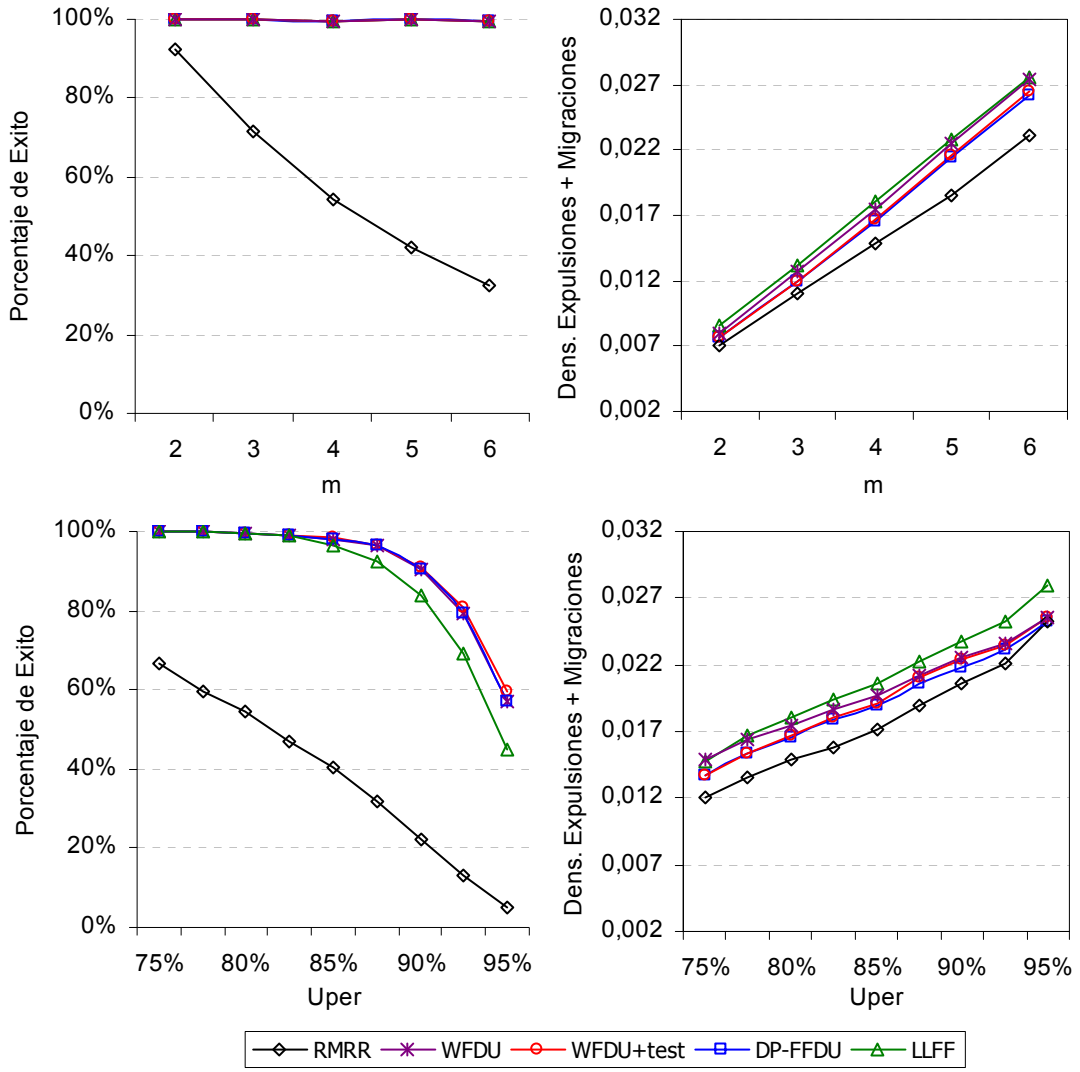


Figura 5.7: A la izquierda, porcentaje de conjuntos de tareas periódicas planificado con éxito con HDP y diversos métodos de distribución de las tareas. A la derecha, suma de las densidades de expulsiones y de migraciones (arriba, en función del número de procesadores, abajo en función de la carga del sistema)

5.3.5 Conclusiones

La fase dinámica de la ejecución de las tareas aporta bastante flexibilidad para realizar balanceo de la carga. Debido a que la ejecución de la tarea está partida en dos fases o prioridades y debido a este balanceo de la carga, el número de las aplicaciones planificables es mayor con el algoritmo HDP que con el resto de los algoritmos experimentados. Este método tiene éxito en un elevado porcentaje de conjuntos de tareas y es independiente de las características de estas. En particular no se ve afectado por el efecto Dhall. Por otra parte, y de importancia capital en los sistemas de tiempo real estrictos o críticos, la fase estática garantiza los plazos de las tareas periódicas.

Las modificaciones establecidas en este apartado permiten mejorar la planificabilidad y la utilización de los recursos de cómputo. Es decir, permiten en tiempo de diseño utilizar un menor número de procesadores que otros métodos o bien permiten realizar más trabajo con un número fijo de procesadores.

Cuando HDP se quiera utilizar para sistemas de tiempo real estricto, los diseñadores de la aplicación deberían realizar la simulación durante un hiperperiodo para verificar si el sistema es planificable (en el caso que haya alguna tarea periódica no garantizada). Sin embargo, cuando se quiera utilizar en sistemas de tiempo real laxos, no se requiere ninguna simulación y el sistema funcionará correctamente con una probabilidad muy alta. Además, las tareas susceptibles de incumplir algún plazo se pueden identificar para poder prever las consecuencias en la aplicación.

5.4 Servicio a tareas aperiódicas sin plazo

En este apartado describimos como usar la flexibilidad de HDP para dar servicio a tareas aperiódicas sin plazo de forma que el tiempo de respuesta medio que éstas obtengan sea relativamente bajo. El esquema general del planificador utilizado es el mismo de la Figura 5.2 en la página 156.

En tiempo de ejecución solo hace falta ajustar la asignación de las prioridades de las tareas aperiódicas. A estas tareas se les asignará una única prioridad en el nivel LPL pero será superior a las prioridades de las tareas periódicas en este nivel. Todas las tareas aperiódicas tendrán la misma prioridad pero se encolarán en la cola GRQ según el orden de llegada al sistema, o sea, se les dará un tratamiento FIFO en la cola. Sin embargo, cabe matizar que el orden de finalización entre ellas no será estrictamente FIFO debido a las posibles expulsiones del procesador cuando ocurren las promociones de las tareas periódicas.

Asignando la prioridad más alta de LPL a las tareas aperiódicas, éstas tendrán preferencia sobre las tareas periódicas no promocionadas, la ejecución de las cuales podrá ser retardada hasta su promoción. De esta forma el tiempo de respuesta de las tareas aperiódicas será posiblemente menor que si tuvieran que esperar a que terminasen todas las tareas periódicas (ejecución en segundo plano), o a que se activase un servidor de tareas aperiódicas (como los utilizados en el Capítulo 3).

En tiempo de diseño se debe realizar una distribución de las tareas periódicas de forma que favorezca el servicio a las tareas periódicas. Esta distribución es la que se analiza en el siguiente apartado.

5.4.1 Distribución de las tareas periódicas en tiempo de diseño

La distribución de las tareas periódicas en tiempo de diseño para hallar las promociones de éstas afectará al servicio que se dará en tiempo de ejecución a las tareas aperiódicas sin plazo. El objetivo los experimentos de este apartado es el averiguar que tipo de distribución es la más favorable para HDP. Las parametrizaciones de las tareas han sido las mismas que las utilizadas en el Capítulo

3 y el Capítulo 4. El resultado se utilizará en el próximo apartado para realizar nuevos experimentos con distintas caracterizaciones de la carga y para compararlo con otros algoritmos.

En los experimentos de este apartado se ha usado un método de distribución balanceado y dos que concentran la carga. La distribución *Balanced* es la realizada por el generador de conjuntos de tareas sintéticas. En ésta, todos los procesadores tienen la misma carga, con igual número de tareas, periodos similares y factores de utilización similares. Del apartado 5.3.4 utilizamos la distribución balanceada WFDU+test y la distribución FFDU+test[35] (*First-Fit Decreasing-Utilization*). Del apartado 5.5.1 utilizamos la distribución no balanceada LLFF (*Least-Laxity First-Fit*).

Aunque, como se ha visto anteriormente, el planificador puede equilibrar la carga en tiempo de ejecución, en las gráficas de la Figura 5.8 y la Figura 5.9 se puede observa que la mejor opción para el tratamiento de las tareas aperiódicas es siempre balancear la carga periódica en tiempo de diseño. Las distribuciones balanceadas permiten ejecutar tareas aperiódicas en cualquier procesador. Por el contrario, si cargamos mucho unos procesadores y dejamos otros más descargados, los primeros podrán servir pocas tareas aperiódicas (puesto que tendrán tareas promocionadas durante más tiempo) y se formarán largas colas en los segundos, aumentando así el tiempo de respuesta medio de las tareas aperiódicas.

En las gráficas (a) y (b) de la Figura 5.8 se puede observar que las distribuciones balanceadas son las que propician los menores tiempos de respuesta medios de las tareas aperiódicas, independientemente de la proporción entre la carga periódica y la carga aperiódica (eje de las abscisas). La diferencia en rendimiento entre LLFF y FFDU+test está alrededor del 5%. FFDU+test es en promedio un 15% peor que las opciones balanceadas. Solo cuando la carga periódica es muy alta y la aperiódica muy baja el rendimiento es equivalente, puesto que no existen dificultades en servir una carga aperiódica baja. Así, la máxima diferencia se obtiene cuando la carga periódica es alta y la carga aperiódica significativa (75+20 y 65+30), puesto que

[35] Equivale a RM-FFDU+respan de aquel apartado, pero solo la distribución, no la planificación.

cuando son bajas todos los procesadores están descargados, úsese la distribución que se use. Esto se confirma en la Figura 5.8-(c), con una distribución de las cargas 70+25, donde FFDU+test es aproximadamente un 30% peor que las distribuciones balanceadas. En la misma gráfica, en general las diferencias entre distribuciones son bastante constantes, independientemente de U_{max}. Solo para U_{max} muy grandes se reducen un poco las diferencias (bajan al 20%), puesto que todos los métodos deben repartir las tareas grandes de forma equilibrada para que sean planificables.

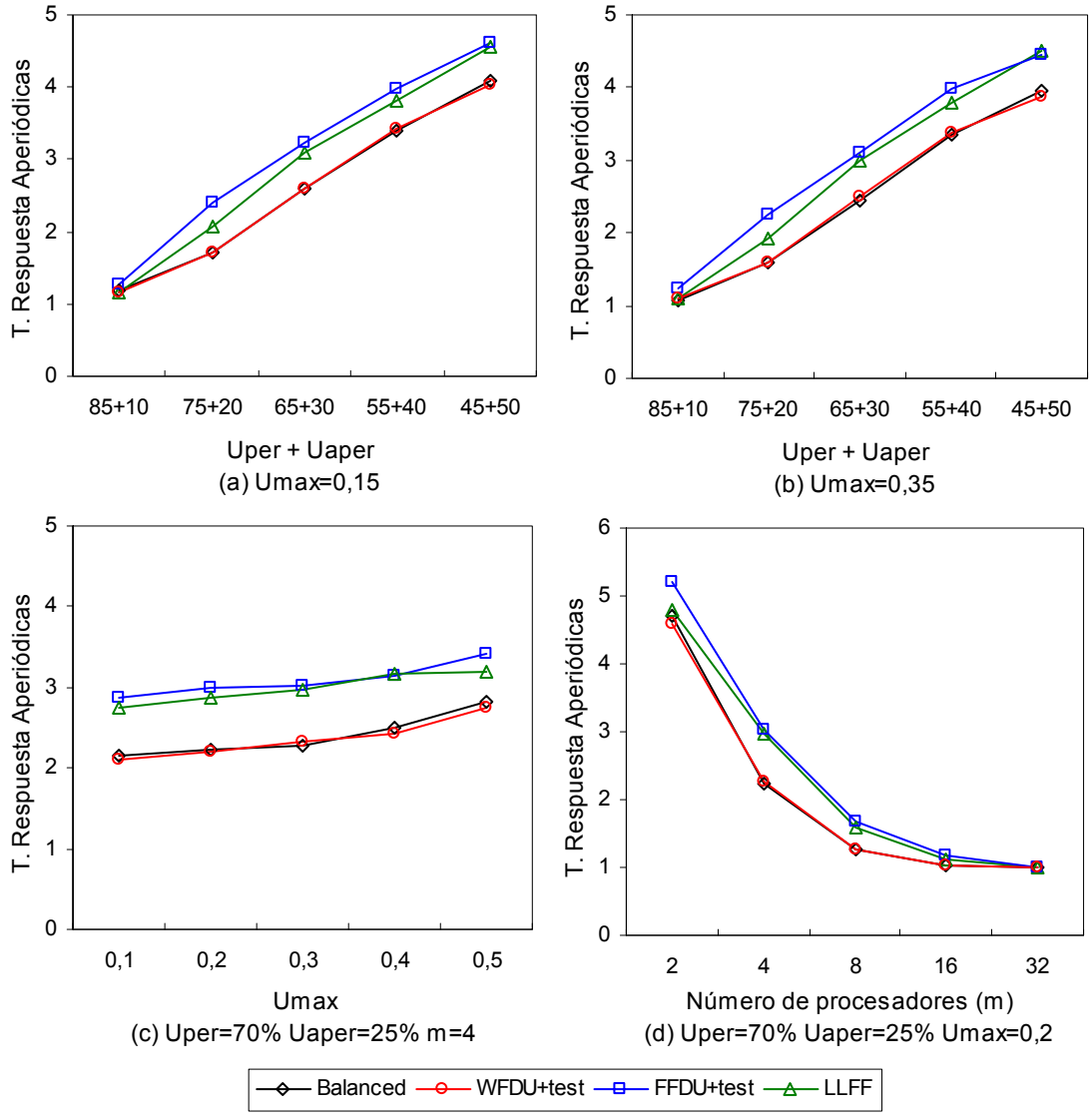


Figura 5.8: Tiempos de respuesta medios obtenidos con HDP y diversos métodos de distribución de las tareas periódicas con la parametrización PCT#0. En (a) y (b) se varía la distribución de la carga, manteniéndose la carga total en el 95%. En (c) y (d) la carga es 70%+25% y se varía Umax y el número de procesadores respectivamente.

En la Figura 5.8-(d) se varía el número de procesadores. También aquí las mejores opciones son las balanceadas pero cuando el número de procesadores es superior a 16 entonces las diferencias se reducen notablemente puesto que el mecanismo de balanceo automático de HDP es más efectivo y contrarresta la concentración de carga de la distribución en tiempo de diseño.

En los experimentos de la Figura 5.9 se ha utilizado la parametrización PCT # 1 y PCT # 4 (véase el Anexo B). Los resultados confirman que el balanceo de la carga es el mejor método independientemente de la proporción entre la carga periódica y la carga aperiódica (eje de las abscisas en las gráficas), independientemente de los periodos de las tareas y independientemente del factor de utilización máximo.

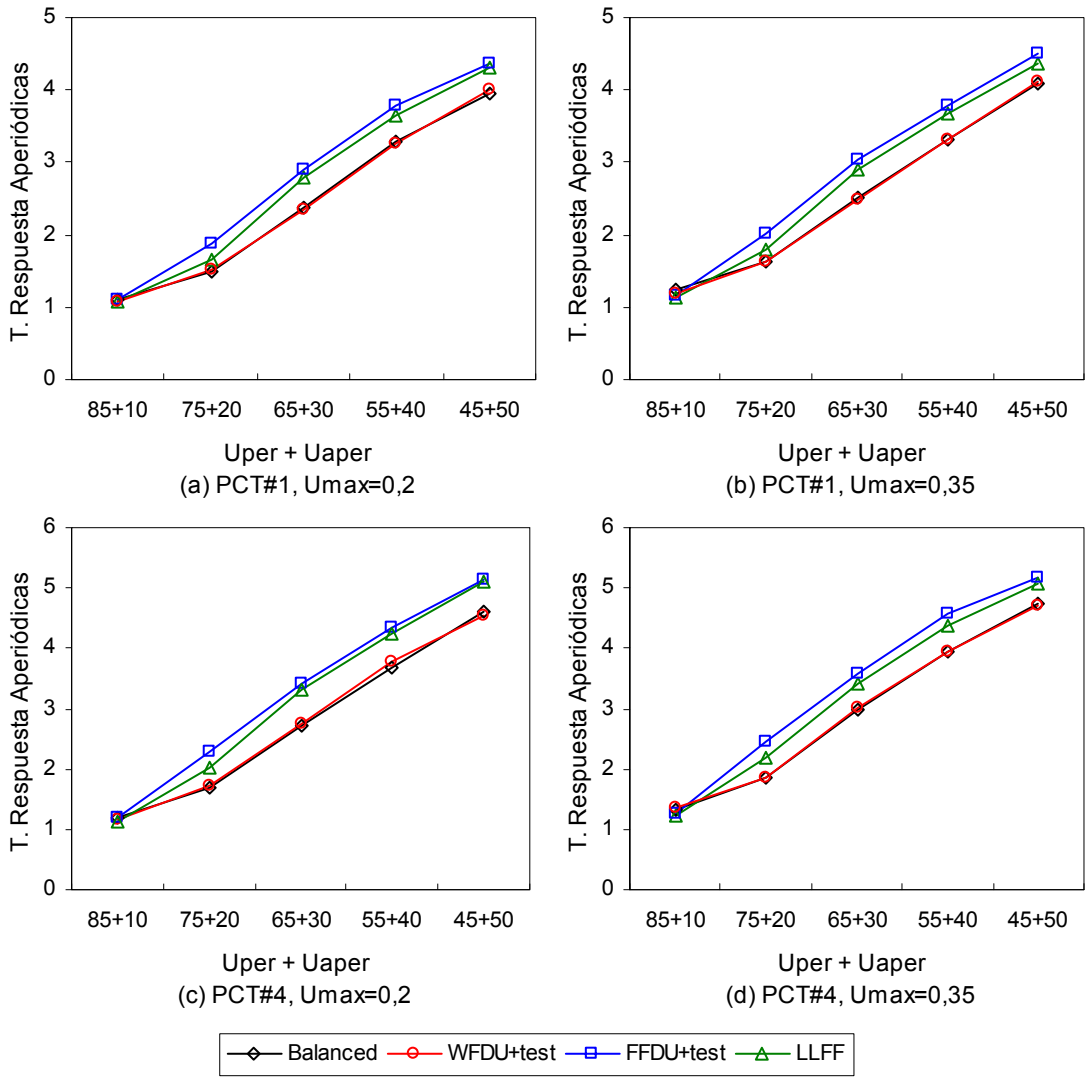


Figura 5.9: Tiempos de respuesta medios obtenidos con HDP y diversos métodos de distribución de las tareas periódicas con las parametrizaciones utilizadas en los capítulos anteriores (PCT#1 y PCT#4). Se varía la distribución de la carga, manteniéndose la carga total en el 95%.

En conclusión, si fuese necesario, HDP puede utilizar una distribución asimétrica de la carga entre los procesadores pero si el objetivo es dar un buen servicio a las tareas aperiódicas sin plazo, entonces las distribuciones balanceadas son las que obtienen mejores resultados. Si el número de procesadores es alto entonces el mecanismo de balanceo automático de HDP es más efectivo y se obtienen resultados equivalentes con las distribuciones no balanceadas.

5.4.2 Evaluación experimental del rendimiento

En el Capítulo 4 analizamos el tiempo de respuesta medio de las tareas aperiódicas con un planificador DP global. Para este apartado hemos realizado los mismos experimentos para comparar el planificador híbrido con el planificador global, es decir, para comprobar en que medida afecta al tiempo de respuesta medio de las tareas aperiódicas la restricción de ejecutar una tarea promocionada en un procesador concreto. El método de distribución en tiempo de diseño utilizado es el balanceado, ya que en el apartado anterior se ha visto que es el que mejor resultados obtenía. La parametrización de los conjuntos de tareas sintéticos y la metodología de las simulaciones son los mismos que los descritos en el apartado 4.3.3. La métrica de rendimiento utilizada es la media aritmética de los tiempos de respuesta medios normalizados de las tareas aperiódicas (TRA).

En la Figura 5.10 se puede observar como el rendimiento de ambos métodos es equivalente. HDP es en todos los casos un poco mejor a GDP_A pero no es estadísticamente significativo. De hecho, es equivalente a GDP_S, el que proporciona menores tiempos de respuesta a las tareas aperiódicas pero necesita más reajustes de las promociones que GDP_A (no lo hemos mostrado para evitar solapamientos innecesarios en las gráficas). Es importante remarcar que con GDP_A y GDP_S era posible que las tareas periódicas incumplieran algunos plazos (de ahí los reajustes de promociones) pero esto no sucede en ningún caso con HDP. Es decir, HDP proporciona un servicio a las tareas aperiódicas equivalente al de los mejores

planificadores globales y además todas las tareas periódicas tienen los plazos garantizados.

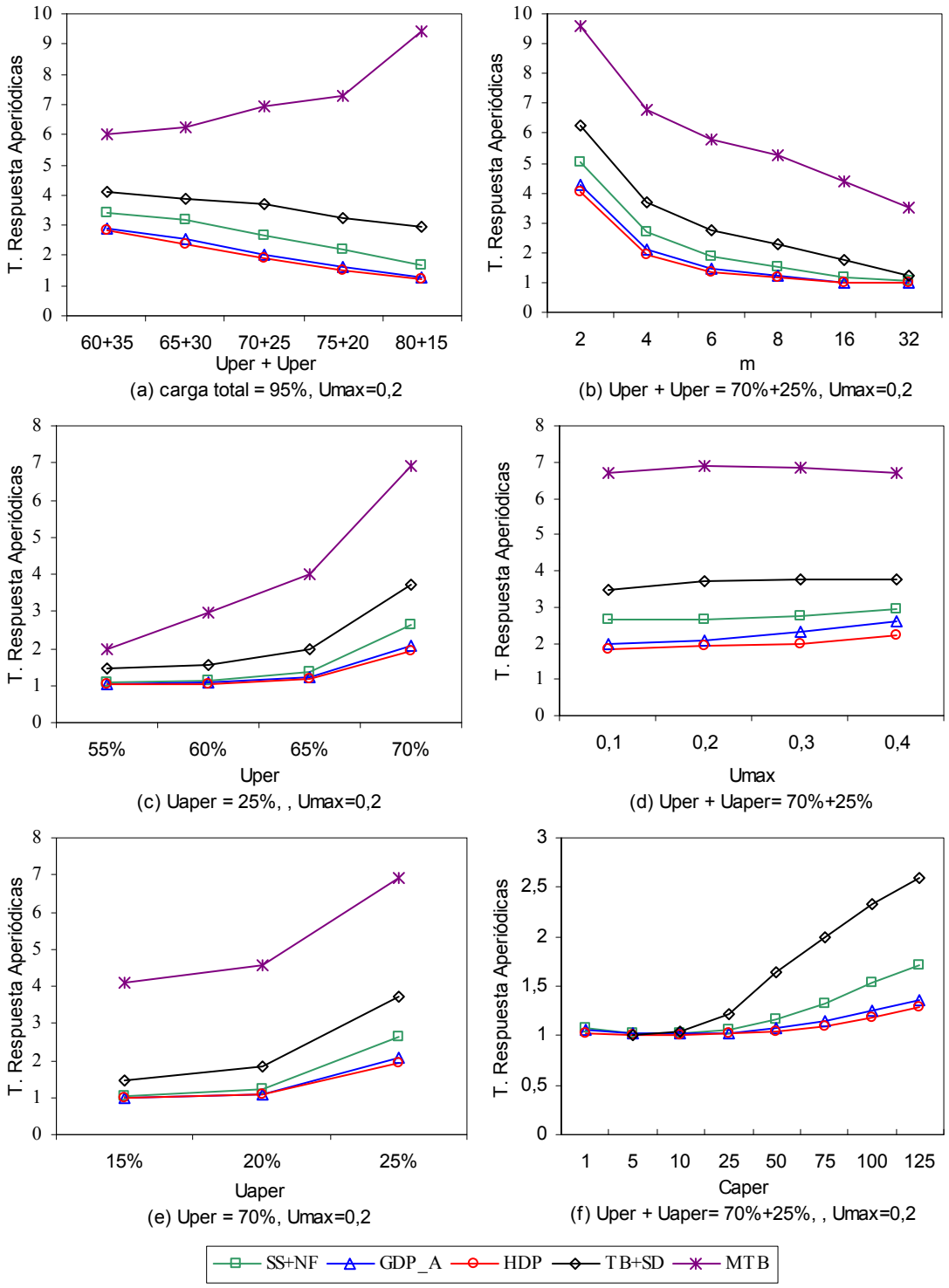


Figura 5.10: Tiempo de respuesta medio de las tareas aperiódicas sin plazo con el planificador HDP (comparado con los resultados obtenidos en el apartado 0).

5.4.3 Conclusiones

Cuando el objetivo es dar un buen servicio a las tareas aperiódicas sin plazo las distribuciones balanceadas permiten a HDP proporcionar los mejores resultados. No obstante, HDP tiene la capacidad de balancear parte de la carga y se pueden utilizar distribuciones asimétricas de la carga entre los procesadores en caso de ser necesario. Si el número de procesadores es alto entonces no importa tanto el tipo de distribución utilizado puesto que el mecanismo de balanceo automático de HDP es suficientemente efectivo.

En ningún caso se han observado diferencias significativas en el servicio a las tareas aperiódicas sin plazo entre el planificador HDP de este capítulo y los planificadores DP globales puros del capítulo anterior (véase el apartado 4.3.2). Sin embargo, HDP tiene la enorme ventaja sobre el planificador GDP que garantiza todos los plazos de las tareas periódicas, siendo así también aplicable a sistemas de tiempo real estrictos. Además, por si esto fuera poco, en el siguiente apartado veremos que HDP, al ser determinista, también es capaz de aceptar y garantizar tareas aperiódicas con plazo.

5.5 Servicio a tareas aperiódicas con plazo

En este apartado describimos como usar la flexibilidad de HDP para dar servicio a tareas aperiódicas con plazo[36] de forma que el planificador decida si puede ejecutarla antes del plazo y la acepte o bien que la rechace por ser incapaz de garantizarlo.

Para incorporar tareas aperiódicas con plazo en el modelo usaremos la información proporcionada por los tiempos de promoción de las tareas periódicas en el test de aceptación. Para ello, en tiempo de ejecución el planificador necesitará saber cuando se van a producir las siguientes promociones de tareas periódicas[37] en cada procesador (**$Next\_Promotion_p$**). La idea es que si un procesador tiene una promoción posterior al tiempo de cálculo de una determinada petición aperiódica con plazo, este procesador podrá ejecutarla de inmediato y por lo tanto podrá ser aceptada.

Para implementarlo es necesario tener una cola adicional de eventos de promociones para cada procesador. La longitud de esta cola es igual al número de tareas periódicas asignadas al procesador ya que estamos en el caso de tareas periódicas con $T_i \leq D_i$. El orden de las colas se mantiene según el instante de llegada de cada promoción. Cada vez que se produce una promoción τ_j HDP aprovecha para quitar el primero de la cola (correspondiente a la tarea promocionada) y volverlo a encolar pero sumando al instante de activación del evento un periodo (T_j) puesto que esta es la separación entre dos promociones consecutivas de una misma tarea (τ_j). Así, $Next\_Promotion_p$ se corresponde con el tiempo de activación del primer evento de la cola. Cuando un procesador no tiene ninguna tarea periódica asignada esta cola está vacía y entonces se asume que $Next\_Promotion_p = \infty$.

[36] Las tareas esporádicas podrán ser tratadas de la misma forma sin utilizar la información adicional de la separación mínima entre activaciones consecutivas.

[37] Como se verá más adelante, las tareas aperiódicas con plazo también tendrán promociones, pero esto no se contemplará en Next_Promotion, puesto que en las ecuaciones se usará la variable Last_DL, o sea, el plazo asignado a la última tarea aperiódica aceptada

Cuando una tarea aperiódica con plazo $H^k = \{r_h^k, C_h^k, D_h^k\}$ llega, el planificador HDP busca un procesador *p* tal que

$$r_h^k + C_h^k \leq \text{Next\_Promotion}_p \qquad \text{(ecuación 5.1)}$$

Ecuación 5.1 Condición para que la tarea aperiódica H^k sea planificable en el procesador p

Si tal procesador existe, entonces la tarea puede ser aceptada y asignada a este procesador. En caso contrario, la tarea es rechazada.

Además de las tareas periódicas también es necesario tener en cuenta las tareas aperiódicas con plazo ya aceptadas y asignadas a un procesador. Para ello se necesita una variable adicional para cada procesador que tenga en cuenta el último plazo asignado a una tarea aperiódica con plazo (**Last_DL_p**). Así, las condiciones para la aceptación de la tarea $H^k = \{r_h^k, C_h^k, D_h^k\}$ en el procesador *p* y el instante $t = r_h^k$ son:

$$\max(r_h^k, \text{Last\_DL}_p) + C_h^k \leq D_h^k \qquad \text{(ecuación 5.2)}$$

$$\max(r_h^k, \text{Last\_DL}_p) + C_h^k \leq \text{Next\_Promotion}_p \qquad \text{(ecuación 5.3)}$$

La ecuación 5.2 garantiza que es factible que la tarea H^k tenga tiempo de finalizar antes de su plazo en el mejor de los casos, cuando no reciba ninguna interferencia. La ecuación 5.3 garantiza que la tarea no será interrumpida por ninguna tarea periódica promocionada. Las tareas aperiódicas con plazo aceptadas con anterioridad si podrán interrumpirla (puesto que el orden de ejecución de este tipo de tareas es EDF), pero ya están contempladas en el término Last_DL_p. En la Figura 5.11 se muestran los límites impuestos por la ecuación 5.2 y la ecuación 5.3.

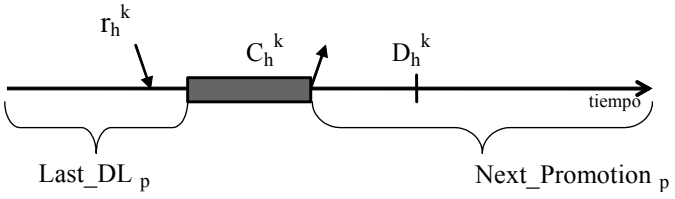


Figura 5.11: Límites para Next_Promotion_p en la elección del procesador. La siguiente promoción en el procesador *p* tiene que ser mayor que $\text{Last\_DL}_p + C_h^k$

Una vez la tarea aperiódica ha sido asignada a un procesador, su plazo es modificado para la ejecución en EDF. Llamamos a este nuevo plazo el plazo efectivo (*effective deadline*, D'^{k}_{h}). Este tiene que satisfacer la siguiente desigualdad:

$$\max(r_h^k, Last\_DL_p) + C_h^k \leq D'^{k}_{h} \leq \min(Next\_Promotion_p, D_h^k) \quad \text{(ecuación 5.4)}$$

La parte izquierda de la ecuación 5.4 representa el plazo mínimo asignable. Cuando a una tarea aperiódica se le asigna este mínimo plazo ésta es ejecutada sin ninguna expulsión, puesto que en EDF no habrá ningún plazo entre $Last\_DL_p$ y su plazo y tampoco habrá ninguna promoción (y por consiguiente ninguna tarea de alta prioridad la expulsará del procesador).

La parte derecha de la ecuación 5.4 representa el plazo máximo asignable. De otra forma, o bien la tarea no terminaría antes de que venza su plazo o bien llegaría la siguiente promoción, con la consiguiente expulsión por parte de una tarea mayor nivel de prioridad. En este último caso, no se puede asegurar, con la información disponible, si la tarea aperiódica con plazo podría terminar a tiempo y por lo tanto no se puede aceptar la tarea con garantías.

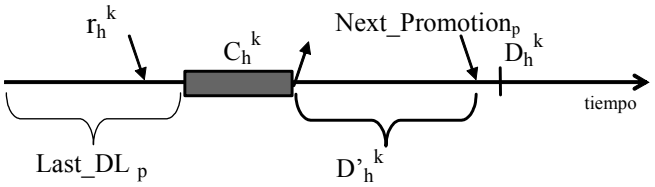


Figura 5.12: Límites para el *plazo efectivo* (D'^{k}_{h}) una vez se ha elegido el procesador. La tarea no puede empezar antes del último plazo efectivo asignado anteriormente. Su plazo efectivo no puede ser posterior al plazo real ni posterior a la siguiente promoción del procesador *p*

Las ecuaciones anteriores permiten establecer el plazo efectivo para la ejecución EDF en el nivel de baja prioridad para las tareas aperiódicas con plazo. Ahora bien, ¿que plazo efectivo se debe asignar?. La respuesta va a depender de los objetivos de la aplicación, en particular de si las tareas aperiódicas con plazo van a coexistir con tareas aperiódicas sin plazo. Así, asignaremos a una tarea H^k un plazo mayor a su mínimo cuando queramos que ésta pueda ser interrumpida por tareas aperiódicas sin plazo, a las que queremos dar un buen tiempo de respuesta. En este caso, aplicaremos

a H^k el mismo mecanismo de promoción que a las tareas periódicas, donde su promoción (Y_h^k) se calcula de la siguiente forma:

$$Y_h^k = r_h^k + D'^k_h - C_h^k \qquad \text{(ecuación 5.5)}$$

Sin embargo, usar plazos efectivos mayores aumentará la capacidad reservada para las tareas aperiódicas con plazo ya aceptadas (puesto que los $Last\_DL_p$ serán mayores) y por lo tanto aumentará la probabilidad de rechazo de este tipo de tareas en el futuro inmediato.

Teniendo en cuenta todas las condiciones expuestas por las ecuaciones anteriores, cuando en tiempo de ejecución se requiera la ejecución de una nueva tarea aperiódica con plazo, el Algoritmo 5-4 nos dirá si podemos aceptar esa tarea o si debemos rechazarla por no tener garantías para cumplir su plazo. En caso afirmativo el mismo algoritmo calcula su plazo efectivo y su promoción (líneas 4 y 5 respectivamente). Obsérvese que la complejidad algorítmica de este algoritmo es *O(m)*.

Una vez aceptada una tarea aperiódica con plazo, el planificador HDP programará el evento para su promoción, la encolará en la cola GRQ y la planificará como una activación de una tarea periódica cualquiera (siguiendo el esquema de la Figura 5.2). Así, mientras no se promocione, esta tarea podrá ser interrumpida por otras tareas que tengan mayor prioridad en LPL (como por ejemplo tareas aperiódicas sin plazo u otras tareas con plazo anterior al suyo). Una vez promocionada puede coincidir en la cola $HPRQ_p$ con otras aperiódicas con plazo, en cuyo caso el orden dentro de esta cola es FIFO (puesto que en HPL la prioridad de las aperiódicas es r_h^k). Es importante notar que en esta cola no habrá simultáneamente tareas periódicas promocionadas ya que, según la ecuación 5.4, las tareas aperiódicas con plazo han de finalizar antes de la promoción de las tareas periódicas locales al procesador al que han sido asignadas.

```
Algoritmo Test_Aceptacion(H_h^k) retorna booleano
1: Hallar procesadores sin tareas promocionadas que cumplan la
ecuación 5.2 y la ecuación 5.3
2: Si no se halla ninguno entonces retorna falso
3: p= Elegir uno de de estos procesadores          ; ver heurísticas
4: D'_h^k=plazo efectivo que satisfaga la ecuación 5.4 ; ver heurísticas
5: Y_h^k = r_h^k + D'_h^k   - C_h^k                ; ecuación 5.5
6: Last_DL_p = D'_h^k
7: retorna cierto
```

Algoritmo 5-4: Algoritmo de la prueba de aceptación de una tarea aperiódica con plazo (o esporádica). En caso afirmativo, se le asigna un plazo efectivo y un tiempo de promoción para poderla tratar en HDP como las activaciones de las tareas periódicas

En la línea 3 del Algoritmo 5-4 se pueden utilizar diversos criterios para elegir uno de los posibles procesadores que cumplan las condiciones. Se han realizado diversos experimentos para comparar varias heurísticas posibles, como por ejemplo elegir el procesador con la promoción más cercana o elegir el procesador con la promoción más lejana. En la línea 4 también se pueden utilizar diferentes heurísticas. Por ejemplo asignar el plazo efectivo más corto (C_i) o el más largo (pero no mayor al de la propia tarea). En ambos casos elegir una heurística u otra proporcionará unas determinadas prestaciones. En este apartado solo se pretende prestar servicio a tareas aperiódicas con plazo, pero no debemos perder de vista que existen otros tipos de tareas y que las distintas heurísticas pueden permitir su coexistencia.

Los principales algoritmos para escoger un procesador y para asignar el plazo efectivo a la tarea derivados de las diferentes heurísticas comentadas son los siguientes:

HDPmin: consiste en elegir el procesador con la promoción más cercana y asignar el plazo efectivo más corto (parte izquierda de la ecuación 5.4). Esta heurística esta pensada para dar buen servicio a futuras peticiones de tareas aperiódicas con plazo: no es avaricioso al elegir el procesador y se busca un $Last\_DL_p$ menor. Así, si en el futuro inmediato se solicita la activación de una nueva tarea aperiódica con un tiempo de cálculo grande las probabilidades de aceptarla serán mayores (pues serán más fáciles de cumplir la ecuación 5.2 y la ecuación 5.3).

HDPmax: consiste en elegir el procesador con la promoción más lejana y asignar el plazo efectivo más largo (parte derecha de la ecuación 5.4). Esta heurística esta pensada para dar buen servicio a futuras peticiones de tareas aperiódicas sin plazo: asignar plazos efectivos lo más largos posibles, para que la tarea aperiódica con plazo sea interrumpible y aplazable (antes de su promoción) por tareas aperiódicas sin plazo.

Antes de pasar al apartado de los experimentos, es necesario analizar que tipo de distribución previa de las tareas periódicas es más conveniente. Utilizaremos como métrica de rendimiento el porcentaje de tareas aperiódicas con plazo aceptadas (PAA), que se calcula como el número de tareas aperiódicas con plazo aceptadas sobre el número total de peticiones realizadas.

5.5.1 Distribución de las tareas periódicas en tiempo de diseño

En los apartados 5.3.2 y 5.4.1 los experimentos realizados mostraron que la distribución en tiempo de diseño de las tareas periódicas tiene una importancia relativa. Sin embargo, para aceptar las tareas aperiódicas con plazo esta distribución es de vital importancia. Dada una distribución de tareas determinada, es posible que la línea 1 de Algoritmo 5-4 siempre falle, siendo imposible hallar un procesador con promociones suficientemente lejanas para cumplir con la ecuación 5.3. Cuando esto ocurre, el algoritmo no acepta ninguna tarea aperiódica con plazo y por lo tanto esta situación debe ser evitada a toda costa. Esta situación no deseable ocurre particularmente cuando no se puede cumplir con la ecuación 5.1, es decir, cuando el tiempo de proceso requerido por las tareas aperiódicas es mayor que las promociones de las tareas periódicas asignadas a los procesadores. Por ejemplo, supongamos que cada procesador tenga asignada una tarea periódica de máxima prioridad τ_1= $\{C_1=10,T_i=D_i=100\}$. La laxitud de esta tarea es 90 y todas sus promociones se producirán 90 unidades de tiempo después de su activación. En este ejemplo, para cada procesador $\text{Next\_Promotion}_p \leq 100$ y así el Algoritmo 5-4 no puede aceptar ninguna tarea aperiódica con $C_h^k > 100$. En estos casos hay que intentar redistribuir las tareas periódicas de forma que haya procesadores con Next_Promotion_p suficientemente grandes. Ahora bien, esto no es sencillo debido a que

$Next\_Promotion_p$ es un parámetro dinámico. Es preferible realizar una distribución que tenga en cuenta los parámetros estáticos.

Según la ecuación 2.3 y la ecuación 4.1 la promoción de una tarea es menor a su laxitud, entonces las promociones de un procesador tienen como límite superior la laxitud mínima, es decir:

$$\text{Next\_Promotion}_p \leq time\_now + \min_{\tau_i \in p}(D_i - C_i) \qquad \text{(ecuación 5.6)}$$

Entonces, teniendo en cuenta la ecuación 5.1 y la ecuación 5.6, para que una tarea aperiódica con plazo $H^k = \{r_h^k, C_h^k, D_h^k\}$ se pueda aceptar como mínimo debe cumplirse la siguiente condición:

$$C_h^k \leq \max_{p \in PS}\left(\min_{\tau_{ip}}(D_i - C_i)\right) \qquad \text{(ecuación 5.7)}$$

Esta ecuación nos indica que el algoritmo de distribución de las tareas periódicas en tiempo de diseño puede basarse en la laxitud. La condición impuesta por esta ecuación es necesaria pero no suficiente. Así, esta condición tan solo refleja que no se puede servir tareas aperiódicas con plazo si el procesador que tiene el máximo tiempo sobrante entre activaciones de las tareas periódicas no puede servir el tiempo de cálculo requerido por las tareas aperiódicas con plazo. En realidad, para tener más garantías, se deberían utilizar las promociones en lugar de las laxitudes, pero esto añadiría un mayor coste al algoritmo de distribución de tareas.

Cuando la ecuación 5.7 no se cumpla proponemos utilizar el algoritmo *Least-Laxity First-Fit* (LLFF) para la distribución de tareas periódicas en tiempo de diseño que aumente las posibilidades que HDP acepte tareas aperiódicas con plazo en tiempo de ejecución (véase el Algoritmo 5-5). Aún en el caso que la ecuación 5.7 se cumpla, esta distribución puede ser beneficiosa para una mayor aceptación de tareas aperiódicas con plazo, puesto que concentra la carga en unos procesadores y el resto o tienen promociones muy distantes o incluso pueden llegar a no tener tareas periódicas asignadas.

```
1: Algoritmo LLFF (τ, PS)            { Task Set, Processor Set }
2:    ordenar (τ, orden decreciente de laxitud)
3:    para cada tarea τi ∈ τ hacer
4:          p= primer procesador de PS tal que passe Test_J&P(p, τi)
5:          quitar τi de τ
6:          asignar τi a p
7:    fin_para
```

Algoritmo 5-5: Algoritmo *Least-Laxity First-Fit* para distribuir las tareas periódicas en tiempo de diseño y aumentar así las posibilidades de aceptar tareas aperiódicas con plazo

En los siguientes experimentos compararemos el hecho de utilizar dos distribuciones balanceada (Balanced y WFDU+test, usadas en el apartado 5.4.1) versus las distribuciones no balanceadas LLFF (Algoritmo 5-5) y la distribución FFDU+test (utilizada anteriormente en los apartados 5.3.1 , 5.3.4 y 5.4.1).

Para los experimentos de este apartado se ha utilizado la parametrizaciones PCT # 1 y PCT # 4 en la Figura 5.13 y la parametrización PCT#0 en la Figura 5.14 (véase el Anexo B). De esta forma podemos comprobar que los resultados son equiparables en distintos escenarios de los periodos de las tareas.

En todas las gráficas la Figura 5.13 y la Figura 5.14 la distribución LLFF es la que proporciona unos índices más elevados de aceptación de tareas aperiódicas con plazos.

En las gráficas de la Figura 5.13 se puede observar que la distribución LLFF es la mejor independientemente de la proporción entre la carga periódica y la carga aperiódica (eje de las abscisas). En las gráficas (a) y (c) podemos observar como la distribución LLFF supera el porcentaje de tareas aceptadas en aproximadamente un 10% al segundo mejor, que es FFDU+test. Así mismo, es remarcable que la distribución balanceada es la menos adecuada. Mientras LLFF se mantiene por encima del 70% en todos los casos, la distribución balanceada baja hasta el 20%, puesto que la mayor parte de tareas aperiódicas tienen unas necesidades de cálculo mayores que los Next_Promotion de cada procesador. En las gráficas (b) y (d), donde las tareas periódicas tienen unos periodos más armónicos, las tendencias se

mantienen, pero LLFF reduce su ventaja en unos dos puntos y la distribución balanceada y WFDU+test incrementan aproximadamente un 9% su rendimiento pero todavía obtienen unos resultados inaceptables cuando la carga es más aperiódica. Las diferencias entre las distintas distribuciones son las parecidas tanto para HDPmin (gráfica (a) versus (b)) como para HDPmax (gráfica (a) versus (b)), lo que muestra una independencia de las heurísticas utilizadas.

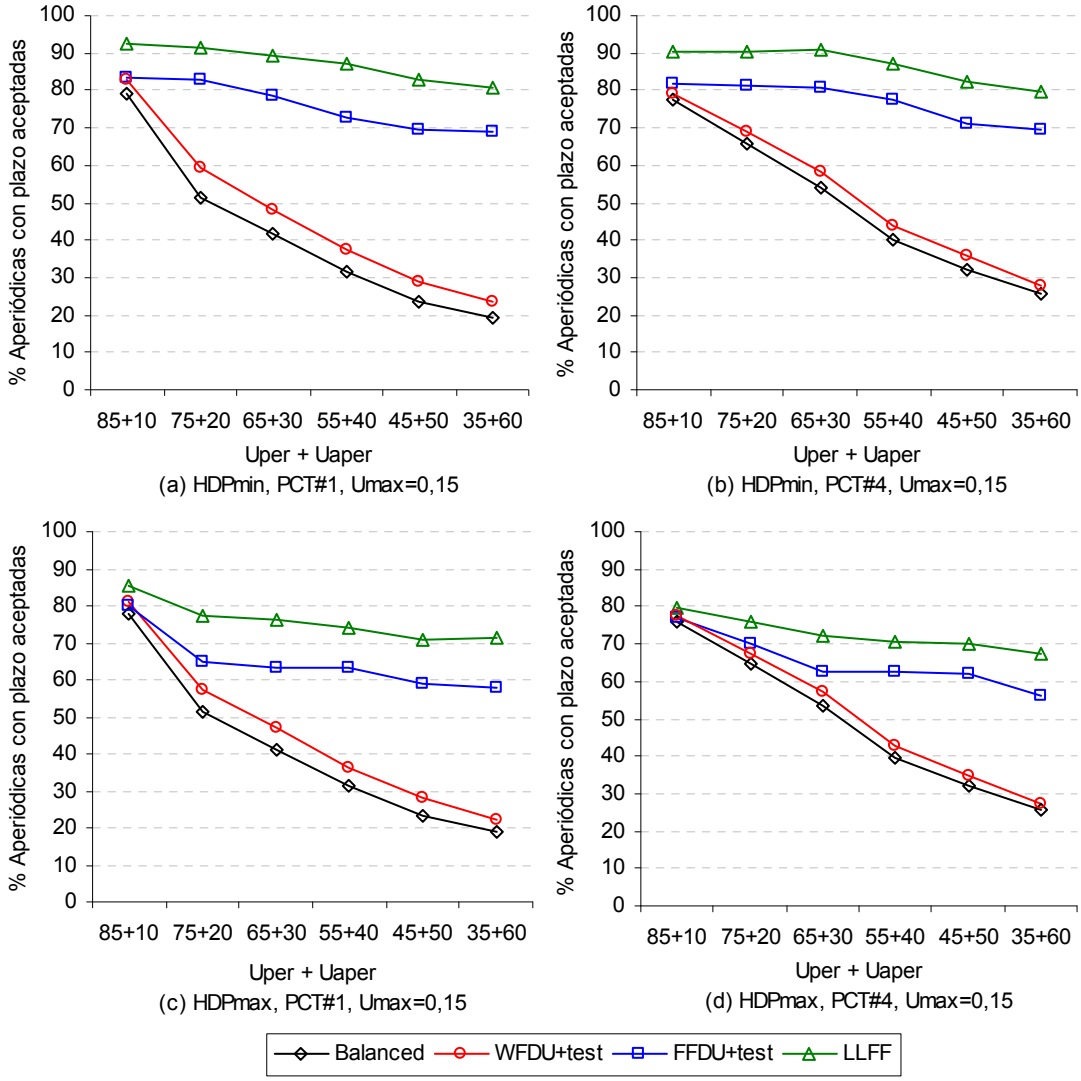


Figura 5.13: Porcentaje de tareas aperiódicas con plazo garantizadas con HDPmin o HDPmax y varios métodos de distribución de las tareas periódicas en tiempo de diseño con m=4 procesadores y cuando varía la distribución de la carga, que es del 95%

Para los experimentos de la Figura 5.14 la distribución de la carga es U_{per}+U_{aper}=70%+25% y se han variado otros parámetros. En las gráficas (a) y (b) se ha variado el número de procesadores. Como es lógico, el porcentaje de éxito es

mayor cuanto mayor es el número de procesadores. Las distribuciones no balanceadas convergen, mientras que las balanceadas crecen a un ritmo más bajo. Es importante remarcar que en la gráfica (a) HDPmin alcanza un éxito del 100% a partir de 16 procesadores con la distribución LLFF.

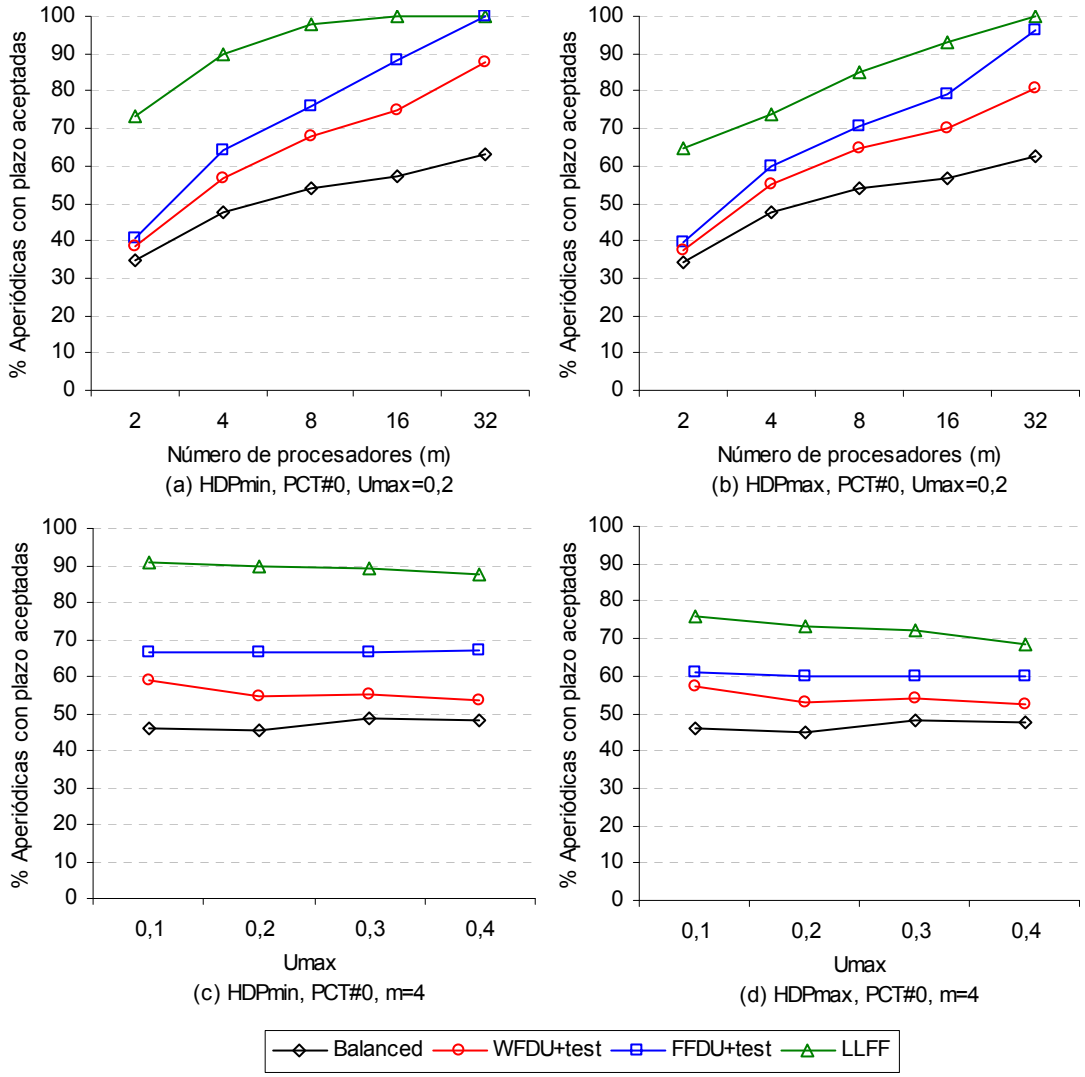


Figura 5.14: PAA (% de tareas aperiódicas con plazo garantizadas) con HDPmin o HDPmax y varios métodos de distribución de las tareas periódicas en tiempo de diseño cuando varía el número de procesadores o cuando varía U_{max} , siendo la carga $U_{per}+U_{aper}$= 70%+25%

En las gráficas (c) y (d) se ha variado el factor de utilización máximo (U_{max}). En ellas se observa que el tener tareas mayores o menores apenas afecta. Para HDPmin y las cargas analizadas, la distribución LLFF supera el porcentaje de tareas aceptadas en aproximadamente un 22% a FFDU+test, que es el segundo mejor.

En conclusión, cuando el objetivo es aumentar el número de tareas aperiódicas con plazo aceptadas con HDP las distribuciones no balanceadas son mejores que las balanceadas. Concretamente, la distribución LLFF es la mejor, entre un 10% y un 20% mejor que FFDU+test. Este resultado contrasta fuertemente con el obtenido en el apartado 5.4.1, donde, con el objetivo de reducir el tiempo medio de respuesta a las tareas aperiódicas sin plazo, las mejores distribuciones eran las balanceadas.

Comparación con otros algoritmos

Para evaluar el funcionamiento del algoritmo propuesto también se han utilizado otros dos algoritmos de planificación: un método de particionado con planificador local y un planificador global.

El método de particionado utiliza como planificador local el servidor de ancho de banda total (TBS, *Total Bandwidth Server*) [SpB94, SpB96] adaptado a los multiprocesadores (véase la sección 3.2.2.2 en la página 95). Las tareas periódicas se distribuyen estáticamente de forma uniforme entre los procesadores en tiempo de diseño. Por otro lado las tareas aperiódicas serán asignadas a un procesador en tiempo de ejecución por un distribuidor global. Este distribuidor también les asignará un plazo efectivo según la ecuación 2.7. Una vez asignadas a un procesador el planificador local las mandará ejecutar siguiendo una política EDF local. Hemos elegido esta opción porque este algoritmo tiene características similares al DP: es simple y robusto, tiene requisitos de cómputo y de memoria bajos. Se basa en EDF y el mecanismo de asignación de plazos sirve directamente como prueba de aceptación de nuevas tareas aperiódicas con plazo. Además, sus autores concluyeron que los tiempos de respuesta aperiódicos medios o los porcentajes de aceptación de tareas aperiódicos con plazo son comparables a los obtenidos con otros algoritmos casi óptimos (véase IPE y EDL en [SpB96] y TB* en [BuS99]).

De la misma forma que con HDP, a la hora de distribuir las tareas aperiódicas con plazo ahora también surgen distintas heurísticas. Hemos probado dos métodos de asignación de las tareas aperiódicas con plazo: TBmin y TBmax. **TBmin**[38] asigna

[38] Equivale a TB-SD del apartado 3.2.2.2 pero se ha adecuado la nomenclatura al apartado actual.

tareas aperiódicas con plazo al procesador que reduce al mínimo la asignación del plazo efectivo, según la ecuación 2.7. **TBmax** asigna tareas aperiódicas con plazo al procesador que maximiza la asignación del plazo efectivo (siempre que este sea menor o igual al plazo propio de la tarea). La asignación del plazo para las tareas aperiódicas sin plazo se hace igual, pero el procesador elegido es el que reduce al mínimo la asignación del plazo, de forma que su tiempo de respuesta medio se reducirá. A diferencia de HDP, con TBS no tiene sentido alargar el plazo efectivo puesto que esto no beneficiaría ni a las futuras tareas aperiódicas con plazo o sin plazo.

En [SVC03] se utiliza un método de particionado de las tareas periódicas, con planificadores locales EDF y la asignación de tareas aperiódicas con plazo utilizando una variante del método propuesto en [SVC99]. En este caso, el planificador global de tareas aperiódicas tiene que hallar y reservar una secuencia de huecos en diversos procesadores de forma que la finalización de la tarea se realice antes de su plazo. Esto implica un alto coste en tiempo de ejecución que no es compatible con el objetivo de esta tesis de mantener los costes del planificador bajos. Si bien los mismos autores diseñaron en [SVC99b] un coprocesador para ayudar al planificador global, éste no sería aplicable al uso de multiprocesadores de propósito general.

También se utilizó el planificador global del servidor de ancho de banda total para multiprocesadores (M-TBS, *Multiprocessor Total Bandwidth Server,* véase el apartado 2.4.2.7 y [BaL04a]) ya que es el único que conocíamos que puede ocuparse de conjuntos de tareas heterogéneos. Es un EDF global tanto para las tareas periódicas como para las aperiódicas. Sus autores indican que las tareas aperiódicas tienen asignado los plazos iguales a su tiempo de finalización si se ejecutasen en segundo plano, pero no realizan ningún experimento para comprobar su rendimiento. Para esta tesis hemos realizado simulaciones con este algoritmo para tener una referencia pero el rendimiento ha sido pésimo, puesto que asigna plazos muy largos y no puede aceptar ninguna tarea, al menos con las parametrizaciones probadas, a pesar de que los conjuntos de tareas satisfacen sus condiciones (véase [BaL04a]). Ya

en el capítulo anterior comprobamos que el tiempo medio de respuesta a las tareas aperiódicas sin plazo eran muy malos (véanse las figuras del apartado 0).

En el apartado anterior (5.5.1) experimentamos con diversas distribuciones para HDP y se concluyo que, de entre las opciones probadas, la distribución no balanceada LLFF era la mejor pero esto no tiene porque ser así para TB, así que repetimos los mismos experimentos para ver que distribución es la más favorable para TB.

En las gráficas de la Figura 5.15 podemos observar como las mejores distribuciones para TB son las balanceadas (Balanced y WFDU+test), que obtienen exactamente los mismos resultados. En general, LLFF es la distribución que peor resultados proporciona para TB, justo todo lo contrario que para HDP. Esto es debido a que ambos TB distribuyen las tareas aperiódicas de forma más o menos equitativa y a que no obtienen ningún partido de concentrar la carga en algunos procesadores (cosa que si hace HDP puesto que obtiene Next_Promotions más lejanos).

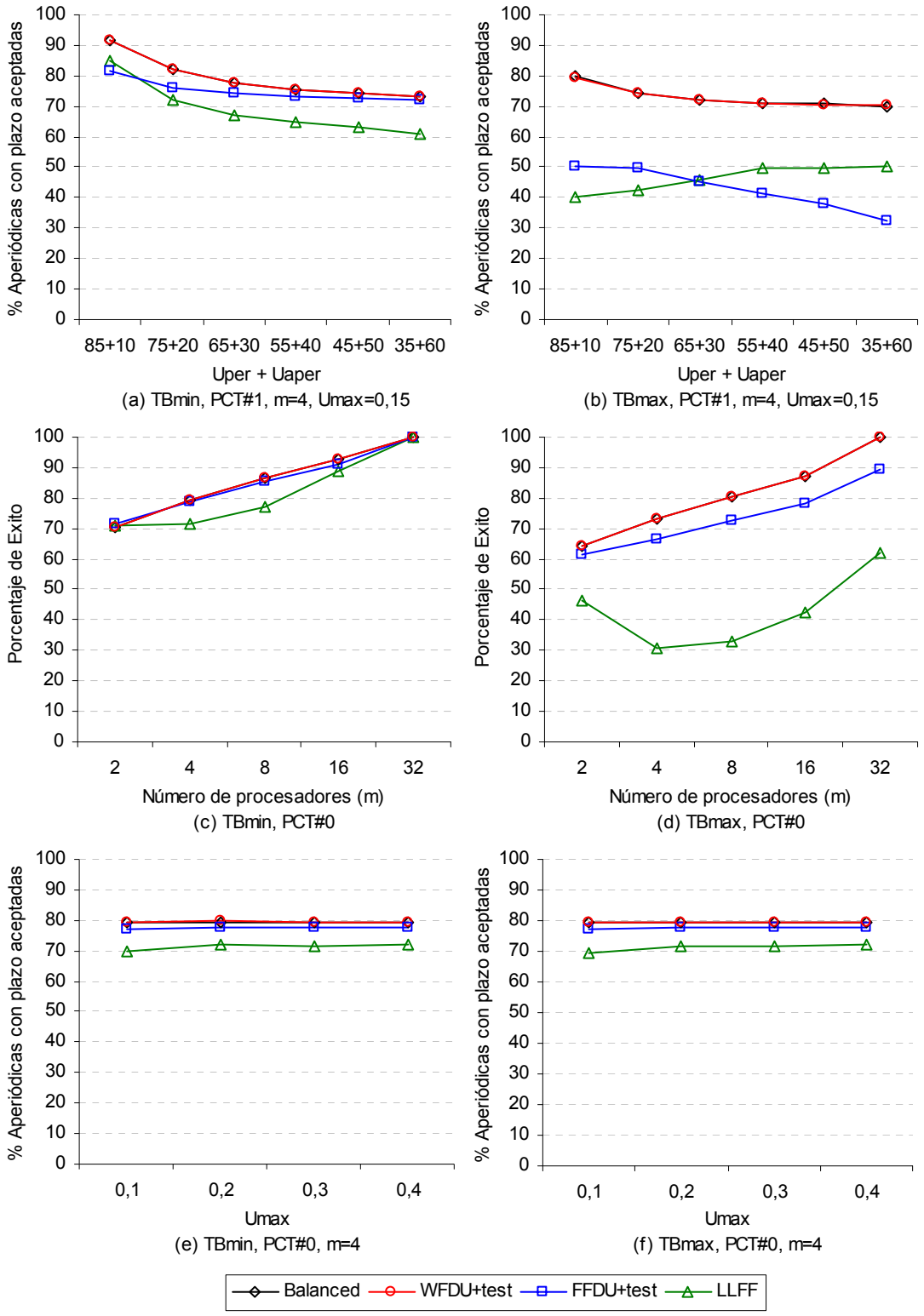


Figura 5.15: Porcentaje de aceptación obtenido con TBmin o TBmax y varios métodos de distribución de las tareas periódicas en tiempo de diseño (de (c) a (f) U_{per}+U_{aper}= 70%+25%)

Como era de esperar, TBmin obtiene mejores resultados que TBmax (gráficas de la izquierda versus gráficas de la derecha). Con TBmin las diferencias entre las distribuciones oscilan alrededor del 10% pero con TBmax estas son bastante mayores. Por ejemplo, en la Figura 5.15-(c) cuando el número de procesadores crece las diferencias con TBmin se minimizan y en cambio en la Figura 5.15-(d) el rendimiento con TBmax cae entre 40 y 50 puntos. Esto es debido a que TBmin utiliza más los procesadores cargados y TBmax los descargados, es decir, TBmin reserva los procesadores menos cargados para peticiones futuras. Cuando el número de procesadores crece, la probabilidad con LLFF de tener procesadores descargados crece. TBmax utiliza estos procesadores y las peticiones posteriores recibirán plazos efectivos más tardíos.

En las gráficas (e) y (f) se varía Umax. Como se puede observar los resultados son constantes. Así, el hecho de tener tareas periódicas pequeñas o grandes no afecta a las diferencias entre las distribuciones.

En conclusión, cuando se utilizan los planificadores propuestos derivados de TB la distribución balanceada en tiempo de diseño de las tareas periódicas es la distribución más adecuada.

5.5.2 Evaluación experimental del rendimiento

En esta sección evaluaremos el funcionamiento promedio de HDP bajo diversas cargas de trabajo, tanto periódicas como aperiódicas, comparándolo con el funcionamiento de TB. Recordemos que las pruebas del planificabilidad de las tareas periódicas se realizan fuera de línea y, por consiguiente, HDP garantiza todos sus plazos. Cada tarea aperiódica con plazo pasará una prueba de aceptación y, si es aceptada entonces esa tarea concreta tendrá el plazo garantizado. El objetivo es comprobar si HDP y TB pueden aceptar un elevado porcentaje de peticiones aperiódicas con plazo y bajo que circunstancias esto ocurre.

La metodología de la evaluación que hemos utilizado se basa en la simulación de extensos conjuntos de tareas sintéticos generados aleatoriamente. Esto permite que probemos el funcionamiento del caso medio y la robustez de los algoritmos de

planificación bajo una amplia gama de situaciones. Utilizaremos como métrica de rendimiento el porcentaje de tareas aperiódicas con plazo aceptadas (PAA), que se calcula como el número de tareas aperiódicas con plazo aceptadas sobre el número total de peticiones realizadas.

Para los experimentos de este apartado se ha utilizado la parametrizaciones PCT#0, PCT # 1 y PCT # 4 (véase el Anexo B) pero generalmente solo se documentan los resultados obtenidos con PCT # 1 puesto que las diferencias obtenidas entre parametrizaciones no son significativas y las diferencias entre los algoritmos son las mismas. El número de procesadores *m* generalmente es 4. El número de tareas periódicas por procesador, *n*, sigue una distribución uniforme en el rango [4..8]. El factor de utilización máximo de una tarea periódica, si no se dice lo contrario, está fijado a 0,2. La carga total del sistema ha sido fijada en el 95%. Como siempre, para probar el máximo estrés del sistema, cada tarea ejecuta su peor tiempo de cómputo entero.

La distribución de las tareas periódicas en tiempo de diseño utilizada para los algoritmos HDP es la LLFF mientras que para los algoritmos TB se utiliza una distribución balanceada, puesto que en apartados anteriores comprobamos que estas eran las distribuciones más efectivas para cada algoritmo de planificación.

Las llegadas de las tareas aperiódicas se generan siguiendo una distribución de Poisson. El tiempo entre llegadas medio (I_h) se elige aleatoriamente en la misma gama de los periodos de las tareas periódicas. Los requisitos del cómputo también se eligen aleatoriamente, según la carga aperiódica prevista total ($U_{aper}=C_{aper}/ I_h$)[39]. Por lo tanto, todas las tareas generadas tienen unas características equiparables. El plazo para las tareas aperiódicas con plazo se ha establecido igual al tiempo entre llegadas medio ($D_h=I_h$), a excepción del experimento 2, donde se ensayan plazos más estrictos.

[39] Al no ser tareas periódicas la carga instantánea puede ser superior a Uaper pero la carga promedio generada a lo largo de una simulación será ésta

La duración de las simulaciones se ha fijado en 20 hiperperiodos, lo que da lugar a obtener muestras para entre 3300 y 50400 tareas aperiódicas. Para cada conjunto de tareas se han ido lanzando simulaciones hasta que el valor obtenido de la métrica estaba dentro de un intervalo de confianza del 95% con una probabilidad del 5%. En todas las gráficas cada punto representa el promedio de la métrica medida en la simulación de 5.000 conjuntos de tareas.

Experimento

El objetivo de este experimento es probar que porcentaje de aceptación de tareas aperiódicas con plazo se puede alcanzar con los algoritmos de planificación propuestos. Los resultados se muestran en la Figura 5.16. En todas las gráficas de esta figura HDPmin es el planificador que obtiene mejores resultados.

En la Figura 5.16-(a) se puede observar como el rendimiento aumenta para todos los planificadores cuando el número de procesadores aumenta, mostrando que el aumento de recursos facilita la ubicación de las peticiones aperiódicas. En la gráfica (b) el porcentaje de garantía alcanzado por HDPmin es en promedio un 14,6% más alto que el alcanzado por TBmax, un 11% más alto que HDPmax y un 8,7% mayor que TBmin. En general, cuando U_{per} decrece y U_{aper} crece el rendimiento disminuye debido al incremento de la competencia entre las tareas aperiódicas con plazo, pero HDPmin se mantiene siempre por encima del 80% porque es la política menos avariciosa. En las gráficas (c) y (e) se aprecia que TBmin es mejor que HDPmax cuando las características de la carga total es menor. En la gráfica (d) se observa que el PAA de los planificadores no depende del tamaño de las tareas periódicas. Tan solo HDPmax experimenta una reducción significativa, que es del 10%. En la gráfica (f) cuando Caper es mayor el rendimiento de HDPmax se reduce un 16,5% pero HDPmin se mantiene estable cercano al 100%.

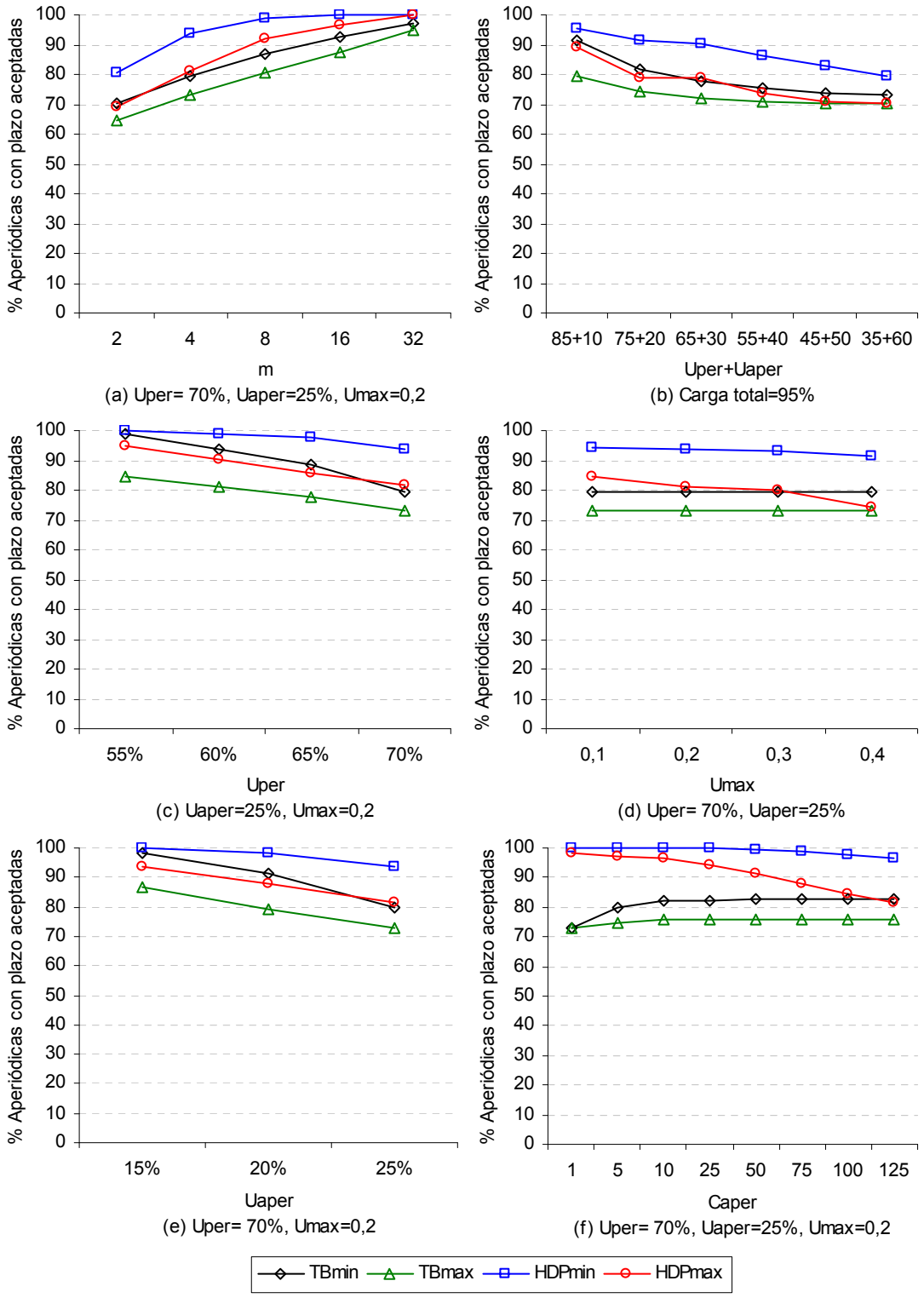


Figura 5.16: Porcentaje de aceptación de tareas aperiódicas con plazo cuando varía (a) el número de procesadores, (b) la distribución de la carga, (c) la carga periódica, (d) la máxima utilización periódica (e) la carga aperiódica y (f) el cómputo aperiódico (m=4 excepto en (a))

5.5.3 Conclusiones

Con la caracterización de las tareas experimentada y en todas las distintas variaciones de los parámetros individuales, HDPmin es el planificador que mejores porcentajes de aceptación obtiene, manteniéndose siempre por encima del 80%, porcentaje que podemos considerar alto, siendo a su vez el que más estable.

El funcionamiento de HDPmax es peor que el de HDPmin porque asigna plazos más largos a las tareas aperiódicas con plazo, es decir reserva más capacidad para cada tarea aperiódica con plazo ($Last\_DL_p$ mayores en las ecuaciones) y esto dificulta la aceptación de futuras peticiones aperiódicas. Por otra parte, el funcionamiento del peor HDP (HDPmax) es similar al funcionamiento del mejor TB (TBmin).

El mayor rendimiento de los algoritmos HDP frente a los TB es atribuible a la mayor información que estos poseen (las promociones) y a la capacidad de HDP de migrar tareas periódicas dando preferencia a las tareas aperiódicas. Conviene recordar que los planificadores TB son planificadores locales (PF/AD según la nomenclatura del Capítulo 3) en cambio los HDP son planificadores híbridos. En este punto también es importante remarcar que los autores de TB desarrollaron versiones optimizadas como TB* [BuS99] pero en este caso el coste de la prueba de aceptación es bastante más alto (*O(mNn)* siendo *m* el número de procesadores, *n* el número de tareas del procesador y *N* el número de iteraciones (variable) que requiere el test de aceptación por procesador).

Cuando el objetivo es aumentar el número de tareas aperiódicas con plazo aceptadas con HDP las distribuciones de las tareas periódicas en tiempo de diseño no balanceadas son mejores que las balanceadas. Concretamente, la distribución basada en la laxitud de las tareas periódicas (LLFF) es la mejor. Este resultado contrasta fuertemente con el obtenido en el apartado 5.4.1, donde, como el objetivo era reducir el tiempo medio de respuesta de las tareas aperiódicas sin plazo, las mejores distribuciones para HDP eran las balanceadas. Para los planificadores propuestos derivados de TB la distribución balanceada también es la distribución más adecuada.

5.6 Servicio conjunto a tareas aperiódicas con y sin plazo

Al principio del apartado 5.5 se han propuesto dos métodos de planificación basados en HDP fruto de distintas heurísticas aplicables al Algoritmo 5-4 (p. 195), pero en el apartado anterior solo se evaluó la eficiencia con únicamente tareas aperiódicas con plazo. Así, en el presente apartado se pretende evaluar el tiempo medio de respuesta que obtendrían las tareas aperiódicas sin plazo en convivencia con tareas aperiódicas con plazo y como esto afecta al porcentaje de tareas aperiódicas con plazo aceptadas. Así pues, utilizaremos dos métricas de rendimiento distintas. Para las tareas aperiódicas con plazo utilizaremos el porcentaje de aceptación (PAA), al igual que en el apartado anterior, mientras que para las tareas aperiódicas sin plazo utilizaremos la media aritmética del tiempo de respuesta medio normalizado (TRA).

Recordemos que HDP ya está preparado para trabajar con ambos tipos de tareas. Así, primero pasamos a realizar una primera evaluación del rendimiento y a posteriori se propondrán algunas modificaciones para mejorar el rendimiento obtenido.

5.6.1 Evaluación experimental del rendimiento

En este apartado, primero realizaremos un experimento equivalente al de las figuras Figura 5.10-(a) y Figura 5.16-(b) (apartados 5.4.2 y 5.5.2 respectivamente). En un segundo experimento utilizaremos plazos más estrictos para las tareas aperiódicas con plazo.

Experimento 1

Para este experimento se ha utilizado la misma parametrización (PCT#0, m=4, Umax=0,15, Carga total= 95%) y la misma metodología que en el apartado 5.5.2. Los resultados de las simulaciones se representan en el Figura 5.17. La diferencia principal con el experimento del apartado 5.5.2 estriba en que la carga aperiódica se ha repartido a partes iguales entre las tareas con plazo y las tareas sin plazo. Así, en el eje de las abscisas de las gráficas una carga de $U_{per}+2*U_{aper}=65+30$ indica un 65%

de carga periódica, un 15% para las tareas aperiódicas con plazo más un 15% para las tareas aperiódicas sin plazo.

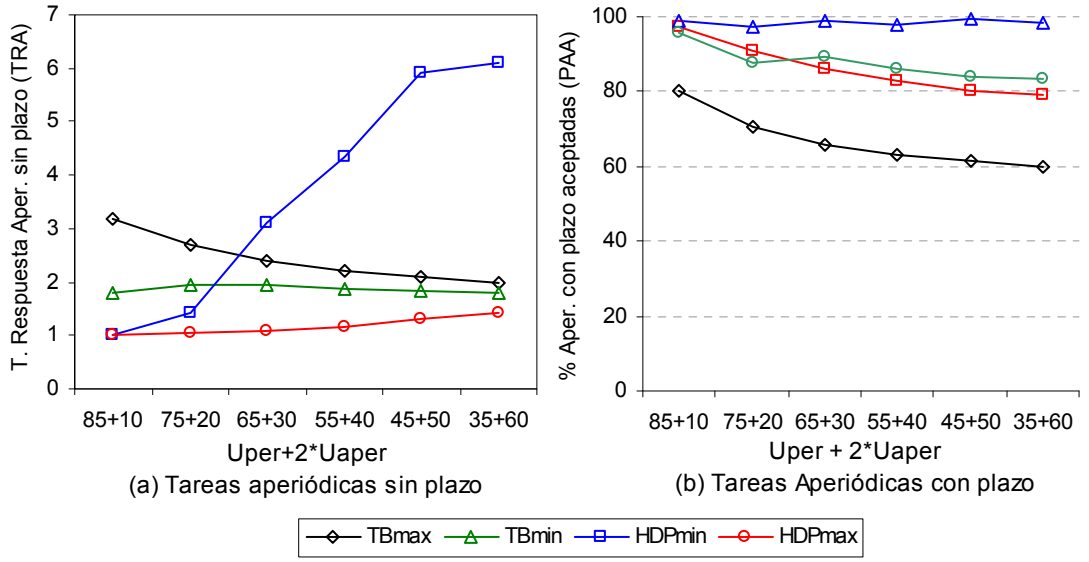


Figura 5.17: Conjuntos de tareas con ambos tipos de tareas aperiódicas con la carga aperiódica repartida al 50%

En primer lugar, observemos la Figura 5.17-(a) para contrastar con el apartado 5.4.2 como ha cambiado el tiempo de respuesta medio de las tareas aperiódicas sin plazo (comparándola con la Figura 5.10-(a)). Como en el experimento actual la carga aperiódica sin plazo es la mitad, en la Figura 5.17-(a) se muestran TRA mejores que los de la Figura 5.10-(a), a pesar de tener la interferencia de las tareas aperiódicas con plazo. Sin embargo, a diferencia de la Figura 5.10-(a), cuando la carga es menos periódica y más aperiódica ahora los TRA obtenidos con HDP tienden a aumentar.

En Segundo lugar, observemos la Figura 5.17-(b) para contrastar con el apartado anterior como ha cambiado el porcentaje de aceptación (comparándola con la Figura 5.16-(b)). En este experimento los porcentajes de aceptación son más altos que en el experimento anterior. Esto es normal y sucede porqué la carga aperiódica con plazo se ha reducido a la mitad de la carga (aunque la suma de ambas cargas aperiódicas sigue siendo la misma) y al ser los WCET de estas tareas menores ahora son más fáciles de servir: TB asigna plazos efectivos más cortos y con HDP es más fácil satisfacer la ecuación 5.3 (página 192).

Cuando la carga es menos periódica y más aperiódica en el experimento actual los TRA obtenidos con HDP tienden a aumentar. En estas situaciones, la carga aperiódica con plazo aumenta y las tareas aperiódicas con plazo son más grandes. Así, con HDPmax el PAA se reduce y el amento del TRA no es muy grande porque, el mayor rechazo de tareas aperiódicas con plazo en la práctica provoca una disminución de la carga aperiódica. En cambio, HDPmin mantiene un PAA cercano al 100% y esto provoca un incremento del TRA muy importante, convirtiéndose HDPmin en la peor opción, con un servicio a las tareas aperiódicas sin plazo casi inaceptable. Esto es debido a que las tareas aperiódicas con plazo se asignan a los procesadores más ocupados y no son expulsables. Como las tareas aperiódicas con plazo tienen una prioridad mayor que las tareas sin plazo y con HDPmin no son expulsables, estas tienden a vetar una gran parte de los procesadores y, por lo tanto, los tiempos de respuesta de las tareas aperiódicas sin plazo pueden llegar a ser muy altos.

En este experimento el mejor planificador es HDPmax, ya que proporciona los TRA siempre inferiores al 1,4 y mantienen el PAA por encima del 83,5%. Esto es debido a que las tareas aperiódicas con plazo se asignan a los procesadores menos cargados y allí, al tener plazos efectivos mayores, pueden ser expulsadas con una probabilidad alta. Como consecuencia, todos los procesadores están disponibles para las tareas aperiódicas sin plazo (los más cargados con periódicas porque apenas los usan las aperiódicas con plazo y los menos cargados porque estas tareas son expulsables). El PAA proporcionado por TBmin es casi el mismo, pero el TRA es en promedio un 63% peor. Por otro lado, TBmax proporciona la peor combinación de resultados: TRA es el doble del WCET de las tareas aperiódicas y PAA baja hasta el 60%.

En conclusión, HDPmin es el mejor planificador para las tareas aperiódicas con plazo y HDPmax el mejor para las tareas sin plazo pero ninguno de los dos métodos tiene una prevalecía total respecto al otro.

Experimento 2

En este experimento se ha usado la parametrización PCT#0 con una carga total del 95% (U_{per}=65% y U_{aper}= 15%+15%). El objetivo es probar plazos más estrictos que los usados en el experimento anterior para ver como afecta a la aceptación de tareas aperiódicas con plazo y al servicio a las tareas sin plazo. Para ello, el plazo de las tareas aperiódicas se disminuirá a partir del tiempo entre llegadas medio (I_h) a un 10% de su laxitud (I_h - C_h), es decir, el plazo se calcula aplicando el factor de escalado *x* a su laxitud, tal como se muestra en la siguiente ecuación:

$$D_h = C_h + x\,(I_h - C_h) \qquad \text{(ecuación 5.8)}$$

En el Figura 5.18, PAA tiende a disminuir cuando los plazos se hacen más estrictos puesto que es más difícil encontrar procesadores que cumplan la ecuación 5.2. Esto implica que hay más capacidad disponible para las peticiones aperiódicas sin plazo. Así el TRA en la gráfica de la izquierda tiende a mejorar cuando los plazos son más estrictos. Aquí, HDPmin y HDPmax tienden a converger, porque D_h tiende a C_h y el plazo efectivo asignable por ambos tiende a ser el mismo (véase la ecuación 5.4). Con HDPmin el PAA sigue siendo el mejor porqué es menos codicioso y deja procesadores con mayor capacidad (es decir, las promociones son lo más tardías posible) libres para servir las futuras peticiones aperiódicas con plazos más exigentes. Por otra parte, los algoritmos de TB no pueden servir ninguna tarea aperiódica con plazo cuando el factor de escalado es 30% o inferior. Esto sucede porque la capacidad del servidor es siempre inferior a los servicios solicitados. En este caso, U_{per}= 0,65, U_{server}= 0,35, U_{aper} = C_h/I_h = 0,15, pero la laxitud se escala con D_h = 0,405I_h y consecuentemente U_{aper} = C_h/D_h = 0,37 > 0,35. Por lo tanto, las nuevas tareas aperiódicas con plazo no caben en ningún servidor de TB (es decir U_{aper}> U_{server}, para cada procesador p). Este fenómeno no sucede con HDP ya que no se utiliza ningún servidor.

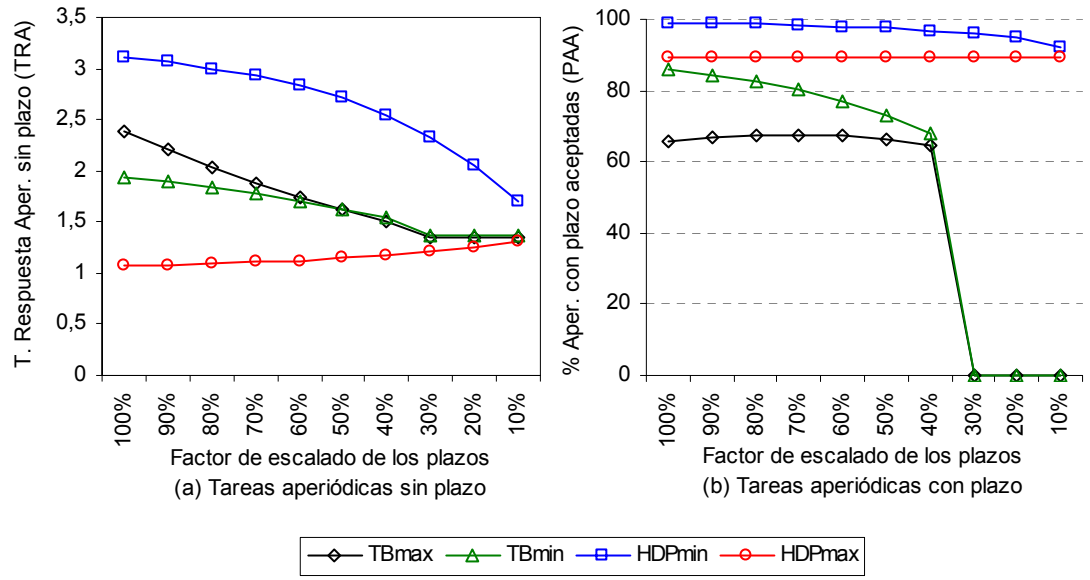


Figura 5.18: Uso de plazos más estrictos para las tareas aperiódicas con plazo. El plazo de las tareas aperiódicas se calcula aplicando el factor de escalado *x* ($D_h = C_h + x\ (I_h - C_h)$). U_{per}=65%, U_{aper}= 15%+15%, U_{max}=0,2 y m=4.

Una posible solución al problema de TB con los plazos más estrictos consiste en utilizar distribuciones no balanceadas, de forma que algunos procesadores dispongan de servidores más grandes. En el apartado 5.5.1 observamos en la Figura 5.15 (p.204), con solo tareas aperiódicas con plazo, que las distribuciones balanceadas eran las que proporcionaban a TB el mejor rendimiento. Con las no balanceadas, LLFF era bastante peor, pero FFDU+test no era muy inferior.

En la Figura 5.19 se ha repetido el experimento de la que en la Figura 5.18 para TBmin con los distintos algoritmos de distribución de las tareas periódicas en tiempo de diseño. Con las distribuciones no balanceadas TBmin obtiene un TRA entre un 20% y un 30% peor que con una distribución equilibrada, mientras que el PAA es aproximadamente 3 puntos inferior pero cuando los plazos son más estrictos PAA se mantiene por encima del 40%. Las distribuciones no balanceadas permiten plazos más exigentes porque U_{server} varía de procesador a procesador y es más fácil acomodar tareas aperiódicas con plazo en algún procesador. De hecho, en este experimento particular, con el 65% de la carga periódica redistribuida entre 4 procesadores, un procesador tendría una alta probabilidad de quedar vacío, y entonces este procesador tendría U_{server}= 1.

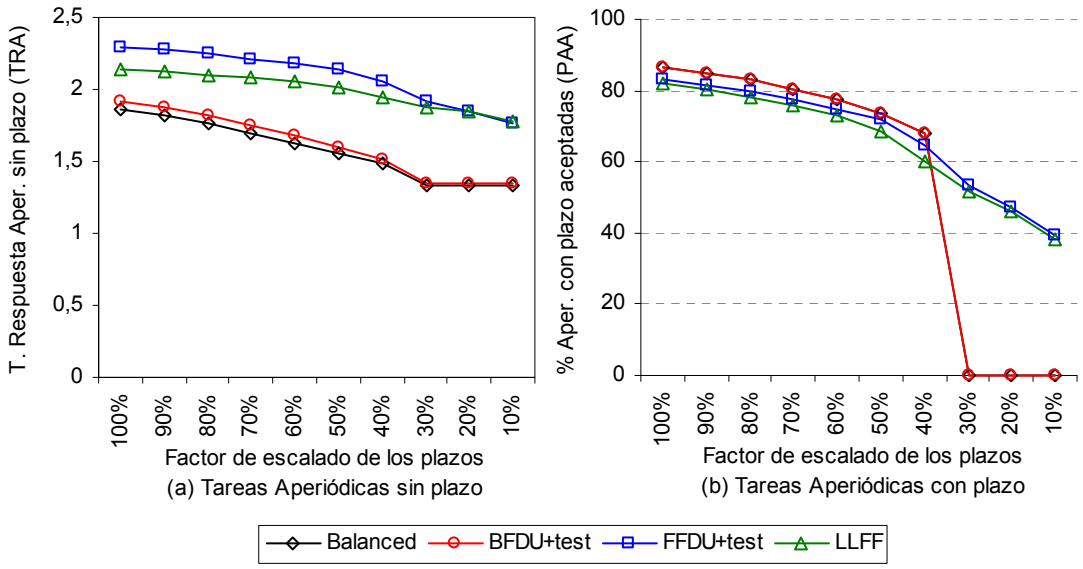


Figura 5.19: Como afectan los plazos más estrictos para las tareas aperiódicas con plazo al rendimiento de TBmin con distintas distribuciones. U_{per}=65%, U_{aper}= 15%+15%, U_{max}=0,2 y m=4

5.6.1.1 Balanceo dinámico del servicio entre aperiódicas con y sin plazo

En los experimentos anteriores hemos comprobado como HDPmin obtiene los mejores resultados para las tareas aperiódicas con plazo y HDPmax para las tareas sin plazo. Los diseñadores de aplicaciones de tiempo real, durante la fase de diseño, deben elegir uno de los dos métodos. Sin embargo, es posible balancear dinámicamente el servicio entre los dos tipos de tareas aperiódicas con la utilización de un parámetro que permita al diseñador indicar el nivel de calidad de servicio requerido para las tareas aperiódicas sin plazo, teniendo en cuenta que un tiempo de respuesta menor para estas implica una menor tasa de aceptación de las tareas aperiódicas con plazo. Así, una primera aproximación consiste en conmutar entre la utilización de HDPmin y HDPmax. Con el algoritmo **HDPswitch** si las tareas sin plazo están obteniendo un tiempo de respuesta medio inferior a la calidad de servicio requerida entonces se utiliza HDPmin. En otro caso se utilizará HDPmax para favorecer a las tareas aperiódicas sin plazo.

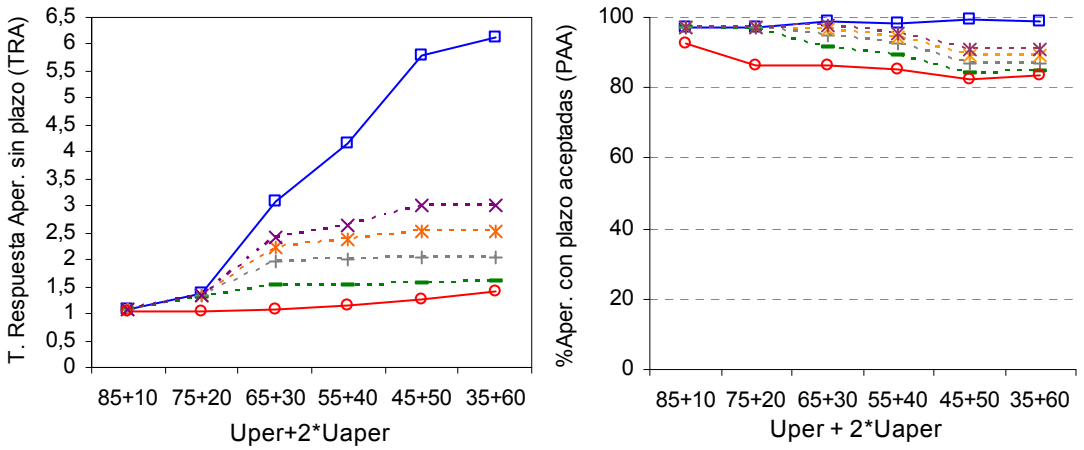
Una variante algo más sofisticada es **HDPqos**, que consiste en elegir el procesador con la promoción más lejana (como haría HDPmax) y asignar el plazo efectivo mayor o menor de forma dinámica. El diseñador de la aplicación fija un objetivo para

el tiempo de repuesta medio de las tareas aperiódicas sin plazo (parámetro *a*). Mientras estas obtengan un tiempo de respuesta inferior a *a*, el plazo efectivo de las nuevas tareas aperiódicas con plazo se puede acortar favoreciendo la aceptación de futuras peticiones (en el límite, el plazo efectivo menor r_h+C_h, como haría HDPmin). En caso contrario habrá que alargar los plazos efectivos asignados para que la tarea aperiódica con plazo sea interrumpible (antes de su promoción) por las tareas aperiódicas sin plazo y así ir mejorando el tiempo de respuesta medio de las tareas aperiódicas. Obsérvese que HDPqos utiliza el parámetro *a* para comprobar el TRA obtenido pero nunca actúa directamente sobre las tareas aperiódicas sin plazo. Es decir, si se obtiene un TRA por debajo de *a* puede asignar a las tareas aperiódicas con plazo los mayores plazos efectivos posibles pero esto no implica necesariamente que TRA ascienda hasta el parámetro *a*.

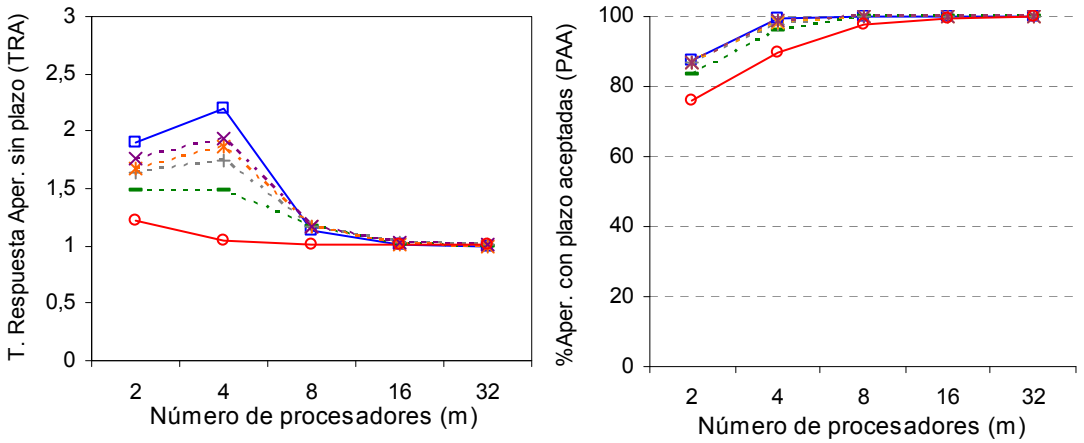
Experimento 3

Para este experimento usaremos HDPqos (Figura 5.20) y HDPswitch (Figura 5.21) con el parámetro *a* de calidad de servicio para las tareas aperiódicas sin plazo. El objetivo consiste en ver hasta que punto el parámetro *a* proporciona a los diseñadores de aplicaciones una capacidad de control sobre el balanceo del servicio entre las peticiones aperiódicas con o sin plazo.

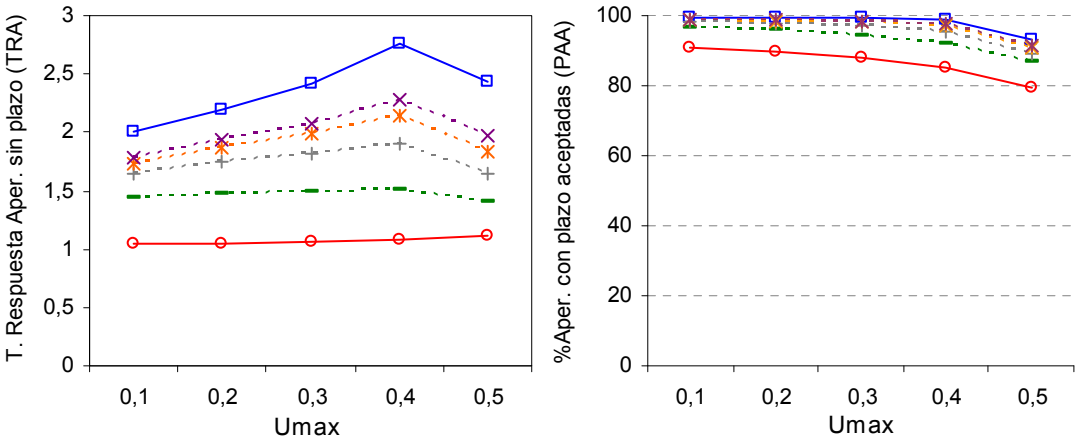
En la Figura 5.20-(a) se puede observar que cuando las cargas periódicas son altas el umbral *a* no afecta a HDPqos (a excepción del valor 1,0, el cual equivale a HDPmax y es demasiado exigente con las tareas aperiódicas sin plazo). En estos casos, las cargas aperiódicas son demasiado bajas para hacerlo sensible. Hasta cargas aperiódicas del 40% el TRA se mantiene por debajo del parámetro *a*. Solo cuando las cargas aperiódicas son más altas podemos ver que TRA se estabiliza alrededor del umbral, tal como es deseable. Obviamente, PAA también se ve afectado: cuanto más bajo es el umbral menor es PAA. Para la parametrización del experimento, el umbral 1,5 podría ser un buen compromiso: el tiempo de espera de las tareas aperiódicas sin plazo es solamente un 50% de su tiempo de cómputo y el PAA de las aperiódicas con plazo está por encima del 86%.



(a) PCT#0, Umax=0,15, m=4



(b) PCT#1, Umax=0,2, Carga=65%+30%



(c) PCT#1, m=4, Carga=65%+30%

Figura 5.20: HDPqos: Uso del parámetro *a* para balancear el servicio entre las tareas aperiódicas con y sin plazo. La carga aperiódica se reparte al 50% entre ambos tipos de tareas.

En la Figura 5.20-(b) se aprecia como aumentando el número de procesadores las diferencias desaparecen y cualquiera que sea el parámetro de calidad de servicio éste es excelente, tanto para las tareas aperiódicas con plazo como para las tareas sin plazo. Cuando el número de procesadores es 2, PAA se reduce ante la dificultad de hallar un procesador donde ubicar las tareas aperiódicas con plazo y, por lo tanto estas se rechazan dejando más margen para la ejecución de tareas sin plazo. Cuando el número de procesadores es 4, el PAA ya es muy alto y el TRA se ve perjudicado, puesto que la capacidad de proceso no ha aumentado suficientemente. Por este motivo en el resto de las gráficas el número de procesadores se ha fijado en 4.

En la Figura 5.20-(c) se ha ensayado con tareas periódicas de distinto tamaño. En general el TRA y el PAA empeoran cuando existen tareas periódicas grandes pero los resultados son en todos los casos aceptables. Por ejemplo, para *a*=1,5 el TRA se mantiene por debajo de 1,5 y el PAA es superior al 90%. TRA solo experimenta una ligera mejoría cuando Umax=50%, puesto que PAA baja y se rechazan más tareas aperiódica con plazo, en beneficio de las tareas sin plazo (que no se rechazan nunca).

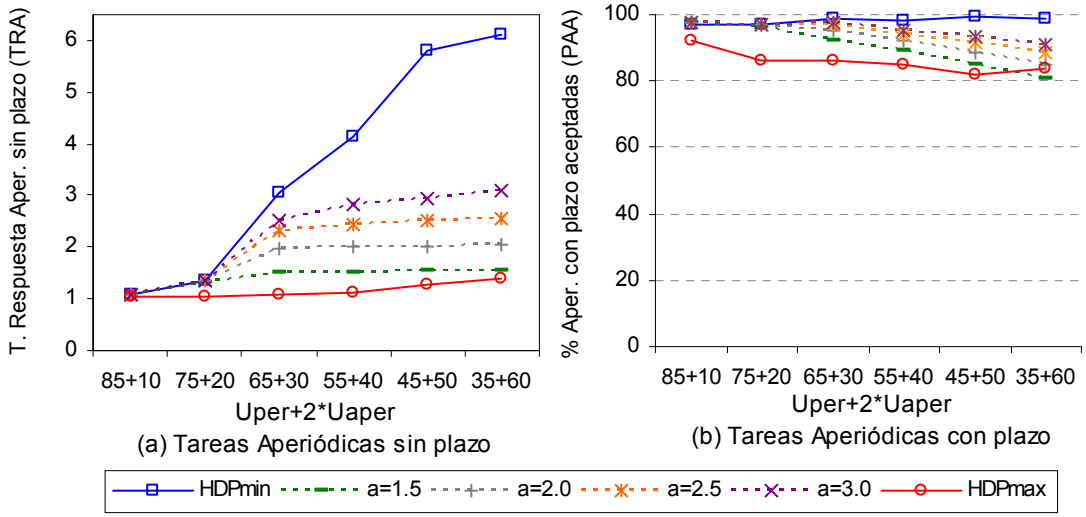


Figura 5.21: HDPswitch: Uso del parámetro *a* para balancear el servicio entre las tareas aperiódicas con y sin plazo. La carga aperiódica se reparte al 50% entre ambos tipos de tareas.

En la Figura 5.21 se muestran los resultados obtenidos con el planificador HDPswitch para el mismo experimento de la Figura 5.20-(a). Los resultados en media son muy similares. Tan solo existe una ligera pérdida en el PAA cuando la carga es 65+30. Sin embargo, la principal diferencia no se muestra en esta gráfica.

Por la naturaleza de HDPswitch las muestras instantáneas de las métricas oscilan más que con HDPqos. Se ha medido el porcentaje de tareas aperiódicas sin plazo que obtienen un tiempo de respuesta inferior a *a* y éste es inferior en un 1% con HDPswitch. Por otro lado las tareas aperiódicas con plazo son rechazadas más a ráfagas con HDPswitch que con HDPqos. Por consiguiente, dado que los resultados con las métricas TRA y PAA no son mejores, podemos descartar a HDPswitch.

En conclusión, HDPqos permite al diseñador automatizar el servicio entre ambos tipos de tareas aperiódicas. Así, el diseño se facilita y solo hace falta establecer el umbral de servicio a las tareas aperiódicas sin plazo a modo de indicador de calidad de servicio deseada para estas, teniendo en cuenta que esto va a repercutir en la cantidad de tareas aperiódicas con plazo que van a ser aceptadas. El parámetro *a* permite reducir el TRA a base de empeorar el PAA pero no permite hacer lo contrario, es decir, incrementar TRA hasta *a* para incrementar el PAA.

5.6.1.2 Distribución de las tareas periódicas en tiempo de diseño

En los apartados anteriores utilizamos el algoritmo LLFF para la distribución de tareas periódicas en tiempo de diseño y se ha mostrado efectivo. Al concentrar la carga en unos procesadores el resto pueden aceptar más tareas aperiódicas con plazo. Por otro lado en el apartado 5.4.1 constatamos como las tareas aperiódicas sin plazo obtienen mejores tiempos de respuesta con una distribución balanceada.

En este apartado evaluamos un método válido para ambos tipos de tareas cuando se conocen en tiempo de diseño el tiempo de cómputo mayor de las tareas aperiódicas con plazo (C_{h_max}). La idea consiste en distribuir las tareas periódicas de forma balanceada (para satisfacer las tareas aperiódicas sin plazo) de forma que las promociones sean al menos mayores que el tiempo de cómputo mayor de las tareas aperiódicas con plazo en la mayoría de procesadores (para que estas puedan ser aceptadas). Esto es lo que precisamente pretende hacer el algoritmo de distribución MaxPromo+LLFF (véase el Algoritmo 5-6). Primero ordena las tareas periódicas por orden de laxitud decreciente (línea 2). El bucle de las líneas 6 a 12 busca el procesador en el que una tarea periódica es planificable (promoción mayor que cero) y que le asigne una promoción máxima y mayor que C_{h_max}. Esto en la práctica balanceará la carga, puesto que los procesadores más descargados tenderán a proporcionar los tiempos de respuesta menores y por lo tanto las promociones mayores. Cuando no es posible hallar un procesador que asigne una promoción mayor a C_{h_max} entonces se aplica LLFF (línea 16). Esto concentrará la carga restante en los primeros procesadores, inhabilitándolos para aceptar tareas aperiódicas con plazo grandes pero dejando el resto de procesadores libres y habilitados.

Para comprobar si este método de asignación de tareas periódicas en tiempo de diseño es efectivo primero se han repetido los experimentos con sólo tareas aperiódicas con plazo (véase la Figura 5.14 del apartado 5.5.1) comparándolo con los métodos de distribución ya ensayados.

```
1: Algoritmo MaxPromo+LLFF (τ, PS)        { Task Set, Processor Set }
2:    ordenar (τ, orden decreciente de laxitud)
3:    para cada tarea τi ∈ τ hacer
4:          pmax= 0
5:          max_promo= Ch_max
6:          para cada procesador P ∈ PS hacer
7:                Yi= promocion(τi,p)
8:                si (Yi >= max_promo) entonces
9:                      pmax= p
10:                     max_promo= Yi
11:               fin_si
12:         fin_para
13:         si pmax>0 entonces
14:               p= pmax
15:         sino                              { LLFF }
16:               p= primer procesador de PS que pase Test_J&P(p, τi)
17:         fin_si
18:         asignar τi a p
19:   fin_para
```

Algoritmo 5-6: Algoritmo MaxPromo+LLFF, alternativa a LLFF para la distribución de tareas periódicas en tiempo de diseño.

En la Figura 5.22 podemos observar en las gráficas de la izquierda que el nuevo método de distribución es un poco peor que LLFF para HDPmin (entre un 3% y un 5% en la mayoría de casos). En cambio, en las gráficas de la derecha se refleja que el porcentaje de aceptación con HDPmax mejora (entre un 3,3% y un 9,3% en la mayoría de casos). Esta diferencia es debida a que HDPmin utiliza primero los procesadores no favorecidos por MaxPromo+LLFF al contrario que HDPmax, que utiliza primero los procesadores menos cargados y con promociones más separadas y, como la carga esta más balanceada que con LLFF, el resto de procesadores todavía tienen suficiente capacidad de servicio.

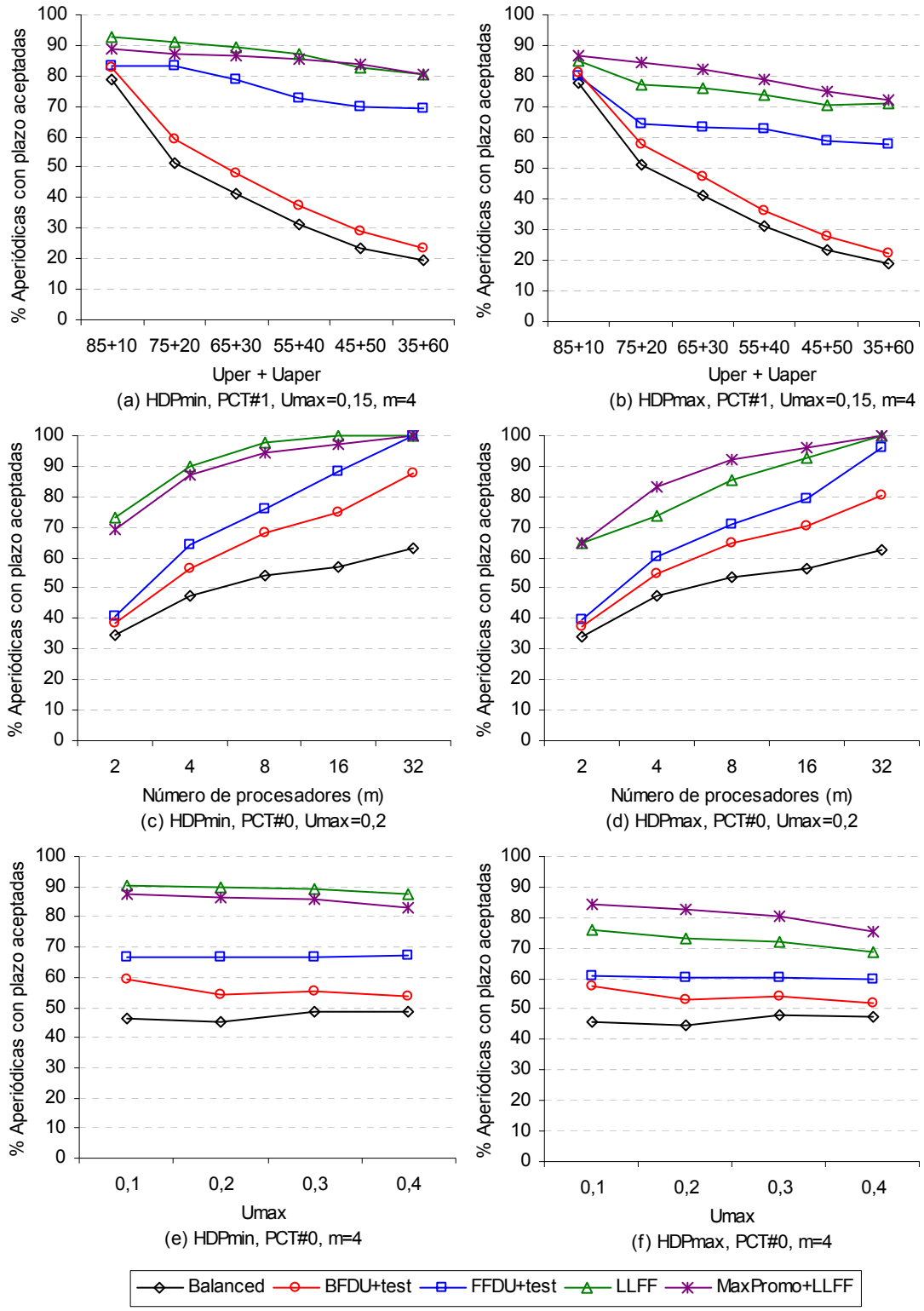


Figura 5.22: Porcentaje de tareas aperiódicas con plazo aceptadas con HDPmin y HDPmax y varios métodos de distribución de las tareas periódicas en tiempo de diseño

Una vez se ha comprobado que la distribución MaxPromo+LLFF obtiene unos resultados que pueden ser algo inferiores pero del mismo orden que LLFF se ha

comprobado si en realidad es efectivo con ambos tipos de tareas aperiódicas. Para ello se han repetido los experimentos del apartado 5.6.1.1 (véase la Figura 5.20). Así, en la Figura 5.23 se muestran los nuevos resultados obtenidos divididos por los resultados anteriores (en el eje de las ordenadas se describe como MaxPromo / LLFF). En las gráficas de la izquierda podemos observar como el tiempo medio de respuesta a las tareas aperiódicas sin plazo en la mayoría de casos ha mejorado considerablemente. Por ejemplo, HDPmin en la gráfica (a) llega a dividir por 5. Por el contrario, en las gráficas de la derecha se observa como en general la tasa de aceptación de tareas aperiódicas con plazo se ha reducido, aunque en menor grado. Por ejemplo, HDPmin en la gráfica (a) llega a perder cerca de 14 puntos, pasando del 98,8% con LLFF al 85% con MaxPromo+LLF. Esta pérdida en la tasa de aceptación ha propiciado una mejora espectacular para el tiempo de respuesta de las tareas aperiódicas sin plazo.

HDPmax es el más favorecido por la distribución MaxPromo+LLFF. Antes era el método que obtenía menores tiempos de respuesta para las tareas aperiódicas y ahora sigue siéndolo, puesto que en las gráficas de la izquierda se mantiene en la paridad. Antes era el que peor porcentaje de aceptación de tareas aperiódicas con plazo tenía y ahora, en las gráficas de la derecha de la Figura 5.23 es el único que ha mejorado (confirmando lo observado en las gráficas de la derecha de la Figura 5.22).

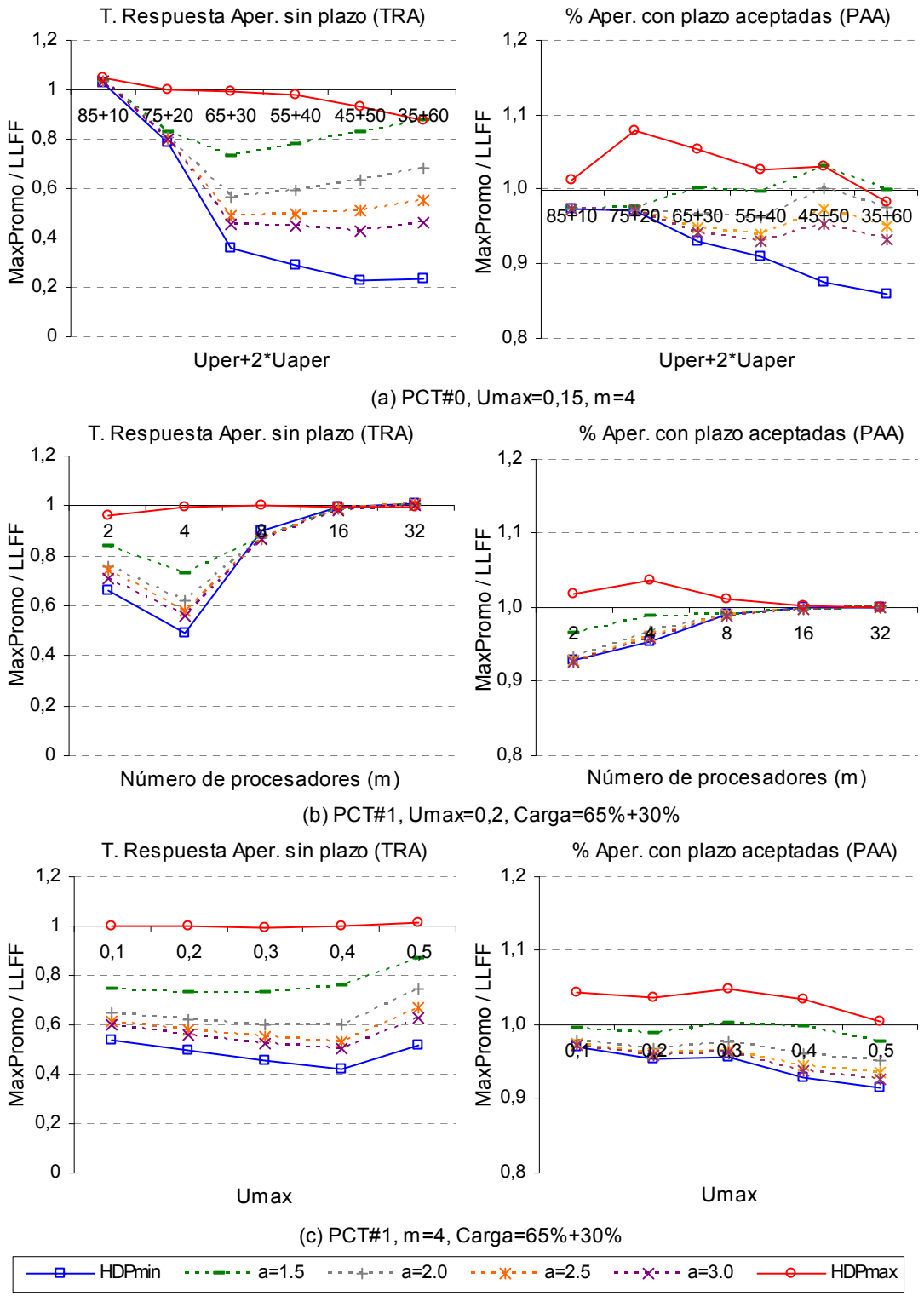


Figura 5.23: Rendimiento de la distribución MaxPromo+LLFF versus LLFF. Se muestra la división entre los resultados obtenidos con la distribución MaxPromo+LLFF (Algoritmo 5-6) con los obtenidos para la Figura 5.20 con LLFF

Los valores absolutos del experimento de la Figura 5.23-(a) se muestran en la Figura 5.24. Como puede observarse, TRA y PAA se igualan indistintamente del parámetro *a*. HDPmin ha reducido el TRA a un nivel aceptable a costa de tan solo perder un promedio del 2% el PAA. Por su parte, HDPmax no ha empeorado el TRA y sí ha incrementado un promedio del 3%. En definitiva, con la distribución MaxPromo+LLFF no hace falta el mecanismo del umbral de calidad de servicio y el diseñador puede elegir entre HDPmin o HDPmax indistintamente. Sin embargo, ya que la diferencia en el PAA no es apenas significativa y el TRA obtenido con HDPmax llega a ser un 14% menor que con HDPmin entonces podemos concluir que la distribución con el planificador HDPmax es la mejor opción para la planificación de tareas periódicas conjuntamente con tareas aperiódicas con y sin plazo.

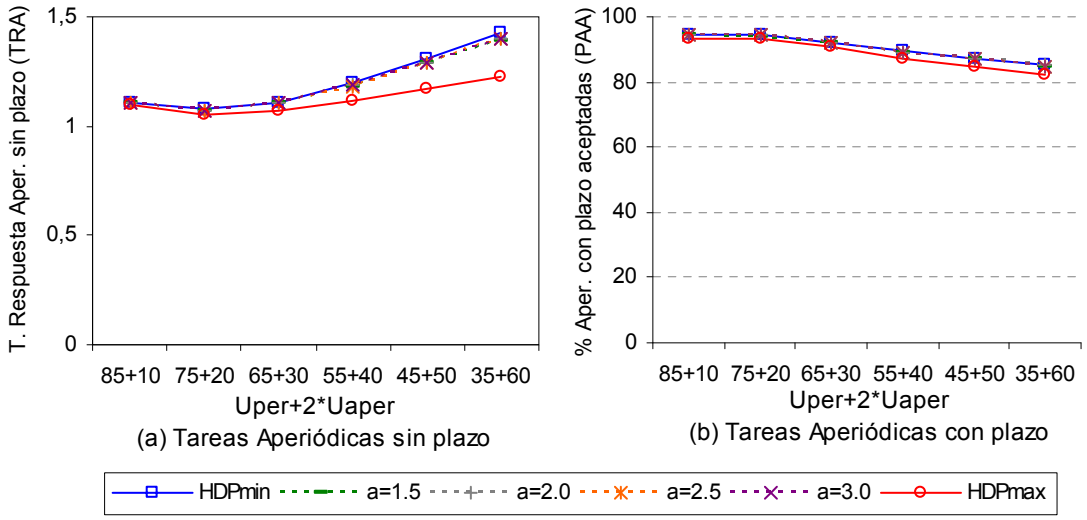


Figura 5.24: Rendimiento de la distribución MaxPromo+LLFF versus LLFF. Se muestran los resultados obtenidos con la distribución MaxPromo+LLFF (Algoritmo 5-6) para el experimento de la Figura 5.20-(a)

5.6.2 Conclusiones

En el apartado 5.6 se ha considerado la posibilidad de utilizar el algoritmo de planificación HDP (detallado en el apartado 5.5) para planificar tareas periódicas junto con tareas aperiódicas sin plazo y con tareas aperiódicas con plazo.

Los resultados han sido excelentes, puesto que nuestro método de planificación aprovecha de la capacidad de retrasar las tareas periódicas y la capacidad de migrar las tareas periódicas entre los procesadores para dar buen servicio al resto de tareas. Los porcentajes de aceptación (PAA) alcanzados para las cargas de trabajo generadas sintéticamente están en el rango [80%..100%]. Aunque no hay un límite superior al tiempo de respuesta aperiódico sin plazo, el promedio del TRA alcanzado en la mayoría de los casos es inferior a dos veces el WCET.

HDPmin resulto ser el mejor planificador para las tareas aperiódicas con plazo y HDPmax el mejor para las tareas aperiódicas sin plazo. Sin embargo, ninguno de los dos métodos tiene una prevalencia total respecto al otro. Así, los diseñadores de aplicaciones de tiempo real, en la fase de diseño, deberían elegir uno de los dos métodos propuestos (HDPmin o HDPmax), según sean las necesidades de cada aplicación y entonces decidir qué tareas aperiódicas requieren plazo y cuales no, en consonancia a su importancia y con respecto al método elegido.

También se han propuesto dos aproximaciones para asignar peticiones aperiódicas a los procesadores combinados con los planificadores locales con TB para poder contrastar los resultados (TBmin y TBmax). Estos métodos de planificación resultaron ser los menos efectivos.

Se realizaron experimentos con tareas aperiódicas con plazos más estrictos y los planificadores HDP mantuvieron tasas de aceptación elevadas y tiempos de respuesta buenos. No ocurrió así con los TB, los cuales no aceptan tareas aperiódicas con plazos muy urgentes. Se comprobó que las distribuciones de tareas periódicas no balanceadas pueden mejorar el rendimiento de TB pero el porcentaje de aceptación no llego a superar el 40%.

Está claro que existe una competencia directa entre las tareas aperiódicas sin plazo y las tareas aperiódicas con plazo. Así, introducimos un umbral a modo de indicador de la calidad de servicio recibida por las tareas aperiódicas sin plazo para balancear automáticamente el rendimiento de las peticiones aperiódicas con plazo o sin plazo. Este parámetro se utilizo en dos versiones del planificador, HDPswitch y HDPqos,

aunque con unos resultados equivalentes. Estos métodos permiten al planificador mantener el tiempo de respuesta de las tareas aperiódicas sin plazo por debajo de este umbral con una alta probabilidad y el porcentaje de aceptación de las tareas aperiódicas con plazo en un nivel aceptable.

Finalmente, se ha diseñado un nuevo método para distribuir las tareas periódicas en tiempo de diseño (MaxPromo+LLFF) que permite dar mejor servicio a ambos tipos de tareas aperiódicas, incluso para las versiones del planificador que no utilizan el parámetro de calidad de servicio.

5.7 Conclusiones del capítulo

La fase dinámica de la ejecución de las tareas periódicas da bastante flexibilidad para realizar el balanceo de la carga. Debido a que la ejecución de la tarea está partida en dos fases y debido a este balanceo de la carga, el número de las aplicaciones planificables es mayor con el algoritmo HDP que con el resto de los algoritmos experimentados. Este método tiene éxito en un elevado porcentaje de conjuntos de tareas y es muy independiente de las características de estas. En particular no se ve afectado por el efecto Dhall. Por otra parte, y de importancia capital en los sistemas de tiempo real estrictos o críticos, la fase estática garantiza los plazos de las tareas periódicas.

Las modificaciones realizadas en el apartado 5.3 permiten mejorar la planificabilidad y la utilización de los recursos de cómputo. Es decir, permiten en tiempo de diseño utilizar un menor número de procesadores que otros métodos o bien permiten realizar mayor cantidad de trabajo con un número fijo de procesadores.

Cuando HDP se quiera utilizar en sistemas de tiempo real estricto, los diseñadores de la aplicación deben realizar la simulación durante un hiperperiodo para verificar si el sistema es planificable (en el caso que haya alguna tarea periódica no garantizada). Sin embargo, cuando se quiera utilizar en sistemas de tiempo real laxos, no se requiere ninguna simulación y el sistema funcionará correctamente con una

probabilidad muy alta. Además, las tareas susceptibles de incumplir algún plazo se pueden identificar y se pueden prever las consecuencias en la aplicación.

Cuando el objetivo es dar un buen servicio a las tareas aperiódicas sin plazo las distribuciones balanceadas permiten a HDP proporcionar los mejores resultados. No obstante, HDP tiene la capacidad de balancear parte de la carga y se pueden utilizar distribuciones asimétricas de la carga entre los procesadores si fuese necesario. Cuando el número de procesadores es alto entonces no importa tanto el tipo de distribución utilizado puesto que el mecanismo de balanceo automático de HDP es más efectivo.

En ningún caso se han observado diferencias significativas en el servicio a las tareas aperiódicas sin plazo entre el planificador híbrido de este capítulo y los planificadores DP globales puros del capítulo anterior (apartado 4.3.2). Sin embargo, el planificador híbrido tiene la enorme ventaja sobre el planificador global puro que garantiza todos los plazos de las tareas periódicas, siendo así también aplicable a sistemas de tiempo real estrictos.

Además, al ser determinista HDP, también es capaz de aceptar y garantizar tareas aperiódicas con plazo. Así, en el apartado 5.5 se ha desarrollado una prueba de aceptación para las tareas aperiódicas con plazo basada en los tiempos de promoción de la de las tareas periódicas. La prueba de aceptación propuesta es simple y tiene unos requisitos muy bajos de cómputo y de memoria. Al ser una prueba de aceptación de bajo costo tiene el inconveniente de no ser muy ajustada. Por este motivo se ha propuesto una asignación de las tareas periódicas en tiempo de diseño distinta para aumentar porcentajes de garantía de las tareas aperiódicas con plazo. Así, con la metodología de planificación presentada, todas las tareas periódicas cumplen su plazo, las tareas aperiódicas con plazo aceptadas también cumplen.

Con la caracterización de las tareas experimentada y en todas las distintas variaciones de los parámetros individuales, HDPmin es el planificador que mejores porcentajes de aceptación obtiene, manteniéndose siempre por encima del 80%, porcentaje que podemos considerar alto, y es el que más estable. El funcionamiento

de HDPmax es peor que el de HDPmin porque asigna plazos más largos a las tareas aperiódicas con plazo, es decir reserva más capacidad para cada tarea aperiódica con plazo y esto dificulta la aceptación de futuras peticiones aperiódicas. Por otra parte, el funcionamiento del peor HDP (HDPmax) es similar al funcionamiento del mejor TB (TBmin). La ventaja de los algoritmos HDP frente a los TB es atribuible a la mayor información que estos poseen (las promociones) y a la capacidad de HDP migrar tareas periódicas dando preferencia a las aperiódicas. Conviene recordar que los planificadores TB son planificadores locales (PF/AD según la nomenclatura del Capítulo 3) en cambio los HDP son planificadores híbridos.

Cuando el objetivo es aumentar el número de tareas aperiódicas con plazo aceptadas con HDP las distribuciones de las tareas periódicas en tiempo de diseño no balanceadas son mejores que las balanceadas. Concretamente, la distribución basada en la laxitud de las tareas periódicas (LLFF) es la mejor. Este resultado contrasta fuertemente con el obtenido en el apartado 5.4.1, donde, con el objetivo era reducir el tiempo medio de respuesta a las tareas aperiódicas sin plazo, las mejores distribuciones para HDP eran las balanceadas. Para los planificadores propuestos derivados de TB la distribución balanceada también es la distribución más adecuada.

En el apartado 5.6 se ha considerado la posibilidad de utilizar el algoritmo de planificación HDP para planificar tareas periódicas junto con tareas aperiódicas sin plazo y con tareas aperiódicas con plazo. Los resultados han sido excelentes, puesto que nuestro método de planificación aprovecha de la capacidad de retrasar las tareas periódicas y la capacidad de migrar las tareas periódicas entre los procesadores para dar buen servicio al resto de tareas. Los porcentajes de la garantía alcanzados para las cargas de trabajo generadas sintéticamente están en el rango [80%..100%]. Aunque no hay un límite superior al tiempo de respuesta aperiódico sin plazo, el promedio del TRA alcanzado en la mayoría de los casos es inferior a dos veces el WCET.

HDPmin resulto ser el mejor planificador para las tareas aperiódicas con plazo y HDPmax el mejor para las sin plazo. Sin embargo, ninguno de los dos métodos tiene una prevalecía total respecto al otro. Así, los diseñadores de aplicaciones de tiempo real, en la fase de diseño, deberían elegir uno de los dos métodos propuestos

(HDPmin o HDPmax), según sean las necesidades de cada aplicación y entonces decidir qué tareas aperiódicas requieren plazo y cuales no, en consonancia a su importancia y con respecto al método elegido.

También se han propuesto dos aproximaciones para asignar peticiones aperiódicas a los procesadores combinados con los planificadores locales con TB para poder contrastar los resultados (TBmin y TBmax). Estos métodos de planificación resultaron ser los menos efectivos.

Se realizaron experimentos con tareas aperiódicas con plazos más estrictos y los planificadores HDP mantuvieron tasas de aceptación elevadas y tiempos de respuesta buenos. No ocurrió así con los TB, los cuales no aceptan tareas aperiódicas con plazos muy urgentes. Se comprobó que las distribuciones de tareas periódicas no balanceadas pueden mejorar estas situaciones pero el porcentaje de aceptación puede no superar el 40%.

En los experimentos ha quedado claro que existe una competencia directa entre las tareas aperiódicas sin plazo y las con plazo. Así, introducimos un umbral a modo de indicador de la calidad de servicio recibida por las tareas aperiódicas sin plazo para balancear automáticamente el rendimiento de las peticiones aperiódicas con plazo o sin plazo. Este parámetro se utilizo en dos versiones del planificador, HDPswitch y HDPqos, aunque con unos resultados equivalentes. Estos métodos permiten al planificador mantener el tiempo de respuesta bajo este umbral con una alta probabilidad y el porcentaje de aceptación de tareas aperiódicas con plazo en un nivel aceptable.

Finalmente, se ha diseñado un nuevo método (MaxPromo+LLFF) para distribuir las tareas periódicas en tiempo de diseño que permite dar mejor servicio a ambos tipos de tareas aperiódicas, incluso para las versiones del planificador que no utilizan el parámetro de calidad de servicio. En concreto, HDPmin es la mejor opción puesto que con la nueva distribución el tiempo de respuesta de las tareas aperiódicas sin plazo se ha equiparado al de HDPmax.

CONCLUSIONES

En esta tesis se ha abordado el problema de la planificación de sistemas de tiempo real utilizando una arquitectura de multiprocesador de memoria compartida. La complejidad y las anomalías en la planificación de los sistemas multiprocesador junto con el determinismo requerido por los sistemas de tiempo real estrictos derivan en un problema realmente difícil de solucionar. Tradicionalmente se ha venido simplificando la problemática tratando el multiprocesador como monoprocesadores independientes, distribuyendo la carga de forma estática y planificándola de forma local. Sin embargo, con la reciente proliferación de sistemas multiprocesador de bajo coste, sistemas multicore y sistemas on-chip-mp, a lo largo de los últimos años la planificación global del multiprocesador ha centrado la atención y los esfuerzos de diversos investigadores. El objetivo general es el poder hacer un uso más exhaustivo de la capacidad de proceso de este tipo de arquitecturas. La teoría de la planificabilidad ha madurado pero todavía tiene algunas lagunas. Por un lado, el grado de utilización para garantizar las tareas periódicas es bajo. Por otro lado, la capacidad de proceso sobrante es difícilmente aprovechable para dar servicio a tareas no periódicas.

En numerosos experimentos de simulación hemos observado que si los recursos de cómputo son abundantes entonces los planificadores que obtenían buenos rendimientos con sistemas monoprocesador también son capaces de obtenerlos en sistemas multiprocesador. Incluso pueden llegar a mejorar su rendimiento si las cargas son medio-bajas (digamos inferiores al 65%) o si disponen de una cantidad considerable de procesadores (pongamos a partir de 8 procesadores). Concretamente, para estos casos, en esta tesis se ha diseñado una planificación con servidores y se ha abordado el problema de la distribución dinámica de tareas aperiódicas entre los procesadores, tanto para planificadores con servidores como sin ellos.

Cuando las cargas son altas y el número de procesadores es medio (entre dos y ocho) entonces los planificadores más extendidos sufren un empeoramiento importante en

su rendimiento. En estos casos se ha detectado la necesidad de diseñar nuevos métodos basados en la planificación global, puesto que esta permite una mayor flexibilidad. Sin embargo, se ha desestimado la planificación global con servidores por la complejidad en su dimensionado. Así, hemos estudiado la posible adaptación de otros tipos de algoritmos de planificación para monoprocesadores. Se han considerado algoritmos que (i) proporcionen tiempos de respuesta de las tareas aperiódicas (ii) su complejidad y sus costes fueran bajos.

En esta tesis hemos propuesto dos algoritmos de planificación que no sufren de este efecto: GDP y HDP. El primero no garantiza los plazos de las tareas periódicas y por eso solo es aplicable a sistemas de tiempo real no estrictos. Sin embargo, al ser global, proporciona una gran flexibilidad y un buen rendimiento para las tareas aperiódicas. El segundo, HDP, sí que garantiza los plazos de las tareas periódicas y, en consecuencia, es aplicable a sistemas de tiempo real estrictos. Además, se ha diseñado un test de aceptación de tareas aperiódicas con plazo y los experimentos muestran que puede conseguir unos porcentajes de aceptación altos. Además, el servicio conjunto de tareas aperiódicas sin plazo es posible y el rendimiento para estas es equiparable al obtenido con GDP.

HDP consigue garantizar los plazos de las tareas periódicas ya que es un método de planificación híbrido que basa sus garantías en la distribución de las tareas de forma pseudo-estática. Así, en tiempo de ejecución el mecanismo de promoción de las tareas las fuerza a ejecutarse en un procesador concreto. El precio a pagar por esta garantía de plazos es una sobrecarga adicional motivada por las migraciones y expulsiones. Así, se ha diseñado un mecanismo de renombrado de procesadores sin apenas coste alguno que reduce drásticamente el número de expulsiones y migraciones

En esta tesis también se han diseñado diversos métodos de distribución pre-runtime de las tareas periódicas para el algoritmo HDP, ya que estas conducen a distribuciones de las promociones diversas. Los experimentos han mostrado que el tipo de distribución pre-runtime determina en gran medida el rendimiento. En general, hemos observado que las distribuciones balanceadas favorecen al servicio a

tareas aperiódicas sin plazo mientras que las distribuciones que concentran la carga favorecen la admisión de nuevas tareas periódicas o aperiódicas con plazo. Así, se ha diseñado el algoritmo MaxPromo+LLFF que distribuiye balanceadamente la carga buscando promociones tardías y, cuando no lo consigue, concentra la carga en algunos procesadores, de forma que HDP puede dar servicio a los tres tipos de tareas.

Ambos algoritmos de planificación, GDP y HDP, requieren de muy poca información adicional. En concreto, necesitan saber cuando se producirán las promociones de las tares periódicas. Las necesidades de proceso también son muy bajas, puesto que los cálculos se realizan en tiempo de diseño. En consecuencia escalan bien y se podrían utilizar en sistemas con un gran número de procesadores.

En definitiva, se ha presentado un método de planificación que elude la problemática de la utilización conjunta de sistemas de tiempo real en sistemas multiprocesador y que cumple con los objetivos propuestos para esta tesis.

En cuanto al trabajo futuro, estamos considerando varias posibles líneas. La primera y más natural, consistiría en expandir la planificación global buscando el ahorro energético cuando no hay tareas aperiódicas ni esporádicas que servir, ralentizando el reloj de algunos procesadores. La segunda línea consistiría en incluir recursos compartidos entre tareas, puesto que el cálculo de los peores tiempos de respuesta permite añadir un factor de bloqueo. En este caso, el algoritmo de planificación en tiempo de ejecución descrito en el Capítulo 5 no variaría pero se debería estudiar la combinación de los métodos de distribución en tiempo de diseño de las tareas que acceden a los recursos compartidos con las técnicas de distribución propuestas en esta tesis. Una tercera línea interesante debida al auge de los sistemas multicore, sería el optimizar la planificación inter-chip, entre los diversos *cores* dentro de un chip, y intra-chip, utilizando una planificación jerárquica. Finalmente, también estamos considerando el extrapolar estos métodos de planificación a entornos no de tiempo real, ni con tareas periódicas, pero si con tareas que tengan patrones repetitivos, que se puedan retrasar permitiendo la ejecución más temprana de otras tareas.

REFERENCIAS

[ABJ01] Anderson, B., Baruah, S., Jonsson, J. “Static-priority Scheduling on Multiprocessors”, Real-Time Systems Symposium, pp. 193-202, 2001

[Amd67] Amdahl, G., "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings, (30), pp. 483-485, 1967

[And03] Anderson, B., “Static-priority Scheduling on Multiprocessors”, Tesis doctoral, Chalmers University of Technology, Suecia, 2003

[AnJ00] Anderson, B., Jonsson, J., “Some Insights on Fixed-Priority Preemptive Non-Partitioned Multiprocessor Scheduling”, Real-Time Systems Symposium, Work-in-Progress Session, pp. 53-56, 2000

[AnJ00b] Anderson, B., Jonsson, J., "Fixed-Priority Pre-emptive Multiprocessor Scheduling: To Partition or not to Partition". Real-Time Computing Systems and Applications, pp. 337-346, 2000

[AnS00] Anderson, J., Srinivasan, A., "Pfair Scheduling: Beyond Periodic Task Systems", Seventh International Conference on Real-time Computing Systems and Applications, IEEE Computer Society Press, pp. 297-306, December 2000

[Bak03] Baker, T. P., "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis”, Real-Time Systems Symposium, 2003

[Bak05] Baker, T. P., "An Analysis of EDF Schedulability on a Multiprocessor”, IEEE Trans. Parallel Distributed Systems, v. 16, n. 8, pp. 760-768, 2005

[BAL02] Banús, J.M., Arenas, A., Labarta, J., "An Efficient Scheme to Allocate Soft-Aperiodic Tasks in Multiprocessor Hard Real-Time Systems", Parallel and Distributed Processing Techniques and Applications, v.II, pp.809-815, Las Vegas, 2002

[BAL03] Banús, J.M, Arenas, A., Labarta, J., "Dual Priority Algorithm to Schedule Real-Time Tasks in a Shared Memory Multiprocessor", Workshop on Parallel and Distributed Real-Time Systems, Nice, 2003

[BAL03b] Banús, J.M, Arenas, A., Labarta, J. "Improving Multiprocessor Average-Case Schedulability using the Global Dual Priority Algorithm", Real-Time Systems Symposium WIP, pp.89-92, Cancún, 2003

[BaL04a] Baruah, S., Lipari, G., “A Multiprocessor Implementation of the Total Bandwidth Server”, International Parallel and Distributed Processing Symposium, 2004

[BaL04b] Baruah, S., Lipari, G., “Executing Aperiodic Jobs in a Multiprocessor Constant-Bandwidth Server Implementation”, Euromicro Conference on Real-Time Systems, pp. 109-116, 2004

[BAL05] Banús, J.M, Arenas, A., Labarta, J., "Extended Global Dual Priority Algorithm for Multiprocessor Scheduling in Hard Real-Time Systems", Euromicro Conference on Real-Time Systems WIP, pp.13-16, Palma de Mallorca, 2005

[BCL05] Bertogna, M., Cirinei, M., Lipari, M, "Improved Schedulability Analysis of EDF on Multiprocessor Platforms", Euromicro Conference on Real-Time Systems, 2005

[BCP96] Baruah, S.K., Cohen, N.K., Plaxton, C.G., Varvel, D.A., "Proportionate Progress: A Notion of Fairness in Resource Allocation", Algorithmica, vol. 15, no. 6, pp. 600-625, 1996

[Ber98] Bernat, G., "Specification and Analysis of Weakly Hard Real-Time Systems", tesis doctoral, Universitat de les Illes Balears, 1998

[BGL02] Baruah, S., Goossens, J., Lipari, G., "Implementing Constant-Bandwidth Servers upon Multiprocessor Platforms", IEEE Real-Time and Embedded Technology and Applications Symposium, 2002

[BGP95] Baruah, S.K., Gehrke, J.E., Plaxton, C.G., "Fast Scheduling of Periodic Tasks on Multiple Resources", Proc. Ninth Int'l Parallel Processing Symp., pp. 280-288, Apr. 1995

[BLY95] Burchard, A., Liebeherr, J., Yingfeng Oh, Sang H. Son, "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems", IEEE Transactions on Computers, vol. 44, no. 12, December 1995

[BML99] Banús, J.M, Moncusí, M.A., Labarta, J., "Estudio de la planificación de tareas en sistemas en tiempo real con asignación dinámica en entornos multiprocesador", VII Jornadas de Concurrencia, Gandia, 1999

[Bur91] Burns, A., "Scheduling Hard Real-Time Systems: a Review", Software Engineering Journal, 6 (3), pp. 116-128, 1991

[Bus96] Busquets Mataix, J. V., "Técnicas para la Utilización Eficaz de Memorias Cache en Sistemas de Tiempo Real", Tesis Doctoral, DISCA, UPV, 1996

[BuS99] Buttazzo, G., Sensini, F., "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments", IEEE Transactions on Computers, Vol. 48, No. 10, pp. 1035-1052, 1999

[But00] Buttazzo, G.C. "Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications", Kluwer Academic Publishers, 2000

[But05] Buttazzo, G.C. "Rate Monotonic vs. EDF: Judgment Day", Real Time Systems, 29, pp. 5-26, 2005

[BuW01] Burns, A., Wellings, A., "Sistemas d Tiempo Real y Lenguajes de programación, 3a Edición", Addison Wesley, 2001

[BuW93] Burns, A., Wellings, A.J., "Dual Priority Assignment: A Practical Method for Increasing Processor Utilization", Fifth Euromicro Workshop on Real-time Systems, pp. 48-53, 1993

[ChC89] Chetto, H., Chetto, M., "Some results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, v.15, n.10, 1989

[DaD86a] Davari, S., Dhall, S.K., “On a Periodic Real-Time Task Allocation Problem," Proc. of 19th Annual International Conference on System Sciences, pp. 133-141, 1986

[DaD86b] Davari, S., Dhall, S.K., “An On Line Algorithm for Real-Time Task Allocation”, Real-Time Systems Symposium, pp. 194-199, 1986

[DaW95] Davis, R., Wellings, A., "Dual Priority Scheduling", Real-Time Systems Symposium, pp. 100-109, 1995

[DeM89] Dertouzos, M.L., Mok, A.K., "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks", IEEE Transactions on Software Engineering, v.15, n.12, pp. 1497-1506, 1989

[Der74] Dertouzos, M.L., “Control Robotics: the Procedural Control of Physical Processes” Information Processing 74, North-Holland Publishing Company, 1974

[Dha77] Dhall, S. K., “Scheduling Periodic-Time-Critical Jobs on Single Processor and Multiprocessor Computing Systems”, PhD thesis, University of Illinois at Urbana-Champaign, 1977

[DhL78] Dhall, S. K., Liu, C. L., “On a Real-Time Scheduling Problem”, Operations Research, Vol. 26, No. 1, pp. 127-140, 1978

[DiS00] Di Natale, M., Stankovic, J.A., “Scheduling Distributed Real-Time Task with Minimum Jitter“, IEEE Transactions on Computers, v.49, n.4, pp. 303-316, 2000

[DTB93] Davis, R.I., Tindell, K.W., Burns, A., "Scheduling Slack Time in Fixed-Priority Pre-emptive Systems", Real-Time System Symposium, pp. 222-231, 1993

[Foh95] Fohler, G., ”Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems”, Real-Time Systems Symposium, 152-161, 1995

[GaJ75] Garey, M.R., Johnson, D.S., "Complexity Results for Multiprocessor Scheduling under Resource Constraints". SIAM Journal on Computing, 4(4): 397-411, 1975

[GaJ79] Garey, M.R., Johnson D.S., “Computers and Intractability. A guide to the Theory of NP-Completeness”, W.H. Freeman and Company, 1979

[GFB02] Goossens, J., Funk, S., Baruah, S., "EDF Scheduling On Multiprocessor Platforms: Some (Perhaps) Counterintuitive Observations", Real-Time Computing Systems and Applications, pp. 321-330, 2002

[GFB03] Goossens, J., Funk, S., Baruah, S., “Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors", Real-Time Systems, 25 (2-3): pp. 187-205, 2003

[Gra76] Graham, R.L., “Bounds on the Performance of Scheduling Algorithms”, Computer and Job-Shop Scheduling Theory, E. G. Coffman, ed. John Wiley and Sons, N.Y., pp.165–225, 1976

[HoS94] Chao-Ju Hou, Kang G. Shin, “Replication and Allocation of Task Modules in Distributed Real-Time Systems”, Fault-Tolerant Computing Sympossium, pp. 26-35, 1994

[IsF00] Isovic, D., Fohler, G., “Efficient Scheduling of Sporadic, Aperiodic and Periodic Tasks with Complex Constrains”, Real-Time Systems Symposium, 2000

[JoP86] Joseph, M., Pandya, P., "Finding Response Times in a Real-Time System", British Computer Society Computer Journal, 29(5): 390-395, Cambridge University Press, 1986

[KaW91] Kamenoff, N.I., Weiderman, N.H., "Hartstone Distributed Benchmark: Requirements and Definitions", Real-Time Systems Symposium, pp. 199-208, 1991

[Law83] Lawler, E.L., “Recent results in the theory of machine scheduling”, Mathematical Programming: The State of the Art, Springer, pp. 202–234, 1983

[LDG01] López, J.M., Díaz, J.L., García, D.F., “Minimum and Maximum Utilization Bounds for Multiprocessor Rate Monotonic Scheduling”, 13th Euromicro Conference on Real-time Systems, pp. 67-75, 2004

[LDG04] López, J.M., Díaz, J.L., García, D.F., “Minimum and Maximum Utilization Bounds for Multiprocessor Rate Monotonic Scheduling”. IEEE Trans. Parallel Distrib. Syst. v. 15, n. 7, pp. 642-653, 2004

[LeW82] Leung, J.Y.-T., Whitehead, J., “On the Complexity of Fixed-Priority Scheduling of Periodic Tasks, Real-Time Tasks”, Performance Evaluation, 2(4), pp. 1429-1442, 1982

[LiL73] Liu, C.L., Layland, J.W., "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", Journal of the Association for Computing Machinery, vol. 20(1), pp. 46-61, 1973

[LMM03] Lauzac, S., Melhem, R. Mossé, D., “An Improved Rate-Monotonic Admission Control and Its Applications”, IEEE Transactions on Computers, v. 52, no 3, pp. 337-350, 2003

[LMM98] Lauzac, S., Melhem, R. Mossé, D., “Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor”, The 10th Euromicro Workshop on Real Time Systems, pp. 188-195, 1998

[LRT92] Lehoczky, J.P., Ramos-Thuel, S., "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems", Real Time Systems Symposium, pp. 110-123, 1992.

[LRT94] Lehoczky, J.P., Ramos-Thuel, S., "Chapter 8: Scheduling Periodic and Aperiodic Tasks using the Slack Stealing Algorithm", pp. 175-197, Principles of Real-Time Systems, Prentice Hall, 1994

[LSD89] Lehoczky, J.P., Sha, L., Ding, Y., "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", Real-Time Systems Symposium, pp. 166-171, 1989

[LSS87] Lehoczky, J.P., Sha, L., Stronsnider, J.K., "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", Real-Time Systems Symposium, pp. 261-270, 1987

[MBL97] Moncusí, M.A., Banús, J.M, Labarta, J., Llamosí, A., "The Last Call Scheduling Algorithm for Periodic and Soft Aperiodic Tasks in Real-Time Systems", V Jornadas de Concurrencia, Vigo, 1997

[OhB98] Oh, D., Baker, T. P., "Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment", Real Time Systems, v.15, pp. 183–192, 1998

[OhS93] Oh, Y., Son, S.H., "Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem", Tech. Report CS-93-24, Dept. of Computer Science, University of Virginia, 1993

[OhS95] Oh, Y., Son, S.H., "Fixed-Priority Scheduling of Periodic Tasks on Multiprocessor Systems", Technical Report CS-95-16, Dept. of Computer Science, University of Virginia, 1995

[OhY98] Oh, S.H.; Yang, S.M; "A Modified Least-Laxity-First scheduling algorithm for real-time tasks", Real-Time Computing Systems and Applications, pp. 31-36, 1998

[PLA01] Palopoli, L., Lipari, G., Abeni, L., Di Natale, M., Ancilotti, P., Conticelli, F., "A Tool for Simulation and Fast Prototyping of Embedded Control Systems". ACM SIGPLAN Workshop on Languages, Compilers and Tools For Embedded Systems, pp, 73-81, 2001

[PSA97] Peng, D., Shin, K. G., Abdelzaher, T. F., "Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems", IEEE Trans. Softw. Eng., v. 23, n. 12 , pp. 745-758, 1997

[Ram95] Ramamritham, K., "Allocation and scheduling of precedence-related periodic tasks", IEEE Trans. on Parallel and Distributed Systems, v. 6, n. 4, pp. 412-420, 1995

[RSS90] Ramamritham, K., Stankovic J. A., Shiah P-F., "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems", IEEE Transactions on Parallel and Distributed Systems, Vol. 1, No. 2, pp. 184-194, April 1990

[RSZ89] Ramamritham, K., Stankovic, J.A., Zhao W., "Distributed Scheduling of Tasks with Deadlines and Resource Requirements", IEEE Transactions on Computers, C-38, (8), pp. 1110-1123, 1989

[RtL94] Thuel, S.R. Lehoczky, J.P. , “Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Prioritysystems Using Slack Stealing”, Real-Time Systems Symposium, pp. 22-33, 1994

[SHA02] Srinivasan, A., Holman, P., Anderson, J., "Integrating Aperiodic and Recurrent Tasks on Fair-scheduled Multiprocessors", 14th Euromicro Conference on Real-time Systems, IEEE Computer Society Press, pp. 19-28, June 2002

[SLS95] Strosnider, J.K., Lehoczky, J.P., Sha, L., "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", IEEE Transactions on Computers, v. 44, n. 1, pp 73-91, 1995

[SpB94] Spuri, M., Buttazzo, G.C., "Efficient Aperiodic Service Under Earliest Deadline Scheduling", Real-Time Systems Symposium, pp. 2-11, December 1994

[SpB96] Spuri, M., Buttazzo, G.C., "Scheduling Aperiodic Tasks in Dynamic Priority Systems", Real-Time Systems Journal, vol. 10, pp. 179-210, 1996

[SrA02] Srinivasan, A., Anderson, J., "Optimal Rate-based Scheduling on Multiprocessors", Proceedings of the 34th ACM Symposium on Theory of Computing, ACM Press, pp. 189-198, May 2002

[SrB02] Srinivasan, A., Baruah, S., “Deadline-based scheduling of periodic task systems on multiprocessors”, Inf. Process. Lett. 84, 2 , pp. 93-98, 2002

[SRL90] Sha, L., Rajkumar, R., Lehoczky, J.P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Transactions on Computers, vol. 39 no. 9, pp. 1175-1185, 1990

[SSL89] Sprunt, B., Sha, L., Lehoczky, J.P, "Aperiodic Task Scheduling for Hard Real-Time Systems", Journal of Real-Time Systems, vol. 1, pp. 27-60, 1989

[SSR98] Stankovic, J.A., Spuri, M., Ramamritham, K. and Buttazzo, G.C., “Deadline Scheduling for Real-Time Systems EDF and related Algorithms”, Kluwer Academic Publishers, 1998

[Sta88] Stankovic, J. A., “Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems”, IEEE Computer, 21(10):10-19, Oct. 1988

[SVC96] Sáez, S., Vila, J., Crespo, A., "Dynamic Scheduling Solutions for Real-Time Multiprocessor Systems", 21th IFAC/IFIP Workshop on Real-Time Programming, pp. 57-62, 1996

[SVC98] Sáez, S., Vila, J., Crespo, A., "Using Exact Feasibility Tests for Allocating Real-Time Tasks in Multiprocessor Systems”, 10th Euromicro Workshop on Real-Time Systems, pp. 53-60, 1998

[SVC99] Sáez, S., Vila, J., Crespo, A., "Soft Aperiodic Task Scheduling on Real-Time Multiprocessor Systems", 6th International Conference on Real-Time Computing Systems and Applications, pp. 424-427, 1999

A hardware scheduler for complex real-time systems

[SVC99b] Sáez, S., Vila, J., Crespo, A., Garcia, A., "A Hardware Scheduler for Complex Real-Time Systems", IEEE International Symposium on Industrial Electronics, v. 1, pp. 43-48, 1999

[SVC03] Sáez, S., Vila, J., Crespo, A., "Firm Aperiodic Task Scheduling in Hard Real-Time Multiprocessor Systems", 27th IFAC/IFIP/IEEE Workshop on Real-Time Programming, 2003

[TBW92] Tindell, K., Burns, A., Welling, A.J., "Allocating Real-Time Tasks (An NP-Hard Problem made Easy)", Journal of Real-Time Systems, Vol.4, No. 2, pp.145-165, June 1992

[TLS96] Tia, T.S., Liu, J.W.S, Shankar, M., "Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems", Journal of Real-Time Systems, v.10, n.1, pp. 23-43, 1996

[Ull73] Ullman, J.D., "Polynomial Complete Scheduling Problems", Symposium on Operating Systems Principles, 1973

[Xu93] Xu, J., "Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations," IEEE Transactions on Software Engineering, v. 9, no. 2, pp. 139-154, Feb., 1993

A. Anexo A

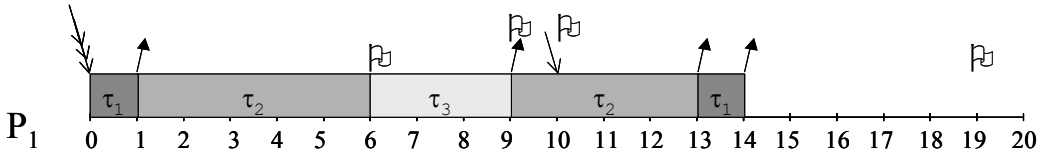
MEJORAS AL ALGORITMO DUAL PRIORITY

En el apartado 4.2 se describe y analiza el algoritmo Dual Priority básico [DaW95]. A pesar de las buenas características de este algoritmo mencionadas anteriormente, se pueden realizar algunas mejoras, sobretodo para conjuntos de tareas con periodos armónicos. Los principales criterios que se van a buscar serán una mejora en el tiempo de respuesta de las tareas aperiódicas y, cuando sea posible, ahorrarse algunos costes, como puede ser reducir el número de cambios de contexto producidos por apropiaciones del procesador.

A.1 Optimizar el orden de prioridades inicial

Optimizar el orden de prioridades inicial para obtener promociones más tardías. *No esta muy claro cual es el criterio de optimización*. Además, al utilizar ordenaciones diferentes a las bien conocidas como *Rate-Monotonic* o *Deadline-Monotonic* nos hace perder sus características principales, como por ejemplo las condiciones de planificabilidad. Se ha ensayado el maximizar la media aritmética o geométrica de las promociones pero no se han obtenido resultados concluyentes. Sin embargo, existe un caso particular del *Rate-Monotonic,* que se produce cuando hay tareas con idéntico periodo. En este caso hemos llegado a la conclusión que se deben ordenar por utilización creciente. En el siguiente ejemplo (Figura A.1) las tareas τ_2 y τ_3 tienen el mismo periodo pero diferente cómputo.

Tarea	Periodo	Plazo	WCET	WCRT	Promoción
τ_1	10	10	1	1	9
τ_2	20	20	9	10	10
τ_3	20	20	3	14	6



Tarea	Periodo	Plazo	WCET	WCRT	Promoción
τ_1	10	10	1	1	9
τ_3	20	20	3	4	16
τ_2	20	20	9	14	6

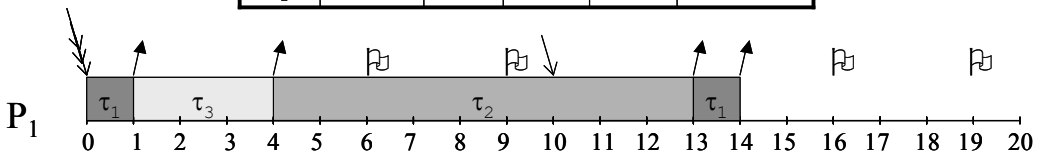


Figura A.1 Ejemplo donde reordenando las tareas con igual periodo por cálculo mejora la distribución de las promociones

En la parte superior de la figura se ordenan las tareas con igual periodo de forma que la que tiene una mayor utilización (τ_2) tiene mayor prioridad. Esto hace que la tarea menor tenga una interferencia muy grande y por lo tanto un tiempo de respuesta muy largo. Sin embargo, intercambiando las prioridades de estas dos tareas, en la parte inferior de la figura, la tarea τ_3 ya no solo no recibe la interferencia de la tarea grande, sino que además se ahorra una interferencia de la tarea τ_1, obteniendo un tiempo de respuesta muy corto y, por lo tanto una promoción más tardía. Obsérvese que en ningún caso la promoción de la tarea con menor prioridad se ve afectada (en el ejemplo es 6), puesto que la interferencia de las tareas de mayor prioridad no se altera y la suma de los cómputos de las tareas de igual periodo se mantiene. Con esta ordenación se pretende dar mejor tiempo de respuesta a las tareas aperiódicas, puesto que las promociones estarán más dispersas. También observamos que de esta forma se pueden ahorrar algunas apropiaciones. En el ejemplo, en la parte inferior de la

figura, la tarea τ_2 se ejecuta sin ninguna interrupción. Esto es debido a que las tareas pequeñas ya han terminado cuando se produce su promoción.

A.2 Optimizar el orden de prioridades de bajo nivel (LPL)

Para el algoritmo DP el orden de las prioridades de alto nivel (HPL) es vital para garantizar los plazos. Sin embargo, el orden LPL puede ser cualquiera puesto que no influye en la garantía de los plazos. Mantener el orden original (el mismo que en HPL, por ejemplo RM) no tiene porque ser la mejor opción. Por un lado, el número de apropiaciones puede ser mayor, pudiéndose llegar a doblar. Este efecto negativo es debido al hecho de tener dos niveles de prioridad y promocionar las tareas de un nivel a otro. Esto puede provocar que las apropiaciones que se hayan producido en el nivel bajo de prioridades se repitan después una vez promocionadas las tareas, doblando así el número de apropiaciones. Por otro lado, en ausencia de la necesidad temporal de prestar servicio a otras tareas debería darse más prioridad a la tarea que pueda perjudicar más a las futuras demandas de servicio. En ambos casos, la ejecución *Earliest Promotion First* (EPF), es decir, la ejecución de la tarea con la promoción más cercana cuando no hay tareas promocionadas ni otro tipo de servicio pendiente, es una opción razonable. Con esto se puede reducir el número de apropiaciones, puesto que en otro caso, probablemente, la tarea con la promoción más cercana realizaría una expulsión al promocionarse. Veámoslo con el ejemplo de la Figura A.2. En este ejemplo, con el orden LPL=RM el número de apropiaciones es 6, el doble de las producidas con el orden LPL=EPF. Esto es debido a que en este ejemplo el orden de las promociones ha resultado inverso al de las prioridades RM.

Tarea	Periodo	Plazo	WCET	WCRT	Promoción
τ_1	8	8	5	5	3
τ_2	16	16	4	14	2
τ_3	32	32	3	31	1

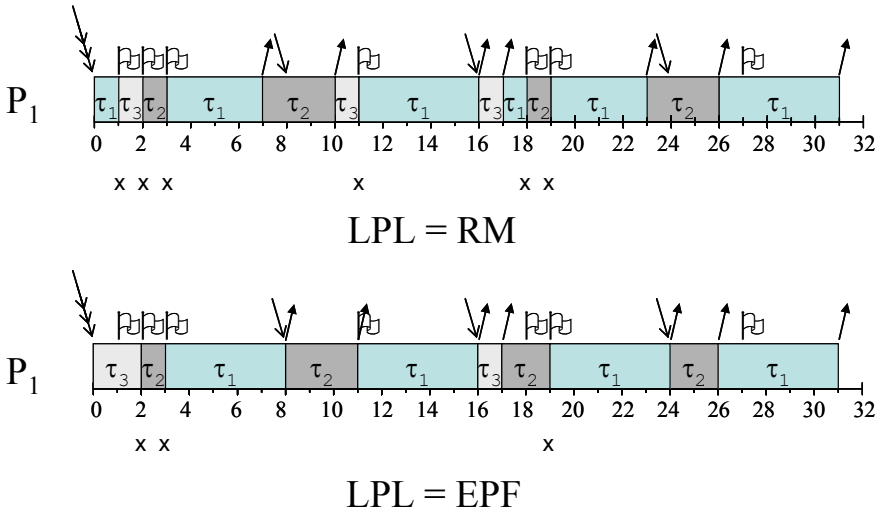


Figura A.2 Ejemplo donde un orden EPF para el LPL ahorra apropiaciones. Las flechas descendentes indican activaciones de tareas y las ascendentes indican finalizaciones. Las banderas en la parte superior indican las promociones y, en la parte inferior, debajo del tiempo, una equis marca las apropiaciones.

En lo referente a otros servicios, avanzar en la ejecución de la tarea que tiene la promoción más cercana puede favorecer al servicio a una presunta demanda inmediata. Sin embargo, desde este punto de vista, una estrategia razonable sería la ordenación de las prioridades LPL según la carga de cada tarea (U_i), puesto que las tareas con más carga serán las que más interferirán con los servicios a prestar. Desde el punto de vista del número de apropiaciones esta asignación de prioridades puede ser nefasta si las tareas con más carga son las menos frecuentes. En estas situaciones, la tarea más grande será interrumpida en numerosas ocasiones por las tareas más frecuentes.

Para comprobar cual de las tres propuestas de ordenación de LPL es la mejor realizamos un estudio estadístico comparativo, junto con una cuarta opción, el orden EDF. En este caso esperamos unas cualidades parecidas a EPF, puesto que frecuentemente ocurrirá que la tarea con la promoción más cercana sea también la

tarea con el plazo más cercano. Para ello utilizamos conjuntos de tareas generados sintéticamente utilizando la parametrización detallada en la Tabla 14.

Conjunto de parámetros	Número de tareas periódicas	Periodo mínimo	Periodo máximo	Hiperperiodo	Carga máxima tarea	Carga periódica total	Carga aperiódica total
CP#1a	12	100	1600	16000	15%	75%	0%
CP#1b	12	100	1600	16000	15%	75%	20%
CP#2a	12	100	1800	264600	15%	75%	0%
CP#2b	12	100	1800	264600	15%	75%	20%

Tabla 14: Parámetros para las simulaciones del estudio de las apropiaciones según el orden de prioridades usado en el LPL

Los parámetros del conjunto CP#1a y CP#1b conducen a conjuntos de tareas con periodos bastante múltiplos entre si (hiperperiodo = 16000 = 2^7 5^3) mientras que con los parámetros CP#2a y CP#2b la multiplicidad disminuye (hiperperiodo = 254600 = 2^3 3^3 5^2 7^2). Los conjuntos CP#1a y CP#2a van a servir para medir las apropiaciones cuando las tareas periódicas no son retardadas por no tener que servir a tareas aperiódicas, mientras que en los otros dos conjuntos las tareas periódicas si van a ser retardadas hasta su promoción en el caso de presencia de tarea aperiódicas.

En la Figura A.3 se puede observar que cuando LPL=U_i se obtiene un menor número de apropiaciones que con el resto de asignaciones, seguida por LPL=EPF. En la Figura A.4 podemos ver que cuando los periodos no son tan múltiplos LPL=1-U_i es la asignación que proporciona un menor número de apropiaciones. Es interesante destacar que, en general, por el mayor desfase en los periodos en estos conjuntos de tareas se dan menos apropiaciones.

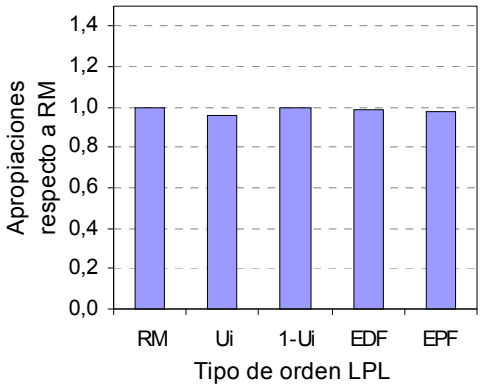


Figura A.3: CP#1, sin aperiódicas

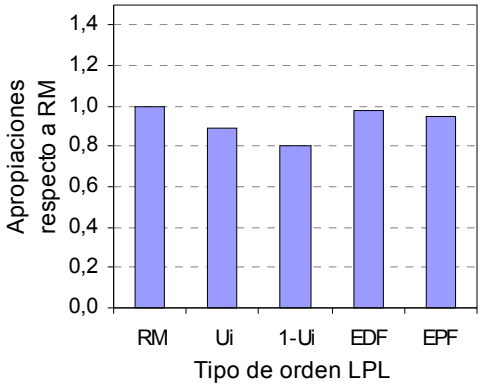


Figura A.4: CP#2, sin aperiódicas

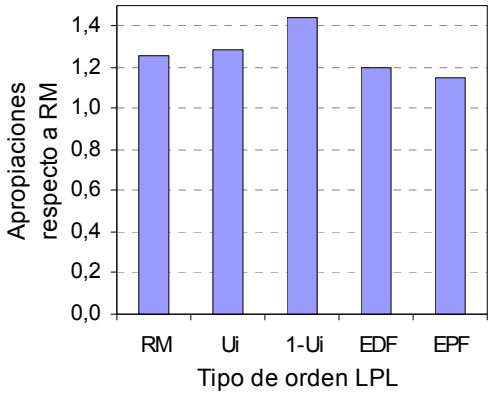


Figura A.5: CP#1a, con aperiódicas

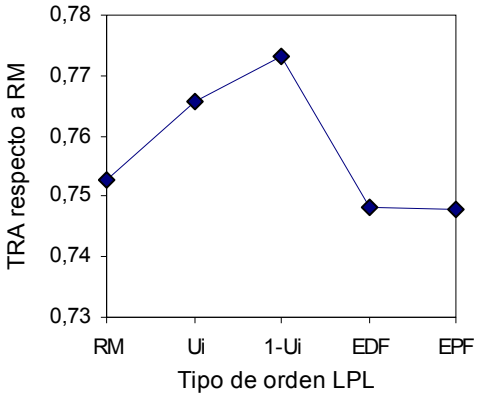


Figura A.6: CP#1a, con aperiódicas

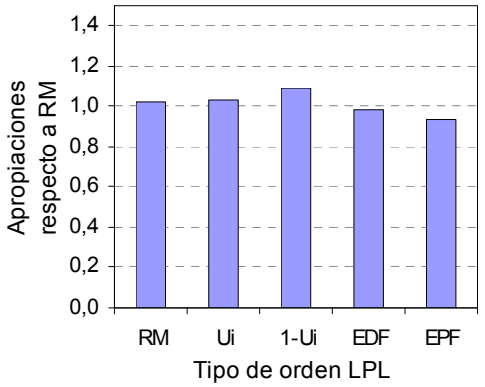


Figura A.7: CP#2a, con aperiódicas

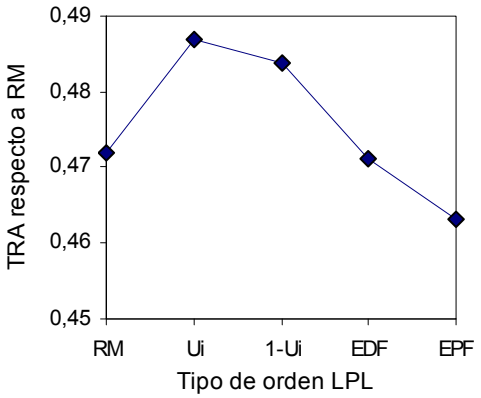


Figura A.8: CP#2a, con aperiódicas

En presencia de tareas aperiódicas, las periódicas tienen que retrasarse y en estos casos las asignaciones de prioridad LPL basadas en la carga (U_i) tienen el peor comportamiento, tanto en lo referente al número de apropiaciones como al tiempo medio de respuesta de las tareas aperiódicas. La asignación LPL=1-U_i es la peor en apropiaciones (Figura A.5) y también en TRA (Figura A.6). Por el contrario, la asignación LPL=EPF exhibe el mejor comportamiento. En la Figura A.5 y en la Figura A.7 podemos observar que es la asignación con menor número de apropiaciones, mientras que en la Figura A.6 y en la Figura A.8 se observa un comportamiento entre un 3% y un 5% mejor que LPL=U_i.

Finalmente, observamos que las diferencias entre EPF y EDF se hacen más evidentes cuando los periodos no son múltiples. Así, en la Figura A.7 podemos observar que LPL=EPF tiene una media de apropiaciones 5% menor que LPL=EDF y en la Figura A.8 podemos observar que LPL=EPF tiene una media de tiempo de respuesta a las tareas aperiódicas 2% mejor.

En consecuencia, podemos concluir que el mejor orden de prioridades para las tareas periódicas en el nivel LPL en general es el orden EPF.

A.3 Posponer una promoción si ya hay una parte ejecutada

Cuando una tarea es promocionada ésta puede haber realizado parte de su ejecución. Esto va a depender del momento en que se active, del orden LPL y de la demanda de otros servicios pendientes antes de su promoción. En estas situaciones se puede posponer la promoción en un tiempo igual al tiempo ya ejecutado, puesto que no va a comprometer su plazo. Esto es equivalente a reducir el componente C_i de la ecuación 2.2. Esta medida puede ahorrar apropiaciones de cpu, puesto que retardando una promoción puede dar tiempo a la tarea que se está ejecutando a terminar o a ser también promocionada. Si se combina con la medida propuesta en el apartado anterior (LPL=EPF) los retardos de los tiempos de promoción pueden llegar a ser substancialmente mayores. Desde el punto de vista del servicio a otras tareas, puede

ir bien para los servicios pendientes pero puede ser contraproducente si no hay servicios pendientes y llegan peticiones justo en el momento de la promoción retardada[40].

A.4 Tener en cuenta todo el trabajo anticipado

En el apartado anterior se tenía en cuenta que una tarea puede haber ejecutado parte de sus necesidades antes de su promoción pero no se tenía en consideración que las tareas de mayor prioridad, las cuales se han contabilizado como interferencia en el cómputo de las promociones, también pueden haberse ejecutado en parte o en su totalidad. Si el planificador lleva una contabilidad de todos estos trabajos anticipados las promociones pueden posponerse todavía más.

La idea básica de esta mejora es aprovechar el trabajo realizado por todas las tareas periódicas en el nivel bajo de prioridad (LPL) para ejecutar las tareas aperiódicas si las hay, mejorando así su tiempo de respuesta.

Su implementación continua siendo simple precisando únicamente de un vector de longitud igual al número de tareas periódicas (cada tarea periódica tiene asignada una prioridad distinta). Este vector servirá para memorizar el "trabajo avanzado" en cada nivel de prioridad. Cuando una tarea periódica abandona el procesador, actualiza la posición del vector correspondiente a su nivel de prioridad, incrementándola con el número de unidades de proceso que ha realizado estando en el nivel bajo de prioridad. Cuando se cumple el plazo de la tarea, si ésta tenía trabajo avanzado, éste ya no puede aprovecharse, por lo que la posición del vector se pone a cero.

El planificador en el momento de decidir que tarea ejecuta tiene varias opciones:

- Si existen tareas periódicas promocionadas y no existe trabajo avanzado al nivel de las tareas promocionadas, escogerá la tarea promocionada de mayor prioridad.

[40] En particular es contraproducente cuando se trata del servicio a tareas aperiódicas con plazo (apartado 5.5).

- Si existen tareas periódicas promocionadas pero existe trabajo avanzado de un nivel superior a las tareas promocionadas, escogerá primero una tarea aperiódica ejecutándola como máximo las unidades de tiempo proporcionadas por el trabajo avanzado al nivel de las tareas promocionadas. En este caso se disminuirá el trabajo avanzado en las unidades de tiempo ejecutadas por la tarea aperiódica.

- Si no existen tareas promocionadas escogerá primero una tarea aperiódica.

Los eventos necesarios serán:

- Activación de una tarea periódica: la tarea se encola en la cola de baja prioridad, se programan los eventos promoción y comprobación de plazo.

- Activación de una tarea aperiódica: la tarea se pone en la cola de tareas aperiódicas en orden FIFO.

- Promoción de una tarea periódica: se actualiza el trabajo avanzado realizado por la tarea, se promociona al nivel prioritario y se incrementa la variable que indica el número de tareas promocionadas.

- Fin del plazo de una tarea: se reinicia el trabajo avanzado realizado por la tarea y se decrementa la variable que indica el número de tareas promocionadas.

Para el articulo [MBL97] se realizaron diversos experimentos. Los resultados mostraron una mejora en el tiempo de respuesta de las tareas aperiódicas. Esta mejora fue substancial en el caso de tener periodos armónicos, que es el peor de los casos para el algoritmo DP. En uno de estos casos llegó a reducir el TRA en un 90%.

A.5 Promociones individuales para cada activación

En el algoritmo DP básico se usa como tiempo de promoción para cada activación de las tareas periódicas un valor muy pesimista: el tiempo de respuesta obtenido por una tarea en el instante crítico si la ejecución es con prioridades fijas (ecuación 2.2). Sin embargo, en tiempo de ejecución en realidad las tareas se ejecutan en base a las promociones, y bien puede suceder que el caso del instante crítico no ocurra nunca. En la tesis [Ber98] se propone una forma de calcular el mínimo de los tiempos de promoción teniendo en cuenta que en tiempo de ejecución se usará el algoritmo DP. Estos valores pueden ser mayores que las promociones encontradas con la ecuación 2.2, con la consiguiente mejora del algoritmo.

En este apartado proponemos una mejora que va más allá. Consiste en utilizar un vector para cada tarea donde guardar los peores tiempos de respuesta particulares de cada activación teniendo en cuenta que la ejecución será en DP, pero con promociones variables, es decir, promociones individualizadas para cada activación de cada tarea periódica. La diferencia con el algoritmo propuesto en [Ber98] estriba en que su método calcula promociones constantes para todas las activaciones.

En el Algoritmo A-1 se detalla el pseudo-código para calcular en tiempo de diseño la promoción individualizada para la k-ésima activación de una tarea. Este algoritmo para calcular la interferencia de las tareas de mayor prioridad empieza a partir del plazo relativo y busca periodos de tiempo con interferencia (*busy periods*) y periodos de tiempo disponibles para la ejecución de la tarea (*idle periods*). Para ello se asume la ejecución límite, es decir que las tareas se retardarán al máximo i finalizarán justo en su plazo.

La implementación de la variante de DP que utilice estos valores tendrá un mayor coste en términos de memoria, puesto que deberá almacenar los vectores de promociones. Este efecto será más notable cuando los hiperperiodos de las tareas periódicas son grandes, es decir, en aplicaciones con periodos no armónicos. A pesar de ello, puede utilizarse como punto de referencia en las evaluaciones de otros

algoritmos. En particular, es de especial interés su aplicación en conjuntos de tareas con periodos armónicos, puesto que se ha comprobado que son los peores casos para DP básico y, además, son los que requerirán vectores de menor longitud. El resultado final serán unas promociones más tardías, que permitirían mayor flexibilidad en la atención a las tareas aperiódicas y, consecuentemente, un mejor rendimiento.

```
Algoritmo calcular_promoción_para_una_activacion(τi,k)
retorna booleano

1:  tiempo_activacion= k*Ti
2:  plazo_absoluto= tiempo_activacion + Di
3:  inicio_con_promos= plazo_absoluto
4:  run= 0
5:  repetir
6:    repetir
7:   ;ir hacia atrás mientras haya interferencias: periodo con promociones
8:      inicio= inicio_con_promos;
9:      para j=i-1 hasta j=0 decrementando j en 1
10:       kj= ⌊inicio_con_promos/Tj⌋
11:       si (kj*Tj ≥ inicio_con_promos)  ;activada después: la anterior
12:          si (kj>0)
13:               kj= kj-1
14:          sino iterar                  ; no ∃ anterior
15:       promo_j_anterior= kj*Tj + Dj - promociones[j,kj]
16:       si (promo_j_anterior < inicio_con_promos) y
17:          (promo_j_anterior+promociones[j,kj] ≥ inicio_con_promos)
18:          ; empieza antes y termina después: interfiere
19:            inicio_con_promos= promo_j_anterior
20:    finpara
21:   mientras (inicio_con_promos < inicio);
22:   ; situados justo al final del IDLE PERIOD
23:
```

```
24:    inicio_sin_promos= 0;
25:    para j=0 hasta j=i-1 incrementando j en 1
26:      ; buscar la que antes más cercana
27:       kj= ⌊inicio_con_promos/Tj⌋
28:       si (kj*Tj + Dj > inicio_con_promos)
29:          ; finaliza después: la anterior
30:          si (kj>0)
31:               kj= kj-1
32:          sino    iterar         ; no ∃ anterior
33:       si (kj*Tj + Dj > inicio_sin_promos)   ; la más cercana
34:          inicio_sin_promos= kj*Tj + Dj
35:    finpara
36:    ; situados en el BUSY PERIOD con tareas ∈ HP(i) promocionadas
37:
38:    si (inicio_con_promos - inicio_sin_promos > Ci -run)
39:      ; el IDLE PERIOD excede a las necesidades
40:      inicio_sin_promos= inicio_con_promos - (Ci -run)
41:    run= run + inicio_con_promos - inicio_sin_promos
42:    inicio_con_promos= inicio_sin_promos;
43:  mientras (run<Ci) y (inicio_con_promos ≥ tiempo_activacion)
44:
45:  si (inicio_con_promos < tiempo_activacion)
46:    retorna Test no pasado
47:  sino
48:    promociones[i,k]= plazo_absoluto - inicio_con_promos
49:  retorna Test pasado
```

Algoritmo A-1 calcular la promoción más tardía posible para la k-ésima activación de la tarea periódica τ_i

B. Anexo B

PARÁMETROS POR DEFECTO DE LAS TAREAS PERIÓDICAS

Parámetro	Valor
Número de procesadores	4
Tareas periódicas/procesador	8
Periodo/Plazo mínimo	100= $2^2 5^2$
Periodo/Plazo máximo	1600= $2^6 5^2$
Hiperperiodo máximo	16000= $2^7 5^3$

Tabla 15: PCT # 0: Parámetros por defecto para los conjuntos de tareas generados sintéticamente

Parámetro	Valor
Periodo/Plazo mínimo	100= $2^2 5^2$
Periodo/Plazo máximo	1500= $2^2 3^1 5^3$
Hiperperiodo máximo	63000= $2^3 3^2 5^3 7^1$

PCT # 1

Parámetro	Valor
Periodo/Plazo mínimo	100= $2^2 5^2$
Periodo/Plazo máximo	1000= $2^3 5^3$
Hiperperiodo máximo	378000= $2^4 3^3 5^3 7^1$ = 6 * 63000

PCT # 2

Parámetro	Valor
Periodo/Plazo mínimo	100= $2^2 5^2$
Periodo/Plazo máximo	3000= $2^3 3^1 5^3$
Hiperperiodo máximo	378000= $2^4 3^3 5^3 7^1$

PCT # 3

Parámetro	Valor
Periodo/Plazo mínimo	128= 2^7
Periodo/Plazo máximo	3125= 5^5
Hiperperiodo máximo	400000= $2^7 5^5$

PCT # 4

Parámetro	Valor
Periodo/Plazo mínimo	128= 2^7
Periodo/Plazo máximo	4096= 2^{12}
Hiperperiodo máximo	4096= 2^{12}

PCT # 5

Parámetro	Valor
Periodos/Plazos	100,200,300,..,1500,1600
Hiperperiodo Máximo	72072000= $2^6 3^2 5^3 7^1 11^1$

PCT # 6

C. Anexo C

SIGLAS

DP	Planificador *Dual Priority*
GRQ	Cola de tareas preparadas global (*Global Ready Queue*)
GARQ	Cola de tareas aperiódicas preparadas global (*Global Aperiodic R. Q.*)
LRQ	Cola de tareas preparadas local (*Local Ready Queue*)
LHPRQ	Cola de tareas preparadas local de alta prioridad (*Local High Priority R. Q.*)
GRM	Planificador global con *RateMonotonic*
GS	Planificador global (*Global Scheduler*)
LPL	Nivel de prioridades bajas (*Low Priority Level*)
HPL	Nivel de prioridades altas (*High Priority Level*)
LS	Planificador local (*Local Scheduler*)
PAA	Porcentaje de aceptación de tareas aperiódicas con plazo. Se calcula dividiendo el número de tareas aperiódicas aceptadas entre el total de las requeridas
PF/AF	distribución de tareas periódicas fijas y aperiódicas fijas
PF/AD	distribución de tareas periódicas fijas y aperiódicas dinámicas
PD/AF	distribución de tareas periódicas dinámicas y aperiódicas fijas
PD/AD	distribución de tareas periódicas dinámicas y aperiódicas dinámicas
SS	Planificador *Slack Stealing*
TB	Planificador *Total Bandwidth*
TRA	Tiempo de respuesta de las tareas aperiódicas. Es la media aritmética del TRMAN obtenido en simulaciones de diversos conjuntos de tareas
TRAN	Tiempo de respuesta de las tareas aperiódicas normalizado. Es la diferencia entre el tiempo de finalización de una tarea y el tiempo de su activación, dividido por el tiempo consumido en su ejecución
TRMAN	Tiempo de respuesta medio de las tareas aperiódicas normalizado. Es la media aritmética del TRAN obtenido en una simulación de un conjunto de tareas

WCET Peor tiempo ejecución del código de una tarea en una arquitectura concreta (*Worst Case Response Time*)

WCRT El peor tiempo de respuesta obtenido por cualquiera de las activaciones de una tarea (*Worst Case Response Time*)