UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS,
COMPUTADORES Y AUTOMÁTICA

Planificación con Prioridades Dinámicas en Sistemas de Tiempo Real Crítico

TESIS DOCTORAL PRESENTADA POR:

José Ismael Ripoll Ripoll

DIRIGIDA POR:

Alfons Crespo i Lorente

Valencia, junio de 1996

Título:
“Planificación con Prioridades Dinámicas en Sistemas de Tiempo Real Crítico”

Autor:
José Ismael Ripoll Ripoll

Tribunal:
Presidente:
Vocales:

Acuerdan otorgar la calificación de:

Resumen

En esta tesis se estudian los métodos de planificación de tiempo real basados en prioridades dinámicas y se aportan soluciones a algunas de las cuestiones abiertas.

La principal propiedad que ha de poseer cualquier política de planificación de tiempo real es la ***predecibilidad****. Se ha de disponer de herramientas teóricas que permitan conocer a priori si el sistema podrá, en todo momento, garantizar la correcta ejecución de todas las tareas críticas. De ahí la necesidad de un eficiente test de planificabilidad. Los planificadores basados en prioridades dinámicas, a pesar de ser capaces de garantizar un mayor número de tareas que los de prioridades estáticas, no disponen de un test de planificabilidad eficiente. En esta línea se han propuesto dos nuevas caracterizaciones de conjuntos de tareas planificables, a partir de las cuales se ha propuesto un nuevo y más eficiente test de planificabilidad.*

Típicamente, los sistemas de tiempo real están compuestos tanto por tareas periódicas, las cuales suelen asociarse con las actividades críticas, como por tareas aperiódicas. A pesar de no haber ninguna urgencia en la ejecución de las tareas aperiódicas, es deseable que éstas completen su ejecución lo antes posible, claro está, sin poner en peligro los plazos de las tareas periódicas. Se han propuesto dos algoritmos para servir tareas aperiódicas basados en el concepto de ***holgura*** *(cantidad de tiempo de cómputo que se puede substraer del trabajo periódico). El primero de ellos se apoya en una tabla precalculada de holgura para aceptar las peticiones aperiódicas. El segundo, a cambio de reducir la complejidad espacial, tiene mayor coste temporal pues realiza todos los cálculos dinámicamente. Ambos algoritmos son óptimos: ofrecen, a las tareas aperiódicas, el menor tiempo de respuesta posible.*

Índice General

Abreviaturas

T_i Tarea periódica o esporádica convertida a periódica, $T_i = (C_i, D_i, P_i)$.

C_i Tiempo de cómputo en el peor de los casos de la tarea T_i.

D_i Plazo de ejecución relativo máximo, la cantidad de tiempo máximo entre la activación de la tarea y su finalización.

P_i Intervalo entre activaciones. En caso de ser una tarea esporádica representa el tiempo mínimo entre dos activaciones consecutivas.

τ Conjunto de tareas periódicas y esporádicas. En el capítulo 4.

n Número de tareas periódicas, cardinal de τ.

A_x x-ésima activación (petición) producida por tareas periódicas, $A_x = (s_x, c_x, d_x)$.

c_x Tiempo de cómputo, en el peor de los casos, de la activación A_x.

d_x Plazo de ejecución absoluto máximo, la activación A_x no ha de finalizar su ejecución despúes de d_x.

s_x Instante de llegada de la activación A_x.

$\mathcal{P}$ Hiperperiodo del conjunto de tareas τ.

$\mathcal{U}_\tau$ Factor de utilización del procesador producido por τ. Se obtiene como: $\sum_{i=1}^{n} \frac{C_i}{P_i}$.

N Número de activaciones de tareas periódicas durante un hiperperiodo.

θ Conjunto de tareas que representan la tabla de holgura (Capítulo 4, página 48).

J_a Tarea aperiódica (con o sin plazo), $J_a = (C_a, D_a)$.

m Número de tareas periódicas activas.

$\mathcal{K}$ Duración del intervalo ocioso (Capítulo 5, página 72).

$\mathcal{L}$ Duración del intervalo de holgura (Capítulo 5, página 75).

$n_{i,t_{now}}$ Instante de inicio de la activación más próxima de la tarea T_i posterior al instante t_{now}.

$c_{i,t_{now}}$ Tiempo de cómputo que resta por consumir de la última activación, suponiendo que el instante actual es t_{now}, de la tarea T_i.

$d_{i,t_{now}}$ Plazo de ejecución, relativo a t_{now}, de la activación actual de la tarea T_i.

e Cantidad de tiempo sobrante disponible en un momento dado (Página 60).

$(x)_0$ Expresión que representa el máximo entre 0 y x, $\max\{0, x\}$.

f Número de tareas aperiódicas con plazo firme activas.

Capítulo 1

Planteamiento y objetivos de la tesis

1.1 Ámbito de la tesis

Los Sistemas de Tiempo-Real (STR) se han convertido en una pieza fundamental en el diseño de multitud de aplicaciones, entre las que se incluyen: telecomunicaciones, control de sistemas, aviónica, guiado de vehículos, sistemas médicos, multimedia, etc.. Una característica común a todas ellas es que su correcto funcionamiento no sólo depende de la corrección lógica, sino también del instante en el que los resultados son obtenidos. Las restricciones temporales vienen impuestas por la propia dinámica del sistema que se controla. El incumplimiento de estos requisitos temporales puede provocar un fallo, el cual, en función del tipo de aplicación, puede desembocar en resultados catastróficos.

La alta complejidad de los sistemas reales hace inviable una solución monolítica, en donde una única tarea es la encargada de dar servicio a todo el sistema. Por ello, los STR suelen estar compuestos por un conjunto de *tareas* que se ejecutan concurrentemente, y donde cada tarea tiene definidas ciertas restricciones temporales. Esta descomposición del sistema global en tareas trae aparejado el problema de contención, o lucha por los recursos, por parte de las distintas tareas. El procesador es un claro ejemplo de recurso compartido que tiene que ser *planificado*.

Varias son las políticas de planificación que se pueden emplear para ordenar la ejecución de las tareas. En una primera clasificación podemos hablar de dos grandes categorías: (1) planificadores estáticos y (2) planificadores basados en prioridades. Dentro de la segunda categoría la asignación de prioridades se puede realizar de dos formas distintas: (a) asignación fija y (b) asignación dinámica (en lo que sigue también los referenciaremos como *planificadores dinámicos*). En el capítulo 2 se hace un rápido repaso de los logros alcanzados por cada una de estas políticas.

En la mayoría de los sistemas los parámetros de algunas tareas son conocidos a priori (durante la fase de diseño del sistema), dado que éstas tienen un comportamiento *periódico*, son las llamadas *tareas periódicas*. Tareas con este tipo de comportamiento han de ser las encargadas de mantener el funcionamiento básico y continuado del sistema. Diremos que una tarea es *crítica* si el retraso en la obtención del resultado puede producir una situación de peligro. De aquí en adelante supondremos que todas las tareas críticas son periódicas.

En los STR críticos, sistemas con al menos una tarea crítica, la política de planificación ha de *garantizar* la correcta ejecución de todas las tareas críticas durante todo el tiempo que el sistema esté activo. Por consiguiente, es necesario disponer de herramientas de análisis que nos permitan conocer a priori si el sistema podrá ser planificador correctamente. Este tipo de análisis se denomina *test de planificabilidad*. El test de planificabilidad ha de ser tal que su satisfacción sea condición *suficiente* para que las tareas críticas cumplan sus plazos. Esto descarta los planificadores del tipo optimista y probabilista. También es deseable que la satisfacción del test sea condición *necesaria* para la planificación de las tareas críticas.

Además de las tareas periódicas, los STR han de dar servicio a peticiones que tienen un comportamiento *aperiódico* (no se conocen a priori los instantes en los que estas tareas tendrán que ejecutarse), tales como ordenes de control u operaciones de recuperación. Sobre este tipo de tareas no se puede realizar ningún tipo de análisis previo de planificabilidad dada la escasa información que de ellas se dispone, y por consiguiente, ninguna función crítica puede estar a cargo de tareas de este tipo. Está clase de tareas a pesar de no necesitar cumplir ningún plazo de ejecución, es deseable que el planificador las ejecute lo antes posible.

Si bien la política de planificación más prometedora en cuanto a prestaciones es la basada en prioridades dinámicas, también es cierto que es la menos desarrollada tanto a nivel teórico como aplicado. En la actualidad, esta situación de abandono tiende a desaparecer y son muchas las publicaciones sobre el tema que últimamente han aparecido [Baruah90b, Tia95, Ghazalie95, Spuri96, Delacroix96].

1.2 Objetivos de la tesis y contribuciones

La presente tesis se centra en el estudio y ampliación de la teoría de la planificación por prioridades dinámicas, con la convicción que este tipo de políticas se convertirán en una buena elección de diseño cuando finalmente se alcance el requerido grado de madurez.

En concreto, los objetivos principales de esta tesis han sido:

- Revisión de los test de planificabilidad para planificadores dinámicos.
- Extensión de los métodos de análisis empleados en planificadores basados en prioridades fijas a los basados en prioridades dinámicas.
- Estudio de la planificación conjunta de tareas periódicas y aperiódicas.

Las principales contribuciones de la presente tesis son:

- Extensión del concepto de instante crítico, ampliamente utilizado en sistemas con prioridades fijas, al ámbito de los planificadores por prioridades dinámicas.
- Se ha propuesto una mejora sobre el test de planificabilidad más eficiente existente hasta la fecha [Baruah90a].

- Paralelamente al anterior test, se ha propuesto un nuevo test de planificabilidad basado en el concepto de instante crítico.
- La combinación de los dos anteriores representa en la actualidad el test de planificabilidad más eficiente, de entre los conocidos por los autores.
- Se han desarrollado las herramientas teóricas necesarias para la el cálculo de la holgura[1].
- Se han propuesto dos algoritmos de aceptación de tareas aperiódicas, basados en la extracción de holgura. El primero de ellos se apoya en una tabla, que contiene los huecos de holgura, para aceptar tareas, mientras que el segundo calcula el tiempo de holgura disponible en tiempo de ejecución.
- Se han propuesto varias ampliaciones a los dos algoritmos básicos: aprovechar el tiempo sobrante, tiempo asignado a tareas periódicas, que no ha sido consumido por éstas, aceptar o rechazar tareas aperiódicas con plazo y precalcular/actualizar la holgura durante los tiempos ociosos del procesador.

Sin ánimo de ser exhaustivos, en la siguiente figura se trata de enmarcar las distintas aportaciones presentadas en esta tesis.

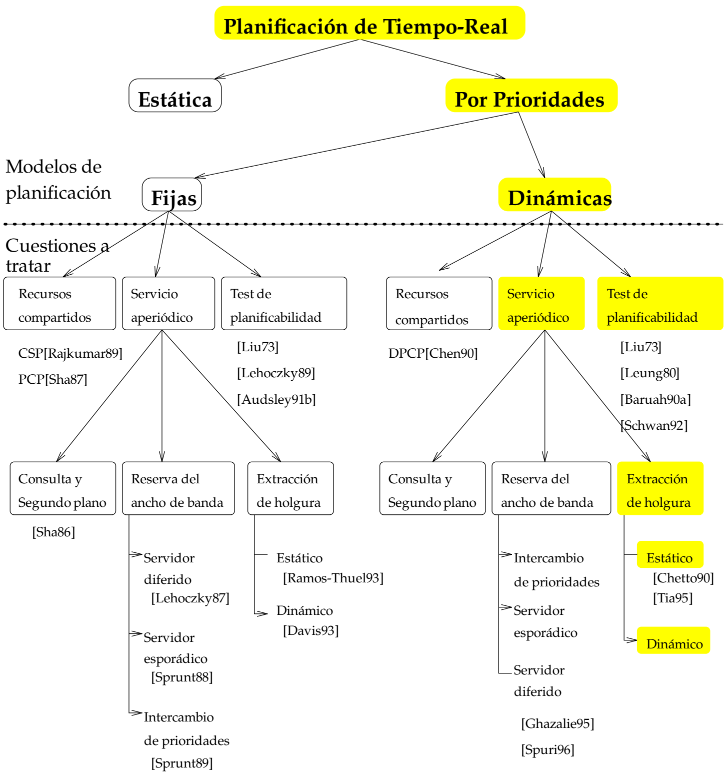


Figura 1.1: Clasificación de las políticas de panificación de tiempo real.

[1] slack

1.3 Organización de la tesis

La presente tesis está estructurada en seis capítulos y dos apéndices:

En el capítulo 2 se introducen los sistemas de tiempo real, el modelo tareas y la terminología general empleada en esta tesis. Seguidamente, tras identificar las características requeridas por toda política de planificación, se realiza un resumen de los logros más desatacados de las distintas políticas de planificación.

En el capítulo 3 se estudian los distintos tests de planificabilidad para sistemas basados en prioridades dinámicas, tras lo cual se proponen dos nuevos tests (necesarios y suficientes) . Un detallado análisis comparativo de las prestaciones de los distintos tests aparece al final del capítulo.

En el capítulo 4 se trata el problema de la planificación conjunta de tareas periódicas y aperiódicas. El método propuesto se basa en el cálculo previo de la holgura y su posterior uso en tiempo de ejecución para dar servicio a las tareas aperiódicas. El algoritmo propuesto es óptimo.

El capítulo 5 continua con la problemática tratada en el capítulo anterior, pero abordado desde un punto de vista dinámico. El cálculo de la holgura se realiza en tiempo de ejecución, por lo que el coste espacial es mucho menor a costa de elevar el coste temporal.

El capítulo 6 contiene las conclusiones más importantes del trapajo realizado y esboza las posibles líneas de trabajo futuro.

El primer apéndice contiene la demostración de un teorema que aparece en el capítulo 4. En el segundo apéndice se describen los símbolos empleados en los distintos cronogramas que aparecen a lo largo de la tesis.

Capítulo 2

Antecedentes

2.1 Introducción

Un sistema informático se puede definir como un *sistema de tiempo real* si la corrección depende tanto de la validez lógica de la respuesta, como del instante de tiempo en el que ésta se produce.

Los principales campos de aplicación de los sistema de tiempo real son los sistemas de control.Típicamente un sistema de control está compuesto por un conjunto de sensores, una o varias unidades de proceso (procesadores) y finalmente otro conjunto de actuadores. El software que se ejecuta en los procesadores monitoriza, a través de los sensores, el estado del sistema que ha de controlar, evalúa las posibles acciones de control y eventualmente envía a los actuadores las señales de control oportunas. El sistema informático (sistema operativo y programas que realizan el trabajo) que da soporte a todo el sistema ha de cumplir con una serie de requisitos temporales impuestos tanto por la propia dinámica del sistema físico que controlan como por el tipo de algoritmo de control que se emplee.

En función de las consecuencias que se producen cuando una tarea no es capaz de obtener el resultado en el momento que se necesitaba, podemos hablar de varios tipos de sistemas de tiempo real. Los *sistemas de tiempo real crítico*[1] son aquellos en los que un fallo (bien lógico o temporal) puede causar resultados catastróficos (esto es, los daños producidos son mucho mayores que los beneficios que se obtienen por el servicio que ofrece el sistema durante el funcionamiento correcto) [Burns91]. Entre otras, podemos citar las siguientes aplicaciones: sistemas de producción industrial, control de radares y otros sistemas de monitorización, guiado de vehículos, control del tráfico aéreo, sistemas médicos, etc..

En el otro extremo de los sistemas de tiempo real se sitúan aquellos sistemas que si bien tienen restricciones temporales, la perdida o retraso en la obtención de algún resultado no pone en peligro la integridad del sistema ni de las personas que lo utilizan[2]. Incluidos en esta categoría están los sistemas multimedia.

[1] Hard real time systems
[2] Soft real time systems

Una importante cuestión a resaltar es la diferencia entre “sistema *en* tiempo real” y “sistema *de* tiempo real”. Los sistemas en tiempo real son sistemas en los que el factor más importante es la velocidad de respuesta para conseguir un “funcionamiento realista”. Por ejemplo, podríamos mencionar los sistemas de animación por computador, o los sistemas multimedia. En estos sistemas el retraso en el envío de una imagen puede quizás provocar algún parpadeo en el monitor, pero no una situación peligrosa. Por contra, la característica más importante de los sistemas de tiempo real es la garantía, y no la rapidez. Por garantía se entiende que se ha de poder predecir y asegurar que el sistema será capaz de ejecutar correctamente todos los trabajos encomendados [Stankovic88a].

2.2 Modelo de sistema

Típicamente , un sistema de tiempo real está dividido en varias tareas que colaboran entre si. Una tarea puede trabajar de forma aislada, donde ella sola se encarga del control de alguno de los subsistemas a controlar, o bien en colaboración con otras tareas para resolver problemas más complejos. En este segundo caso serán necesarios mecanismos para la gestión de la comunicaciones.

El sistema que se toma como base está formado por un único procesador, encargado de ejecutar todas las tareas que dan soporte al sistema. También es el propio procesador el que ha de administrar los recursos del sistema (comunicación entre tareas, control de la violación de los plazos, planificación de las tareas, aceptación de tareas aperiódicas, etc.).

Fundamentalmente, podemos realizar dos clasificaciones de las tareas de tiempo real en función del criterio que empleemos: (1) considerando la importancia del trabajo que realiza cada tarea en el mantenimiento de la integridad del sistema; (2) atendiendo a sus características temporales.

Atendiendo a la importancia del problema que resuelven tenemos la siguiente clasificación:

Críticas Un fallo en una de estas tareas puede conducir el sistema a una situación catastrófica. El fallo puede ser debido a un error en el código del programa, o a un retraso en la obtención del resultado.

Acríticas Tareas que colaboran en el funcionamiento del sistema sin comprometer la integridad de éste, en caso de fallar o incluso no ejecutarse. Tareas de este tipo podrían ser las encargadas de llevar el registro de los sucesos del sistema, o la visualización de variables no críticas, etc.

Dependiendo de las características temporales de las tareas, aparece la siguiente taxonomía:

Periódicas Tareas que se activan (ejecutan) repetidamente con una cadencia fija. Cada activación ha de finalizar antes de un determinado plazo máximo de respuesta. Con estas dos características (periodo y plazo máximo) se pueden implementar fácilmente algoritmos de control de procesos continuos, en los que cada cierto tiempo se ha de muestrear el estado del sistema, periodo, y actuar sobre el sistema antes que derive el plazo máximo. Tres son los parámetros que caracterizan estas tareas, $T_i = (C_i, D_i, P_i)$ [Liu73, Chetto89]:

- C_i: Tiempo de cómputo, en el caso más desfavorable, requerido en cada activación.
- D_i: Plazo relativo. Tiempo máximo del que dispone la activación para completar su ejecución. Si por cualquier causa el resultado de la ejecución no está disponible D_i unidades de tiempo (u.t.) después de iniciarse una activación diremos que se ha producido un **fallo**.
- P_i: Periodo de activación. Distancia temporal entre dos peticiones consecutivas.

En los primeros trabajos de tiempo real [Liu73] se asumía que el plazo coincidía con el periodo. Actualmente se tiende a eliminar esta restricción permitiendo que $D_i \leq P_i$, e incluso en algunos estudios podemos encontrar tareas con plazos mayores al periodo [Baruah90b]. En el presente trabajo asumiremos que $D_i \leq P_i$.

En algunos sistemas no todas las tareas han de comenzar su ejecución tan pronto se inicia el sistema, sino que algunas tareas tienen que comenzar su ejecución con un cierto *desfase*, S_i, desde el arranque del sistema. Diremos que un conjunto de tareas es *síncrono* si el desfase de todas las tareas periódicas es igual a cero.

Aperiódicas Estas tareas, J_i, han de atender a eventos que aparecen de forma impredecible, como por ejemplo una orden dada por un operador, o la aparición de un obstáculo (por ejemplo, en un sistema de guiado de vehículos), etc.

En función de la urgencia podemos, a su vez, subdividirlas en los siguientes grupos:

Sin plazo Su ejecución no es crítica. No tienen ningún plazo limite para su finalización. Pueden emplearse para mejorar la precisión de las acciones de control, o generar informes del estado del sistema, etc. Si bien no tienen plazos limite de finalización, es deseable que el planificador las ejecute lo antes posible para conseguir un corto tiempo de respuesta. El único parámetro que las caracteriza es el tiempo de cómputo, $J_i = (C_i)$.

Con plazo firme Al igual que las tareas aperiódicas sin plazo, éstas tampoco son críticas. El resultado de su ejecución sólo tiene utilidad para el sistema si finaliza antes de que expire su plazo. En caso de que no se pueda cumplir el plazo, el planificador debe cancelar lo antes posible su ejecución, con el fin de no "malgastar" tiempo de procesador. El tiempo de cómputo y el plazo son sus atributos, $J_i = (C_i, D_i)$.

Esporádicas Este tipo de tareas poseen un plazo máximo de ejecución, y su incumplimiento puede resultar catastrófico para el sistema [Mok83]. Por ello, el planificador debe garantizar su correcta ejecución. Los parámetros de esta clase de tareas son los mismos que los de las tareas periódicas, con la salvedad que el periodo en lugar de indicar el tiempo entre activaciones representa la distancia mínima entre dos activaciones consecutivas, $T_i = (C_i, D_i, P_i)$.

En todos los trabajos [Liu73, Leung80, Mok84, Stankovic88b, Xu93] para garantizar la correcta ejecución, se basan en el conocimiento explícito de los periodos de las tareas o al menos de el tiempo mínimo entre peticiones. Por consiguiente, aquellas tareas consideradas críticas han de ser transformadas en periódicas, en caso de no serlo. En concreto, las tareas esporádicas se han de considerar como si fueran periódicas durante la fase de diseño y análisis del sistema.

Asumiremos en la presente tesis que tanto las características temporales de las tareas (tiempo de cómputo, plazo y periodo) como el tiempo (t) están representados por variables enteras. Este convenio no supone ningún tipo de restricción en la aplicación de la teoría a casos prácticos, y sí que simplifica en gran medida la comprensión de los resultados presentados. En cualquier caso, el extender los resultados para valores reales del tiempo no entraña cambios fundamentales en las demostraciones presentadas. Una detallada discusión se puede encontrar en [Baruah90a].

Aparte de las características temporales de las tareas, en los primeros estudios de tiempo real se asumían la siguiente serie de restricciones, si bien conforme ha ido evolucionando y madurando la teoría se han ido eliminando restricciones:

- Todas las tareas periódicas son críticas.
- Todas las tareas pueden ser expulsadas del procesador en cualquier instante.
- Todas las tareas pueden comenzar su ejecución tan pronto están activas.
- $C_i \leq D_i \leq P_i$.
- La sobrecarga debida al cambio de contexto se supone despreciable.
- No existe ninguna dependencia entre tareas: ni comparten recursos comunes ni hay relaciones de precedencia entre ellas. Un conjunto de tareas con relaciones de precedencia puede ser transformado en otro nuevo sin problemas de precedencia mediante la modificación de los plazos [Chetto90, Henn78].
- Ninguna tarea se suspende voluntariamente, a excepción de las expulsiones causadas por el algoritmo de planificación.

Llamaremos A_x a cada una de las activaciones que una tarea periódica o esporádica realiza. La activación k-ésima de una tarea T_i se produce en el instante $(k-1) \cdot P_i$. Para ejecutar correctamente la activación, el planificador deberá asignarle C_i u.t. a esta tarea en el intervalo $[(k-1) \cdot P_i\,, (k-1) \cdot P_i + D_i)$. Una activación está definida por la terna (s_x, c_x, d_x), donde $s_x = (k-1) \cdot P_i$ es el instante de inicio, $c_x = C_i$ el tiempo de cómputo, y $d_x = (k-1) \cdot P_j + D_j$ el plazo (absoluto) de finalización. Las activaciones se supondrán ordenadas por el instante de finalización d_x, de modo que $x < y \rightarrow d_x \leq d_y$ (en la página 19 se justifica el uso de este criterio de ordenación). Obsérvese que se utilizan letras minúsculas para describir activaciones, mientras que las letras mayúsculas representan los parámetros de las tareas.

Representaremos por τ al conjunto de tareas periódicas (y esporádicas) que componen la carga del procesador. Asumiremos que n es el cardinal del conjunto τ.

El parámetro más importante de la carga periódica es el *factor de utilización* del procesador, $\mathcal{U}_\tau$. Se define como:

$$\mathcal{U}_\tau = \sum_{i=1}^{n} \frac{C_i}{P_i}$$

El *hiperperiodo*, $\mathcal{P}$, es otra característica importante del conjunto de tareas. Se define como el mínimo común múltiplo del periodo de todas las tareas de τ, $\mathcal{P} = mcm(P_1, P_2, \cdots, P_n)$.

Un conjunto de tareas, τ, es **planificable** (*feasible*)si y sólo si existe algún algoritmo de planificación capaz de cumplir los plazos de todas la activaciones de τ. Obsérvese que la condición de planificabilidad es una propiedad del conjunto de tareas, y no del planificador. Un conjunto de tareas, τ, es planificable por un planificador dado si y sólo si éste es capaz de cumplir los plazos de todas la activaciones de τ.

Partiendo de estas definiciones queda claro que pueden existir conjuntos de tareas que sean planificables pero que un determinado algoritmo de planificación sea incapaz de planificarlas correctamente. Se dice que un planificador es *óptimo* si y sólo si puede planificar correctamente todo conjunto de tareas planificable.

2.3 Requisitos de planificación

En los sistemas informáticos clásicos, el objetivo principal del planificador suele ser minimizar el tiempo de respuesta de las tareas Shortert Job First (SJF), o repartir lo más equitativamente posible el tiempo de procesador entre las distintas tareas Round Robin (RR), o evitar realizar cambios de contexto con el First Come First Serverd (FCFS). Ninguna de estas políticas es apropiada en sistemas de tiempo real [Stankovic88b, Stankovic95]. El principal objetivo de cualquier política de planificación de tiempo real ha de ser el asegurar que todas las tareas cumplan sus restricciones temporales. Además de este objetivo básico, es deseable que la política de planificación considere las siguientes cuestiones:

- Gestión los recursos cuando son compartidos entre distintas tareas.
- Recuperación de fallos.
- Minimización del tiempo de respuesta de todas las tareas.
- Ejecución de tareas acríticas sin plazo.
- Reemplazo de tareas en tiempo de ejecución (cambios de modo).
- Considerar la sobrecarga temporal del propio algoritmo de planificación, del cambio de contexto, etc.

- Planificación de tareas cuyo tiempo de computo no es conocido o predecible.
- Si es posible, conviene que sea sencillo y fácil de utilizar en realizaciones prácticas.

La única forma de poder garantizar que un determinado algoritmo de planificación va a poder planificar correctamente un determinado conjunto de tareas es mediante el análisis, previo a la puesta en marcha del sistema, mediante métodos analíticos, o de forma exhaustiva. Como se apunta en [Xu93] la simulación no es un método válido que garantice la correcta planificación: , "la simulación muestra la presencia de errores, no la ausencia de éstos" [Dijkstra72].

Por consiguiente, todo algoritmo de planificación de tiempo real ha de ir acompañado obligatoriamente de un test de planificabilidad (test de garantía). Evidentemente, cualquier extensión del algoritmo de planificación se ha de reflejar en el test de garantía.

2.4 Planificación estática

También conocida como *ejecutivo cíclico*[3] o *dirigido por tiempo*[4] [Kopetz89, Zamorano92, Xu93], es la clase de planificadores más empleados en la actualidad, debido posiblemente a haber sido los primeros en aparecer. La mayor parte del trabajo de planificación se realiza durante el diseño del sistema. Se construye una tabla (llamada también plan estático o calendario) indicando los instantes en los que cada tarea debe ponerse en ejecución y cuando ha de finalizar. Cuando el sistema está en ejecución lo único que ha de hacer el planificador es seguir repetidamente las instrucciones almacenadas en el plan. El tamaño de la tabla está determinado por el hiperperiodo del conjunto de tareas, lo que aveces resulta en un tamaño excesivo de ésta. La garantía de correcta ejecución de todas las tareas (análisis de planificabilidad) se hace implícitamente durante la construcción del propio plan.

Como principales ventajas de esta política podemos citar la predecibilidad y eficiencia en tiempo de ejecución. Mientras que entre los inconvenientes tenemos su ineficiencia para gestionar eventos aperiódicos, la poca flexibilidad para modificar las características de la carga (un pequeño cambio en alguna tareas implica el tener que reconstruir todo el plan), y el posible excesivo tamaño de la tabla.

A pesar de ello, Xu y Parnas [Xu93] hacen una férrea defensa de los planificadores cíclicos, y defienden que algunos tipos de problemas sólo pueden ser resueltos con este tipo de algoritmos.

2.5 Planificación por prioridades

En contraposición a los planificadores cíclicos tenemos los planificadores controlados por prioridades. En estos, se elimina la necesidad de un plan prefijado, dejando la toma

[3] ciclic scheduling
[4] time-driven

de decisiones en manos del planificador en ejecución.

El esquema básico de funcionamiento es el siguiente: en cada instante (típicamente cuando llega o finaliza alguna tarea), el planificador elige aquella tarea cuya *función de prioridad* tenga mayor valor de entre todas las tareas activas. Dependiendo de cual sea la función de prioridad tendremos: planificadores con *prioridades fijas* si a cada tarea se le asigna una prioridad fija inicialmente, y se emplea este valor como función de prioridad; o planificadores con *prioridades dinámicas* en caso de que la función de prioridad tenga como parámetros datos relativos a la carga que haya en el instante de su evaluación.

Estos métodos precisan de un test previo que garantice que la política concreta de planificación empleada será capaz de planificar correctamente todas las tareas críticas.

2.5.1 Prioridades fijas

Dentro del esquema de prioridades fijas, los dos métodos más extendidos son el de *prioridad a la tarea más frecuente*[5] (RM) y el de *prioridad a la tarea más urgente*[6] (DM). En los últimos años han recibido una gran atención por parte de la comunidad científica, alcanzado un alto grado de madurez y reconocimiento [Stankovic87, Lehoczky87, Lehoczky89, Sha90, Burns91, Audsley93], etc. En [Audsley95] se puede encontrar un estudio histórico de los avances en la planificación por prioridades fijas.

El RM fue introducido por Liu y Layland [Liu73], y consiste en asignar a las tareas de periodos cortos prioridades altas, esto es, la prioridad de cada tarea es inversamente proporcional a su periodo. Para poder aplicar esta política, el plazo de respuesta de cada tarea ha de coincidir con su periodo. La obtención del primer test de garantía se basó en la siguiente observación:

Definición 2.1 (Liu y Layland [Liu73])
Sea τ un conjunto de tareas periódicas. El tiempo de respuesta más largo para cualquier tarea ocurre cuando se activan todas las tareas al mismo tiempo. A este instante se le denomina instante crítico.

Si el conjunto de tareas es síncrono (todas las tareas tienen la misma fase), el instante crítico inicial es aquel que se produce a partir del instante cero.

El siguiente test representa una condición suficiente de planificabilidad:

Teorema 2.1 (Liu y Layland [Liu73])
El conjunto de tareas τ es planificable bajo el RM si:

$$\mathcal{U}_\tau \leq n(2^{1/n} - 1)$$

Posteriormente Lehoczky, Sha y Ding [Lehoczky89] propusieron el siguiente método exacto de análisis de planificabilidad. Está basado en el cálculo del peor tiempo de respuesta de cada tarea, condición que se da durante el instante crítico inicial.

[5] Rate Monotonic
[6] Deadline Monotonic

Teorema 2.2 (Lehoczky, Sha y Ding [Lehoczky89])
El conjunto de tareas τ es planificable bajo el RM si y sólo si:

$$\forall_{1\leq i\leq n} \quad \min_{t\leq 0\leq P_i} \sum_{\forall j\in hp(i)} \frac{C_i}{t}\left\lceil\frac{t}{P_j}\right\rceil \leq 1$$

Leung y Whitehead [Leung82] extendieron el RM para trabajar con tareas que tuvieran plazos menores que el periodo, asignando mayor prioridad a tareas con plazos más cortos, y demostrando que esta asignación de prioridades es óptima entre todas las asignaciones estáticas. Estamos hablando del DM, quedando el RM como una caso particular del DM. Lehoczky, Sha y Ding [Lehoczky89] ampliaron el análisis de planificabilidad para el DM, obteniendo como resultado un test muy similar al del RM:

Teorema 2.3 (Lehoczky, Sha y Ding [Lehoczky89])
El conjunto de tareas τ es planificable bajo el DM si y sólo si:

$$\forall_{1\leq i\leq n} \quad \min_{t\leq 0\leq D_i} \sum_{\forall j\in hp(i)} \frac{C_i}{t}\left\lceil\frac{t}{P_j}\right\rceil \leq 1$$

El test consiste en verificar que la primera activación de cada tarea es capaz de cumplir su plazo de ejecución teniendo en cuenta la cantidad de tiempo de cómputo pedido por tareas más prioritarias. Este test no sólo puede ser empleado con asignaciones de prioridades basadas en el DM, sino que es un test exacto para cualquier asignación de prioridades [Lehoczky90]. Resultados similares fueron obtenidos por Audsley et al. [Audsley91b].

La compartición de recursos entre distintas tareas también ha sido tenida en cuenta dentro de la teoría del DM. Algoritmos tales como el protocolo del techo de prioridad[7] [Sha87, Sha90] o el protocolo del techo del semáforo[8] [Baker90, Rajkumar89], por nombrar algunos, se pueden emplear para gestionar el acceso a secciones críticas. A la hora de analizar la planificabilidad del sistema, tan sólo se ha de incluir un termino que representa el factor de bloqueo en la desigualdad del test. El factor de bloqueo representa la cantidad de tiempo que una tarea está bloqueada a causa de tareas menos prioritarias que están dentro de secciones críticas.

Una interesante revisión sobre los distintos métodos de control de recursos se puede encontrar en [Audsley91a].

El reemplazo de tareas en tiempo de ejecución y los cambios de modo ha sido estudiados en [Alonso94, Sha89].

2.5.2 Servicio de tareas aperiódicas

Otro importante aspecto que no ha sido descuidado y que ha desencadenado una gran cantidad de resultados [Audsley95], es el servicio de tareas aperiódicas. Desde los ser-

[7] Priority Ceiling Protocol (PCP)
[8] Ceiling Semaphore Protocol (CSP)

vidores de segundo plano[9] o por consulta[10] [Sha86], con muy poca sobrecarga pero con pobres resultados; pasando por los algoritmos de mantenimiento del ancho de banda[11] , como el servidor diferido[12] [Lehoczky87], el intercambio de prioridades[13] [Sprunt88], o el servidor esporádico[14] propuesto por Sprunt et al. [Sprunt89], que ofrecen un mejor tiempo de respuesta a las tareas aperiódicas a costa de una mayor complejidad en la gestión del tiempo disponible; para finalizar con los algoritmos de extracción de holgura, que estresan al máximo las tareas periódicas (sin llegar a producir la perdida de ningún plazo) para conseguir dedicar la mayor cantidad de tiempo posible a servir tareas aperiódicas. Estas últimas propuestas son óptimas, esto es:

Teorema 2.4 ([Ramos-Thuel93, Davis93])
Para cualquier conjunto de tareas periódicas planificado por un planificador por prioridades fijas y cualquier secuencia de peticiones aperiódicas servidas en orden FIFO, los extractores de holgura minimizan el tiempo de respuesta de cada una de las tareas aperiódicas, de entre los algoritmos que garantizan el cumplimiento de todos los plazos de todas las tareas periódicas (exceptuando los algoritmos clarividentes).

Dos son las propuestas basadas en la extracción de holgura, una estática y otra dinámica. En la versión estática [Ramos-Thuel93]se dispone de una tabla en la que se tienen almacenados los instantes en los que se le puede "robar" tiempo de cómputo a las tareas periódicas. Cada vez que hay que servir una tarea aperiódica se consulta esta tabla para buscar los huecos de holgura disponibles, para asignárselos a la tarea. El espacio requerido para almacenar la tabla de consulta puede llegar a ser totalmente intolerable en una realización práctica, de hecho, tal como comentan los propios autores, la principal utilidad del algoritmo es la de servir de referencia para evaluar el rendimiento de otros algoritmos.

Davis, Tindell y Burns [Davis93] propusieron la versión dinámica. El cálculo de la holgura se realiza totalmente en tiempo de ejecución, sin emplear para ello ninguna tabla precalculada. Esta solución evita la necesidad de un gran espacio de memoria para almacenar la tabla, pero por contra, sufre una mayor sobrecarga en tiempo de ejecución, debida al mayor número de cálculos necesitados.

2.5.3 Prioridades dinámicas

Los dos algoritmos más significativos, dentro de esta clase, son el de prioridad a la tarea más urgente[15] y el de prioridad a la tarea con menor holgura[16]. El EDF asigna mayor prioridad a las activaciones con plazos más próximos. Obsérvese que a diferencia del DM, donde las prioridades se asignaban inicialmente (la prioridad es una función de D_i),

[9] background
[10] pooling
[11] bandwidth-preserving
[12] deferable server
[13] priority exchange
[14] sporadic server
[15] Earliest Deadline First (EDF)
[16] Least Laxity First (LLF)

el EDF evalúa los plazos absolutos de las activaciones y asigna prioridad en función de estos plazos (la prioridad es una función de d_i). El LLF asigna mayor prioridad a las activaciones con menor holgura, entendiendo holgura como la longitud del intervalo desde el instante actual hasta el plazo de la activación menos el tiempo de cómputo que le resta por ejecutar.

Ambos algoritmos son óptimos[17] [Dertouzos74, Mok78]. Por consiguiente, si alguna tarea pierde su plazo es seguro que ningún otro algoritmo habría sido capaz de evitarlo. Debido a esta característica, tanto el EDF como el LLF consiguen aprovechar al máximo la potencia del procesador, garantizando más tareas periódicas críticas que cualquier otro algoritmo. Esta clase de algoritmos de adaptan muy bien a entornos altamente dinámicos, en los cuales la carga del sistema puede no ser conocida inicialmente.

A pesar de estas interesantes propiedades, y a diferencia de la atención prestada a la planificación por prioridades fijas, esta clase de planificadores ha estado relegada, hasta hace algunos años, a un segundo plano dentro de la comunidad de tiempo real, debido principalmente a que presentaban los siguientes problemas:

1. La sobrecarga del planificador es mayor a la causada por planificadores estáticos.
2. No es estable ante sobrecargas. En caso de que alguna tarea consuma más tiempo del estimado, no hay forma de saber qué tarea perderá su plazo –si es que alguna lo pierde.
3. La mayoría de los sistemas operativos actuales disponen de un planificador basado en prioridades, por lo que sería necesario construir un planificador a medida.
4. La carencia de una bases teóricas adecuadas.

En la presente tesis se trata de dar respuesta a algunos de estos problemas, y conseguir acercar un poco más los planificadores dinámicos a las necesidades de los sistemas de tiempo real estricto.

Veamos ahora un brevísimo repaso de los trabajos más relevantes del área.

En su famoso artículo [Liu73], Liu y Layland también demostraron que el EDF es capaz de garantizar la correcta ejecución de todas las tareas cuando la utilización del procesador es menor del 100% (cuando el plazo era igual que el periodo). Por tanto, es óptimo para este tipo de tareas.

Dertouzos [Dertouzos74] demostró que el EDF es también óptimo para tareas con plazos menores al periodo.

Leung y Merril [Leung80] observaron que la planificación producida por cualquier planificador expulsivo se repite cíclicamente cada hiperperiodo. En otras palabras, el planificador realiza las mismas acciones en el instante t que en $t + k \cdot \mathcal{P}$, siendo k un número entero. Empleando esta importante propiedad propusieron el primer test de planificabilidad para tareas con plazos menores al periodo. Un conjunto de tareas periódicas es planificable si y sólo si ninguna tarea pierde ningún plazo durante el primer hiperperiodo.

[17] El EDF deja de ser óptimo en sistemas multiprocesador

Un test de planificabilidad más eficiente fue propuesto por Baruah et al. [Baruah90a]. Demostraron que para la mayor parte de conjuntos de tareas no es necesario comprobar la correcta ejecución de todas las tareas a lo largo de todo el hiperperiodo, sino que basta comprobarla durante un pequeño intervalo. También, en el mismo papel, se demuestra que el problema la planificabilidad para monoprocesador es co-**NP**-completo en en el sentido fuerte. Una característica que parecen poseer los problemas de este tipo es que pequeñas variaciones en la instancia del problema (en nuestro caso, podría ser modificar el periodo o el tiempo de cómputo de alguna tarea) resultan en importantes variaciones en la complejidad. Gracias a esta característica, el análisis de planificabilidad se ve notablemente simplificado en los casos reales.

Chen y Lin [Chen90] han estudiado el problema de la compartición de recursos para el algoritmo de prioridad al más urgente. El Protocolo del Techo de Prioridad Dinámico[18] evita los interbloqueos y los bloqueos encadenados.

Otro enfoque sobre el uso de recursos es el ofrecido por Mok [Mok83]. Las tareas que están ejecutando código dentro de alguna sección crítica no son expulsadas, pero a cambio el tiempo que permanecen dentro de ésta ha de ser despreciable.

2.5.4 Servicio de tareas aperiódicas

En lo relativo a la planificación conjunta de tareas periódicas y aperiódicas, los primeros trabajos fueron realizados por Chetto y Chetto [Chetto89]. Estudiaron la localización de los intervalos ociosos del procesador, y propusieron el algoritmo *earliest deadline last* (EDL). El EDL trata de planificar las tareas periódicas los más tarde posible (pero sin llegar a violar ningún plazo), con lo que los intervalos ociosos se "desplazan" hacia adelante, pudiendo ser empleados para dar servicio a trabajo aperiódico. Chetto, Silly y Bouchentouf [Chetto90] proponen un test óptimo de aceptación de grupos de tareas aperiódicas para ser planificadas bajo el EDF. El conjunto de tareas aperiódicas es aceptado si ninguna tarea pierde su plazo en el intervalo $[t_{now}, t_{now} + \mathcal{P} + D_a)$, donde t_{now} es el instante actual y D_a el plazo más largo de todas las tareas aperiódicas. El coste del algoritmo es alto, $O(N^2)$.

En 1990 Baruah, Mok y Rosier [Baruah90b] presentan un test de planificabilidad para tareas esporádicas, que puede ser empleado para aceptar tareas aperiódicas con plazo firme. La complejidad de este algoritmo es muy inferior al propuesto por Chetto et al.. En el peor caso el coste es $O(n^2 \max\{P_i - D_i\}/\min\{P_i\})$.

Schwan y Zhow [Schwan92] proponen un algoritmo de planificación dinámico basado en el EDF y con un coste $O(n\log(n)$. El problema de su propuesta radica en que todas las tareas, incluidas las tareas periódicas, deben pasar un test de aceptación. Las tareas que han sido rechazadas son tratadas por otro planificador de más alto nivel.

Los tres algoritmos de conservación de ancho de banda: *priority exchange*, *deferable server* y *sporadic server* han sido adaptados para trabajar con el EDF por Ghazalie y Baker

[18] Dynamic Priority Ceiling Protocol (DPCP)

[Ghazalie95] y posteriormente por Spuri y Buttazzo [Spuri96]. A pesar de no ser algoritmos óptimos, mantienen un buen compromiso entre coste y eficiencia.

Tia y Liu [Tia95] desarrollaron un servidor aperiódico basado en la extracción de holgura. Lo más llamativo de esta solución es su elevado coste espacial, $O(N^2)$. Otro problema es que en cada punto de planificación se han de actualizar ciertas variables con un coste de $O(n)$. Aunque el servidor es óptimo, el elevado coste espacial lo hace inutilizable en aplicaciones prácticas.

Capítulo 3

Análisis de planificabilidad

3.1 Introducción

El presente capítulo se centra en el análisis de la planificabilidad de conjuntos de tareas. Las principales aportaciones presentadas consisten en dos nuevas caracterizaciones de conjuntos de tareas planificables, teoremas 3.3 y 3.4 . La primera de ellas es una extensión de los resultados obtenidos por Baruah et al., mientras que la segunda caracterización está basada en la idea de instante crítico. La idea de *Instante Crítico* y las condiciones que se dan durante éste han sido ampliamente empleadas en el estudio de los sistemas de tiempo real, y más concretamente en el estudio de los sistemas basados en prioridades fijas [Liu73, Lehoczky89, Audsley91b].

A partir de las dos caracterizaciones de planificabilidad, se propone un nuevo test de planificabilidad. No nos consta que exista otro test de planificabilidad más eficiente. A pesar de ello, no hemos de olvidar que el problema de decidir si un conjunto de tareas es planificable o no pertenece a la clase de problemas co-**NP**-completos en el sentido fuerte [Baruah90a], por consiguiente, en el peor caso el coste del algoritmo continuará siendo exponencial.

Parte de las herramientas formales que aquí desarrollamos serán empleadas en los siguientes capítulos.

3.1.1 Modelo de sistema

El modelo básico de sistema es el descrito en el capítulo 2. Se dispone de un sólo procesador, y un conjunto de n tareas periódicas o esporádicas convertidas a periódicas. Las tareas son independientes y no se suspenden voluntariamente hasta finalizar su ejecución. Una tarea T_i se representa por la terna (C_i, D_i, P_i), donde C_i, D_i y P_i representan el tiempo máximo de ejecución, el plazo de respuesta máximo y el periodo de activación respectivamente. El plazo de respuesta puede ser menor que el periodo.

Inicialmente supondremos que el sistema es síncrono, al final del capítulo estudiamos los sistemas asíncronos.

Diremos que una activación *pendiente* es aquella a la que todavía no ha finalizado su ejecución. Diremos que el procesador está *ocioso* durante un intervalo de tiempo si no hay ninguna activación pendiente durante este intervalo. Es interesante destacar que el procesador puede estar ocioso durante intervalos de duración cero. Estos casos se dan cuando sólo hay una activación pendiente y la finalización de esta coincide con la llegada de una nueva activación.

Ejemplo de planificación: Supongamos un sistema con tres tareas $T_1 = (2, 3, 5)$, $T_2 = (1, 6, 7)$ y $T_3 = (3, 7, 10)$. El hiperperiodo es de 70 u.t.. El cronograma de la figura 3.1 muestra una posible planificación de este conjunto de tareas, empleando para ello la política de planificación EDF. Como vemos, todas las activaciones han podido cumplir su plazo límite, pero ¿ Se violará el plazo de alguna activación posterior al instante 55 ?

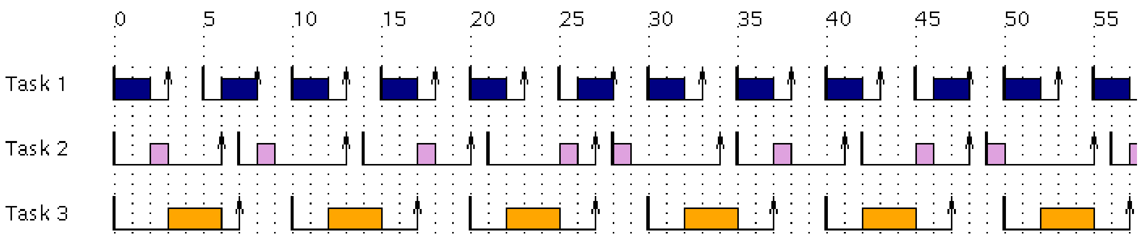


Figura 3.1: Ejemplo de planificación.

3.2 Análisis de planificabilidad

En esta sección estudiaremos distintas caracterizaciones de conjuntos de tareas planificables.

De la definición de planificador óptimo, se obtiene la siguiente equivalencia: determinar si un conjunto de tareas es planificable o no, es equivalente a decidir si ese conjunto de tareas es planificable bajo un planificador óptimo. A lo largo del capítulo nos apoyaremos el hecho de que el EDF sea un planificador óptimo [Dertouzos74] para ilustrar ciertas ideas.

Sea $shift(A_x, k)$ una función que desplaza la activación A_x y todas las siguientes activaciones de la misma tarea k unidades de tiempo. Si la operación de $shift$ se produce sobre la primera activación de una tarea, entonces el resultado es equivalente a modificar la fase de la tarea en k unidades de tiempo. Sin perdida de generalidad supondremos que k es siempre un número positivo. Aplicar el operador $shift$ con un valor negativo de k es similar a no aplicar ningún desplazamiento sobre la tarea T_i y aplicar el mismo desplazamiento pero con signo positivo sobre el resto de las tareas que componen el sistema.

Por otra lado vamos a necesitar las dos siguientes funciones:

Seq $G_\tau(t)$ la cantidad de tiempo de cómputo que ha sido pedido desde el instante inicial hasta el tiempo t, que puede calcularse como:

$$G_\tau(t) = \sum_{i=1}^{n} C_i \cdot \left\lceil \frac{t}{P_i} \right\rceil \tag{3.1}$$

Directamente a partir de las propiedades de las funciones enteras *suelo* y *techo*, se observa que esta es una función creciente escalonada, donde los escalones se producen para los valores de t múltiplo de alguno de los periodos.

H_τ es la cantidad de tiempo de cómputo que ha sido pedida por activaciones cuyo plazo es menor o igual a t, y puede calcularse como:

$$H_\tau(t) = \sum_{i=1}^{n} C_i \cdot \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor \tag{3.2}$$

Dado que P_i no es menor que D_i, entonces $(t + P_i - D_i)$ es siempre un número positivo, y por tanto, $H_\tau(t)$ es una función positiva.

De aquí en adelante supondremos que las activaciones están ordenadas en orden creciente de d_x, y denotaremos por A_x a la activación que ocupe la posición x-ésima según esta ordenación. De este modo (ver figura 3.2) la función H_τ se puede calcular de la siguiente forma:

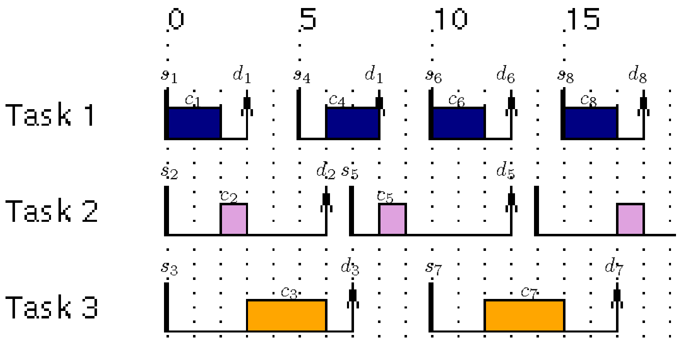


Figura 3.2: Ordenación de las activaciones.

$$\begin{aligned}
c_1 &= H_\tau(d_1)\\
c_2 + c_1 &= H_\tau(d_2)\\
c_3 + c_2 + c_1 &= H_\tau(d_3)\\
\vdots \quad &\vdots \quad \vdots\\
c_x + \cdots + c_2 + c_1 &= H_\tau(d_x)\\
\vdots \quad &\vdots \quad \vdots
\end{aligned}$$

Vamos a extender la definición de instante crítico dada para sistemas basados en prioridades fijas para tener en cuenta las particularidades de los sistemas dinámicos. En los sistemas con prioridades fijas el instante crítico de una tarea finalizaba con el plazo de ejecución de ésta. Por contra, en los sistemas dinámicos la activación que sufre un mayor estrés puede no ser la primera. Es por ello por lo que hay que definir el instante crítico no como un punto de tiempo, sino como un intervalo.

Definición 3.1 *Sea el Intervalo Crítico Inicial (ICI) de un conjunto síncrono de tareas, el intervalo de tiempo comprendido entre el instante inicial y el primer momento, $\mathcal{R}$, en el que no hay ninguna activación pendiente. Suponiendo que el procesador no queda ocioso mientras haya tareas pendientes.*

Sea $[0, \mathcal{R})$ *este intervalo.*

$\mathcal{R}$ puede calcularse mediante la siguiente fórmula de recurrencia:

$$\begin{aligned} K_0 &= 1 \\ K_{i+1} &= G_\tau(K_i) \end{aligned} \tag{3.3}$$

Si $K_{i+1} = K_i$ entonces la recurrencia termina y $\mathcal{R} = K_i$. Por ejemplo: la secuencia de valores de K_i para calcular el ICI del conjunto de tareas de la figura 3.2 es: $K_i = 1$, $K_2 = 6$, $K_3 = 8$, $K_4 = 9$ y $K_5 = 9$; por lo que el ICI es el intervalo $[0, 9)$.

Una formula de recurrencia similar es utilizada por Audsley et al. [Audsley93] para determinar el peor tiempo de respuesta de una tarea.

Nótese que la definición del ICI no está ligada a ningún planificador en especial, y que el intervalo ICI puede ser de mayor duración que el mayor plazo de todas las tareas.

3.2.1 Caracterización de sistemas planificables

Las caracterizaciones de los sistemas planificables las enunciaremos en forma de teoremas.

Teorema 3.1 (Leung y Merrill [Leung80])
τ es planificable si y sólo si

$$\forall t < \mathcal{P}, H_\tau(t) \leq t.$$

Teorema 3.2 (Baruah et al. [Baruah90a])
Sea

$$K = \frac{\mathcal{U}_\tau}{1 - \mathcal{U}_\tau} \max_{i=1}^{n}\{P_i - D_i\}.$$

τ es planificable si y sólo si

$$\forall t < K, H_\tau(t) \leq t.$$

Baruah et al. desarrollaron un test de planificabilidad basado en este teorema, así como un test de aceptación de tareas esporádicas [Baruah90b]. Fundamentalmente, su test consiste en simular el comportamiento del EDF hasta que o bien alguna tarea pierde su plazo, o bien se alcanza el instante K, en cuyo caso se concluye que el sistema es planificable.

El siguiente teorema es en esencia idéntico al propuesto por Baruah et al., en el sentido de que ofrece un límite a la longitud de la "simulación" para realizar el test. Este límite está también basado en algunas propiedades de la función H_τ.

Teorema 3.3 (Cota a H_τ mejorada)
Sea

$$L = \frac{\sum_{i=1}^{n} C_i \cdot (1 - \frac{D_i}{P_i})}{(1 - \mathcal{U}_\tau)}.$$

τ es planificable si y sólo si

$$\forall t < L, H_\tau(t) \leq t.$$

DEMOSTRACIÓN: Supongamos que τ no es planificable. Entonces $H_\tau(t) > t$ implica que $t < \frac{\sum_{i=1}^{n} C_i \cdot (1 - \frac{D_i}{P_i})}{(1 - \mathcal{U}_\tau)}$.

$H_\tau(t) = \{$De la definición de $H_\tau\}$

$$\sum_{i=1}^{n} C_i \cdot \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor$$

$\leq \{$Eliminando la función suelo$\}$

$$\sum_{i=1}^{n} C_i \cdot \frac{t + P_i - D_i}{P_i}$$

$= \{$Reorganizando términos, y de la definición de $\mathcal{U}_\tau$ $\}$

$$t \cdot \mathcal{U}_\tau + \sum_{i=1}^{n} C_i \cdot (1 - \frac{D_i}{P_i})$$

$\{$Substituyendo t por $H_\tau(t)$, y de la asunción que $H_\tau(t) > t\}$

$$t < t \cdot \mathcal{U}_\tau + \sum_{i=1}^{n} C_i \cdot (1 - \frac{D_i}{P_i})$$

$\equiv \{$Reorganizando términos$\}$

$$t \cdot (1 - \mathcal{U}_\tau) < \sum_{i=1}^{n} C_i \cdot (1 - \frac{D_i}{P_i})$$

$\equiv$ {Dividiendo por el número positivo $(1 - \mathcal{U}_\tau)$}

$$t < \frac{\sum_{i=1}^{n} C_i \cdot (1 - \frac{D_i}{P_i})}{1 - \mathcal{U}_\tau}$$

$\square$

Por lo tanto, si un conjunto de tareas no es planificable, alguna tarea violará su plazo de ejecución antes de que transcurran L unidades de tiempo desde el instante inicial.

El siguiente lema muestra que la cota que obtiene este teorema, L, es más ajustada que la conseguida por el teorema anterior, K.

Lema 3.1 *Sean K y L las cotas obtenidas en los teoremas 3.2 y 3.3 respectivamente.*

Entonces $K \geq L$.

DEMOSTRACIÓN: $L \cdot (1 - \mathcal{U}_\tau) = \sum_{i=1}^{n} C_i \cdot (1 - \frac{D_i}{P_i})$ = {Reorganizando términos}

$$\sum_{i=1}^{n} \frac{C_i \cdot (P_i - D_i)}{P_i}$$

$\leq$ {Substituyendo $(P_i - D_i)$ por $\max_{j=1}^{n}\{P_j - D_j\}$}

$$\sum_{i=1}^{n} \frac{C_i \cdot \max_{j=1}^{n}\{P_j - D_j\}}{P_i}$$

= {Sacando $\max_{j=1}^{n}\{P_j - D_j\}$ fuera del sumatorio, y por la definición de $\mathcal{U}_\tau$}

$$\max_{j=1}^{n}\{P_j - D_j\} \cdot \mathcal{U}_\tau$$

= {De la definición de K}

$$K \cdot (1 - \mathcal{U}_\tau)$$

Por tanto $L \leq K$

$\square$

Ambas cotas, L y K, se pueden obtener en tiempo $O(n)$.

Los teoremas 3.2 y 3.3 pueden entender de una forma quizás más intuitiva si consideramos una representación gráfica.

Los dos teoremas anteriores hacen uso del hecho de que la función H_τ puede ser acotada por una función lineal de la forma: $f(t) = a \cdot t + b$ (ver figura 3.3). La función cota que se obtiene del teorema 3.2 es $f^1(t) = \mathcal{U}_\tau \cdot t + (\mathcal{U}_\tau \cdot \max_{i=1}^{n}\{P_i - D_i\})$; y la función cota para el teorema 3.3 es $f^2(t) = \mathcal{U}_\tau \cdot t + (\sum_{i=1}^{n} C_i \cdot (1 - D_i/P_i))$. Ambas cotas se obtienen directamente a partir de la función H_τ, y tienen la misma pendiente. La diferencia reside en el término constante.

Podemos definir geométricamente la condición de planificabilidad como:

> Un conjunto síncrono de tareas periódicas es planificable si y sólo si: la curva escalonada (que representa la función H_τ) no corta la bisectriz del plano[1], $y = t$.

Los valores de K y L se obtienen de la intersección de las rectas $y = f^1(t)$ y $y = f^2(t)$ con la bisectriz. Como puede observarse, una vez que las rectas f^1 y f^2 han cruzado la diagonal, y dado que estas representan una cota superior de la función escalonada, es imposible que la curva escalonada corte a la bisectriz del plano. Es por ello, por lo que sólo es necesario verificar los valores de la función escalonada hasta este punto. Formalmente se puede expresar como: $\forall d_x \geq (b/(1-a)) : H_\tau(d_x) \leq d_x$. Substituyendo los valores de a y b por los correspondientes a cada teorema se obtienen los valores de K y L.

	a	b	Cota
teorema 3.2	$\mathcal{U}_\tau$	$\mathcal{U}_\tau \cdot \max_{i=1}^n\{P_i - D_i\}$	$K = \frac{\mathcal{U}_\tau}{1-\mathcal{U}_\tau} \max_{i=1}^n\{P_i - D_i\}$
teorema 3.3	$\mathcal{U}_\tau$	$\sum_{i=1}^n C_i \cdot (1 - \frac{D_i}{P_i})$	$L = \frac{\sum_{i=1}^n C_i \cdot (1-\frac{D_i}{P_i})}{(1-\mathcal{U}_\tau)}$

Tabla 3.1: Parámetros de las rectas que acotan la función$H_\tau(t)$, $f(t) = a \cdot t + b$.

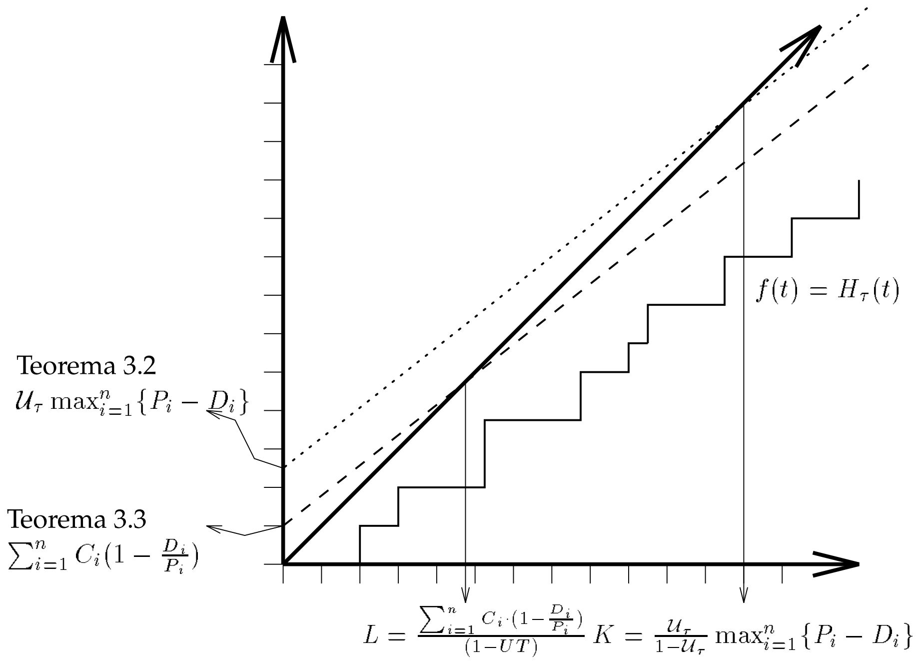


Figura 3.3: Representación gráfica de los últimos teoremas

La idea que inspira el siguiente teorema es completamente diferente a la utilizada en los dos anteriores. Si bien, también obtendremos una cota en la "longitud de la simulación" pero basada en algunas propiedades del ICI.

[1] La curva escalonada puede "tocar" a la bisectriz pero no traspasarla.

Teorema 3.4 (Cota ICI)
Sea τ un conjunto síncrono de tareas periódicas y $[0, \mathcal{R})$ el intervalo crítico inicial. τ es planificable si y sólo si:

$$\forall t \leq \mathcal{R} \ : \ H_\tau(t) \leq t$$

En otras palabras, un conjunto de tareas será planificable si y sólo si el EDF puede planificarlas correctamente durante el ICI. Para la demostración de este teorema nos apoyaremos en los resultados de los dos siguientes lemas.

Lema 3.2 *Sea τ un conjunto de tareas periódicas, y $\tau\prime$ el mismo conjunto pero desplazadas (a las que se les ha aplicado el operador shift) una cantidad de tiempo arbitraria –el desplazamiento puede ser diferente para cada tarea–.*

Si τ es planificable en el intervalo $[0, S)$, entonces $\tau\prime$ es también planificable en el intervalo $[0, t_{break})$, donde t_{break} es el primer instante en el que procesador está ocioso.

DEMOSTRACIÓN: Un conjunto de tareas es planificable en un intervalo si todas las activaciones de todas las tareas cumplen sus plazos de ejecución. Puesto que las activaciones con plazo menor son ejecutadas primero, una activación A_x cumplirá su plazo de ejecución si la suma de los tiempos de ejecución de las activaciones que la preceden (activaciones cuyo plazo es menor o igual que d_x) más su propio tiempo de cómputo es menor o igual que su plazo de ejecución. Esto se puede formular como: $H_\tau(d_x) \leq d_x$. Extendiéndolo para todas las activaciones del intervalo: $\forall A_x \in [0, S) : H_\tau(d_x) \leq d_x$, y puede ser desglosada en la siguiente secuencia de desigualdades:

$$\begin{array}{rcl} c_1 & \leq & d_1 \\ c_2 + c_1 & \leq & d_2 \\ c_3 + c_2 + c_1 & \leq & d_3 \\ \vdots & \vdots & \vdots \\ c_x + \cdots + c_2 + c_1 & \leq & d_x \qquad (3.4) \\ c_{x+1} + c_x + \cdots + c_2 + c_1 & \leq & d_{x+1} \qquad (3.5) \\ \vdots & \vdots & \vdots \\ c_n + \cdots + c_x + \cdots + c_1 & \leq & d_n \leq S \end{array}$$

Supongamos ahora que aplicamos el operador shift a la activación A_x, $shift(A_x, t_r)$. Su nuevo plazo será $d_{x\prime} = d_x + t_r$, y si tiempo de comienzo $s_{x\prime} = s_x + t_r$. Dependiendo del valor de t_r pueden suceder los siguientes casos:

- Mientras $d_x + t_r$ sea menor o igual que d_{x+1}, la activación A_x no altera su orden en la ejecución y el conjunto de desigualdades permanece inalterado, por lo que todos los plazos de todas las activaciones se cumplen.

- Si $d_x + t_r > d_{x+1}$, entonces se produce una permutación en el orden de ejecución entre las activaciones A_x y A_{x+1}. Claramente, la activación A_{x+1} cumple su plazo de ejecución ya que el tiempo de cómputo requerido por A_x es descontado de la parte izquierda de su desigualdad. Queda por verificar si $A_{x'}$ cumple también su nuevo plazo. De la expresión 3.5 y de la condición de permutación obtenemos:

$$\left.\begin{array}{rcl} c_{x+1} + c_x + \cdots + c_2 + c_1 & \leq & d_{x+1} \\ d_{x'} \;=\; d_x + t_r & > & d_{x+1} \end{array}\right\}$$

  Lo que implica que: $c_{x+1} + c_x \cdots + c_2 + c_1 \leq d_{x'}$.

  Ni las activaciones previas a A_x ni las posteriores a A_{x+1} cambian su posición en la secuencia de ejecución, permaneciendo invariables sus condiciones de planificabilidad.

- Supongamos ahora que a m activaciones, $A_{x+1} \cdots A_{x+m+1}$, les vence el plazo en el mismo instante, $d_{x+1} = \cdots = d_{x+m+1}$. Supongamos también que $d_x + t_r > d_{x+1}$.

$$\begin{array}{rcl} \vdots & \vdots & \vdots \\ c_{x+1} + 0 + \cdots + c_2 + c_1 & \leq & d_{x+1} \\ \vdots & \vdots & \vdots \\ c_{x+m+1} + \cdots + 0 + \cdots + c_2 + c_1 & \leq & d_{x+1} \\ c_x + c_{x+m+1} + \cdots + 0 + \cdots + c_2 + c_1 & \leq & d_x \\ \vdots & \vdots & \vdots \end{array}$$

  Puesto que A_x se ejecuta después de $A_{x+1} \cdots A_{x+m+1}$, todas estas disponen de c_x unidades de tiempo extra para completar su ejecución, conservando su correcta ejecución. A_x cumple su plazo ya que:

$$\left.\begin{array}{rcl} c_{x+m+1} + \cdots + c_x + \cdots + c_1 & \leq & d_{x+1} \\ d_{x'} \;=\; d_x + t_r & > & d_{x+1} \end{array}\right\}$$

  Con lo que se llega a que: $c_{x+m+1} + \cdots + c_x + \cdots + c_1 \leq d_{x'}$.

- Supongamos que varias activaciones de una misma tarea aparecen en el intervalo, $[0, S)$. Recordemos que según el modelo de tareas, los plazos de ejecución no pueden ser inferiores a los periodos, $d_x \leq p_x$. Por tanto, si alguna de las activaciones sufre una permutación, los efectos de esta no alteran a la ordenación de las siguientes activaciones, pudiendo analizarse cada una de ellas independientemente.

- Si $t_{break} = \sum_{y:s_y<s_x} c_y$. Asumamos que t_{break} es menor que $s_{x'}$. El procesador queda ocioso después del instante t_{break}. Es evidente que una vez el procesador queda ocioso, ninguna combinación de desplazamientos (*shifts* positivos) puede pedir tiempo de cómputo al procesador después de t_{break}. En este caso, solamente necesitamos considerar el intervalo $[0, t_{break})$ –según el enunciado del lema. Las activaciones que queden en este subintervalo conservan el orden y por tanto la planificabilidad.

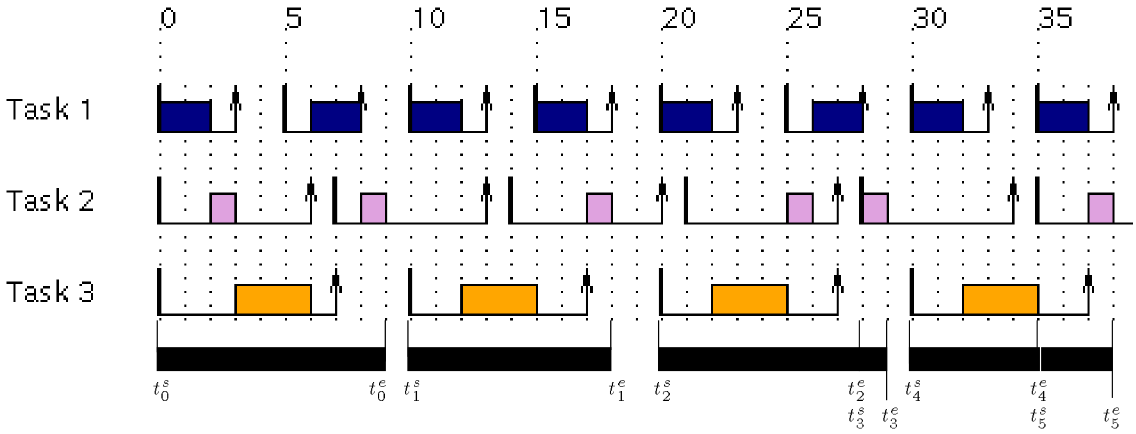


Figura 3.4: División en intervalos de actividad.

Dado un intervalo inicial en el que todas las activaciones cumplen sus plazos, podemos forzar un desplazamiento en una de las activaciones de tal forma que sólo se produzca un de los casos anteriores. De esta forma llegamos a un segundo estado en conjunto de activaciones en el intervalo considerado, estado que continuará siendo planificable. De este segundo estado, podemos pasar a un tercero y así tantas veces como sea necesario, y en el estado final todas las activaciones continuarán cumpliendo sus plazos.

□

Obsérvese que la longitud del intervalo final no es más largo que el de partida.

Lema 3.3 *El intervalo critico inicial de un conjunto síncrono de tareas periódicas tiene el máximo número de activaciones, esto es, no existe otro intervalo de la misma longitud con mayor número de activaciones.*

DEMOSTRACIÓN: El máximo número de activaciones que una tarea T_i puede tener en un intervalo $[t_1, t_2)$ es: $\left\lceil \frac{t_2-t_1}{P_i} \right\rceil$. Luego el intervalo crítico inicial contiene el máximo número de activaciones.[2]

□

DEMOSTRACIÓN: (*del teorema 3.4*)
La demostración está dividida en los siguientes pasos: (1) dividir el tiempo de ejecución en un conjunto de intervalos; (2) construir un método para derivar cada intervalo a partir del inicial; (3) demostrar que ninguna activación pierde su plazo dentro de estos intervalos; y finalmente (4) comprobar que las ejecuciones de todas las activaciones están contenidas en alguno de los intervalos. Vamos a realizar una división *imaginaria* de todo el tiempo de ejecución del sistema en una serie de *intervalos de actividad*, representados por

[2] Gracias a Ramamurt por su detallada revisión

$[t_i^s, t_i^e)$, y definidos como sigue:

$$t_i^s = \begin{cases} i = 0 & 0 \\ i > 0 & \text{mínimo } t \text{ tal que } t > t_{i-1}^e \text{ el procesador no esté ocioso} \end{cases}$$

$$t_i^e = \text{mínimo } t \text{ tal que } t = \sum_{j=1}^{n} C_j \cdot \left\lceil \frac{t - \left\lceil \frac{t_i^s}{P_i} \right\rceil \cdot P_i}{P_j} \right\rceil \text{ y } t > t_i^s$$

Obsérvese que el primero de los intervalos coincide con el ICI.

Pasaremos ahora a desarrollar un método para derivar cualquier intervalo a partir del ICI. Supongamos un intervalo cualquiera, $[t_j^s, t_j^e)$. La primera activación de la tarea T_i dentro del intervalo considerado (esto es, después de t_j^s) se produce en el instante $\lceil \frac{t_j^s}{P_i} \rceil \cdot P_i$. Si a cada una de las tareas le aplicamos el siguiente desplazamiento:

$$\forall_{i:1 \leq i \leq n} \;\; shift(T_i, \left\lceil \frac{t_j^s}{P_i} \right\rceil \cdot P_i - t_j^s)$$

el resultado obtenido es la misma ordenación de tareas que la que se da en el intervalo considerado, pero comenzando en el tiempo cero. Sólo resta desplazar todo el intervalo en bloque t_j^s u.t. para obtener exactamente el intervalo buscado.

El lema 3.3 nos asegura que no hay otro intervalo con más activaciones que el ICI. Por tanto el lema 3.2 puede utilizarse para demostrar que todos los intervalos son planificables.

Por la forma en la que se han construido los intervalos, todos los instantes en los que el procesador está activo están contenidos por algún intervalo. Sólo los instantes en los que el procesador está ocioso no están contenidos en ningún intervalo. Por tanto, todas las activaciones cumplen su plazo de ejecución.

$\square$

En el siguiente lema se da una cota superior a la longitud del ICI. Si bien las simulaciones llevadas a cabo muestran que ésta es una cota bastante pesimista y poco ajustada.

Lema 3.4 *Sea τ un conjunto de tareas planificable. Existe un tiempo $t \leq \min\{\sum_{i=1}^{n} C_i/(1 - \mathcal{U}_\tau)\ ,\ \mathcal{P}_\tau\}$ tal que $G_\tau(t) \leq t$.*

DEMOSTRACIÓN: Primero verifiquemos que $G_\tau(t)$ está acotada por: $\sum_{i=1}^{n} C_i + t \cdot \mathcal{U}_\tau$.

$$G_\tau(t) = \sum_{i=1}^{n} C_i \left\lceil \frac{t}{P_i} \right\rceil$$

$\leq$ {Puesto que $\lceil x \rceil \leq x + 1$}

$$\sum_{i=1}^{n} C_i \left(\frac{t}{P_i} + 1 \right)$$

$=$ {De la definición de $\mathcal{U}_\tau$ y reorganizando términos}

$$\sum_{i=1}^{n} C_i + t \cdot \mathcal{U}_\tau$$

{Substituyendo $G_\tau(t)$ por t}

$$t \leq \sum_{i=1}^{n} C_i + t \cdot \mathcal{U}_\tau$$

$\equiv$ {Reorganizando}

$$t \cdot (1 - \mathcal{U}_\tau) \leq \sum_{i=1}^{n} C_i$$

$\equiv$ {Dividiendo ambos lados por $(1 - \mathcal{U}_\tau)$}

$$t \leq \frac{\sum_{i=1}^{n} C_i}{1 - \mathcal{U}_\tau}$$

El otro término del operador "min" es el hiperperiodo.

Supongamos que $G_\tau(\mathcal{P}) > \mathcal{P}$.

$$G_\tau(\mathcal{P}) = \sum_{i=1}^{n} C_i \cdot \left\lceil \frac{\mathcal{P}}{P_i} \right\rceil$$

$=$ {Puesto que $\forall i : P_i \| \mathcal{P}$ }

$$\sum_{i=1}^{n} C_i \cdot \frac{\mathcal{P}}{P_i}$$

$=$ {Dado que $\mathcal{P}$ es una constante puede salir del sumatorio}

$$\mathcal{P} \cdot \mathcal{U}_\tau$$

{Substituyendo $\mathcal{P}$ por $G_\tau(\mathcal{P})$, y por la suposición que $G_\tau(\mathcal{P}) > \mathcal{P}$}

$$\mathcal{P} < \mathcal{P} \cdot \mathcal{U}_\tau$$

$\equiv$ {Dividiendo ambos lados por $\mathcal{P}$}

$$1 < \mathcal{U}_\tau.$$

Puesto que una condición necesaria de planificabilidad es que $1 \geq \mathcal{U}_\tau$ [Liu73, Leung80], llegamos a una contradicción, y por tanto $G_\tau(\mathcal{P}) \leq \mathcal{P}$.

□

Esta cota superior es bastante pesimista. De hecho, el test propuesto por Baruah et al. es siempre mejor que esta cota, si bien los resultados experimentales (ver sección 3.4) muestran que en más del 50% de los conjuntos de tareas evaluados, el test basado en el ICI es más eficiente –la duración del ICI es menor– que el test basado en el teorema 3.3.

Teorema 3.5 (Cota combinada)
Sea $L = \frac{\sum_{i=1}^{n} C_i \cdot (1 - \frac{D_i}{P_i})}{(1 - \mathcal{U}_\tau)}$, *y* $[0, \mathcal{R})$ *el ICI.*

τ *es planificable si y sólo si:*

$$\forall t < \min(L, \mathcal{R}), H_\tau(t) \leq t.$$

DEMOSTRACIÓN: Inmediato a partir de los teoremas 3.3 y 3.4.

□

3.2.2 Sistemas asíncronos

En la sección precedente hemos estudiado las condiciones necesarias y suficientes que tienen que cumplir un conjunto de tareas para que sea planificable. En esta sección analizamos el problema de la planificabilidad de conjuntos asíncronos de tareas que pueden tener tiempos de inicio distintos de cero.

Teorema 3.6
Un conjunto asíncrono de tareas, τ, es planificable si es planificable el correspondiente conjunto síncrono (conjunto formado por las mismas tareas que τ pero con tiempos de inicio igual a cero).

Podemos emplear exactamente la misma demostración que la utilizada en el teorema 3.4 para conjuntos síncronos. Si el ICI del conjunto síncrono es planificable, entonces todos los intervalos de actividad en los que podemos dividir la ejecución del conjunto asíncrono pueden ser derivados de él utilizando el mismo método.

Éste es un test pesimista (suficiente, pero no necesario). Puede darse el caso que un conjunto de tareas planificable no pase satisfactoriamente el test. Esto puede suceder cuando no se dé ningún intervalo de actividad "similar" al ICI –intervalos con la misma cantidad de activaciones– durante toda la ejecución hasta el hiperperiodo. Un sencillo ejemplo numérico de esta situación es el siguiente conjunto, formado por dos tareas $T_1 = (4, 5, 8)$ y $T_2 = (3, 5, 8)$ con tiempo de inicio cero y tres respectivamente.

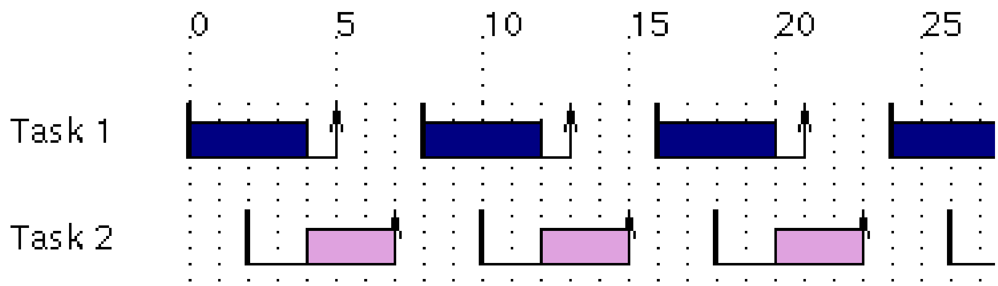


Figura 3.5: Conjunto planificable pero no aceptado por el test.

El test obtiene la respuesta correcta cuando todos los periodos de las tareas son coprimos. En este caso, se dan todas las posibles combinaciones de fases entre todas las tareas, incluida la combinación que produce el ICI.

Sean x e y dos enteros. Existen dos números enteros $x\prime$, $y\prime$ tales que $x \cdot x\prime + y \cdot y\prime =$ gcd(x, y) [Knuth69]. Obsérvese que tanto $x\prime$ como $y\prime$ representan dos posibles valores de una serie infinita de soluciones; y que los valores de $x\prime$, e $y\prime$ son de signo opuestos. Dado que el máximo común divisor de dos números co-primos es 1, podemos multiplicar los dos lados de la expresión anterior para obtener: $x \cdot (x\prime yw) + y \cdot (y\prime x) = w$, lo que significa que podemos encontrar un múltiplo de x y otro de y tal que disten cualquier valor. Eligiendo esta diferencia para que sea igual a la diferencia entre los tiempos de inicio de las dos tareas, obtenemos un instante en el que ambas tareas tienen una activación justo en el mismo instante. Repitiendo este razonamiento con todas las demás tareas, podemos encontrar un instante de tiempo en el que todas las tareas se activen a la misma vez.

3.3 Test de planificabilidad

En esta sección construimos el algoritmo que implementa el test basado en el teorema 3.5.

Básicamente, el algoritmo simula el funcionamiento del EDF hasta que o bien una activación pierde su plazo, o se alcanza la condición del teorema 3.5. La simulación se lleva a cabo de forma implícita al evaluar las funciones H_τ y G_τ.

El teorema 3.5 está compuesto por dos cotas. La cota basada en la función H_τ se calcula antes de entrar en el bucle principal, mientras que la cota basada en el ICI, al ser una ecuación recurrente, se calcula dentro del mismo bucle.

Algoritmo 3.1 (Test de planificabilidad, test combinado)

```
1  proc (τ) ≡
2    var
3      τ;
4      t := 0;
5      c := \sum_{i=1}^{n} C_i / P_i;
6      γ := \sum_{i=1}^{n} C_i · (1 − D_i / P_i);
7      L := γ / (1 − c)
8    end
9    begin
10     comment: Definici
11     G_τ(t) = \sum_{i=1}^{n} C_i · \lceil t / P_i \rceil;
12     H_τ(t) = \sum_{i=1}^{n} C_i · \lfloor (t + P_i − D_i) / P_i \rfloor;
13     if (c ≥ 1)
14       then exitNo_planificable;
15     fi
16     while (t < min(G_τ(t), L)) ∧ (t ≥ H_τ(t)) do
17           t := t + 1
18     od
19     if t ≥ H_τ(t)
20       then Planificable;
21        else No_Planificable;
22     fi
23   end
24 .
```

Varias mejoras se pueden aplicar al algoritmo para acelerar su ejecución. Tanto G_τ como H_τ son dos funciones escalonadas, por lo que la sentencia 17 se puede reemplazar para que los incrementos se produzcan de escalón en escalón, en lugar de ir de unidad en unidad. Cuanto mayores sean los periodos, mayores serán los saltos entre escalones de ambas funciones y mayor será el incremento de velocidad. Por otra parte, el cálculo de G_τ

y H_τ puede realizarse de forma incremental, con lo que en lugar tener una complejidad temporal del orden de $O(n)$, el coste se reduce a $O(1)$ en cada iteración.

A pesar de estas mejoras, si no restringimos la instancia del problema –no imponemos ninguna condición al conjunto de tareas–, en el peor caso el coste temporal es exponencial. Pero si restringimos la aplicación del test a conjuntos de tareas con un factor de utilización menor que el 99%, entonces el algoritmo se ejecuta en tiempo pseudo-polinómico.

Sólo hemos presentado una implementación del test basada en el teorema 3.5, pero también se podrían construir dos nuevos algoritmos utilizando por separado los teoremas 3.3 y 3.4, y aunque no los hemos descrito en pseudo-código, si que los emplearemos en la siguiente sección para demostrar que ninguno de ellos supera al otro para cualquier conjunto de tareas. Es por ello que el test combinado representa la mejor opción.

3.4 Resultados experimentales

En esta sección comparamos, por simulación, las prestaciones de cada uno de los test propuestos. El índice empleado en la comparación ha sido la longitud del intervalo que cada caracterización define como cota para asegurar que el conjunto de tareas es planificable. Cuanto mayor sea este índice, más iteraciones se necesitan para decidir si el conjunto es planificable o no, y por tanto, peor es el algoritmo.

Se han empleado dos tipos de carga –conjunto de tareas–: (1) carga real, y (2) carga sintética.

Carga real Como carga real se ha utilizado el conjunto de tareas que componen el sistema Olympus Attitude and Orbital System [Burns93] . En la tabla 3.4 se detallan las características del conjunto. En la tabla 3.3 aparecen los resultados de todos los tests.

Tabla 3.2: Descripción de las tareas (tiempos en milisegundos).

Nombre de tarea	Periodo	Plazo	Tiempo máx. de ejecución
RTC	50	9	0.28
Read_Bus_IP	10	10	1.76
Command_Actuators	200	14	2.13
Request_DSS_Dada	200	17	1.43
Request_Wheel_Speeds	200	22	1.43
Request_IRES_data	100	24	1.43
Process_IRES_data	100	50	8.21
Control_Law	200	200	52.84
Process_DSS_Data	1000	400	5.16
Calibrate_Gyro	1000	900	6.91

Tabla 3.3: Resultados del HRT-HOOD (tiempo en ms).

Factor de utilización	0.57
Hiperperiodo (teorema 3.1)	1000.00
Test combinado (teorema 3.5)	32.76
Baruah (teorema 3.2)	825.85
H_τ test (teorema 3.3)	32.76
Test del ICI (teorema 3.4)	97.00

En este caso, el test basado en el teorema 3.3 es más de 25 veces más rápido que el basado en el teorema 3.2.

Carga sintética Tal como exponen Ghazalie y Baker [Ghazalie95]:

> . . .nosotros no conocemos ningún trabajo que caracterice la distribución de los periodos y los tiempos de cómputo de los conjuntos de tareas de las aplicaciones de tiempo real.

El único estudio sobre cargas de tiempo real, del que tenemos conocimiento, fue el llevado a cabo por Weiderman y Kamenoff [Weiderman92]. En él se definen una serie de experimentos con el fin de comprobar la habilidad de un sistema monoprocesador para dar servicio a tareas de tiempo real. Por desgracia, la unidad de medida empleada en la especificación de los tiempos de cómputo es el Kilo-Whetstone, lo que lo hace dependiente de la implementación. Por consiguiente no apropiado para nuestros experimentos.

El modelo de carga utilizado es similar al empleado en [Davis93]. La carga ha sido generada a partir de los siguientes cuatro parámetros: tiempo máximo de ejecución, plazo de ejecución, periodo y porcentaje de utilización del procesador; de los cuales los tres primeros se emplean para generar cada tarea, mientras que el último se emplea para determinar el número de tareas.

El núcleo del generador de carga sintética se detalla en el siguiente algoritmo:

Algoritmo 3.2 (Generador de carga)

```
proc (COMPUTE, DEADLINE, PERIOD, TARGET_UTIL) ≡
  var
    C, D, P;
  end
  begin
    while Util(Task_Set) < TARGET_UTIL do
          C := random mod COMPUTE;
          D := C + random mod DEADLINE;
          P := D + random mod PERIOD;
          Task_Set := Task_Set ∪ (C, D, P);
    od
  end
```

14 .

La función *random* genera un número pseudo-aleatorio con una distribución uniforme –también se han estudiado conjuntos generados con otros tipos de distribuciones con resultados muy similares.

La influencia que cada tarea produce sobre los tests no depende de la magnitud de los parámetros de ésta, sino más bien de la relación entre ellos. Por ejemplo, una tarea con tiempo de cómputo 10, plazo de ejecución 20, y periodo 30, presenta una influencia muy similar a otra tarea de la forma $T = (100, 200, 300)$. Es por ello, por lo que se ha optado por mantener uno de los parámetros constante y variar sólo los otros dos. El valor de *COMPUTE* se ha fijado a 20 unidades.

En la confección de las gráficas sólo se han contabilizado los conjuntos de tareas planificables, puesto que todos los tests tienen el mismo coste cuando el conjunto no es planificable.

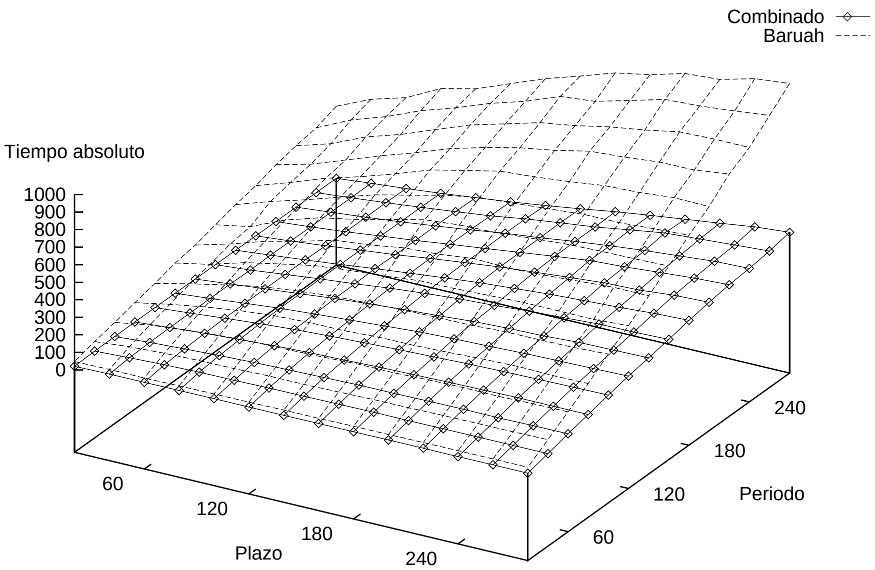
Para cada tipo de tareas –tareas generadas a partir de los mismos parámetros (*COMPUTE, DEADLINE, PERIOD* y *TARGET_UTIL*)– se han generado quinientos conjuntos, a los que se les han aplicado los diferentes tests. A pesar de que la teoría que soporta cada test es distinta, a nivel implementación todos son muy similares. Los tests simulan el comportamiento del EDF hasta llegar a un instante de tiempo a partir del cual se asegura que ninguna tarea perderá su plazo. Este tiempo será distinto en función del test concreto que se haya ejecutado, y es el parámetro que empleamos para comparar los tests.

Los dos primeros grupos de gráficas (figuras 3.6 y 3.7) comparan las prestaciones del test más eficiente propuesto en esta tesis (test combinado, teorema 3.5) frente al mejor test que existía (Baruah, teorema 3.2).

El factor de utilización del procesador es del 80%. El plano superior representa el tiempo utilizado por el algoritmo propuesto por Baruah et al., y el plano punteado inferior el empleado por el test combinado. Como puede apreciarse, para cualquier conjunto de tareas el test combinado requiere menos tiempo de análisis. En la segunda gráfica de cada figura aparecen los resultados de las mismas simulaciones pero expresados como el cociente entre los dos. Valores más bajos indican mayor diferencia a favor del test combinado, llegando éste a ser unas diez veces más rápido.

En la siguiente pareja de gráficas, figura 3.7, se mantiene la misma configuración de tareas y se aumenta el factor de utilización al 95%. Los resultados son similares.

En la siguiente pareja de gráficas, figura 3.8, se muestra la respuesta de los mismos algoritmos cuando varía el grado de utilización del procesador. En la primera se ha mantenido constante el plazo de ejecución en 150 u.t., mientras que en la segunda, ha sido el periodo el que se ha fijado en 150 u.t.. En ambas gráficas, al igual que en cuatro precedentes, el tiempo de cómputo es de 20. En el peor caso el test combinado es el doble de rápido, llegando incluso a ser hasta 20 veces más rápido cuando la utilización es baja o el periodo el grande.



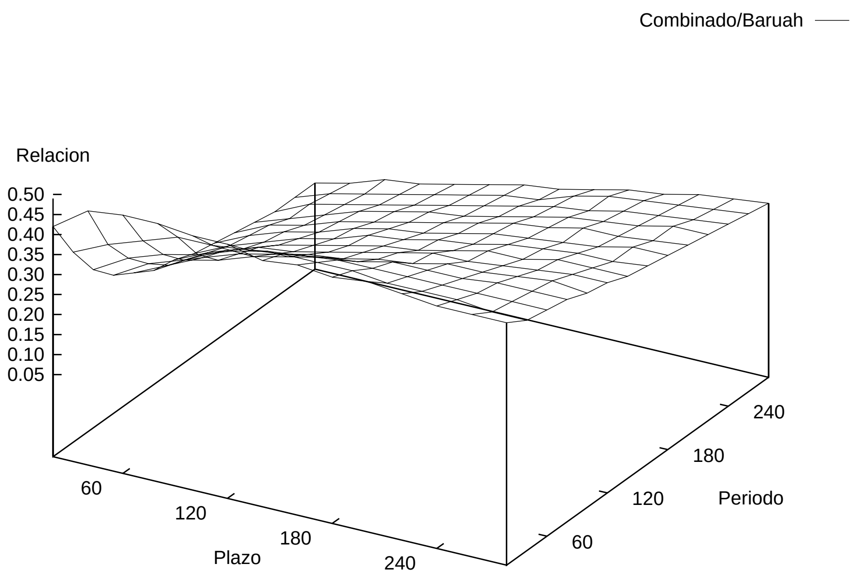
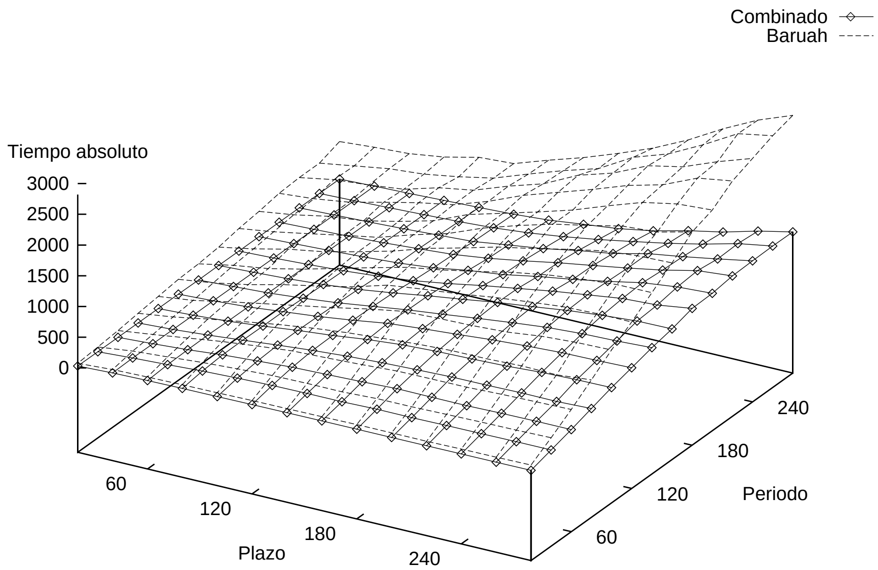


Figura 3.6: Factor de utilización del procesador del 80%.



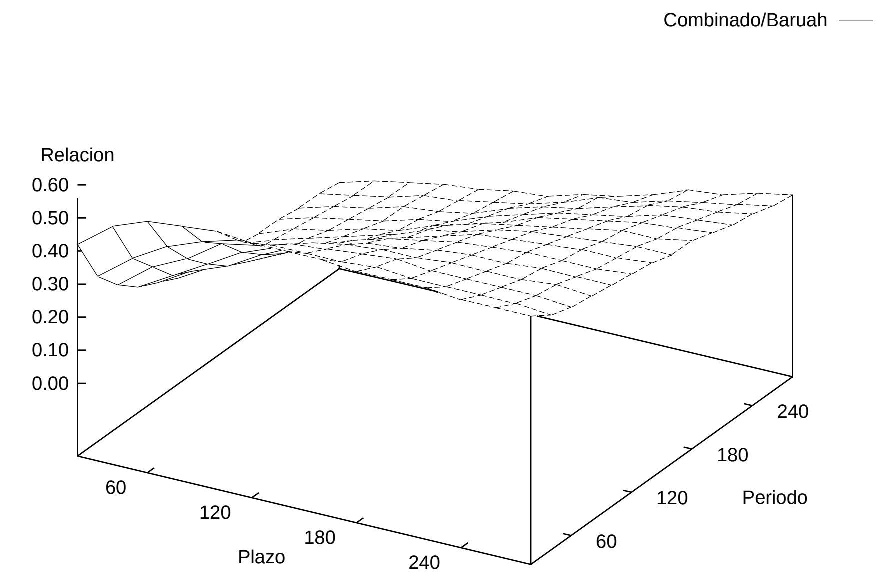
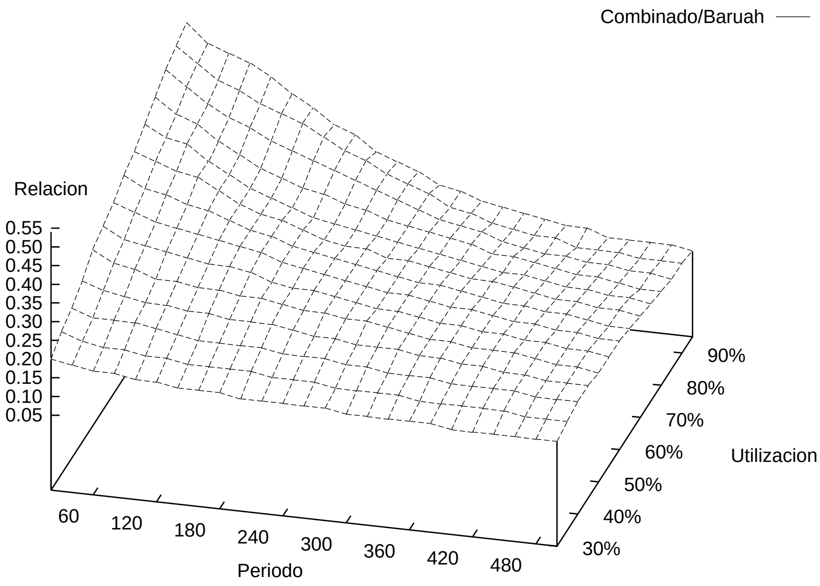


Figura 3.7: Factor de utilización 95%.



Plazo fijado a 150 u.t..

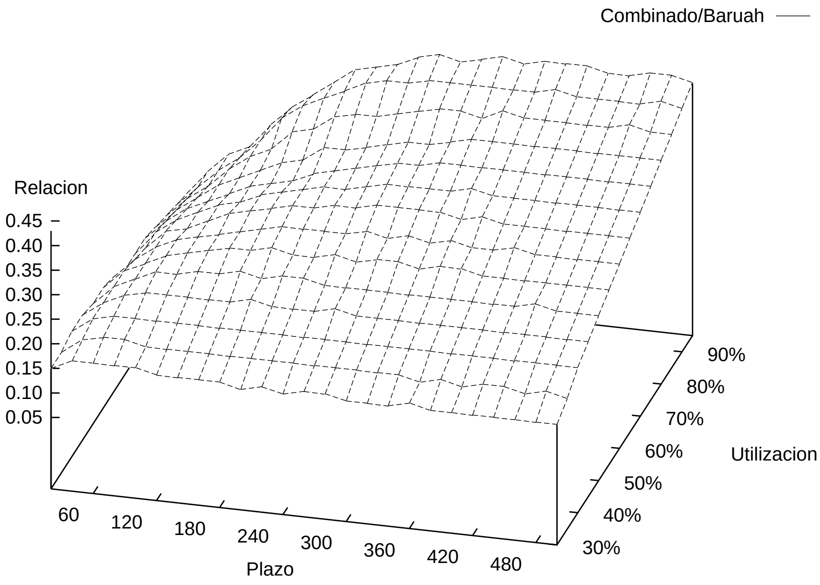


Figura 3.8: Periodo fijados a 150 u.t..

En la figura 3.9 se compara el comportamiento por separado de los dos tests que componen el test combinado. Para apreciar mejor los casos en los que uno supera al otro, se han igualado a uno todos los puntos menores que uno, por lo que la zona plana corresponde a conjuntos que en los que el test mejorado (teorema 3.3) es más rápido. Cuando los plazos son similares a los periodos el test mejorado supera al test del ICI. Ninguno supera al otro en todos los tipos de tareas.

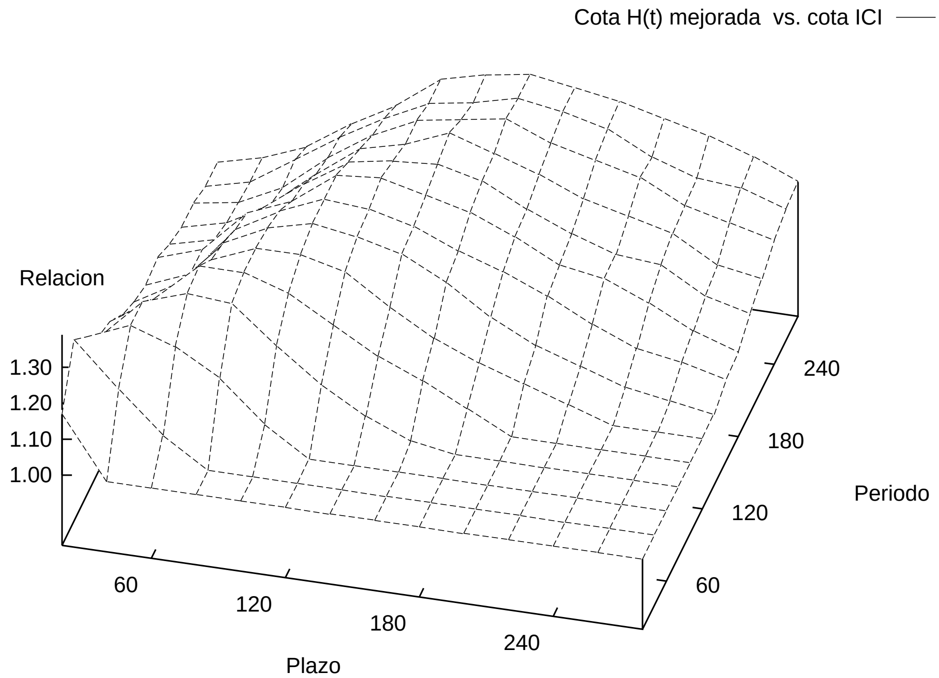


Figura 3.9: Comparación entre el test mejorado y el basado en el ICI, (factor de utilización del 80%).

El tiempo total requerido por las simulaciones que sirvieron para dibujar los gráficos de la figura 3.8 fue de unos 9 minutos para realizar un total de 350.000 ejecuciones distintas. La máquina que se utilizó fue una estación de trabajo Sun Sparc IPX.

3.5 Conclusiones

En este capítulo se ha realizado una revisión de los tests de planificabilidad existentes hasta la fecha. Se ha redefinido el Intervalo Crítico Inicial (ICI), se han propuesto dos nuevas caracterizaciones de conjuntos de tareas planificables, y se ha presentado un nuevo test de planificabilidad. Finalmente, se han comparado las prestaciones de los tests, derivados de las distintas caracterizaciones, comprobándose que el test propuesto es el test más eficiente para cualquier conjunto de tareas.

A pesar del mayor rendimiento del test propuesto, el problema general sigue siendo co-**NP**-completo, y por consiguiente, existen algunos conjuntos de tareas para los que decidir si son planificables o no requiere una gran cantidad de tiempo de cómputo.

Normalmente se supone que las características temporales de las tareas de tiempo real son estrictamente impuestas por las restricciones físicas del sistema que se ha de controlar, de tal forma que cada tarea es una tupla de tres valores inalterables. En la práctica, desde el punto de vista de la planificación, la mayor parte de los sistemas no están descritos de forma tan severa. Por ejemplo: en un sistema de control de temperatura de un horno, el control (medida, cálculo de control y actuación) de la temperatura se ha de realizar al menos cada dos minutos, pero puede ser más frecuentemente.

Por otra parte, la clase de problemas co-**NP**-completos parece tener una curiosa propiedad. Esta clase de problemas muestran un comportamiento exponencial en el peor de los casos, pero modificando mínimamente la instancia del problemas (en nuestro caso, incrementando o decrementando en una o dos unidades de tiempo alguno de los parámetros de alguna tarea), la complejidad puede disminuir drásticamente, hasta el grado de convertirse en un problema resoluble en tiempo polinómico [Garey79]. Esta idea no ha sido formalmente demostrada. Recordemos que el problema de la planificabilidad pertenece a esta clase de problemas.

Uniendo las dos ideas anteriores, podemos afirmar que en la práctica el 100% de los problemas de análisis de planificabilidad de sistemas de tiempo real pueden ser resueltos eficientemente.

Capítulo 4

Servidor de holgura estático

4.1 Introducción

Una vez estudiado el problema básico del análisis de planificabilidad en el capítulo anterior, en éste nos centraremos en el diseño de un algoritmo de planificación que permita ejecutar conjuntamente tareas periódicas y aperiódicas. Evidentemente, y puesto que la integridad del sistema depende de la correcta ejecución de las tareas críticas, el algoritmo de planificación debe garantizar el cumplimiento de todos los plazos de las tareas periódicas y esporádicas, pero por otra parte, ha de ser capaz de ofrecer un corto tiempo de respuesta a las tareas aperiódicas (a veces llamadas *non-real-time tasks*), lo que significa demorar la ejecución de las tareas periódicas, sin por ello poner en peligro los plazos de ejecución de las tareas periódicas.

Las tareas aperiódicas hacen uso del tiempo de procesador no empleado por las tareas periódicas para mejorar las prestaciones del sistema. Muchos son los usos que se le puede dar a estas tareas, tales como: mostrar información del estado del sistema, mejorar la calidad de la respuesta (acciones de control) de las tareas periódicas, o incluso gestionar los cambios de modo o la recuperación de tareas críticas.

El servidor presentado en este capítulo se engloba dentro de la clase de servidores de holgura[1]. El algoritmo propuesto es un *aceptador aperiódico* que se apoya sobre un algoritmo de planificación óptimo (como puede ser el EDF o el LLF). El algoritmo de aceptación asigna a cada tarea aperiódica el plazo más corto compatible con la ejecución de las tareas periódicas. Después de asignarle un plazo a la tarea aperiódica, ésta se envía al planificador del sistema como si de una activación periódica se tratara.

4.1.1 Modelo de sistema

El modelo de tareas críticas es fundamentalmente el mismo que el empleado en el capítulo 3, con la excepción de que todas las tareas deben ser periódicas, el sistema no

[1] slack server

puede tener ninguna tarea esporádica convertida a periódica. En el siguiente capítulo se elimina esta restricción.

Las tareas aperiódicas, J_i, representan acciones cuyo patrón de activaciones no es conocido *a priori*. Tampoco tienen un plazo de ejecución estricto. Cada tarea aperiódica tiene asociado un tiempo de llegada y un tiempo de cómputo, $J_i = (S_i, C_i)$. En la sección 4.5.2 se amplía el aceptador aperiódico para aceptar tareas con plazo firme, $J_i = (S_i, C_i, D_i)$.

Ejemplo ilustrativo Veamos un ejemplo numérico para mostrar cuales son los objetivos y cual la problemática que aparece.

Consideremos el siguiente conjunto de tareas:

Tarea	Tiempo de cómputo	Plazo de ejecución	Periodo
T_1	1	3	6
T_2	3	8	12
T_3	3	9	10

Tabla 4.1: Conjunto de tareas ejemplo.

El hiperperiodo de este conjunto es 60; y la utilización es del 71.67%.

Supongamos que varias peticiones aperiódicas, con una unidad de tiempo de cómputo, piden ser ejecutadas en el instante cero. Si las ejecutamos cuando el procesador esta ocioso, obtendremos un alto tiempo de respuesta, tal como se aprecia en la figura 4.1. Los recuadros inferiores representan los instantes en los que se ejecuta carga aperiódica.

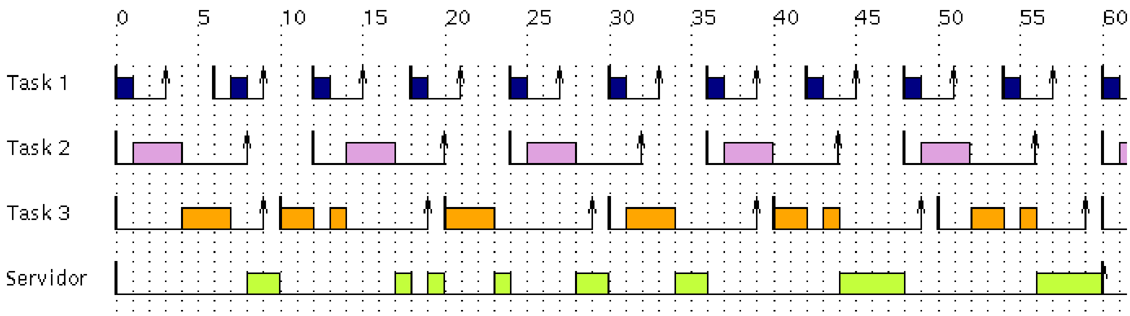


Figura 4.1: Planificación de carga aperiódica en segundo plano (*background*)

Como puede apreciarse, la primera tarea podría retrasarse dos unidades de tiempo, sin por ello perder su plazo de ejecución, para adelantar la ejecución de la primera petición aperiódica y así conseguir un tiempo de respuesta más corto –inmediato– para las dos primeras tareas aperiódicas.

El resultado de esta decisión, como puede apreciarse en la siguiente figura, no es el esperado. La segunda activación de la primera tarea pierde su plazo al haber demasiada carga pedida antes que su plazo.

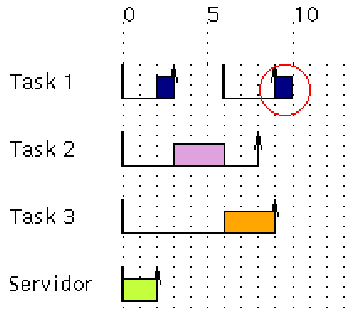


Figura 4.2: Sistema sobrecargado al adelantar la ejecución de trabajo aperiódico.

Si sólo adelantamos la ejecución de una tarea aperiódica, ninguna tarea pierde su plazo. En la figura 4.3 se han planificado las cinco primeras peticiones aperiódicas con el menor tiempo de respuesta posible.

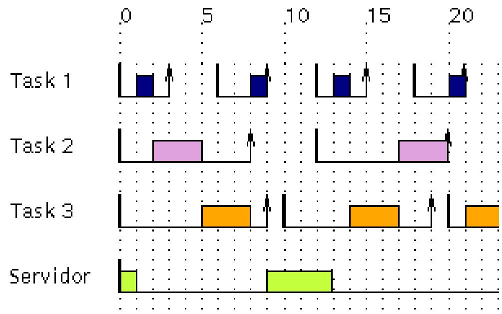


Figura 4.3: Las cinco primeras tareas aperiódicas ejecutadas lo antes posible.

Por otro lado, la figura 4.4 muestra los resultados de intentar dar los mismos tiempos de respuesta a las cinco primeras tareas, pero empleando una política de planificación con prioridades fijas.

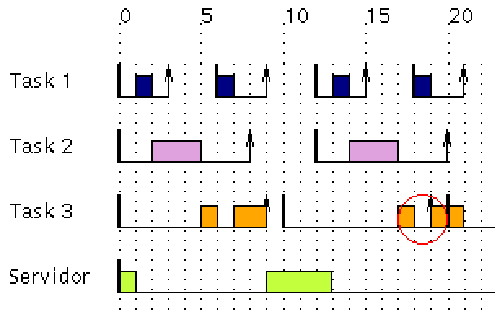


Figura 4.4: La planificación con prioridades fijas no permite unos tiempos de respuesta tan rápidos como con prioridades dinámicas.

En este ejemplo vemos las principales diferencias entre los sistemas basados en prioridades fijas y dinámicas. En la planificación con prioridades fijas, tan sólo se necesita verificar la correcta ejecución de las tareas activas, pues si éstas no fallan entonces ninguna de las siguientes fallará. Con prioridades dinámicas se tiene una mayor flexibilidad en la planificación, con lo que se pueden ejecutar mayor número de tareas periódicas y/o dar un mejor tiempo de respuesta a las tareas aperiódicas. Pero en cambio, el análisis de garantía resulta algo más complejo.

4.2 Fundamentos teóricos

Antes de abordar el cálculo de la holgura (*slack*), precisaremos de algunas herramientas teóricas.

En primer lugar requeriremos las funciones $G_\tau(t)$ (ecuación 3.1) y $H_\tau(t)$ (ecuación 3.2) tal como se definieron en el capítulo anterior, y por otro, varias propiedades de estas funciones:

Propiedad 4.1 *Para cualquier conjunto de tareas, τ, las funciones $H_\tau(t)$ y $G_\tau(t)$ son funciones positivas no decrecientes.*

Propiedad 4.2 *Para todo conjunto de tareas planificable, τ, se cumple que:*

$$\forall t,\ H_\tau(t) \leq G_\tau(t)$$

La demostración se reduce a comprobar que $\left\lfloor \frac{t+P_i-D_i}{P_i} \right\rfloor \leq \left\lceil \frac{t}{P_i} \right\rceil$. Puesto que $0 < D_i \leq P_i$, la expresión se puede reescribir como: $\lfloor x + \epsilon \rfloor \leq \lceil x \rceil$, donde $0 \leq \epsilon < 1$, y x es un número real positivo. Lo que demostraremos por casos:

- Si x es un número entero, entonces $x + \epsilon$ es menor que el siguiente número entero, por lo que $\lfloor x + \epsilon \rfloor = \lfloor x \rfloor = x = \lceil x \rceil$.
- Si x no es entero, entonces tenemos que $\lfloor x \rfloor + 1 = \lceil x \rceil$. Sólo se pueden dar los siguientes casos:
  - $x + \epsilon < \lfloor x \rfloor + 1$, entonces $\lfloor x + \epsilon \rfloor = \lfloor x \rfloor < \lceil x \rceil$.
  - $x + \epsilon \geq \lfloor x \rfloor + 1$, entonces $\lfloor x + \epsilon \rfloor = \lfloor x \rfloor + 1 = \lceil x \rceil$.

Propiedad 4.3 *Sea τ un conjunto de tareas planificable. Sea T_j una tarea perteneciente al conjunto τ. Entonces:*

- $\forall_{t<D_j},\ H_\tau(t) \equiv H_{\tau-T_j}(t)$.
- $\forall_{D_j \leq t \leq P_j},\ H_\tau(t) \equiv H_{\tau-T_j}(t) + C_j$

De la definición de $H_\tau(t) = \sum_{i=1}^{n} C_i \cdot \left\lfloor \frac{t+P_i-D_i}{P_i} \right\rfloor$, se observa que la expresión que aparece dentro de la función suelo, $\frac{t+P_i-D_i}{P_i}$ es menor que 1 para para valores de t menores que D_i, y el suelo de ésta es cero, por lo que el tiempo de cómputo de la tarea no se contabiliza.

Para valores de t comprendidos entre D_i y P_i, el valor de la función suelo es igual a 1, por lo que la tarea T_i sólo colabora con C_i unidades de tiempo al valor final de $H_\tau(t)$.

Propiedad 4.4 *Sea τ un conjunto de tareas planificable, y sea T_j una de las tareas de τ. Entonces:*

- $\forall_{0<t\leq P_j},\ G_\tau(t) \equiv G_{\tau-T_j}(t) + C_j$.

Inmediato a partir de la definición de G_τ y las propiedades de la función techo.

Propiedad 4.5 *Sea τ un conjunto de tareas planificable. Sea t un instante t_x tal que:*

$$G_\tau(t_x) \leq t_x$$

entonces el ICI es de menor duración que el intervalo $[0, t_x)$.

Para que $G_\tau(t_x) \leq t_x$, el procesador ha debido quedar ocioso al menos durante $t_x - G_\tau(t_x)$ unidades de tiempo desde el instante inicial. Como mucho, el ICI llegará hasta el primero de estos tiempos ociosos.

De todas las caracterizaciones de sistemas planificables descritas en el capítulo anterior, tan sólo necesitaremos la descrita por el teorema 3.4.

4.2.1 Características de nuevas tareas

En esta sección desarrollamos los fundamentos teóricos en los que se basa el aceptador aperiódico. Estudiaremos cómo incluir nuevas tareas a un conjunto de tareas dado, intentando asignar la mayor cantidad de tiempo posible a las nuevas tareas pero sin poner en peligro la correcta ejecución del conjunto original.

Supongamos que la utilización del procesador es menor que el 100%. En este caso, es obvio que el procesador estará desocupado durante algún intervalo de tiempo; más concretamente, se dispondrán de $(1-\mathcal{U}_\tau)\cdot\mathcal{P}_\tau$ u.t. libre en cada hiperperiodo. En el siguiente lema se concretan estas ideas.

Lema 4.1 *Sea τ un conjunto de tareas planificable. Si $\mathcal{U}_\tau < 1$, entonces existe una tarea, T_{n+1}, tal que el nuevo conjunto $\tau' = \tau \cup T_{n+1}$ es planificable y $\mathcal{U}_{\tau'} = 1$.*

DEMOSTRACIÓN: Construyamos la nueva tarea T_{n+1} con $C_{n+1} = (1-\mathcal{U}_\tau)\cdot\mathcal{P}_\tau$ y $P_{n+1} = D_{n+1} = \mathcal{P}_\tau$. Nótese que el nuevo conjunto de tareas tiene el mismo hiperperiodo que el conjunto original, pero el ICI se ha modificado, por lo que tenemos que comprobar la planificabilidad del nuevo conjunto en todo el hiperperiodo, esto es, $\forall_{t\leq\mathcal{R}},\ H_{\tau'}(t) \leq t$.

- Si $t < \mathcal{P}_{\tau'}$
  $\{$Dado que $D_{n+1} = \mathcal{P}_\tau = \mathcal{P}_{\tau'}$ y por la Propiedad 4.3$\}$
  $H_{\tau'}(t) = H_\tau(t)$
  $\leq$ $\{$Puesto que τ es planificable$\}$
  $$t$$
  Y por tanto, $H_{\tau'}(t) \leq t$.

- Si $t = \mathcal{P}_{\tau'}$ $\{$De la Propiedad 4.3 , y $D_{n+1} = \mathcal{P}_\tau = \mathcal{P}_{\tau'}\}$
  $H_{\tau'}(\mathcal{P}_{\tau'}) = H_\tau(\mathcal{P}_\tau) + C_{n+1}$
  $=$ $\{$Por definición de $C_{n+1}\}$
  $$H_\tau(\mathcal{P}_\tau) + (1 - \mathcal{U}_\tau) \cdot \mathcal{P}_\tau$$
  $=$ $\{$Substituir H_τ y $\mathcal{U}_\tau$ por sus definiciones$\}$
  $$\sum_{i=1}^{n} C_i \left\lfloor \frac{\mathcal{P}_\tau + P_i - D_i}{P_i} \right\rfloor + \mathcal{P}_\tau - \sum_{i+1}^{n} \mathcal{P}_\tau \frac{C_i}{P_i}$$
  $=$ $\{$Sacar $\mathcal{P}_\tau$ fuera de la función suelo, puesto que $P_i || \mathcal{P}_\tau\}$
  $$\sum_{i=1}^{n} C_i \left(\frac{\mathcal{P}_\tau}{P_i} + \left\lfloor \frac{P_i - D_i}{P_i} \right\rfloor \right) + \mathcal{P}_\tau - \sum_{i=1}^{n} \mathcal{P}_\tau \frac{C_i}{P_i}$$
  $=$ $\{$ $\left\lfloor \frac{P_i - D_i}{P_i} \right\rfloor$ es igual a cero, ya que $P_i - D_i < P_i\}$
  $$\sum_{i=1}^{n} C_i \frac{\mathcal{P}_\tau}{P_i} + \mathcal{P}_\tau - \sum_{i=1}^{n} C_i \frac{\mathcal{P}_\tau}{P_i}$$
  $$\mathcal{P}_\tau = \mathcal{P}_{\tau'}$$

El factor de utilización del nuevo conjunto es el original más la componente de la nueva tarea:

$\mathcal{U}_{\tau'} = \mathcal{U}_\tau + C_{n+1}/P_{n+1}$
$=$ $\{$Definición de $C_{n+1} y P_{n+1}\}$
$$\mathcal{U}_\tau + \frac{(1 - \mathcal{U}_\tau) \cdot \mathcal{P}_\tau}{\mathcal{P}_\tau}$$
$$1$$
□

El anterior lema aporta una solución bastante burda de cómo utilizar el tiempo ocioso del procesador. De hecho, si aplicáramos este resultado para ejecutar carga aperiódica, obtendríamos el servidor en en segundo plano[2]. El siguiente lema refina este primer resultado.

[2] background server

Lema 4.2 *Sea τ un conjunto de tareas planificable. Supongamos que existe t_1, tal que $G_\tau(t_1) < t_1$. Entonces existe una tarea, T_{n+1}, tal que el nuevo conjunto de tareas $\tau' = \tau \cup T_{n+1}$ es planificable y $\mathcal{U}_\tau \leq \mathcal{U}_{\tau'}$*

DEMOSTRACIÓN: Sea $C_{n+1} \leq (t_1 - G_\tau(t_1))$, $D_{n+1} = t_1$ y $P_{n+1} = t_1$. Primero demostraremos que el ICI del nuevo conjunto es menor o igual que el intervalo $[0, t_1)$.

$G_{\tau'}(t_1) = \{$Propiedad 4.4$\}$

$$G_\tau(t_1) + C_{n+1}$$

$\leq \{$De la definición de $C_{n+1}\}$

$$G_\tau(t_1) + t_1 - G_\tau(t_1)$$

$$t_1$$

Y por la Propiedad 4.5 llegamos a que $\mathcal{R}' \leq t_1$.

Ahora comprobemos que el conjunto τ' cumple el teorema 3.4: $\forall_{t \leq t_1},\ H_{\tau'}(t) \leq t$

- Si $t < t_1$
  $H_{\tau'}(t)$
  $= \{$Por la Propiedad 4.3$\}$
  $$H_\tau(t)$$
  $\leq \{$Puesto que τ es planificable$\}$
  $$t$$
- Si $t = t_1$
  $H_{\tau'}(t_1)$
  $= \{$Por la Propiedad 4.3$\}$
  $$H_\tau(t_1) + C_{n+1}$$
  $\leq \{$De la definición de $C_{n+1}\}$
  $$H_\tau(t_1) + t_1 - G_\tau(t_1)$$
  $\leq \{$Por la Propiedad 4.2$\}$
  $$H_\tau(t_1) + t_1 - H_\tau(t_1)$$
  $$t_1$$

Tan sólo queda por comprobar que el factor de utilización del nuevo conjunto de tareas es mayor que el factor original. La utilización del nuevo conjunto es: $\mathcal{U}_\tau + \frac{t_1 - G_\tau(t_1)}{t_1}$. Puesto que $G_\tau(t_1)$ es menor estricto que t_1, tenemos que $\frac{t_1 - G_\tau(t_1)}{t_1}$ es un número positivo mayor que cero.

□

Como ya hemos visto, podemos interpretar la función $H_\tau(t)$ como la cantidad de tiempo que han *necesitado* todas las activaciones cuyo plazo es anterior al instante t. Por tanto, la diferencia entre $H_\tau(t)$ y t podemos considerarla como tiempo ocioso –más concretamente representa la cantidad de tiempo *no necesaria* para finalizar con éxito todas las activaciones con plazo menor o igual que que t. Si el tiempo t coincide con el plazo de alguna activación entonces $H_\tau(t) - t$ es el tiempo de holgura que se le puede quitar a esta activación.

Por ahora, aún no podemos saber qué puede suceder con las activaciones que se ejecuten después del instante t ¿ Podrán cumplir sus plazos de ejecución ? En el siguiente lema se profundiza en estas ideas.

Lema 4.3 *Sea τ un conjunto de tareas planificable. Sean t_1 y t_2 dos instantes tales que:*

1. $t_2 \leq t_1$
2. $G_\tau(t_1) < t_1$
3. $\forall_{t_2 \leq t \leq t_1},\ (t_1 - G_\tau(t_1)) \leq (t - H_\tau(t))$.

Entonces existe una tarea T_{n+1} con las características: $C_{n+1} \leq t_1 - G_\tau(t_1)$, $D_{n+1} = t_2$, y $P_{n+1} = t_1$.

Y el nuevo conjunto $\tau' = \tau \cup T_{n+1}$ es planificable y $\mathcal{U}_\tau \leq \mathcal{U}_{\tau'}$.

DEMOSTRACIÓN: La demostración de que el intervalo $[0, t_1)$ es menor que el ICI es similar a la empleada en el lema anterior.

Para demostrar que τ' es planificable tendremos que verificar que $\forall t \leq t_1,\ H_{\tau'}(t) \leq t$
Si $t < t_2$ {Por la Propiedad 4.3, y puesto que τ es planificable}

$$H_{\tau'}(t) = H_\tau(t) \leq t$$

Si $t_2 \leq t \leq t_1$ {Por la Propiedad 4.3}

$H_{\tau'}(t) = H_\tau(t) + C_{n+1}$
$\leq$ {De la definición de C_{n+1}}

$$H_\tau(t) + t_1 - G_\tau(t_1)$$

$\leq$ {Por las condiciones del lema}

$$H_\tau(t) + t - H_\tau(t)$$

$$t$$

□

Desde el punto de vista de la utilización del procesador, las dos nuevas tareas que aparecen el los lemas 4.2 y 4.3 son equivalentes.

Todos los anteriores lemas suponen la existencia de algunas condiciones temporales a partir de las cuales se obtienen los resultados. En el siguiente lema veremos como estas condiciones si que existen.

Lema 4.4 *Sea W un número real tal que $\mathcal{U}_\tau < W \leq 1$. Entonces*

- *El instante t_1 supuesto en el lema 4.2 existe y es menor que* $\min\left\{\frac{\sum_{i=1}^{n} C_i}{1-W}\ ,\ \mathcal{P}_\tau\right\}$, *y*
- $W \leq \mathcal{U}_{\tau'}$.

DEMOSTRACIÓN: La función $G_\tau(t)$ esta acotada por $\sum_{i=1}^{n} C_i + t \cdot \mathcal{U}_\tau$.

$G_\tau(t) = \sum_{i=1}^{n}\left(C_i \left\lceil \frac{t}{P_i} \right\rceil\right)$
$\leq$ {Puesto que $\lceil x \rceil \leq x + 1$}

$$\sum_{i=1}^{n} C_i \left(\frac{t}{P_i} + 1\right)$$

$=$ {De la definición de $\mathcal{U}_\tau$}

$$\sum_{i=1}^{n} C_i + t \cdot \mathcal{U}_\tau$$

El factor de utilización del procesador del nuevo conjunto de tareas es:

$W = \mathcal{U}_{\tau'} = \mathcal{U}_\tau + \frac{C_{n+1}}{P_{n+1}}$
$=$ {De la definición de T_{n+1}}

$$\mathcal{U}_\tau + \frac{t_1 - G_\tau(t_1)}{t_1}$$

$\equiv$ {Expresando G_τ como función de $W, \mathcal{U}_\tau$ y t_1}

$$G_\tau(t_1) = t_1 \cdot (1 + \mathcal{U}_\tau - W)$$

Utilizando estos dos resultados :

$t_1 \cdot (1 + \mathcal{U}_\tau - W) \leq \sum_{i=1}^{n} C_i + t \cdot \mathcal{U}_\tau$

$$t_1 \leq \frac{\sum_{i=1}^{n} C_i}{(1 - W)}$$

Directamente aplicando el lema 4.1, se puede añadir una tarea con periodo $P_{n+1} = D_{n+1} = \mathcal{P}_\tau$ y tiempo de cómputo $C_{n+1} = (1 - \mathcal{U}_\tau) \cdot \mathcal{P}_\tau$ para alcanzar un factor de utilización del 100%

$\square$

El lema 4.2 indica como localizar tiempo ocioso del procesador, y por el lema 4.3 sabemos qué condiciones se tienen que cumplir para poder emplear esa cantidad de tiempo sin poner en peligro las tareas periódicas del sistema. Nos falta por desarrollar un método para extraer ese tiempo ocioso de la ejecución de las tareas periódicas. Las siguientes funciones permiten obtener una tabla con los instantes y la cantidad de tiempo que se puede substraer para dedicarlo a ejecutar cualquier otra tarea.

Sea t_1 un instante de tiempo tal que $G_\tau(t_1) < t_1$. Definiremos la función β_j como:

$$\beta_j = \begin{cases} j = 0 & t_1 \\ \\ j > 0 & \max(t) : (t < \beta_{j-1}) \wedge (t - H_\tau(t) < \beta_{j-1} - H_\tau(\beta_{j-1})) \end{cases}$$

Obsérvese que β depende de τ y t_1, pero se han omitido de los parámetros de función para simplificar la sintaxis. Esta función está acotada pues es decreciente mayor que cero. Los valores de β nos indican los instantes en los que la cantidad de tiempo ocioso se reduce.

El último valor de j se produce cuando $t - H_\tau(t) = 0$. Sea k este último valor.

La función $\Delta(j)$ se define como:

$$\Delta(j) = \begin{cases} j = 0 & \beta_0 - G_\tau(\beta_0) \\ \\ j > 0 & \beta_j - H_\tau(\beta_j) \end{cases}$$

Con esta función obtenemos la cantidad de tiempo ocioso disponible en los intervalos $[0, \beta_i]$. Finalmente, θ se define como un conjunto de tareas, $\{T_{n+1}, T_{n+2}, \ldots, T_{n+k}\}$, que hace uso de los tiempos libres:

$$\begin{aligned} C_{n+j} &= \Delta(j-1) - \Delta(j) \\ D_{n+j} &= \beta_j + c_{n+j} \\ P_{n+j} &= t_1 \end{aligned}$$

Tomando la carga que aparece en el ejemplo de este capítulo (tabla 4.1), la figura 4.5 presenta una representación gráfica de la función $H_\tau(t)$ y el método geométrico para calcular los valores de la función β. Los rectángulos negros de la base de la figura representan los tiempos de cómputo de las tareas que componen el conjunto θ.

El cálculo de θ se puede desdoblar en dos fases: (1) primero buscar hacia adelante la suficiente cantidad de tiempo libre, a la vez que se obtienen los valores de la función H_τ; (2) en la segunda fase se "baja" hasta el instante inicial intentando "acercar al origen" la mayor cantidad posible de todo el tiempo ocioso encontrado, aquí se calculan los vectores β, Δ y finalmente θ, valores que se pueden obtener simultáneamente.

El siguiente teorema representa la principal aportación de este capítulo. En él, se demuestra que el conjunto de tareas θ es el el más restrictivo que se puede añadir al conjunto original τ conservando la planificabilidad de todas las tareas.

Teorema 4.1 *Sea τ un conjunto de tareas planificable. Sea W un número real tal que, $\mathcal{U}_\tau < W \leq 1$. Entonces existe un tiempo $t_1 \leq \min\{\sum_{i=1}^{n} C_i/(1-W)\ ,\ \mathcal{P}_\tau\}$ y un conjunto de tareas $\theta = \{T_{n+1}, T_{n+2}, \cdots, T_{n+k}\}$ tal que :*

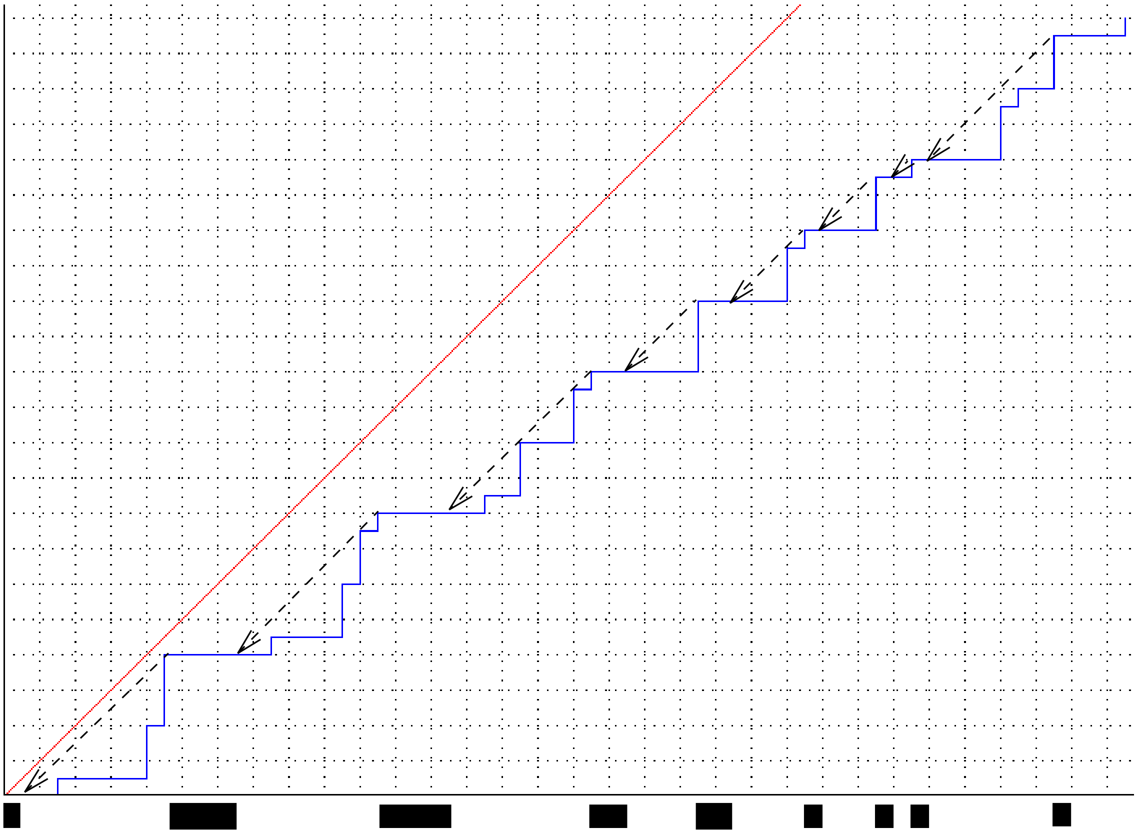


Figura 4.5: Valores de la función β.

1. *El nuevo conjunto de tareas $\tau^k = \tau \cup \theta$ es planificable, y*
2. *$\{D_{n+1}, \cdots, D_{n+k}\}$ son los plazos más cortos que garantizan la planificabilidad de τ^k.*

La demostración de este teorema se encuentra en el apéndice A.

En la figura 4.6 se muestra el conjunto de tareas al que se le ha añadido el conjunto θ. Éste ha sido calculado con un valor de $W = 1$.

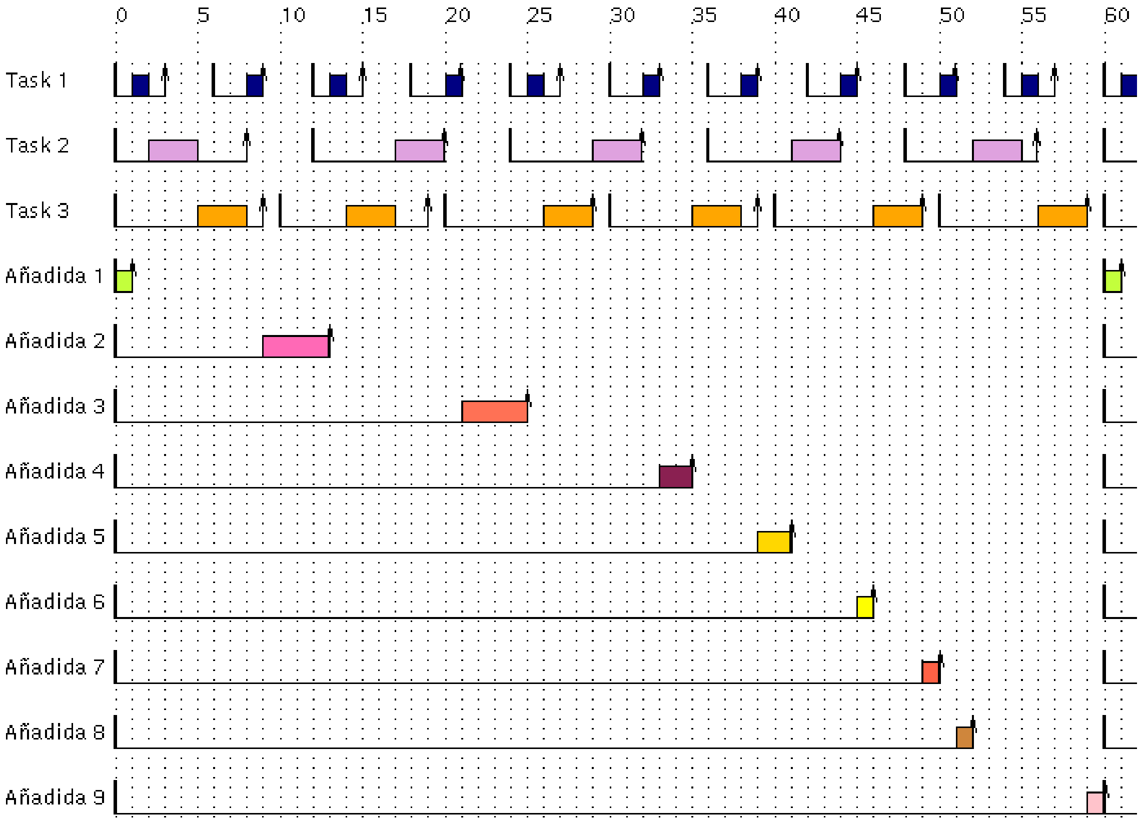


Figura 4.6: Ejemplo de planificación conjunta de τ y θ

Obsérvese como las posiciones de las nuevas tareas coinciden con los rectángulos negros obtenidos en gráfica 4.5.

Corolario 4.1 *Eligiendo $W = 1$ en el teorema anterior, se obtiene un nuevo conjunto de tareas con periodo igual al hiperperiodo, y un factor de utilización total del 100%.*

El conjunto de tareas obtenido con $W = 1$ se puede utilizar de forma muy simular a como se emplea la tabla de holgura en el algoritmo del servidor de holgura[3], tal como veremos en la sección 4.4. Pero en este caso para planificadores óptimos.

En lo que resta de capítulo se hará referencia a θ como tabla de holgura o nuevo conjunto de tareas indistintamente. Igualmente, supondremos que $W = 1$.

[3] Slack Server [Lehoczky92]

4.3 Cálculo de la tabla de holgura

En la sección anterior se han obtenido las condiciones necesarias y suficientes, así como el método para construir la tabla de holgura. En esta sección presentamos los algoritmos derivados de esta teoría para obtener la tabla de holgura.

Los pasos a seguir son: verificar la planificabilidad del conjunto de tareas original, calcular las funciones H_τ, β, Δ y finalmente obtenerla tabla θ.

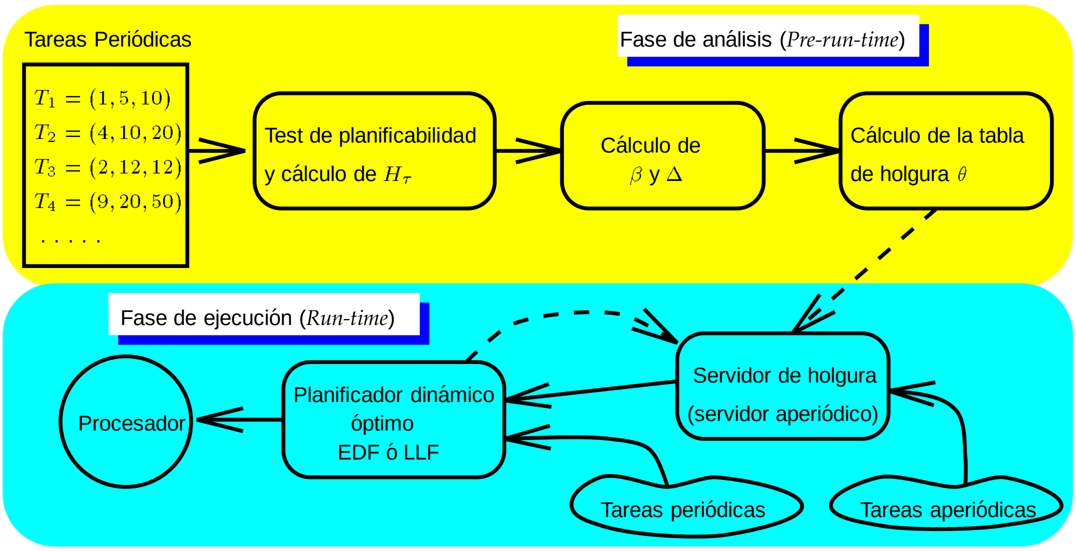


Figura 4.7: Diagrama de flujo desde la descripción de las tareas hasta la planificación.

El primero de los pasos es determinar si el conjunto original de tareas es planificable. Para ello disponemos del algoritmo desarrollado en el capítulo 3. Lo modificaremos ligeramente para que a la vez que verifica la planificabilidad, calcule las funciones G y H.

El algoritmo espera como parámetros el conjunto de tareas periódicas y el nivel de utilización que se quiere alcanzar. Y devuelve si el sistema es planificable o no, el tiempo t_1, y dos vectores con los pares de puntos que definen las funciones G y H. En estos vectores sólo se almacena el instante en el que la función cambia de valor y el nuevo valor que toma (en la componente x se almacena el instante de cambio y en la componente y el nuevo valor).

Algoritmo 4.1 (Obtención de los vectores H y G)

proc $Step1(\tau, W)\ \equiv$
  $Output : G[], H[], H_{len}, G_{len}$
  var $i_H, i_G, t;$
  end
  begin
    $U_\tau := \sum_{i=1}^{n} \frac{C_i}{P_i}$
    $G[0] := (0, \sum_{i=1}^{n} C_i);$
    $H[0] := (0, 0);$

```
    i_H := i_G := t := 0;
    while (W ≥ (U_τ + (t−G[i_G].y)/t)) ∧ (t ≥ H[i_H].y) do
        t := t + 1;
        if (G_τ(t) ≠ G[i_G].y)
          then
            i_G := i_G + 1;
            G[i_G] := (t, G_τ(t));
        fi
        if (H_τ(t) ≠ H[i_H].y)
          then
            i_H := i_H + 1;
            H[i_H] := (t, H_τ(t));
        fi
    od
    if (t ≥ G[i_G].y) ∧ (t ≥ H[i_H].y)
      then Planificable;
       else No_planificable;
    fi
    t_1 := t − 1;
  end
.
```

Este algoritmo finaliza cuando o bien detecta que el sistema no es planificable o bien encuentra suficiente tiempo ocioso con el que conseguir la utilización pedida.

El siguiente algoritmo utiliza los vectores $G[\,]$, $H[\,]$ y el instante t_1 para obtener los vectores $\beta[\,]$, $\Delta[\,]$ y la tabla $\theta[\,]$. Puesto que todas las tareas de θ tienen el mismo periodo, cada componente del vector $\theta[\,]$ sólo tiene dos campos $\theta[i].C$ y $\theta[i].D$, que son el tiempo de cómputo y el plazo de ejecución respectivamente. También retorna en la variable θ_p el periodo de todo el nuevo conjunto, y en θ_c el tiempo total de ejecución de todas las tareas.

Algoritmo 4.2 (Cálculo de las matrices β, Δ y θ)

```
proc Step2(G[], H[], G_len, H_len, t_1) ≡
  Output : β[], Δ[], β_size
  var Slack, last_slack, j, i; end
  begin
    j := 1;
    i := H_len;
    last_slack := t_1 − G[G_len].y;
    β[0] := t_1;
    Δ[0] := last_slack;
    while (i ≥ 0) do
        Slack := H[i].x − H[i].y;
        if (Slack < last_slack)
          then
            β[j] := H[i].x;
            Δ[j] := Slack;
            last_slack := Slack;
            j := j + 1;
        fi
        i := i − 1;
```

od
if $(\mathit{Slack} \neq 0)$
then
$\beta[j] := \Delta[j] := 0;$
$j := j + 1;$
fi
$\beta_{size} := j - 1;$
for $i := 0$ **to** $\beta_{size} - 1$ **do**
$\theta[i].C := \Delta[\beta_{size} - i - 1] - \Delta[\beta_{size} - i];$
$\theta[i].D := \beta[\beta_{size} - i].x + \theta[i].C;$
od
$\theta_{size} := \beta_{size};$
$\theta_c := \theta[\theta_{size}].C;$
$\theta_p := t_1;$
end
.

El vector θ se almacena para poder ser utilizado por el algoritmo de aceptación de tareas durante la ejecución del sistema.

4.4 Aceptador Aperiódico Estático (AAE)

En esta sección se describe el funcionamiento del algoritmo de aceptación, y se presenta una implementación en pseudo-código.

El algoritmo de aceptación en lugar de añadir el conjunto θ de tareas utiliza esta información para decidir como servir las peticiones aperiódicas. El vector θ se puede interpretar como una lista de instantes en los que se puede substraer cierta cantidad de tiempo de las tareas periódicas. Cada nueva tarea aperiódica que llega al sistema es recogida por el AAE, el cual, tras consultar el vector θ, le asigna un plazo de terminación, convirtiéndose de esta forma en una tarea "urgente". Finalmente, tras esta conversión la tarea es enviada a la cola del planificador como si de una tarea periódica se tratara. El algoritmo de planificación la ordenará como cualquier otra tarea, y cuando sea la más prioritaria la ejecutará. Obsérvese que el algoritmo AAE no modifica el funcionamiento del algoritmo de planificación.

Antes de entrar a describir el algoritmo AAE, hay que considerar algunas cuestiones que surgen al interpretar el conjunto θ como una tabla de holgura.

Cada vez que el procesador quede ocioso durante la ejecución, la cantidad de tiempo de holgura disponible se reduce. Por consiguiente, el efecto que producen los intervalos ociosos es el mismo que el producido por la llegada de peticiones de trabajo aperiódico.

Si la carga de trabajo aperiódico es alta, todo el tiempo de holgura almacenado en la tabla se emplea tan pronto como es posible (esto es, justo en los instantes marcados por los plazos de la tabla θ).

El problema surge cuando no se producen peticiones aperiódicas mientras hay disponible tiempo de holgura. En caso de que hubieran tareas periódicas activas en ese momento, el planificador no dejará el procesador ocioso y ejecutará en su lugar tareas periódicas; con lo que el tiempo de holgura no se desaprovecha, sino que se retrasa. El momento en el que nuevamente se podrá hacer uso de este tiempo de holgura depende de qué tarea/s periódica/s hayan sido ejecutadas en lugar de la carga aperiódica esperada.

Llamaremos *tiempo de holgura acumulado* a los intervalos de holgura que han sido consumidos por tareas periódicas y que volverán a estar disponibles cierto tiempo después.

Supongamos el siguiente escenario: en el instante t_h hay disponibles k u.t. de holgura (esto es, hay en θ una entrada, i, con el valor: $\theta[i].D = t_h + k$ y $\theta[i].C = k$); llegado el instante t_h de ejecución, no hay ninguna tarea aperiódica pendiente (y tampoco llega ninguna nueva petición aperiódica), pero si que hay activaciones de tareas periódicas esperando completar su ejecución. Supongamos que tarea que el planificador elegirá para ejecutar es T_j[4] por lo que esta activación adelantará parte de su ejecución, dejando libre esta misma cantidad de tiempo más adelante. En el peor de los casos, las k unidades de tiempo volverán a estar disponibles justo antes de finalizar el plazo de ejecución de T_j, d_j. De hecho, la cantidad total de carga pedida al procesador hasta el instante d_j es independiente de donde se ejecuten las k unidades de tiempo de holgura.

Por tanto, si alguno de los intervalos de holgura es consumido por alguna tarea periódica, será necesario recalcular la tabla de holgura hasta el plazo de la tarea cuyo plazo sea el más largo de las que se han ejecutado. La relocalización de la holgura no es necesario hacerla cada vez que una tarea periódica consuma tiempo de holgura, sino sólo cuando lleguen tareas aperiódicas.

Para poder recalcular la holgura necesitamos de una función similar a $H_\tau(t)$, pero que considere el tiempo que ya ha sido ejecutado por tareas con plazos más largos. Esta función es:

$$H_\tau^{t_{now}}(t) = \sum_{i=1}^{n} C_i \cdot \left\lfloor \frac{(t + P_i - D_i - n_{i,t_{now}})_0}{P_i} \right\rfloor + \sum_{i=1}^{n} c_{i,t_{now}} \cdot \left\lfloor \frac{(2t - d_{i,t_{now}})_0}{t} \right\rfloor$$

donde t_{now} representa el instante de tiempo actual; $n_{i,t_{now}}$ es el instante de la siguiente activación de la tarea i respecto al instante actual (se puede obtener como: $t_{now} \bmod P_i$); $c_{i,t_{now}}$ es la cantidad de tiempo de cómputo que resta por consumir de la última activación de la tarea i, en caso de aún no haber finalizado; y finalmente, $d_{i,t_{now}}$ es el plazo de ejecución, relativo a t_{now}, de la activación actual de la tarea i (se puede obtener bien la cola del planificador, o como $(t_{now} \bmod P_i) - (P_i - D_i)$). En el capítulo 5 se discute más detalladamente el significado de esta función.

Seguidamente se describe el método para relocalizar la holgura consumida por tareas periódicas.

Algoritmo 4.3 (Relocalización de holgura)

```
proc Relocate(C_a, d_max) ≡
  var
    t;
  end
```

[4] Si suponemos que el planificador es el EDF, T_j es la tarea con el plazo más próximo

begin
  $t := d_{max} - t_{now};$
  if $(t \leq C_a)$ **then** **exit**$(t_{now} + C_a);$ **fi**
  repeat
    $t := t - 1;$
  until $(t - H_{\tau}^{t_{now}}(t) \geq C_a);$
  exit$(t_{now} + t + 1);$
end
.

Se puede mejorar la implementación de este algoritmo comprobando la condición del bucle sólo en los valores de t que coinciden con el plazo de alguna tarea, pues son éstos los puntos en los que la función $H_{\tau}^{t_{now}}$ modifica su valor.

El AAE está dividido en dos partes: (1) algoritmo de descuento de tiempo ocioso, y (2) algoritmo de cálculo y asignación de plazo a las peticiones aperiódicas.

Para simplificar el pseudo-código de los algoritmos supondremos que:

- La variable *Idle_Time* contiene la cantidad de tiempo que el procesador ha estado ocioso. El sistema operativo se encarga de incrementarla convenientemente.
- Abusando de la notación, diremos que $\theta(t_i, t_j)$ es la cantidad de tiempo de holgura que hay entre los instantes t_i a t_j. Se obtiene accediendo a la tabla θ. El coste temporal en el peor de los casos es del número de intervalos de holgura haya entre t_i y t_j.
- La función *Insert_into_sched_queue(*J_a, D*)* manda la tarea J_a a la cola del planificador. Este planificador como ya se ha dicho puede ser cualquier planificador óptimo, tal como el EDF o el LLF. En caso que el planificador elegido fuera el EDF esta función sólo tendría que insertar en una lista ordenada por plazo.
- La variable d_{max} guarda el plazo de la tarea con plazo más largo de entre las que se han ejecutado. Es actualizado por el planificador.

La variable *Last_Time* es el último instante consumido de la tabla de holgura. Todas las entradas de la tabla anteriores a *Last_Time* has sido consumidas, bien atendiendo a peticiones aperiódicas, bien desperdiciadas en tiempo ocioso del procesador, o bien ejecutando trabajo periódico y generando tiempo acumulado.

La variable *Acumul* contabiliza la cantidad de tiempo de holgura acumulado. Para hacer uso de este tiempo es necesario saber dónde está localizado.

Algoritmo 4.4 (Algoritmo de Actualización)

proc $Update \equiv$
  $Global$: *Last_Time*, *Idle_Time*;
  var
    temp;
  end

```
begin
  if (Acumul ≥ Idle_Time)
    then
        Acumulated := Acumulated − Idle_Time;
        Idle_Time := 0;
    else
        Idle_Time := Idle_Time − Acumulated;
        Acumulated := 0;
        Sea temp tal que θ(Last_Time, temp) = Idle_Time;
        Last_Time := temp;
  fi
end
.
```

Algoritmo 4.5 (Aceptador Aperiódico Estático)

```
proc AAE(t_now, θ[], J_a) ≡
  Global : Last_Time, Acumul;
  var
    Deadline, temp;
  end
  begin
    comment: Contabilizar tiempo de holgura consumido por tareas peri
    if (t_now > last_time)
      then
          Acumul := Acumul + θ(Last_Time, t_now);
          last_time := t_now;
    fi
    if (θ(Last_Time, Last_Time + C_a) = C_a)
      then
          Last_Time := max{Last_Time, t_now} + C_a;
          Insert_into_sched_queue(J_a, Last_Time);
      else
          comment: Sino, buscar en la tabla la cantidad pedida
          Sea temp tal que θ(Last_Time, temp) = (C_a − Acumul)_0;
          if (temp ≥ d_max)
            then
                comment: No ha sobrado tiempo acumulado
                Deadline := max{temp, t_Now + C_a};
                Acumul := 0;
            else
                comment: Ajustar la variable Acumul
                Deadline := Relocate(C_a, d_max);
                Acumul := Acumul − (C_a − θ(last_time, Deadline))_0;
          fi
    fi
    if (deadline > last_time) then last_time := deadline fi
    Last_Time := Deadline;
    Insert_into_sched_queue(J_a, Deadline);
  end
.
```

Primero se comprueba si se ha producido tiempo de holgura acumulado desde la última vez que el procesador quedó ocioso o se aceptó una tarea aperiódica. Después se comprueba si la cantidad de tiempo que resta por utilizar del último hueco de holgura es mayor que el tiempo pedido por la tarea aperiódica (línea 15). En caso de que así sea, se substraen C_a unidades de tiempo de este hueco y se le asignan a la tarea aperiódica el plazo más estricto posible, $t_{now} + C_a$. Si la cantidad de tiempo de cómputo pedida es superior, entonces se ha de recurrir a la tabla de holgura con el fin de obtener el tiempo necesario para acabar de completar todo el tiempo requerido. Finalmente, se comprueba si aún queda algo de tiempo de holgura acumulado, y se actualizan *Last_Time* y *Acumul*.

Ejemplo En este ejemplo se ilustra el funcionamiento del AAE. El vector θ correspondiente al conjunto de tareas ejemplo de este capítulo es:

θ	1	2	3	4	5	6	7	8	9
D	1	13	25	35	41	46	50	52	60
C	1	4	4	2	2	1	1	1	1

Supongamos que llegan al sistema tres peticiones aperiódicas: $J_a = (3, 1)$, $J_b = (9, 3)$ y $J_c = (15, 6)$. El resultado de la ejecución puede verse en la figura 4.8.

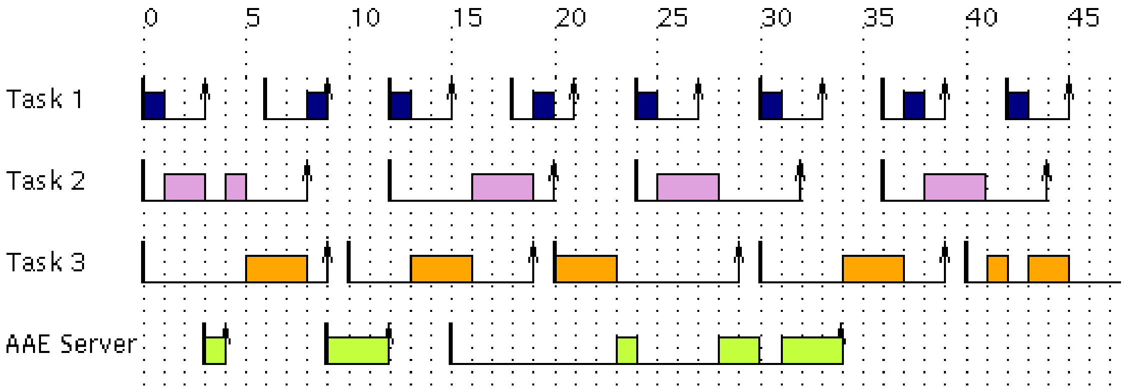


Figura 4.8: Ejemplo de funcionamiento del Aceptador Aperiódico Estático.

En el instante 3, la tarea J_a pide una unidad de tiempo. En ese instante se dispone de una unidad de tiempo acumulado y se llama a la función *Relocate* con los parámetros $C_a = 1$ y $d_{max} = 9$. Función que retorna el valor 4, que es el plazo que finalmente se le asigna a J_1. No queda tiempo acumulado sobrante.

La segunda petición aperiódica llega en el instante 9. En ese momento, hay suficiente tiempo de holgura directamente utilizable para servir inmediatamente la tarea periódica. El plazo asignado es $12 = t_{now} + C_a$.

Finalmente, en el instante 15 llega la última petición. Hay en este instante una unidad de tiempo acumulado y el plazo más largo de las tareas periódicas que se han ejecutado, $dmax$, es de 19 u.t., con lo que hay que buscar en la tabla de holgura $5 = C_a - 1$ u.t.. Se encuentran las 5 u.t. en el intervalo $[15, 34)$, y puesto que el intervalo contiene a d_{max}, el plazo asignado a J_3 es 34 y no hay tiempo acumulado sobrante.

En la siguiente figura (figura 4.9) se pueden apreciar las consecuencias de intentar reducir el tiempo de respuesta de las tareas aperiódicas disminuyendo el plazo que el AAE les asigna. Concretamente, se ha reducido en una u.t. el plazo asignado a la tarea J_3, en lugar del valor 34 se le ha asignado 33.

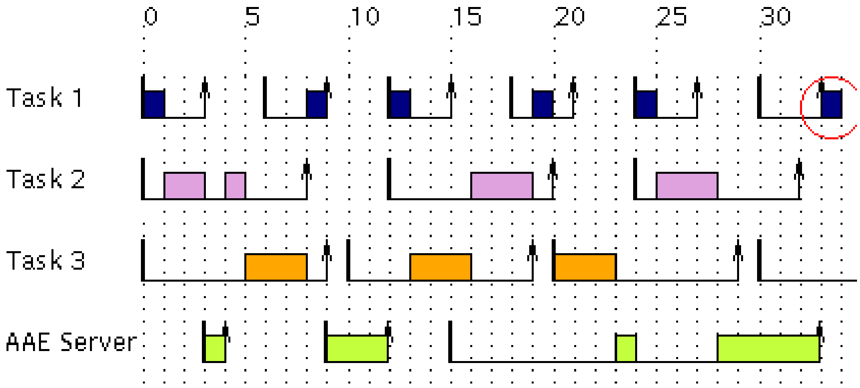


Figura 4.9: Los plazos calculados por el AAE son los más cortos compatibles con la correcta ejecución de todas las tareas.

La planificación por prioridades fijas no es capaz de ofrecer los mismos tiempos de respuesta a las tareas aperiódicas. En la figura 4.10 se puede observar el resultado de intentar planificar el sistema empleando el algoritmo de prioridad a la tarea más urgente[5].

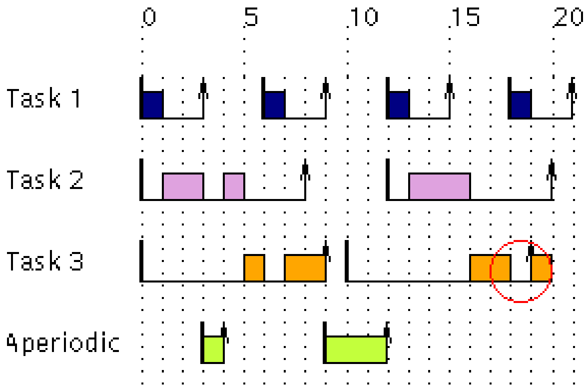


Figura 4.10: Planificación por prioridades fijas las tareas aceptadas por el AAE.

4.4.1 Complejidad

La complejidad del algoritmo AAE depende en parte de la frecuencia con la que hay que servir tareas aperiódicas. Por ello, el análisis lo realizaremos bajo tres supuestos: (1)

[5] Deadline Monotonic

no hay trabajo aperiódico; (2) siempre hay trabajo aperiódico que servir; (3) las peticiones aperiódicas llegan lo suficientemente espaciadas como para que se produzca tiempo acumulado.

Es importante destacar que si el sistema no recibe trabajo aperiódico, la complejidad del aceptador es $O(0)$, ya que el AAE sólo se invoca cuando llega una tarea aperiódica y el algoritmo de actualización se ejecuta sólo cuando el procesador está ocioso, por lo que no añade ninguna sobrecarga y consiguientemente no se considera su complejidad. Otras propuestas de servidores aperiódicos (servidor esporádico o el servidor por intercambio de prioridades) han de mantener constantemente actualizadas las estructuras de datos que emplean, lo que produce una permanente sobrecarga en el sistema aún sin tener que dar servicio a trabajo aperiódico.

En el segundo supuesto el AAE nunca necesita invocar la función *Relocate*, con lo que el coste se reduce a buscar en la tabla de holgura la cantidad de tiempo requerido por la tarea aperiódica. Para ello, se evalúa la expresión $\theta($*Last_Time, Last_Time*$+C_a)$. Lo que tiene un coste de $O(C_a)$, puesto que si suponemos que cada entrada de la tabla de holgura representa tan sólo una unidad de tiempo, el número máximo de entradas en la tabla que hay que recorrer es C_a. Empleando una búsqueda dicotómica en lugar de lineal se puede obtener un coste logarítmico, quedando el coste en $O(\log(C_a))$.

En el tercer supuesto, el AAE ha de recalcular dinámicamente la holgura. El coste de esta operación es de $O(C_a \sum_{i=1}^{n} C_i \frac{1}{1-\mathcal{U}_\tau})$.

Otro aspecto muy importante a tener en cuenta es la complejidad espacial, esto es, la cantidad de memoria necesitada. A excepción de las pocas variables escalares empleadas, la única estructura de datos que consume memoria es la tabla θ. En el peor de los casos la talla de esta tabla de del $O(N)$, donde N es el número de activaciones contenidas en un hiperperiodo.

Es evidente que N puede llegar a ser excesivamente grande. N depende en gran medida de lo grande que sea el hiperperiodo, y éste depende a su vez los valores de los periodos de las tareas y si los periodos son primos entre sí[6].

4.5 Extensiones

Varias son las mejoras que se le pueden realizar sobre algoritmo básico de aceptación.

Dado que todas las tareas críticas tienen que cumplir bajo cualquier circunstancia sus plazos de ejecución. El análisis de planificabilidad se tiene que realizar considerando el tiempo de cómputo más pesimista, a pesar de que este caso sea muy improbable. Por consiguiente, no es de extrañar que las tareas críticas consuman, durante su ejecución normal, menos tiempo del que inicialmente se les ha asignado. A este tiempo asignado pero no consumido se llama *tiempo sobrante*[7]. Una posible extensión del AAE consiste en reapro-

[6] La probabilidad de que dos números enteros elegidos al azar sean primos entre sí es aproximadamente del 60% (exactamente, la probabilidad es $6/\pi^2$) [Knuth69]

[7] Gain time

vechar este tiempo para mejorar el tiempo de respuesta de las tareas aperiódicas.

Otra mejora consiste en dar servicio a tareas aperiódicas con plazo de ejecución firme. Este tipo de tareas no son críticas, por lo que no ejecutarlas no conduce a una situación catastrófica, ahora bien, sólo es útil ejecutarlas si finalizan dentro de su plazo. Una tarea aperiódica con plazo firme sólo se ha de ejecutar si se puede garantizar que va a cumplir su plazo, en caso contrario, deberá rechazarse inmediatamente.

4.5.1 Reutilización de tiempo sobrante

La detección de tiempo sobrante se produce al finalizar la ejecución de cada tarea o parte de ella en caso que ésta esté preparada con puntos de comprobación intermedios (*checkpoints*). Por lo tanto, la diferencia entre el tiempo en el peor caso y el que realmente ha consumido cada una de las activaciones, puede utilizarse inmediatamente sin necesidad de ninguna comprobación.

Al finalizar cada tarea –o cada vez que se alcanza un punto de comprobación– el sistema incrementa la variable *Gain_Time* con la diferencia entre el tiempo consumido y el inicialmente asignado. El tiempo sobrante ha de ser contabilizado como si de tiempo acumulado se tratara. Por consiguiente, tanto el algoritmo de actualización como el de aceptación tienen que asignar el tiempo sobrante a la variable *Acumul* cada vez que son llamadas.

Hasta ahora hemos supuesto que no hay peticiones aperiódicas cuando se detecta el tiempo sobrante, por lo que era acumulado en *Gain_Time* para luego –cuando llegara una tarea aperiódica– ser consumido. Veamos ahora cuales son las acciones a tomar en caso de que el tiempo sobrante se generase mientras una o más tareas aperiódicas están en la cola del planificador. Supongamos que se generan e unidades de tiempo sobrante. Una eficiente aunque subóptima solución, consistiría en reducir el plazo asignado a todas las tareas aperiódicas en e unidades, con la consiguiente reordenación de la cola del planificador. En este caso lo que hacemos es consumir tiempo sobrante en lugar de tiempo de holgura, por lo que tenemos que reponer el tiempo de holgura, decrementando para ello la variable *Last_Time* en e unidades de tiempo. Esta solución, si bien es eficiente y fácil de implementar, no es la que mejor uso hace del tiempo sobrante. Otra solución mucho más costosa, pero que ofrece mejores tiempos de respuesta consiste en recalcular uno por uno todos los plazos asignados a tareas aperiódicas.

4.5.2 Tareas con plazo firme

Para el test de aceptación de tareas con plazo firme hay que considerar dos casos: por un lado, cuando la nueva tarea aperiódica ,J_a, tiene el plazo más tardío de entre todas las tareas aperiódicas con plazo firme activas; y por otro, cuando hay alguna tarea aperiódica activa con plazo mayor.

En el primer caso, el test consiste en intentar asignarle un plazo empleando para ello el algoritmo de aceptación. Si el plazo obtenido por el AAE es menor o igual al plazo requerido por la tarea, entonces se puede garantizar que la tarea cumplirá su plazo, y se puede insertar en la cola del planificador, ordenándose según el plazo de la tarea. Nota, el plazo calculado por el AAE sólo se emplea para decidir si es planificable o no, y no para asignárselo. Evidentemente, el tiempo de cómputo empleado por J_a ha de ser extraído de la tabla de holgura de igual forma que sucedía con las tareas aperiódicas sin plazo.

Con el fin de simplificar el test de aceptación de futuras peticiones, se guarda junto con cada tarea aceptada un parámetro adicional, E_a, que indica la cantidad de tiempo de holgura no empleado dentro de su plazo, esto es, la holgura de que dispone la tarea J_a para completar su ejecución, y que podrá ser asignada a otras tareas aperiódicas. Se obtiene como $E_a = \theta(\mathit{Last\_Time}, D_a)$.

Cuando la nueva tarea –con plazo firme– es más urgente que alguna de las tareas activas con plazo firme previamente aceptadas, el test de aceptación debe considerar las restricciones impuestas por éstas. Los pasos a seguir son:

1. Verificar que ninguna tarea aperiódica con plazo mayor perderá su plazo. Para ello, hay que recorrer la lista de tareas comprobando que el valor de E_i es mayor que C_a en aquellas tareas con plazo mayor que D_a. Tiene un coste de $O(f)$, donde f es el número de tareas tareas aperiódicas activas.

2. Comprobar si la nueva tarea podrá cumplir su plazo. Esto se puede realizar de la siguiente forma: suponiendo que no hay ninguna tarea aperiódica con plazo firme activa y comprobando si se puede aceptar una tarea con tiempo de cómputo igual a la suma de los tiempos de cómputo –que faltan por ejecutar– de todas las tareas aperiódicas con plazo firme activas con plazo menor o igual que D_a, incluyendo la propia tarea J_a.

3. Actualizar el tiempo de holgura E_i de las tareas con plazo mayor o igual que D_a. El valor de E_a se calcula de igual forma que en el caso anterior; y el valor de E_i de todas las tareas con plazo mayor que D_a se decrementa en C_a unidades.

Aperiódicas con plazo firme junto con aperiódicas

Veamos los cambios necesarios (recordemos que la aceptación de peticiones se realiza en orden FIFO):

- Si el plazo de la nueva tarea es menor que el plazo asignado de la última tarea sin plazo, entonces es seguro que no se le podría asignar un plazo menor o igual que su propio plazo, con lo que la nueva tarea ha de ser rechazada.

- Para asignar el plazo a las nuevas peticiones aperiódicas sin plazo. Primero se ha de comprobar si se le puede asignar un plazo menor al de alguna de las tarea con plazo firme activas. Se ha de recorrer en orden decreciente de plazos las tareas aperiódicas con plazo firme, hasta encontrar una con una holgura, E_i, menor que el tiempo de cómputo requerido. El plazo a asignar a J_a se situará entre las dos últimas tareas. La holgura de todas las tareas aperiódicas con plazo menor al asignado se pone a cero, mientras que a las tareas con plazo mayor se les decrementa E_i en C_a unidades.

En caso de no haber suficiente tiempo de holgura en la tarea con plazo más largo, se ha de buscar el tiempo directamente en la tabla de holgura, empleando para ello directamente el algoritmo AAE.

4.6 Comparación experimental

Puesto que vamos comparar las prestaciones de algoritmos óptimos tanto para planificadores estáticos (*Exact Slack Stealing* [Lehoczky92, Davis93]) como para dinámicos (AAE), los resultados que obtendremos medirán más que las características del algoritmo concreto que se utiliza para implementar el experimento, se obtendrán las características intrínsecas de los algoritmos de planificación en lo relativo a la gestión de trabajo aperiódico: DM vs. EDF.

En algunas gráficas se han incluido los resultados del servidor de segundo plano como referencia de las prestaciones del algoritmo menos costoso.

Primero describiremos el marco en el que se han desarrollado los experimentos, para luego pasar a presentar los resultados obtenidos.

4.6.1 Diseño de los experimentos

El criterio empleado para evaluar el rendimiento ha sido el tiempo medio de respuesta de las tareas aperiódicas.

Se ha elegido al EDF como algoritmo de planificación sobre el que se ha montado el AAE. Recuérdese que el AAE puede trabajar junto con cualquier algoritmo de planificación óptimo.

El tiempo de cómputo requerido por cada petición aperiódica es de 1 u.t.. Las peticiones aperiódicas están distribuidas uniformemente a lo largo de tolo el hiperperiodo. El número de peticiones varía en función de la carga que se necesite en cada simulación.

La forma de generar carga la sintética para los experimentos está basada en las simulaciones que Davis et al. [Davis93] emplean para comparar varios algoritmos de servicio aperiódico entre los que se incluye el extractor de holgura óptimo.

Se han generado tres tipos de carga periódica, cada una de ellas con un grado de utilización del procesador superior (50%, 70% y 90%). Cada tareas periódica se genera de la siguiente forma: primero se elige el periodo según la distribución empleada (exponencial o uniforme), se descarta en caso de ser igual que alguno de los periodos anteriormente elegidos; seguidamente se calcula el plazo de forma aleatoria pero restringido a ser menor que el periodo; finalmente, se obtiene el tiempo de cómputo, ésta a su vez inferior al plazo. Conjuntos de tareas con una utilización que difiera en un 1% respecto lo buscado son desechados. De los conjuntos de tareas así generados sólo se han simulado aquellos que eran planificables por el DM (evidentemente el EDF también podrá planificarlos). Se ha experimentado con conjuntos de 12 y 30 tareas. El valor del hiperperiodo también se ha tenido en cuenta, se han realizado experimentos limitando el valor del hiperperiodo a 5000, 10000, 50000 y 100000 u.t..

Las tareas periódicas consumen siempre todo el tiempo de cómputo,esto es, no se produce tiempo sobrante. La sobregarca relacionada con los algoritmos de ejecución no es tenida en cuenta.

El tiempo medio de respuesta empleado en la comparación se obtiene simulando la planificación durante todo el hiperperiodo. Nótese que sólo considerando todo el hiperperiodo se puede conocer cual es el nivel de utilización exacto producido por la carga aperiódica.

Cada uno de los puntos que aparecen en las gráficas representan la media de 50 simulaciones de distintos conjuntos.

4.6.2 Resultado experimentales

Las primeras conclusiones que se han obtenido de los experimentos han sido que el comportamiento (tiempo de respuesta a las tareas aperiódicas) permanece invariable ante el valor del hiperperiodo y el número de tareas (a partir de conjuntos con más de 6 o 7 tareas). Los conjuntos de tareas empleados en los experimentos aquí presentados esta compuesta por 12 tareas y con un hiperperiodo máximo de 50000 u.t..

En todas las gráficas, las líneas etiquetadas como "DM" representan los resultados del servidor de holgura para el DM; las etiquetadas con "BACK" son los resultados del servidor de segundo plano.

En la figura 4.11 se presentan los resultados del primer experimento. La carga periódica se ha fijado al 50% y la carga aperiódica se ha variado desde 1% hasta el 50%.

Tanto el DM como el AAE ofrecen un buen tiempo de respuesta (las peticiones se sirven inmediatamente) hasta llegar a una carga total del 82%, a partir de este momento el sistema comienza a saturarse y el tiempo de respuesta se incrementa de forma prácticamente lineal conforme aumenta el nivel de carga.

En la figura 4.12 se puede apreciar como el AAE consigue siempre menores tiempos de respuesta, si bien sólo en el transito de no saturado a saturado es cuando se obtienen los mejores resultados.

La carga periódica en el segundo experimento, gráficas 4.13 y 4.14, representa el 70% de la carga total. A este nivel de utilización, el procesador comienza a saturarse alrededor del 90%. Con esta carga la diferencia entre los planificadores estáticos y los dinámicos aparece más acusada, llegando a conseguir tiempos de respuesta hasta tres veces menores.

En el tercer experimento, ilustrado en las figuras 4.15 y 4.16, la carga periódica se ha incrementado hasta el 90%. Tal como se aprecia, con esta elevada carga el AAE ofrece tiempos de respuesta hasta 10 veces menores que los conseguidos en planificación estática.

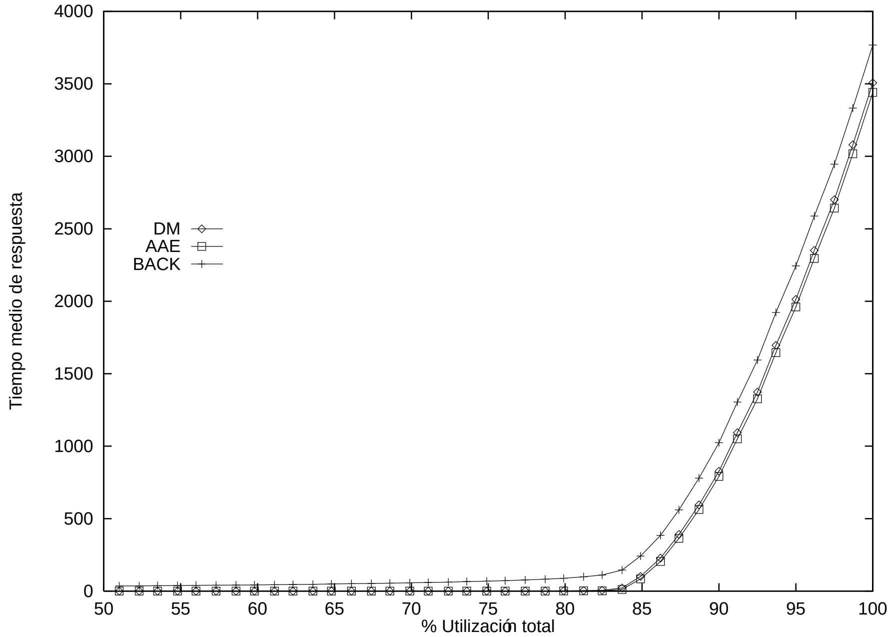


Figura 4.11: Comparación con baja carga periódica: 50%.

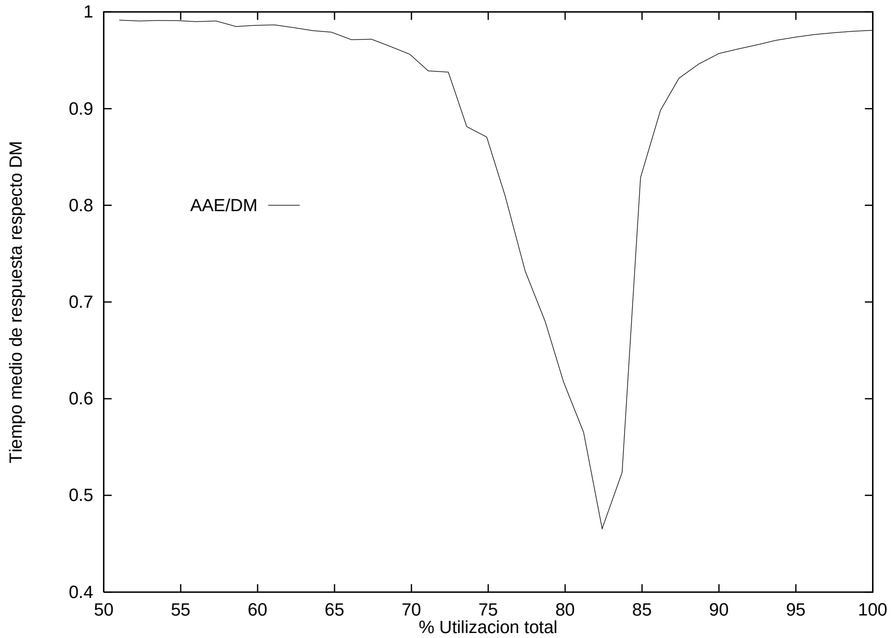


Figura 4.12: Comparación entre el DM y AAE, carga periódica 50%.

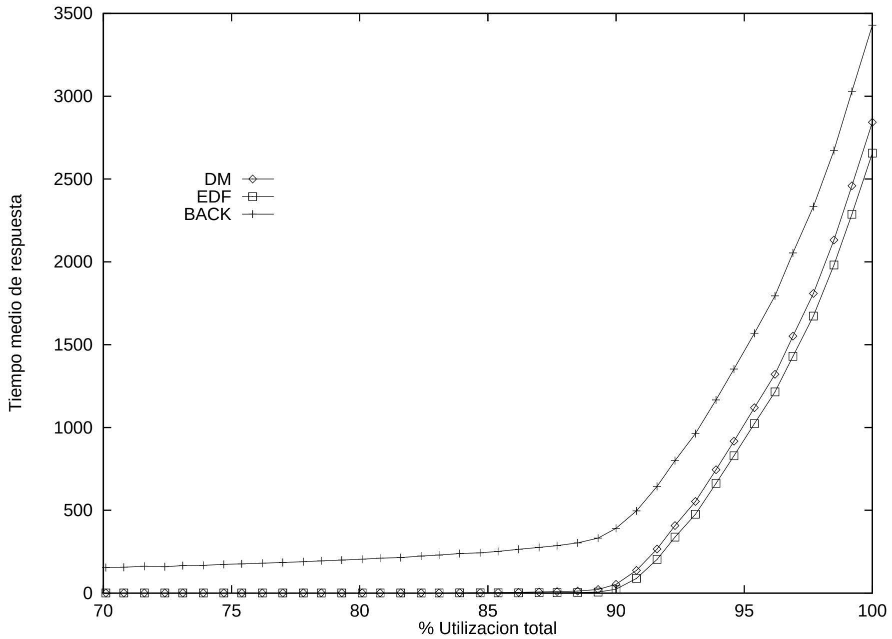


Figura 4.13: Comparación con baja carga periódica: 70%.

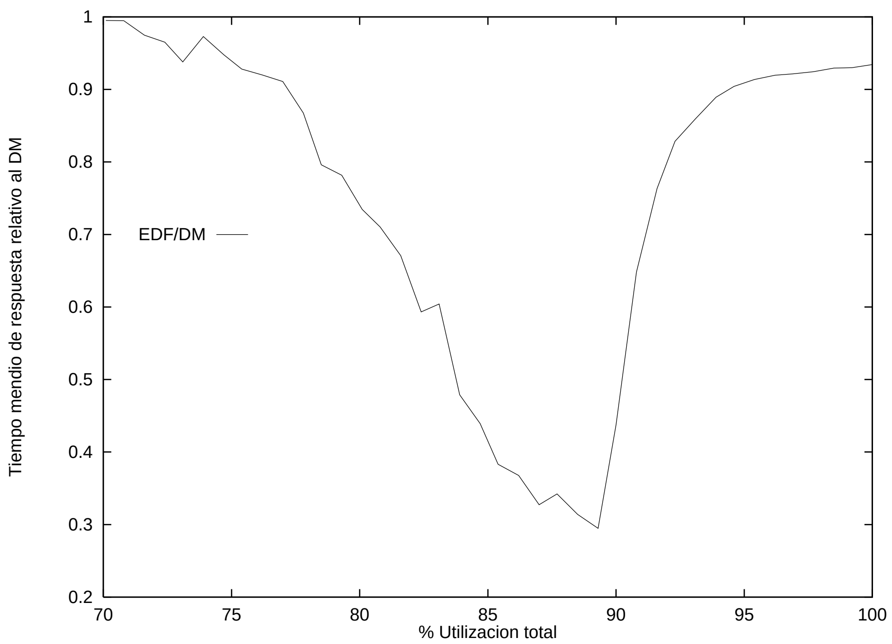


Figura 4.14: Comparación entre el DM y AAE, carga periódica 50%.

Por otra parte, el AAE es capaz de mantener un tiempo de respuesta casi inmediato hasta alcanzar el 96% de carga.

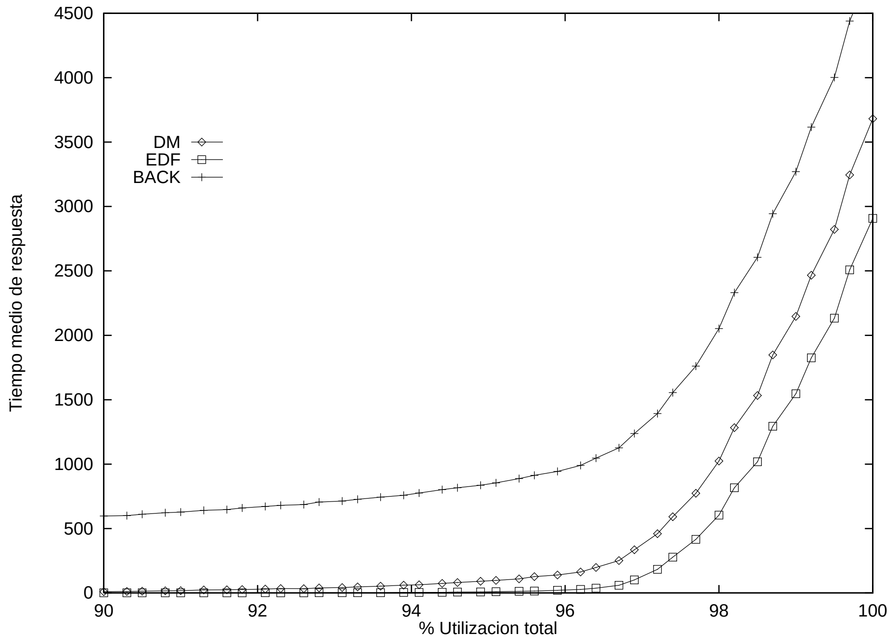


Figura 4.15: Comparación con baja carga periódica: 70%.

Tal como era de esperar a tenor de los resultados teóricos, los planificadores dinámicos son capaces de ofreces, bajo cualquier condición de carga, menores tiempos de respuesta para las tareas aperiódicas que los planificadores estáticos. Cuanto mayor es la carga periódica, mayor es el rendimiento de los planificadores dinámicos.

Por otro lado, recordemos que para poder realizar las comparaciones hemos tenido que restringir los conjuntos de tareas periódicas para que serán planificables bajo el DM. El EDF es capaz de planificar correctamente mayor número de tareas que el DM [Dertouzos74].

Un aspecto que no ha sido tenido en cuenta en estos experimentos es la sobrecarga producida por el servidor de tareas aperiódicas. Sería necesario realizar estudios de prestaciones sobre implementaciones reales en la que se pudiera tener en cuenta los detalles concretos del algoritmo.

4.7 Conclusiones

En este capítulo se han desarrollado los fundamentos teóricos en los que se basa el calculo de la holgura para sistemas dinámicos. A partir de éstos se ha desarrollado un algoritmo de aceptación de tareas aperiódicas (Aceptador Aperiódico Estático AAE), que ofrece una solución al problema de la planificación conjunta de tareas periódicas críticas con aperiódicas no críticas.

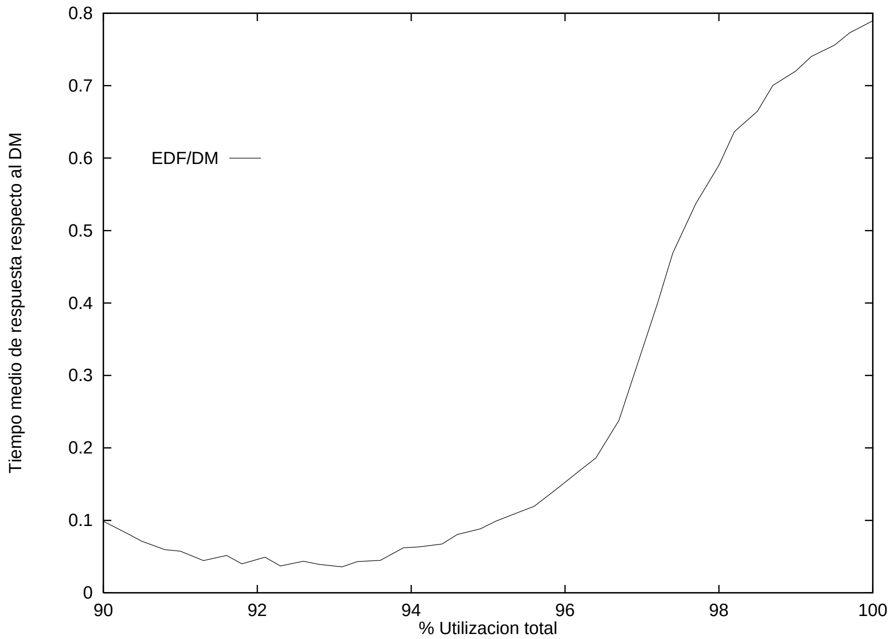


Figura 4.16: Comparison between EESS and OSSDM, hard periodic task set load 90%.

El AAE está diseñado para trabajar junto a un planificador óptimo, como podría ser el EDF. De tal forma que el EDF se encarga de planificar todas las tareas con plazo, y el AAE de asignar plazo a las peticiones aperiódicas.

Decimos que AAE es "estático" porque el calculo de la holgura se realiza durante la fase de diseño del sistema y se almacena en una tabla. posteriormente, en tiempo de ejecución, esta tabla se consulta para dar servició a las peticiones aperiódicas.

El tándem AAE-EDF es óptimo desde dos puntos de vista distintos. Primero, puesto que el EDF es óptimo, no existe otro sistema que sea capaz de planificar más tareas periódicas. Y segundo el AAE ofrece el menor tiempo de respuesta a las tareas aperiódicas.

Se han presentado dos extensiones al algoritmo básico. Una para poder reutilizar el tiempo sobrante (tiempo asignado a tareas activas, pero no empleado por éstas) para ofrecer un menor tiempo de respuesta al trabajo aperiódico. La otra extensión propuesta adapta al AAE para aceptar o rechazar tareas aperiódicas con plazo firme.

En lo relativo a la complejidad, el coste espacial es $O(N)$ donde N es el número de activaciones durante el hiperperiodo. El coste temporal en el peor de los casos es de $O(C_a \sum_{i=1}^{n} C_i \frac{1}{1-\mathcal{U}_\tau})$. Si bien cuanto más cargado está el sistema más bajo es el coste del AAE, de hecho cuando la carga es del 100% el coste es $O(1)$. Los resultados teóricos de optimalidad se han confirmado a través de estudios experimentales.

Esta página contiene intencionadamente sólo esta frase.

Capítulo 5

Servidor de holgura dinámico

5.1 Introducción

En este capítulo se propone la versión dinámica del algoritmo de aceptación de tareas aperiódicas, que llamaremos Aceptador Aperiódico Dinámico (AAD). Como ya vimos, a pesar de ser óptimo en cuanto al tiempo de respuesta que da a las tareas aperiódicas, tiene serias restricciones de implementación debido a la gran cantidad de memoria que requiere. La tabla que precisa el algoritmo AAE para trabajar puede llegar a ser del tamaño del hiperperiodo.

Por otra parte, el AAE sólo está preparado para trabajar con conjuntos de tareas periódicas. Para el correcto empleo de la tabla precalculada, es necesario que todas las activaciones de las tareas críticas comiencen exactamente al inicio de cada periodo. En caso de emplearse el AAE con conjuntos de tareas periódicas y esporádicas, el AAE se convierte en un algoritmo subóptimo, si bien sigue siendo válido pues garantiza la correcta ejecución de todas las tareas.

El algoritmo AAD trata de superar estos problemas. El AAD calcula dinámicamente la cantidad de tiempo de holgura que se puede substraer de las tareas críticas (periódicas y esporádicas) para asignárselo a las peticiones aperiódicas, sin violar por ello ningún plazo. Podemos interpretar que el AAD construye dinámicamente el fragmento de tabla de holgura que necesita para aceptar cada tarea, es por ello que el coste temporal es algo superior al coste del AAE.

Al igual que el AAE el AAD es óptimo pues garantiza el menor tiempo de respuesta posible a todas las tareas aperiódicas.

Podemos considerar que el AAD es el hermano dinámico del algoritmo propuesto por Davis et al.[Davis93], si bien, tanto la teoría como la propia implementación de cada uno de ellos es completamente diferente.

5.1.1 Modelo de sistema

El conjunto de tareas críticas, τ, puede estar compuesto tanto por tareas periódicas como esporádicas. Evidentemente, el conjunto ha de ser planificable, esto es, ha pasado satisfactoriamente el test de planificabilidad descrito en el capítulo 3.

El modelo de carga aperiódica con la que trabaja el AAD es el mismo que el empleado en el capítulo anterior: tareas acríticas aperiódicas sin plazo y posteriormente se presenta una extensión para aceptar tareas acríticas aperiódicas con plazo firme.

Ejemplo de trabajo Todos los ejemplos de este capítulo se basan el siguiente conjunto de tareas:

Tarea	Tiempo de cómputo	Plazo de ejecución	Periodo
T_1	2	4	6
T_2	1	3	8
T_3	4	11	15

Tabla 5.1: Conjunto de tareas ejemplo.

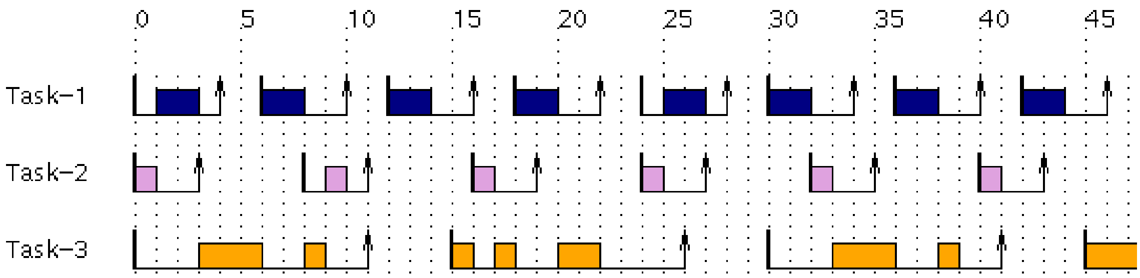


Figura 5.1: Ejemplo de planificación generada por el EDF.

5.2 Extracción dinámica de holgura

En esta sección estudiaremos cómo extraer determinada cantidad de tiempo de holgura de las tareas activas y de las futuras activaciones. No construiremos un mapa de los instantes en los que hay tiempo de holgura disponible, sino que nos centraremos en *encontrar el intervalo más pequeño que contenga el suficiente tiempo de holgura para ejecutar la tarea aperiódica que hay que servir*.

Recordemos que partimos de un conjunto inicial de tareas críticas que son planificables. El sistema dispone de un planificador óptimo, que bien puede ser EDF, el LLF, o cualquier otro que sea óptimo. Y pretendemos añadirle un algoritmo de aceptación de tareas aperiódicas.

Por claridad, inicialmente supondremos que no hay solapamiento de peticiones aperiódicas, esto es, no llega una nueva petición aperiódica hasta que no ha finalizado la anterior. También supondremos que todas las tareas consumen todo el tiempo que piden (el tiempo de ejecución en el peor de los casos). En la sección 5.4 eliminaremos estas dos restricciones.

El análisis está dividido en dos partes. Primero se busca tiempo ocioso del procesador. Se calcula el intervalo más pequeño en el cual el procesador está ocioso la misma cantidad de tiempo que la requerida por la petición aperiódica que hay que servir. Podemos interpretar el intervalo obtenido como el tiempo de respuesta que el servidor en *background* daría a esta tarea.

En la segunda parte, se acota el intervalo más pequeño que comienza en instante actual (momento en el que llega la petición aperiódica) del que se puede extraer el tiempo de holgura requerido y que será asignado a la tarea aperiódica.

Supondremos que la tarea J_a pide C_a u.t. de cómputo en el instante t_{now}. El objetivo consiste en decidir o bien qué plazo asignarle (en caso de ser una tarea aperiódica sin restricciones temporales), o .bien si puede ser garantizada (si se trata de una tarea con plazo firme).

Nota en el resto del capítulo supondremos que la variable t expresa la cantidad de tiempo transcurrida desde el instante t_{now}, a diferencia de capítulos anteriores en los que representaba el tiempo absoluto transcurrido desde la puesta en funcionamiento del sistema.

5.2.1 Intervalo ocioso

Si una tarea aperiódica, J_a, pide ser ejecutada, algunas de las tareas que en ese momento están en ejecución posiblemente tengan que ser retrasadas, y en algunos casos, incluso las siguientes activaciones también pueden verse afectadas por la sobrecarga que produce la aceptación de esta nueva tarea. El intervalo más largo que contiene tareas "afectadas" – tareas que modifican los instantes de ejecución con respecto a la planificación normal– finaliza justo cuando el procesador queda ocioso exactamente C_a u.t..

Para realizar el análisis, necesitaremos la siguiente información (datos que se obtienen fácilmente del sistema operativo o del algoritmo de planificación):

$n_{i,t_{now}}$ Instante de la próxima activación de la tarea T_i. Para las tareas periódicas se puede calcular como $p_i - (t_{now} \mod P_i)$; y para las esporádicas se obtiene como $\min\{s_{x-1} + P_i - t_{now}, 0\}$, donde s_{x-1} representa el instante de su última activación.

$d_{i,t_{now}}$ Plazo de la activación actual de la tarea T_i.

$c_{i,t_{now}}$ Cantidad de tiempo que resta por ejecutar de la activación actual de la tarea T_i. Se obtiene como la diferencia entre el tiempo de ejecución en el peor caso y el tiempo de ya que ha consumido. Vale cero si la activación actual ha completado su ejecución, tanto si ha empleado todo el tiempo C_i como si no. Si t_{now} coincide con el comienzo de la activación actual, entonces $c_{i,t_{now}} = C_i$.

$A_{t_{now}}$ Tiempo total que resta por ejecutar de las tareas activas. Típicamente es $A_{t_{now}} = \sum_{i=1}^{n} c_{i,t_{now}}$

Sea $[t_{now}, t_{now} + \mathcal{K})$ el intervalo de tiempo que contiene exactamente C_a u.t. ocioso. Después del instante $t_{now}+\mathcal{K}$ el orden de ejecución de las tareas periódicas no se verá alterado. Si le añadimos C_a u.t. de cómputo, entonces el procesador estará permanentemente ocupado durante le intervalo ocioso.

Para garantizar que todas las tareas periódicas cumplirán sus plazos, tendremos que suponer las condiciones más desfavorables, esto es, todas las tareas consumen el máximo tiempo, y las tareas esporádicas realizan peticiones a su máxima frecuencia.

Para el cálculo de $\mathcal{K}$ vamos a necesitar una función que nos permita obtener la cantidad de tiempo de cómputo pedido por un determinado conjunto de tareas. Concretamente necesitaremos conocer la cantidad de tiempo de cómputo "pedido" entre el instante t_{now} y algún instante posterior, t. La siguiente función calcula este dato:

$$G_{\tau}^{t_{now}}(t) = A_{t_{now}} + \sum_{i=1}^{n} C_i \cdot \left\lceil \frac{(t - n_{i,t_{now}})_0}{P_i} \right\rceil \quad (5.1)$$

Ésta es una generalización de la función $G_{\tau}(t)$, introducida en el capítulo 3 (función 3.1). Para el valor de t_{now} igual a cero, ambas funciones son equivalentes.

$G_{\tau}^{t_{now}}(t)$ es una función no decreciente que representa la suma total del tiempo de cómputo pedido en el intervalo $[t_{now}, t_{now} + t)$.

A partir de esta función, podemos definir el valor de $\mathcal{K}$ como:

$$\mathcal{K} = \min t \text{ tal que } t = C_a + G_{\tau}^{t_{now}}(t)$$

Esta expresión se puede resolver empleando la siguiente relación de recurrencia:

$$\begin{aligned} K_0 &= C_a + A_{t_{now}} \\ K_{i+1} &= C_a + G_{\tau}^{t_{now}}(K_i) \end{aligned} \quad (5.2)$$

El proceso iterativo finaliza cuando $K_i \equiv K_{i+1}$. El valor de $\mathcal{K}$ es el último valor de la secuencia K_i.

Lema 5.1 *La secuencia de valores dada por la ecuación 5.2 converge en un número finito de iteraciones si y sólo si el factor de utilización del procesador es estrictamente menor que uno.*

DEMOSTRACIÓN: Verifiquemos que la función $G_{\tau}^{t_{now}}(t)$ está acotada por $A_{t_{now}}+\sum_{i=1}^{n} C_i+ \mathcal{U}_{\tau} \cdot t$:
$G_{\tau}^{t_{now}}(t) = A_{t_{now}} + \sum_{i=1}^{n} C_i \cdot \left\lceil \frac{(t-n_{i,t_{now}})_0}{P_i} \right\rceil$

$\leq$ { Eliminando la función techo}

$$A_{t_{now}} + \sum_{i=1}^{n} C_i \left(\frac{t}{P_i} + 1 \right)$$

$=$ {Por definición de $\mathcal{U}_\tau$ y reorganizando términos}

$$A_{t_{now}} + \sum_{i=1}^{n} C_i + \mathcal{U}_\tau \cdot t$$

Por tanto: $G_\tau^{t_{now}}(t) \leq A_{t_{now}} + \sum_{i=1}^{n} C_i + \mathcal{U}_\tau \cdot t$.

Una cota superior de el último valor de K_i es:
$t \leq C_a + A_{t_{now}} + + \sum_{i=1}^{n} C_i \mathcal{U}_\tau \cdot t$
$\equiv$ { Reorganizando términos }

$$t \leq \frac{C_a + \sum_{i=1}^{n} C_i + A_{t_{now}}}{1 - \mathcal{U}_\tau}$$

Por tanto, si $\mathcal{U}_\tau$ es menor que uno (la utilización del procesador es inferior al 100%), entonces el denominador de esta expresión es mayor que cero, con lo que t es un número positivo no infinito.

□

Puesto que el procesador estará ocioso durante C_a u.t. en el intervalo $[t_{now}, t_{now} + \mathcal{K})$, podremos ejecutar la tarea aperiódica J_a con un plazo asignado de $D_a = t_{now} + \mathcal{K}$, sin violar el plazo de ejecución de ninguna otra tarea.

Ejemplo numérico Supongamos que en el instante 6 la primera petición aperiódica pide 4 u.t. de cómputo. El sistema operativo ofrece la siguiente información:

t_{now}	C_a	A_6	$n_{1,6}$	$n_{2,6}$	$n_{3,6}$
6	4	3	6	2	9

Para obtener la duración del intervalo ocioso, calcularemos la secuencia de valores de K_i:

$$K_0 := 7 = 3 + 4 = A_6 + C_a$$

$$K_1 := 10 = 4 + 3 + 2 \left\lceil \frac{7-6}{6} \right\rceil + 1 \left\lceil \frac{7-2}{8} \right\rceil + 4 \left\lceil \frac{7-9}{15} \right\rceil$$

$$K_2 := 15 = 4 + 3 + 2 \left\lceil \frac{10-6}{6} \right\rceil + 1 \left\lceil \frac{10-2}{8} \right\rceil + 4 \left\lceil \frac{10-9}{15} \right\rceil$$

$$K_3 := 17 = 4 + 3 + 2 \left\lceil \frac{15-6}{6} \right\rceil + 1 \left\lceil \frac{15-2}{8} \right\rceil + 4 \left\lceil \frac{15-9}{15} \right\rceil$$

$$K_4 := 17 = 4 + 3 + 2 \left\lceil \frac{17-6}{6} \right\rceil + 1 \left\lceil \frac{17-2}{8} \right\rceil + 4 \left\lceil \frac{17-9}{15} \right\rceil$$

La secuencia de recurrencia se detiene en K_4 , ya que K_3 es igual a K_4. El intervalo ocioso es $[6, 17 + 6)$.

En la figura 5.2 aparece una representación gráfica de como se puede calcular el valor de $\mathcal{K}$.

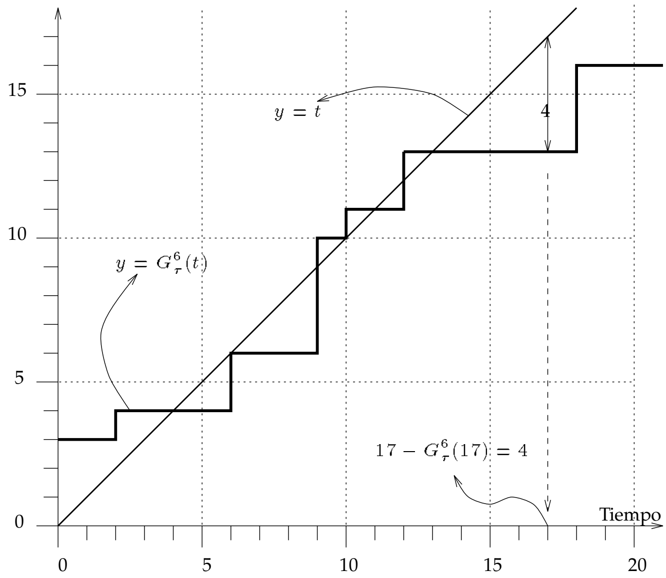


Figura 5.2: Representación gráfica de G^6_τ.

Y en la figura 5.3, aparece el cronograma de la ejecución de las tareas τ junto con la nueva tarea J_a, a la que se le ha asignado el plazo de ejecución $D_a = 17{+}6$. Comparándolo con el cronograma de la figura 5.1, se puede apreciar como ninguna de las activaciones anteriores al instante 6, ni posteriores al instante 23 alteran su ejecución. Sólo la segunda activación de la tarea 3 ve retrasada su ejecución, retraso que no produce la perdida de su plazo.

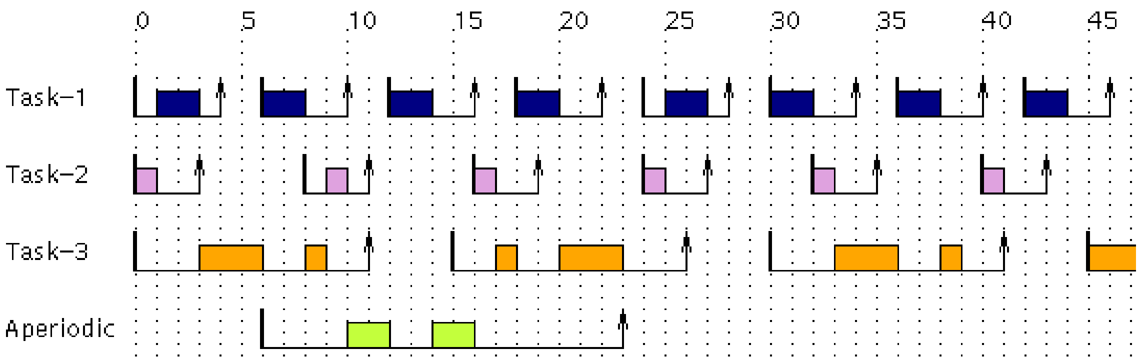


Figura 5.3: Cronograma de ejecución con la inclusión de la tarea aperiódica.

En la siguiente sección, estudiaremos cómo ejecutar lo antes posible el tiempo de cómputo aperiódico dentro del intervalo ocioso.

5.2.2 Intervalo de interferencia

Para poder calcular el intervalo de interferencia, necesitaremos de la siguiente función:

$$H_\tau^{t_{now}}(t) = \sum_{i=1}^{n} c_{i,t_{now}} \cdot \left\lfloor \frac{(2t - d_{i,t_{now}})_0}{t} \right\rfloor + \sum_{i=1}^{n} C_i \cdot \left\lfloor \frac{(t + P_i - D_i - n_{i,t_{now}})_0}{P_i} \right\rfloor$$

Esta función contabiliza la cantidad de tiempo de cómputo "necesitado" por todas las tareas que finalizan su ejecución dentro del intervalo $[t_{now}, t)$. Y al igual que $G_\tau^{t_{now}}$, $H_\tau^{t_{now}}(t)$ es una generalización de la función H_τ descrita en el capítulo 3 (función 3.2).

El primer sumatorio calcula el tiempo de ejecución requerido por las tareas que están activas en el instante t_{now}. La expresión $\left\lfloor \frac{(2t-d_{i,t_{now}})_0}{t} \right\rfloor$ es un artilugio matemático cuyo valor es cero en el intervalo $[t_{now}, d_{i,t_{now}})$ y uno en $[d_{i,t_{now}}, \infty)$. El segundo sumatorio calcula el tiempo de ejecución necesitado por las futuras activaciones –activaciones cuyo tiempo de inicio es posterior a t_{now}.

El valor de $H_\tau^{t_{now}}(t)$ se puede interpretar como la mínima cantidad de tiempo que debe haberse completado en el instante t, para impedir que alguna tarea pierda su plazo. Si $H_\tau^{t_{now}}(t)$ es mayor que t, entonces es seguro que al menos una tarea con plazo menor que t ha violado su plazo de ejecución (Ver Baruah et al. [Baruah90b] Lema 2).

En el capítulo 4 ya analizamos la expresión $t - H_\tau(t)$. Vimos que si el conjunto de tareas τ es planificable, entonces la expresión se evaluaba a un número positivo. Cuando el tiempo t coincide con el plazo de alguna de las activaciones, podíamos interpretarla como el tiempo de holgura que esa activación disponía para completar su ejecución. También vimos que aún disponiendo de suficiente tiempo de holgura una activación, no siempre se podía asignar todo ese tiempo a tareas aperiódicas, pues podía suceder que activaciones posteriores perdieran su plazo.

Supongamos que existe un instante de tiempo, $\mathcal{L}$, tal que $\mathcal{L} \leq \mathcal{K}$; y que para todo t mayor o igual que $\mathcal{L}$ la diferencia entre $H_\tau^{t_{now}}(t)$ y t fuese mayor o igual que C_a.

Teorema 5.1 *Sea τ un conjunto de tareas planificable. C_a u.t. se pueden substraer del intervalo $[t_{now}, t_{now} + \mathcal{L})$ sin producir por ello la perdida de ningún plazo si y sólo si:*

$$\mathcal{L} \geq \min(t) \text{ tal que } \forall t \geq \mathcal{K}, H_\tau^{t_{now}}(t) - t \geq C_a$$

DEMOSTRACIÓN: Supongamos que la substracción de las C_a u.t. se lleva a cabo introduciendo una tarea aperiódica con tiempo de inicio t_{now}, plazo $t_{now} + \mathcal{L}$, y tiempo de cómputo igual a C_a.

Todas las activaciones que comiencen su ejecución con posterioridad al instante $\mathcal{K}$ son planificables ya que el sistema era inicialmente planificable, y tal como vimos en la sección anterior, las activaciones que a partir de entonces se produzcan no se verán afectadas por la introducción de las C_a u.t..

Aquellas activaciones cuyo plazo finaliza dentro del intervalo $[\mathcal{L}, \mathcal{K})$ no pierden su plazo de ejecución pues la condición impuesta por el teorema garantiza que el planificador dispondrá de suficiente tiempo para ejecutar correctamente cada una de ellas.

Las activaciones con plazo menor que $\mathcal{L}$ no sufren ninguna alteración de su condición de planificabilidad, pues la nueva tarea tiene un plazo posterior a todas ellas y por tanto tiene menos “prioridad”.

$\square$

Ejemplo numérico Continuando con el ejemplo anterior, calcularemos ahora el plazo más corto que se le puede asignar. La información que necesitamos se puede obtener fácilmente del sistema operativo (planificador). En la siguiente tabla aparece esta información.

$$\begin{array}{lll} n_{1,6} := 6 & n_{2,6} := 2 & n_{3,6} := 9 \\ d_{1,6} := 4 & d_{2,6} := 5 & d_{3,6} := 5 \\ c_{1,6} := 2 & c_{2,6} := 0 & c_{3,6} := 1 \end{array}$$

El valor de $H_\tau^6(t)$ se calcula como:

$$H_\tau^6(t) \rightarrow \quad 2\left\lfloor \frac{t-4}{6} \right\rfloor + 1\left\lfloor \frac{t+3}{8} \right\rfloor + 4\left\lfloor \frac{t-5}{15} \right\rfloor + 2\left\lfloor \frac{2 \cdot t-4}{t} \right\rfloor + 0\left\lfloor \frac{2 \cdot t-5}{t} \right\rfloor + 1\left\lfloor \frac{2 \cdot t-5}{t} \right\rfloor$$

La secuencia de valores de la expresión $(t - H_\tau^6(t))$ para los valores de t desde 17 hasta 7son: $8, 7, 8, 7, 6, 6, 5, 4, 5, 4, 3$. Para $t = 7$ el valor obtenido es menor que el tiempo de cómputo buscado, y por tanto, el menor valor de t que cumple el teorema 5.1 es 8, que es el valor de $\mathcal{L}$.

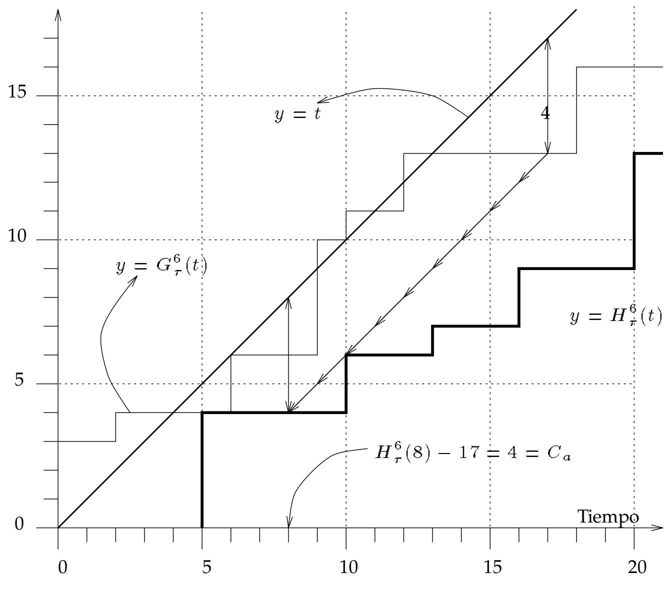


Figura 5.4: Representación gráfica del cálculo de los tiempos $\mathcal{K}$ y $\mathcal{L}$.

En la figura 5.4 aparecen representadas las dos funciones, $G_\tau^6(t)$ y $H_\tau^6(t)$, así como la interpretación geométrica del cálculo de ambos tiempos, $\mathcal{K}$ y $\mathcal{L}$.

El método geométrico para obtener el instante $\mathcal{K}$ consiste en trazar la recta $y = t - C_a$. El valor de t del primer punto de cruce de esta recta con la curva $y = G_{\mathcal{T}}^{t_{now}}(t)$ define el instante $\mathcal{K}$. Y de el último punto de cruce de la recta con la curva $y = H_{\mathcal{T}}^{t_{now}}(t)$ obtenemos el instante $\mathcal{L}$.

5.3 Aceptador Aperiódico Dinámico (AAD)

Los principios básicos de funcionamiento del AAD se han presentado en la sección anterior. En esta sección reunimos todo el estudio previo en la implementación que aquí presentamos es tan sólo a nivel ilustrativo. Los detalles de una implementación práctica se describen al final de la sección.

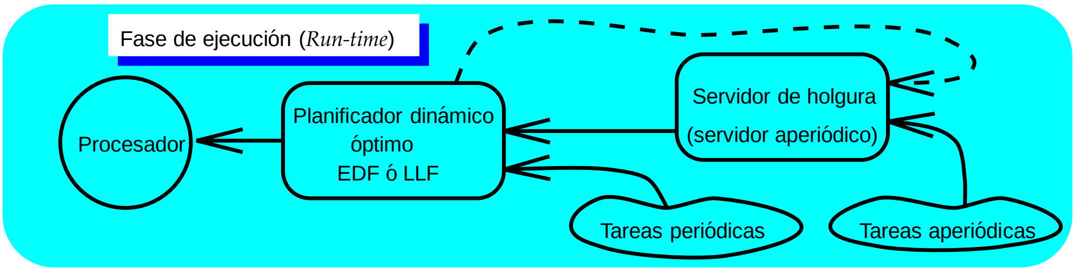


Figura 5.5: Diagrama de bloques del AAD.

El AAD es invocado cada vez que llega una petición aperiódica. El AAD le asigna un plazo de ejecución y la inserta en la cola del planificador del sistema. A partir de ese momento el AAD no prestará más atención a esta tarea. Por otra parte, el AAD no precisa de ningún método de actualización ni seguimiento del estado del sistema mientras no llegan peticiones.

Algoritmo 5.1 (Aceptador Aperiódico Dinámico (AAD))

<u>**proc**</u> **AAD**(J_a) $\equiv$
  <u>**var**</u>
    deadline, *temp*, $\mathcal{K}$, $\mathcal{L}$;
  <u>**end**</u>
  <u>**begin**</u>
    $\mathcal{K} := 0$;
    temp $:= C_a$;
    <u>**while**</u> $(\mathcal{K} \neq$ *temp*$)$ <u>**do**</u>
        $\mathcal{K} :=$ *temp*;
        temp $:= G_{\mathcal{T}}^{t_{now}}(\mathcal{K})$;
    <u>**od**</u>
    <u>**if**</u> $(\mathcal{K} = C_a)$ <u>**then**</u>
            deadline $:= \mathcal{K} + C_a$;
        <u>**else**</u>
            $\mathcal{L} := \mathcal{K}$;
            <u>**repeat**</u>
              $\mathcal{L} := \mathcal{L} - 1$;
            <u>**until**</u> $(\mathcal{L} - H_{\mathcal{T}}^{t_{now}}(\mathcal{L}) \geq C_a)$;

21 *deadline* $:= t_{now} + \mathcal{L} + 1;$
22 **fi**
23 *Insert_into_Scheduler_queue*$(J_a,$ *deadline*$);$
24 **end**
25 .

Tal como se comentó en el capítulo anterior, la función *Insert_into_Scheduler_queue*(J, D) envía la tarea J_a a la cola del planificador con el plazo D.

5.3.1 Optimalidad del AAD

Teorema 5.2 *Para cualquier conjunto de tareas periódicas planificable y para cualquier secuencia de peticiones aperiódicas servidas en orden FIFO (First Come First Served), el algoritmo AAD minimiza el tiempo de respuesta de cada una de las peticiones aperiódicas de entre todos los algoritmos de planificación que garantizan el cumplimiento de todos los plazos de todas las tareas. (nota: excluyendo los planificadores clarividentes que tienen conocimiento de los instantes exactos de llegada de las tareas esporádicas.)*

DEMOSTRACIÓN: La demostración es inmediata a partir del teorema 5.1. Por construcción, el intervalo de interferencia es el más pequeño en el que algún algoritmo puede planificar con éxito todas las tareas, incluida la propia tarea aperiódica.

□

5.3.2 Complejidad del algoritmo AAD

El algoritmo AAD lleva a cabo el trabajo en dos etapas: primero calcula la longitud del intervalo ocioso y luego la longitud del intervalo de interferencia. El número de iteraciones necesarias para obtener la longitud del intervalo ocioso (línea 9) depende principalmente del valor de $\mathcal{U}_\tau$ (ver lema 5.1). En el peor de los casos, el número de iteraciones es del orden de $O(\frac{C_a}{1-\mathcal{U}_\tau})$. Como se deduce de está expresión, el coste es realmente significativo sólo para valores de $\mathcal{U}_\tau$ muy próximos a la unidad. De hecho, si limitamos el factor de utilización a ser menor que el 99%, la complejidad asintótica se convierte en pseudopolinómica.

En el bucle de la segunda parte, líneas de la 18 a 20, para calcular la longitud del intervalo de interferencia, el número de i el número de iteraciones que realizan está acotado por la longitud del intervalo ocioso. Otro factor que afecta a la complejidad de este bucle es la facilidad para "retrasar" la ejecución de las tareas periódicas. Tareas periódicas con plazos similares a los periodos tienen una gran facilidad para acomodar su ejecución en cualquier posición dentro de su periodo, lo que produce que el trabajo aperiódico se pueda situar muy próximo al instante de llegada, y por consiguiente el intervalo de interferencia es muy pequeño, lo que conlleva un mayor número de iteraciones del bucle 18 para calcular el valor de $\mathcal{L}$. Por contra, si las tareas tienen poca holgura de ejecución (plazos muy ajustados), el intervalo de interferencia será muy largo, con lo que el número de iteraciones será reducido.

La complejidad global del AAD está regida por la longitud del intervalo ocioso, lo cual depende de el tiempo de cómputo requerido por la tarea aperiódica y el nivel de utilización producido por las tareas periódicas: $O(\frac{1}{(1-(\mathcal{U}_\tau+1/C_a))})$

En lo relativo a la complejidad espacial del AAD sólo cabe comentar que es despreciable, pues tan sólo se necesitan unas pocas variables escalares, a parte, claro está, de la información sobre las propias tareas críticas.

5.4 Extensiones

5.4.1 Varias peticiones aperiódicas solapadas

Hasta ahora habíamos supuesto que cada vez que llegaba una nueva tarea aperiódica, todas las anteriores peticiones aperiódicas ya habían sido servidas completamente –esto es, una vez aceptada una petición, la siguientes no se produciría hasta después de $\mathcal{K}$ u.t.. Como ahora veremos, se podrá seguir empleando el algoritmo AAD para obtener el plazo asignado de las nuevas peticiones. La única consideración a tener en cuenta es incluir en el cálculo de $G_\tau^{t_{now}}(t)$ y $H_\tau^{t_{now}}(t)$ las peticiones aperiódicas previas.

Sea σ el conjunto de tareas aperiódicas activas en el instante t_{now}. Supongamos que la última petición aperiódica llegó en el instante t_{prev}, el intervalo ocioso era $[t_{prev}, t_{prev}+\mathcal{K}')$, el plazo que se le asignó fue $t_{prev}+\mathcal{L}'$, y finalmente, el instante actual cae dentro del intervalo ocioso de la última petición. Dada esta situación , es fácil observar que el procesador no podrá quedar ocioso hasta después del instante $t_{prev}+\mathcal{K}'$. En consecuencia, el cálculo de $\mathcal{K}$ debe comenzar en $t_{prev}+\mathcal{L}'-t_{now}+C_a$. Por otra parte el hecho de que puedan haber una gran cantidad de tareas aperiódicas en el sistema no incrementa la complejidad de $G_\tau^{t_{now}}$, pues todas ellas aparecen englobadas dentro de la constante $A_{t_{now}}$ por ser tareas activas, y el sumatorio de $G_\tau^{t_{now}}$ sólo ha de considerar las tareas periódicas y esporádicas.

Evidentemente la nueva tarea aperiódica tendrá un plazo asignado mayor que cualquier otra aperiódica aceptada con anterioridad. Por consiguiente, no se tendrán que evaluar la función $H_\tau^{t_{now}}(t)$ para valores de t menores que $\mathcal{L}'$, lo que lleva a la siguiente simplificación:

$$\forall t > \mathcal{L}' \;,\; H_{\tau+\sigma}^{t_{now}}(t) = H_\tau^{t_{now}}(t) + \sum_{i\in\sigma} c_{i,t_{now}}$$

Teniendo en cuenta todas estas observaciones, la complejidad del algoritmo de aceptación cuando las peticiones se solapan es similar a aceptar peticiones aisladas.

5.4.2 Reutilizando tiempo sobrante

El tiempo sobrante se produce cuando alguna tarea activa consume menos tiempo del que inicialmente se le había asignado [Woodbury88]. En principio sólo se puede detectar la aparición de tiempo sobrante tras la finalización de cada tarea comparando el tiempo que efectivamente a empleado con el que inicialmente se le asignó. Si las tareas están especialmente preparadas con puntos de chequeo y se conoce el tiempo máximo necesario para alcanzar cada uno de estos puntos, entonces se puede obtener el tiempo sobrante con mayor antelación.

Si no hay ninguna tarea aperiódica activa en el momento de producirse el tiempo sobrante, no es necesaria ninguna modificación para reutilizar este tiempo, pues ya se tiene en cuenta en el cálculo del tiempo que le resta de ejecución a la tarea que lo ha producido, $c_{i,t_{now}}$.

En caso de producirse el tiempo sobrante mientras hay tareas aperiódicas a la espera de ser ejecutadas, entonces se puede intentar mejorar el tiempo de respuesta de éstas. Un primer método consiste en asignárselo directamente a la tarea aperiódica con plazo más corto, para ello se reduce el plazo de ésta en tantas unidades de tiempo como se hayan obtenido de tiempo sobrante. Por otra parte, el resto de las tareas aperiódicas pueden reducir sus plazos en la misma medida.

El método antes descrito no siempre es capaz de obtener los mejores tiempos de respuesta. Puesto que las condiciones en las que se calculó el plazo asignado de las tareas aperiódicas han podido cambiado. Si ahora se recalculasen nuevamente los plazos, podría suceder que los nuevos plazos fuesen menores que los estimados en el método anterior. En el siguiente ejemplo numérico se ilustrar como una pequeña reducción en el tiempo de cómputo de una tarea puede conducir a importantes reducciones en el plazo asignado:

Ejemplo numérico Una tarea aperiódica, J_a, pide 4 u.t. en el instante 15. La información que almacena el planificador es la siguiente:

$$\begin{array}{lll} n_{1,15} := 3 & n_{2,15} := 1 & n_{3,15} := 15 \\ d_{1,15} := 1 & d_{2,15} := 4 & d_{3,15} := 11 \\ c_{1,15} := 0 & c_{2,15} := 0 & c_{3,15} := 4 \end{array}$$

Bajo estas circunstancias, el valor de $\mathcal{L}$ obtenido por el algoritmo AAD es de 14, lo que representa un plazo asignado de $15 + 14 = 29$.

Supongamos que en el instante $t_1 = 18$ la segunda activación de la tarea T_3 ha empleado dos u.t. y tan sólo le resta una u.t. para finalizar su ejecución. Con lo que disponemos de 1 u.t. de tiempo sobrante. En este momento, el estado del sistema es:

$$\begin{array}{lll} n_{1,18} := 6 & n_{2,18} := 6 & n_{3,18} := 12 \\ d_{1,18} := 4 & d_{2,18} := 1 & d_{3,18} := 8 \\ c_{1,18} := 2 & c_{2,18} := 0 & c_{3,18} := 1 \end{array}$$

El método anterior, reduciría el plazo asignado a J_a de 29 a 28. El resultado de invocar al algoritmo AAD, suponiendo que no hay ninguna tarea aperiódica en el planificador y teniendo en cuenta que a J_a le quedan por ejecutar las cuatro unidades de tiempo, es un plazo asignado de 24.

Por consiguiente, para aprovechar completamente el tiempo sobrante se ha de recalcular nuevamente los plazos asignados de todas las tareas.

5.4.3 Cálculo de la holgura durante el tiempo ocioso

En la formulación original del AAD, todos los cálculos necesarios para obtener el plazo de la tarea aperiódica, $\mathcal{L}$, se realizan justo en el momento en el que se produce la petición. Sin embargo, gran parte de estos cálculos se pueden llevar a cabo por adelantado.

Los intervalos de tiempo en los que el procesador queda ocioso pueden emplearse para calcular los tiempos de holgura de instantes futuros. Estos resultados se guardan en una pequeña tabla para poder consultarlos más adelante cuando llegue la tarea aperiódica. La forma de consultar esta tabla es similar a la explicada en el capítulo anterior.

5.4.4 Tareas aperiódicas con plazo firme

Con ligeras modificaciones se puede transformar el algoritmo AAD en una algoritmo de garantía de tareas con plazo firme.

Cuando llega una tarea con plazo firme se invoca al AAD para determinar el mínimo plazo que se le podría asignar. Si el plazo obtenido es menor que el propio plazo de la tarea, entonces se acepta la tarea y se inserta en la cola del planificador, empleando para ello el plazo de la tarea y no el plazo calculado por el AAD.

Cuando varias tareas firmes se solapan es necesario tener en cuenta las restricciones que las anteriores peticiones impusieron a la planificación del sistema. Las tareas firmes que todavía no han concluido su ejecución se han de considerar como tareas periódicas activas con período infinito. En el cómputo de $G_\tau^{t_{now}}(t)$ estas tareas sólo intervienen en el factor constante $A_{t_{now}} = \sum_{i \in \tau+\sigma} c_{i,t_{now}}$. La función $H_\tau^{t_{now}}(t)$ resultante es:

$$H_\tau^{t_{now}}(t) = \sum_{i=1}^{n} c_{i,t_{now}} \cdot \left\lfloor \frac{(2t - d_{i,t_{now}})_0}{t} \right\rfloor + \sum_{i \in \tau+\sigma} C_i \cdot \left\lfloor \frac{(t + P_i - D_i - n_{i,t_{now}})_0}{P_i} \right\rfloor$$

5.5 Conclusiones

En este capítulo hemos presentado un nuevo algoritmo para el servicio de trabajo aperiódico, el AAD. Al contrario que el algoritmo descrito en el capítulo anterior, los requisitos de memoria de este algoritmo son despreciables.

Puesto que el AAD es un algoritmo completamente dinámico (funciona sobre un planificador dinámico y evalúa la aceptación de las tareas sólo con la información que se dispone en el preciso momento de la llegada de la tarea) está especialmente bien preparado para trabajar en entornos altamente dinámicos, donde se producen frecuentes cambios de modo, e incluso modos de funcionamiento que no habían sido previstos inicialmente. Aún en estas circunstancias, el AAD continuará funcionando correctamente y ofreciendo los tiempos de respuesta más cortos posibles.

Tanto el AAE como el AAD son algoritmos óptimos, y tienen exactamente el mismo comportamiento –ofrecen los mismos tiempo de respuesta– cuando planifican el mismo sistema. Por consiguiente, los resultados experimentales de ambos son identicos. Es por ello que no se ha incluido en este capítulo una sección de experimentos.

La diferencia fundamental entre ambos es la complejidad en tiempo de ejecución. El AAD no produce aumento de carga, ni espacial ni temporal, mientras no llegen peticiones que servir. El coste de aceptar cada petición es independiente de cuantas se hayan aceptado previamente; tan sólo depende del nivel de utilización producido por las tareas periódicas, $\mathcal{U}_{\tau}$, y el tiempo de cómputo pedido por la tarea aperiódica. Si conocemos cual es la tarea aperiódica con más tiempo de cómputo entonces podemos saber *a priori* cuanto tiempo en el pero caso necesitado por AAD.

Capítulo 6

Conclusiones y líneas de trabajo futuras

6.1 Conclusiones

El objetivo de esta tesis ha sido ampliar los métodos de planificación de sistemas de tiempo real crítico basados en prioridades dinámicas. Dos han sido las principales aportaciones: (1) un nuevo y más eficiente test de planificabilidad y (2) dos nuevos algoritmos de planificación conjunta de tareas periódicas y aperiódicas. A continuación se comentan detalladamente cada uno los resultados alcanzados.

Test de planificabilidad

El concepto de sistema de tiempo real está estrechamente ligado a las ideas de *garantía* y *predecibilidad*. Evidentemente, no es una solución el sobredimensionar el sistema con la esperanza de evitar las sobrecargas y las perdidas de los plazos de las tareas. La única forma de poder asegurar que determinado sistema será capaz de cumplir las restricciones impuestas es a través del análisis o test de planificabilidad, de ahí la necesidad de disponer de herramientas de análisis precisas y eficientes.

Una de las aportaciones más interesantes es la extensión del concepto de instante crítico, que tan fructífera se ha demostrado en el campo de los planificadores basados en prioridades fijas, a los sistemas basados en prioridades dinámicas. Se ha propuesto un nuevo test de planificabilidad (nuevo desde el punto de vista de los sistemas dinámicos) basado en algunas propiedades del instante crítico.

Por otra parte, se ha propuesto una mejora al test de planificabilidad propuesto por Baruah et al. [Baruah90a].

Finalmente, se ha propuesto un test de planificabilidad combinando los dos anteriores, que representa el test más eficiente de los existentes[1].

[1] De entre los conocidos por los autores

No hay que olvidar que el problema de decidir si un conjunto de tareas es planificable o no pertenece a la clase de problemas co-**NP**-completos, y por consiguiente existen conjuntos de tareas para los cuales el test tiene un coste exponencial. Por otra parte, esta clase de problemas "parece" tener una extraña propiedad [Garey79]: variando mínimamente alguna característica de alguna tarea, la complejidad puede disminuir drásticamente, pasando de ser un problema intratable a tener un comportamiento casi polinómico. Desde la posición del ingeniero que diseña el sistema de tiempo real, no es difícil cambiar algunos de los parámetros de las tareas, bien sustituyendo el algoritmo de control que implementa una tarea por otro menos costoso, o bien reduciendo o aumentando el periodo de algunas tareas. Con lo que se puede concluir que el problema de decidir si un "sistema" es planificable es perfectamente abordable y resoluble en la práctica.

Los resultados teóricos obtenidos en este capítulo sirven de base para los desarrollos de capítulos posteriores.

Planificación conjunta de tareas periódicas y aperiódicas

Partiendo del hecho que un planificador de tiempo real ha de garantizar la correcta ejecución de todas las tareas críticas, las características más deseables de los algoritmos de planificación conjunta de tareas periódicas (críticas) y aperiódicas (acríticas) son:

- Ofrecer el mínimo tiempo de respuesta a las tareas aperiódicas,
- Ser eficientes, esto es, consumir la menor cantidad de recursos posible, tanto de memoria como de tiempo de procesador y
- Ser sencillos de implementar.

Los dos algoritmos propuestas –Aceptador Aperiódico Estático (AAE) y Aceptador Aperiódico Dinámico (AAD), capítulos 4 y 5 respectivamente– son óptimas, esto es, no existe otro algoritmo de planificación que sea capaz de ofrecer un tiempo de respuesta menor a las tareas aperiódicas sin perder ningún plazo de ninguna tarea crítica. En cuanto a la eficiencia de la implementación, cada uno de los algoritmos propuestos tratan de reducir o bien los requerimientos de memoria, o bien el tiempo de cómputo. Soluciones intermedias, en las que se puede obtener un compromiso entre coste espacial y temporal, se comentan más adelante en la sección de trabajo futuro.

Una característica de ambos algoritmos es el método que emplean para servir las tareas aperiódicas. En los servidores clásicos la ejecución de las tareas la realiza el propio servidor cuando éste decide que puede tomar el procesador. En cambio, los algoritmos propuestos sólo entran en ejecución cuando llegan las peticiones aperiódicas, en este instante asignan el plazo más corto posible compatible con la ejecución de todas las tareas activas y las futuras activaciones de las tareas críticas e insertan la tarea aperiódica (ahora convertida en una activación con plazo) en la cola del planificador, para que éste la ejecute como si se tratase de una activación periódica normal. De esta forma, al estar completamente separados los algoritmos de planificación de los de servicio aperiódico, se simplifica en gran medida tanto la implementación como la depuración. Finalmente, cabe también destacar que al estar centralizado el coste de la aceptación, se pueden construir implementaciones mucho más robustas, pues ante situaciones de estrés se puede suspender la aceptación de nuevas peticiones aperiódicas y la ejecución de las ya aceptadas no supone ninguna sobrecarga adicional respecto a la ejecución de las tareas periódicas. Nótese que las tareas periódicas no han de ser aceptadas, sino que las activaciones de las tareas periódicas pasan directamente a ser planificadas por el planificador.

En el algoritmo AAE, los instantes en los que hay tiempo de holgura, así como la cantidad que hay disponible se precalculan y almacenan en una tabla. Posteriormente, en tiempo de ejecución, se consulta esta tabla para aceptar tareas aperiódicas. La talla de la tabla es del orden del número de activaciones en el hiperperiodo, $O(N)$. El algoritmo AAD calcula la holgura disponible dinámicamente, por lo que no requiere de la tabla, pero por contra es más costoso.

Tanto el AAE como el AAD pueden ser utilizados conjuntamente con cualquier planificador óptimo. No están restringidos a funcionar con el EDF, sino que pueden emplearse junto con el LLF o incluso con cualquier otro nuevo planificador que se proponga en el futuro con tal de que sea óptimo.

Ambos algoritmos se han extendido para trabajar con tareas aperiódicas con plazo firme. El AAE y el AAD se convierten en este caso en test de aceptación, rechazando aquellas peticiones aperiódicas que puedan cumplir su plazo. También se ha contemplado la reutilización del tiempo sobrante (tiempo asignado pero no consumido; situación muy frecuente en sistemas de tiempo real pues los tiempos de ejecución de las tareas críticas están considerados para el peor caso) para dar mejor tiempo de respuesta a las tareas aperiódicas.

En lo relativo a la complejidad, aceptar una petición aperiódica tiene un coste pseudopolinómico, pues depende de los periodos y los tiempos de cómputo de las tareas críticas, y del tiempo de cómputo de la tarea aperiódica. A pesar de ello, si suponemos un tiempo de cómputo máximo para las tareas aperiódicas, y puesto que conocemos todos los parámetros de las tareas periódicas se puede conocer a priori cual será el tiempo que empleará el AAE o el AAD en aceptar una petición.

6.2 Líneas de trabajo futuras

Muchas son las líneas de trabajo que quedan abiertas a partir de los resultados de la presente tesis.

- El test de planificabilidad propuesto tan sólo indica si un conjunto de tareas es planificable o no. Una posible mejora consistiría en refinar el test para que en caso de que el conjunto no fuera planificable ofreciera algún tipo de información de qué tareas son las que impiden que el conjunto sea planificable, y como se deberían modificar y en qué medida (por ejemplo: disminuyendo los tiempos de cómputo o ampliando el plazo...) para conseguir que sea planificable.

- Extender el análisis de planificabilidad a conjuntos de tareas con compartición de recursos.

- Todos los servidores de trabajo aperiódico, incluidos los dos aquí propuestos, no consideran la compartición de recursos entre las tareas.

- La tabla de holgura calculada en el capítulo 4 no tiene por qué ser del tamaño del hiperperiodo, sino que se puede tener un tamaño arbitrario. Recordemos igualmente, que la tabla de holgura es periódica. Sería interesante estudiar cómo se podría utilizar unta tabla reducida para diseñar un servidor aperiódico subóptimo pero con un menor coste espacial que el AAE.

- En la realización de los experimentos se ha observado que la distribución de los huecos de holgura tiende ha ser muy uniforme. En caso de confirmarse esta observación, se podría emplear para mejorar el tiempo de cálculo de la holgura.

Apéndice A

Demostración del teorema 4.1

Antes de entrar en el cuerpo de la demostración, presentaremos una serie de propiedades y definiciones que luego emplearemos.

- Sea τ^j un conjunto de tareas definido recursivamente como: $\tau^{j-1} \cup \{T_{n+k-j+1}\}$, donde $\tau^0 = \tau$. Obsérvese que τ^1 es el conjunto τ más la tarea T_k; τ^2 es τ más las tareas T_k y T_{k-1}, etc; finalmente, $\tau^k = \tau \cup \theta$.

- Directamente de las definiciones de β y Δ obtenemos las siguientes desigualdades:

$$\beta_k < \beta_{k-1} < \cdots < \beta_1 \leq \beta_0 \tag{A.1}$$

$$\Delta(k) < \Delta(k-1) < \cdots < \Delta(1) < \Delta(0) \tag{A.2}$$

$$D_k < D_{k-1} < \cdots < D_2 \leq \beta_1 \tag{A.3}$$

- El valor de $D_{n+1} - H_\tau(D_{n+1})$ es:
$= \{$De la definición de $D_{n+1}\}$

$$\beta_1 + C_{n+1} - H_\tau(\beta_1 + C_{n+1})$$

$\geq \{$Dado que $\beta_1 \leq \beta_1 + C_{n+1}$ y de la definición de $\beta\}$

$$t_1 - G_\tau(t_1)$$

De la expresiónA
$= \{$Substituyendo C_{n+1} por su definición$\}$

$$\beta_1 + t_1 - G_\tau(t_1) - \beta_1 + H_\tau(\beta_1) - H_\tau(\beta_1 + C_{n+1})$$

$= \{$Cancelando términos$\}$

$$t_1 - G_\tau(t_1) + H_\tau(\beta_1) - H_\tau(\beta_1 + C_{n+1})$$

$\leq$ {Propiedad 4.1 , $H_\tau(\beta_1) \leq H_\tau(\beta_1 + C_{n+1})$}

$$t_1 - G_\tau(t_1)$$

Uniendo ambos resultados (A) y (A)

$$t_1 - G_\tau(t_1) \equiv \Delta(0) \tag{A.4}$$

- Empleando el mismo razonamiento tenemos que: si $j > 1$ entonces $D_{n+j} - H_\tau(D_{n+j})$ es:
  = {Por la definición de D_{n+j}}

$$\beta_j + C_{n+j} - H_\tau(\beta_j + C_{n+j})$$

$\geq$ {Puesto que $\beta_j \leq \beta_j + C_{n+j}$ y de la definición de β}

$$\beta_{j-1} - H_\tau(\beta_{j-1}) = \Delta(j-1)$$

De la expresión (A)
= {Substituyendo C_{n+j} por su definición}

$$\beta_j + \beta_{j-1} - H_\tau(\beta_{j-1}) - \beta_j + H_\tau(\beta_j) - H_\tau(\beta_j + C_{n+j})$$

= {Cancelando términos comunes}

$$\beta_{j-1} - H_\tau(\beta_{j-1}) + H_\tau(\beta_j) - H_\tau(\beta_j + C_{n+j})$$

$\leq$ {Empleando la propiedad 4.1 , $H_\tau(\beta_j) \leq H_\tau(\beta_j + C_{n+j})$}

$$\beta_{j-1} - H_\tau(\beta_{j-1}) = \Delta(j-1)$$

Uniendo ambos resultados (A) y (A).

$$D_{n+j} - H_\tau(D_{n+j}) \equiv \beta_{j-1} - H_\tau(\beta_{j-1}) \equiv \Delta(j-1) \tag{A.5}$$

- Otro resultado que necesitaremos para la demostración es:
  $\sum_{i=j}^{k} C_{n+i}$
  = {De la definición de C_{n+i}}

$$\sum_{i=j}^{k} (\Delta(i-1) - \Delta(i))$$

= {Expandiendo el sumatorio}

$$\Delta(j-1) - \Delta(j) + \Delta(j) - \Delta(j+1) \cdots \Delta(k-1) - \Delta(k)$$

$\equiv$ {Cancelando términos equivalentes, y puesto que $\Delta(k) = 0$}

$$\sum_{i=j}^{k} C_{n+i} = \Delta(j-1) \tag{A.6}$$

- El valor de la función $H_{\tau^j}(t)$ es, para valores de t menores que t_1
  {Por la propiedad 4.3}

$$H_{\tau^j}(t) = \begin{cases} t < D_{n+j} & H_{\tau^{j-1}}(t) \\ t \geq D_{n+j} & H_{\tau^{j-1}}(t) + C_{n+k-j+1} \end{cases}$$

  {Por la ordenación de los plazos (A.3)}

$$H_{\tau^j}(t) = \begin{cases} t < D_{n+j} & H_{\tau}(t) + \sum_{i=1}^{j-1} C_{n+k-i+1} \\ t \geq D_{n+j} & H_{\tau}(t) + \sum_{i=1}^{j} C_{n+k-i+1} \end{cases} \tag{A.7}$$

- De forma similar, obtenemos el siguiente resultado para la función G_{τ^j}.
  {Utilizando la propiedad 4.4}

$$G_{\tau^j}(t) = \begin{cases} t \leq t_1 & G_{\tau}(t) + \sum_{i=1}^{j} C_{n+k-i+1} \end{cases} \tag{A.8}$$

DEMOSTRACIÓN: Vamos a demostrar que τ^j es planificable para todo j. Con lo que habremos demostrado que $\tau^k = \tau \cup \theta$ tambien lo es. Emplearemos el lema 4.3 para verificar la planificabilidad, con lo que tenemos que comprobar las tres condiciones requeridas en el lema:

1. $\{D_{n+j} \leq P_{n+j}\}$
   Si $j \neq 1$
   $D_{n+j} = \beta_j + C_{n+j}$
   $=$ {Por definición de C_{n+j}}

$$\beta_j + \beta_{j-1} - H_{\tau}(\beta_{j-1}) - \beta_j + H_{\tau}(\beta_j)$$

   $=$ {Cancelando términos semejantes}

$$\beta_{j-1} - H_{\tau}(\beta_{j-1}) - H_{\tau}(\beta_j)$$

   $\leq$ {De la expresión (A.1) $\beta_j < \beta_{j-1}$, y puesto que τ es planificable}

$$\beta_{j+1}$$

   $\leq$ {Empleando la expresión (A.1)}

$$D_{n+j} \leq t_1 \equiv P_{n+j}$$

   Si $j = 1$
   $D_{n+1} = \beta_1 + C_{n+1}$
   $=$ {Por definición de C_{n+1}}

$$\beta_1 + t_1 - G_{\tau}(t_1) - \beta_1 + H_{\tau}(1)$$

$= \{$Cancelando términos semejantes$\}$

$$t_1 - G_\tau(t_1) + H_\tau(\beta_1)$$

$\leq \{$De la expresión (A.1) $\beta_1 \leq t_1$, y puesto que τ es planificable$\}$

$$D_{n+1} \leq t_1 \equiv P_{n+k}$$

2. $\{G_\tau(t_1) < t_1\}$
   Puesto que el lema 4.4 tiene la misma condición que este teorema, podemos emplearlo para concluir que existe un instante de tiempo t_1 tal que $G_\tau(t_1) < t_1$.

3. $\{\forall D_{n+k} \leq t \leq t_1 : D_{n+k} - H_\tau(D_{n+k}) \leq t - H_\tau(t)\}$
   Primero probaremos que cualquier tarea –pero sólo una– puede ser incorporada al conjunto original de tareas, τ, conservando la tercera condición del lema 4.3. Un caso particular es que la tarea T_k puede ser añadida sin causar la perdida de ningún plazo. Seguidamente, empleadon un método inductivo, supondremos que τ^{i-1} cumple esta propiedad, y comprobaremos que τ^i también la cumple.
   Veamos el resultado general:
   $\{$Por (A.4) y (A.5), para todo $j > 0\}$
   $D_{n+j} - H_\tau(D_{n+j}) = \Delta(j-1)$
   $\{$De la definición de $\beta\}$
   $\forall_{\beta_j \leq t \leq \beta_{j-1}} \Delta(j-1) \leq t - H_\tau(t)$
   $\equiv \{$Con el último resultado, la definición de Δ, y (A.2)$\}$

$$\forall t \geq D_{n+j} : D_{n+j} - H_\tau(D_{n+j}) \leq t - H_\tau(t)$$

   Supongamos que τ^{i-1} cumple la tercera condición. Tenemos que comprobar que:
   $\{\forall D_{n+k-i+1} \leq t \leq t_1 : D_{n+k-i+1} - H_{\tau^{i-1}}(D_{n+k-i+1}) \leq t - H_{\tau^{i-1}}(t)\}$
   $\equiv \{$Empleando el resultado de la expresión (A.7) para substituir $H_{\tau^{i-1}}\}$

$$D_{n+k-i+1} - H_\tau(D_{n+k-i+1}) - \sum_{x=1}^{i-1} C_{n+k-x+1} \leq t - H_\tau(t) - \sum_{x=1}^{i-1} C_{n+k-x+1}$$

   $\equiv \{$Cancelando términos comunes$\}$

$$D_{n+k-i+1} - H_\tau(D_{n+k-i+1}) \leq t - H_\tau(t)$$

   $\{$Empleando el resultado generico antes obtenido$\}$

$$D_{n+j} : D_{n+j} - H_\tau(D_{n+j}) \leq t - H_\tau(t)$$

Por lo que la tercera condición del lema se cumple.

Finalmente, el tiempo de cómputo de $T_{n+k-i+1}$ tiene que ser menor o igual que $\min\{(t_1 - G_\tau(t_1), D_{n+k-i+1} - H_{\tau^{i-1}}(D_{n+k-i+1})\}$ para complir con los requerimientos de lema 4.3.

Comprobemos primero que $D_{n+k-i+1} - H_{\tau^{i-1}}(D_{n+k-i+1})$ es menor o igual que $t_1 - G_{\tau^{i-1}}(t_1)$.

$D_{n+k-i+1} - H_{\tau^{i-1}}(D_{n+k-i+1}) \leq t_1 - G_{\tau^{i-1}}(t_1)$
$\equiv$ {Empleaando los resultados (A.7) y (A.8)}

$$D_{n+k-i+1} - H_\tau(D_{n+k-i+1}) - \sum_{x=1}^{i-1} C_{n+k-x+1} \leq t_1 - G_\tau(T_1) - \sum_{x=1}^{i-1} C_{n+k-x+1}$$

$\equiv$ {Cancelando términos similares}

$$D_{n+k-i+1} - H_\tau(D_{n+k-i+1}) \leq t_1 - G_\tau(t_1)$$

este resultado es cierto (ver resultado (A.2)), y por tanto la premisa también lo es.

Nos queda por verificar que $C_{n+k-i+1}$ es menor o igual que $D_{n+k-i+1} - H_{\tau^{i-1}}(D_{n+k-i+1})$.
$C_{n+k-i+1} \leq D_{n+k-i+1} - H_{\tau^{i-1}}(D_{n+k-i+1})$
$=$ {Empleando la ecuación (A.7)}

$$D_{n+k-i+1} - H_\tau(D_{n+k-i+1}) - \sum_{x=1}^{i-1} C_{n+k-x+1}$$

$=$ {De las expresiones (A.4) y (A.5)}

$$\Delta(k-i) - \sum_{x=1}^{i-1} C_{n+k-x+1}$$

$=$ {Substituyendo $\Delta(k-i)$ por $\sum_{x=k-i+1}^{k} C_{n+x}$ y reordenando los índices del sumatorio}

$$\sum_{x=1}^{i} C_{n+k-x+1} - \sum_{x=1}^{i-1} C_{n+k-x+1}$$

$=$ {Cancelando términos}

$$C_{n+k-i+1}$$

demostración de que τ^k es polanificable.

La segunda parte del teorema —$\{d_{n+1}, \cdots, d_{n+k}\}$ son los plazos más cortos que garantizan la planificabilidad de τ^k— se puede demostrar más fácilmente. Recordemos que la expresión $d_i - H_\tau(d_i)$ representa la cantidad de tiempo de holgura que la activación con plazo d_i tiene para completar su ejecución. En el último paso de la tercera condicón del lema 4.3 obtuvimos que:

$$D_{n+k-i+1} - H_{\tau^{i-1}}(D_{n+k-i+1}) \;=\; C_{n+k-i+1}$$

Por la propiedad 4.4

$$D_{n+k-i+1} - H_{\tau^i}(D_{n+k-i+1}) \;=\; 0,$$

Con lo que llegamos a que no hay la tarea T_{+k-i+1} finaliza su ejecución justo cunado se cumple su plazo. Con lo que un plazo menor produciría el incumplimiento de este.

□

Esta página contiene intencionadamente sólo esta frase.

Apéndice B

Simbología de los cronogramas

La mayor parte de los cronogramas se han obtenido a partir de un programa de simulación propio, sobre el que se han implementado todos los algoritmos propuestos.

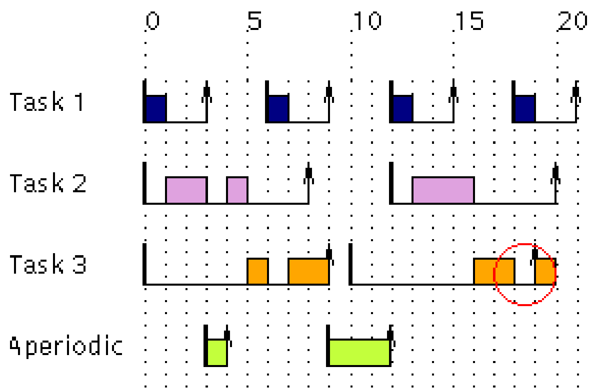


Figura B.1: Cronograma ejemplo

Petición: Barra vertical gruesa.

Plazo: Pequeña flecha ascendente.

Ejecución: Rectangulo sólido. Cada tarea tiene un color distinto.

Plazo sobrepasado: Círculo sobre el instante aproximado.

La línea vertical que representa la llegada de una petición está unida con la flecha de finalización del plazo de forma que el intervalo en el que se han de ejecutar la tarea aparece como una caja.

Esta página contiene intencionadamente sólo esta frase.

Bibliografía

[Alonso94] A. Alonso-Muñoz, "*Extensiones a los Métodos de Planificación de Sistemas de Tiempo Real Críticos basados en Prioridades*", Tesis Doctoral, Universidad Politécnica de Madrid, Facultad de Infromática, Feb 1994.

[Audsley91a] N.C. Audsley, "*Resource Control for Hard Real-Time Systems: A Review*", YCS-159(1991), University of York, Dep. of Computer Science, Ago 1991.

[Audsley91b] N.C. Audsley, A. Burns, M.F. Richardson y A.J. Wellings, "*Hard Real-Time Scheduling: The Deadline Monotonic Aproach*" Proceedings 8^{th} WorkShop on Real-Time Operating Systems and Software, 1991, pp. 127–132.

[Audsley93] N.C. Audsley, A. Burns, M. Richardson, K.W. Tindell y A.J. Wellings, "*Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling*", Software Engineering Journal, Vol. 8(5) Sept 1993 pp. 284–292.

[Audsley95] N.C. Audsley, A. Burns, R.I. David, K.W. Tindell y A.J. Wellings, "*Fixed Priority Pre-emptive Scheduling: An Historical Prespective*", Real-Time Systems, Vol 8(2/3), Mar/May 1995, pp. 173–198.

[Baker90] T. Baker, "*Protected Records, Time Management and Distribution*", ACM ADA Letters X(9), 1990, pp. 17–102.

[Baruah90a] S. Baruah, L. Rosier y R.R. Howell, "*Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor*", The Journal of Real-Time Systems, Dec 1990, pp. 301–324.

[Baruah90b] S.Baruah, A. Mok y L. Rosier, "*Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One periodic*", 11^{th} Real-Time Systems Symposium, Dec 1990, pp. 182–190.

[Bihari91] T. Bihari y K. Schwan, "*Dynamic adaptation of Real-Time software*", ACM Transactions Computer Systems and Software, May 1991.

[Burns91] A. Burns, "*Scheduling Hard Real-Time Systems: A Review*", Software Ingineering Journal, 6(3) 1991, pp. 116–128.

[Burns93] A. Burns, A.J. Wellings, C.M. Bailey y E. Fyfe, "*The Olympus Attitude and Orbital Control System: A Case Study in Hard Real-Time System Design and Implementation*", Department of Computer Science, University of York, report YCS 190 1993.

[Chen90] M.I. Chen y K.J. Lin , "*Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems*", The Journal of Real-Time Systems, Feb 1990, pp. 325–346.

[Chetto89] H. Chetto y M. Chetto, "*Some results of the Earliest Deadline Scheduling Algorithm*", IEEE Transactions Software Engineering, Oct 1989.

[Chetto90] H. Chetto, M. Silly y T. Bouchentouf, "*Dynamic Scheduling of Real-Time Tasks under Precedence Constrains*", The Journal of Real-Time Systems, Feb 1990, pp. 181–194.

[Davis93] R.I. Davis, K.W. Tindell y A.J. Burns, "*Scheduling Slack Time in Fixed Priority Pre-emptive Systems*", Proceedings Real-Time Systems Symposium, Dec 1993, pp. 222-231.

[Delacroix96] J. Delacroix, "it Towards a Stable Earliest Deadline Scheduling Algorithm", The Journal of Real-Time Systems, May 1996, pp. 263–291.

[Dertouzos74] M. Dertouzos, "*Control robotics : the procedural control of physical processors*", Proceedings of the IFIP Congress, 1974, pp. 807–813.

[Dijkstra72] E.W. Dijkstra, "*Notes on Structured Programing*", Structured Programming, O.J. Dahl, E.W. Dijktra y C.A.R. Hoare, Eds, London: Academic Press, 1972.

[Ghazalie95] T.M. Ghazalie y T.P. Baker, "*Aperiodic Servers in a Deadline Scheduling Environment*", The Journal of Real-Time Systems, Sep 1995, pp. 31–67.

[Garey79] M.R. Garey y D.S. Johnson, "*Computer And Intractability, A Guide to the Theory of NP-Completeness*", Ed: W.H. Freeman And Company, 1979.

[Henn78] R. Henn, "*Antwortzeitgesteuerte Prozessorzutilung unter Strengen Zeitbedingungen*", Computing, (19) 1978, pp. 202–220.

[Homayoun94] N. Homayoun y P. Ramanathan, "*Dynamic Priority Scheduling of Periodic and Aperiodic Tasks in Hard Real-Time Systems*", The Journal of Real-Time Systems, (6) 1994, pp. 207-232.

[Kopetz89] H. Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft y R. Zainlinger, "*Distributed fault-tolerant real-time systems: The MARS Approach*", IEEE Micro, Feb 1989, pp. 25–40.

[Knuth69] D. Knuth, "*The art of computer programming*", Vol. 2, 1969, pp. 293–304,324.

[Lehoczky87] J.P. Lehoczky, L. Sha y J.K. Strosnider, "*Enhanced Aperiodic Responsiveness in Hard-Real-Time Environments*", Proceedings of the Real-Time Systems Symposium, Dec 1987, pp. 261–270

[Lehoczky89] J.P. Lehoczky, L. Sha y Y. Ding, "*The Rate Monotonic Scheduling Algorithm: Exact Characterisation and Average Case Behaviour*", Proceedings of the Real-Time Systems Symposium, Dec 1989.

[Lehoczky90] J.P. Lehoczky, "*Fixed priority scheduling of periodic task sets with arbitrary deadlines*", Proceedings of the 11^{th} Real-Time Systems Symposium, Dec 1990, pp. 201–209.

[Lehoczky92] J.P. Lehoczky y S. Ramos-Thuel, "*An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks Fixed-Priority Preemptive Systems*", Proceedings Real-Time Systems Symposium, Dec 1992, pp. 110-123.

[Leung80] J. Leung y M. Merrill, "*A note on the preemptive scheduling of periodic, Real-Time tasks*", Information Processing Letters, Vol. 11, 1980, pp. 115–118.

[Leung82] J. Leung y J. Whitehead, "*On the complexity of fixed-priority scheduling of periodic, Real-Time tasks*", Performance Evaluation, Vol. 2(4), Dec 1982, pp. 237–250.

[Liu73] C. Liu y J. Layland, "*Scheduling algorithms for multiprogramming in a hard Real-Time environment*", Journal of the ACM, Vol. 20(1) 1973, pp. 46–61.

[Mok78] A.K. Mok y M.L. Dertouzos, "*Multiprocessor Scheduling in a Hard Real-Time Environment*", Proceedings of the Seven Texas Conference on computing Systems, Nov 1978.

[Mok83] A.K. Mok, "*Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*", PhD Thesis, MIT Laboratory for Computer Science, May 1983.

[Mok84] A.K. Mok, "*The design of real-time programing systems based on process models*", Proceedings Rel-Time Systems Symposium, Dec 1984, pp. 5–7.

[Rajkumar89] R. Rajkumar, L. Sha y J.P. Lehoczky, "*En Experimental Experimentarion of Synchronisation Protocols*", Proceedigns 6^{th} Workshop on Real-Time Operating Systems and Software, May 1989, pp. 11–17.

[Ramos-Thuel93] S. Ramos-Thuel, y J.P. Lehoczky, "*On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems*", Proceedings Real-Time Systems Symposium, Dec 1993, pp. 160-171.

[Ripoll96] I. Ripoll, A. Crespo y A. Mok, "*Improvement in Feasibility Testing for Real-Time Tasks*", I. Ripoll, A. Crespo y A.K. Mok. The Real-Time Systems Journal (11) pp. 19–39 1996. Ed: Kluwer Academic Publichers. Academic Publishes.

[Ripoll95] "*An OnlLine Optimal Aperiodic Server for Dynamic Scheduling Environments*", I. Ripoll, A. García-Frones y A. Crespo. Workshop Design and Analysis of Real-Time Systems, Brussels, Belgium, Nov. 1995.

[Schwan92] K. Schwan y H. Zhou, "*Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads*", IEEE Transactions on Software Engineering, Vol 18 Aug 1992, pp. 736–747.

[Sha86] L. Sha, R. Rajkumar y J.P. Lehoczky, "*Solutions for some Practical Problems in Prioritised Preemptive Scheduling*", Proceedings of the Real-Time Systems Symposium, Dic 1986, pp. 181–191.

[Sha87] L. Sha, R. Rajkumar y J.P. Lehoczky, "*Priority inheritance protocols: An approach to real-time synchronization*", CMU-CS-87-181, Computer Science Department, Carnegie-Mellon University, Dic 1987.

[Sha89] L. Sha, R. Rajkumar, J.P. Lehoczky, K. Ramamritham, "*Mode Change Protocols for Priority-Driven Preemptive Scheduling*", The Journal of Real-Time Systems, 1(3), 1989.

[Sha90] L. Sha, R. Rajkumar y J.P. Lehoczky, "*Priority inheritance protocols: An approach to real-time synchronization*", IEEE Transactions on Computers, Vol. 93, Sep 1990, pp. 1175–1185.

[Shawan92] K. Schawan y H. Zhou, "*Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads*", IEEE Transactions on Software Engineering, vol 18, Aug 1992, pp. 736-747.

[Sprunt88] B. Sprunt, J.P. Lehoczky y L. Sha, "*Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm*", Proceedings of the Real-Time System Symposiun, Dec 1988, pp. 251–258.

[Sprunt89] B. Sprunt, L. Sha y J.P. Lehoczky, "*Aperiodic Task Scheduling for Hard Real-Time Systems*", The Journal of Real-Time Systems, 1989, pp. 27–60.

[Spuri96] M. Spuri y G. Buttazzo, "*Scheduling Aperiodic Tasks in Dynamic Priority Systems*", The Journal of Real-Time Systems, Mar 1996, pp. 179–210.

[Stankovic87] J.A. Stankovic, "*Tutorial on Hard Real-Time Systems*", IEEE Computer Society Press, 1987.

[Stankovic88a] J.A. Stankovic, " Real-Time Computing Systems: The Next Generation", Tech report TR-88-06, COINS Dep., Univ. of Masachusetts, Ene 1988.

[Stankovic88b] J.A. Stankovic, "*Misconceptions About Real-Time Computing: A Serious Problem for Next Generation Systems*", IEEE Computer, Oct 1988, pp. 10–19.

[Stankovic95] J.A. Stankovic, M. Spuri, M.D. Natale y G.C. Buttazzo, "*Implications of Classical Scheduling Results for Real-Time Systems*", IEEE Computer, Jun 1995, pp. 16–25.

[Tia95] T.S. Tia, "*Utilizing Slack Time For Aperiodic and Sporadic Requests Scheduling in Real-Time Systems*", PhD Thesis, University of Illinois at Urbana-Champaign, 1995.

[Tia96] T.S. Tia, J.W.-S. Liu y M. Shankar, "*Algorithms and Optimality of Scheduling Soft Aperiodic Requests in fixed-Priority Preemptive Systems*", The Journal of Real-Time Systems, Ene 1996, pp. 23–43.

[Weiderman92] N.H. Weiderman, N.I. Kamenoff, "*Hartsone Uniprocessor Benchmark: Definitons and Experiments for Real-Time Systems*", The Journal of Real-Time Systems, Apr 1992, pp. 353–382.

[Woodbury88] M.H. Woodbury y K.G. Shin, "*Evaluation of the Probability of Dynamic Failure and Processor Utilization for Real-Time Tasks*" Proceedings of the Real-Time Systems Symposium, Dec 1988, pp. 222–231.

[Xu93] J. Xu y D.L. Parnas "*On Satisfying Timing Constraints in Hard-Real-Time Systems*", IEEE Trans. on Software Engineering, (19) Jan 1993, pp. 70–84.

[Zamorano92] J. Zamorano, J.L. Redondo, C. Blanco, J.I. Tortosa y J.A. de la Puente, "*Ejecutivo cíclico, Manual de Usuario*", Ene 1992.