Sáez Barona, S.; Crespo, A. (2013). Mejora de los test de planificabilidad para asignación incremental de tareas en sistemas multiprocesadores de tiempo real. Revista Iberoamericana de Automática e Informática Industrial RIAI. 10(2):197-203. doi:10.1016/j.riai.2013.03.006.

Mejora de los Test de Planificabilidad para Asignación Incremental de Tareas en Sistemas Multiprocesadores de Tiempo Real

Sergio Sáez y Alfons Crespo
{ssaez,acrespo}@ai2.upv.es
Grupo de Informática Industrial y Sistemas de Tiempo Real
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia

Resumen

Durante la etapa de diseño de un sistema multiprocesador de tiempo real, los test de planificabilidad son una parte clave de los algoritmos de asignación de tareas. El uso de test de planificabilidad exactos permite aumentar la eficiencia de los algoritmos de asignación a costa de un incremento en el tiempo de computo necesario para validar una partición. Aunque existen varios estudios que mejoran el rendimiento de dichos test de planificabilidad, ninguno se ha centrado en el contexto en que dichos test se utilizan. Este trabajo presenta diversas mejoras en los test de planificabilidad exactos basándose en la naturaleza incremental del proceso de asignación.

La versión final de este artículo ha sido publicada por Elsevier España, S.L. en Revista Iberoamericana de Automática e Informática Industrial (RIAI). 10(2):197-203. doi:10.1016/j.riai.2013.03.006.

1. Introducción

Los sistemas de tiempo real son cada vez más complejos y las plataformas de ejecución basadas en sistemas multiprocesadores están convirtiéndose en una solución habitual para dichos sistemas. Los sistemas complejos suelen estar formados por un conjunto de tareas que se ejecutan concurrentemente sobre la plataforma de ejecución con el fin de conseguir cumplir los requisitos u objetivos establecidos. Los sistemas de tiempo real, además de la corrección funcional, deben cumplir un conjunto de requisitos no funcionales que incluyen restricciones temporales sobre la ejecución del conjunto de tareas, como puede ser la entrega de los resultados, por parte de cada tarea, antes de un determinado plazo de entrega.

La consecución de las restricciones temporales por parte de un sistema de tiempo real se consigue mediante la adecuada combinación de:

- Un *algoritmo de planificación* de tareas que se encargue del reparto de los recursos de procesamiento entre las tareas activas siguiendo algún criterio bien conocido (prioridades fijas, primero la tarea con el plazo de entrega más cercano, etc).
- Un *test de planificabilidad* que garantice, antes de la puesta en marcha del sistema, que un determinado conjunto de tareas va

a cumplir las restricciones temporales sobre una plataforma de ejecución dada bajo el control de un determinado algoritmo de planificación.

Los test de planificabilidad suelen considerar un modelo del sistema en el que cada tarea es una secuencia infinita de activaciones con un intervalo fijo (o mínimo) entre las mismas. Cada activación solicita una cierta cantidad de cómputo que debe atenderse antes de un plazo de entrega determinado. Con esta información el test de planificabilidad debe garantizar que bajo un cierto algoritmo de planificación el conjunto de tareas siempre va a cumplir las restricciones temporales.

En un sistema monoprocesador, las dos aproximaciones mayoritarias se basan en el uso de un algoritmo de planificación basado en prioridades fijas [1, 2] o en prioridades dinámicas [2]. En el caso de los esquemas basados en prioridades fijas, asociado al test de planificabilidad suele ir un determinado *algoritmo de asignación de prioridades* que determinará la forma en que el algoritmo de planificación seleccionará las tareas para su ejecución [2, 3].

Sin embargo, en una plataforma de ejecución con múltiples procesadores el sistema debe determinar no sólo los parámetros que permitan realizar la ordenación de las tareas en ejecución, sino también el criterio utilizado para la selección del procesador en que debe ejecutar cada tarea. Dependiendo de si estas decisiones se toman en la etapa de diseño o durante la ejecución del sistema, y si se permite que una tarea pueda cambiar o no de procesador, existen múltiples aproximaciones [4, 5]. Aunque algunas de estas aproximaciones ofrecen una gran flexibilidad, una de las opciones más aceptadas, sobretodo en los sistemas críticos [6, 7] es el particionado o asignación de tareas a procesadores [8].

En los esquemas de asignación estática de tareas entra en juego un nuevo componente: el *algoritmo de asignación* de tareas. El algoritmo de asignación es el encargado de generar una partición viable del conjunto de tareas es decir, una partición en la que cada subconjunto de tareas asignadas a un procesador sea planificable. Para asegurar la planificabilidad de cada subconjunto, el algoritmo de asignación utiliza un test de planificabilidad para sistemas monoprocesador acorde con el algoritmo de planificación utilizado en cada procesador.

Estos test de planificabilidad suelen clasificarse en test de condición *suficiente* [2] o condición *suficiente y necesaria* o test exactos [9, 10]. Ambos tipos de test de planificabilidad han sido utilizado en los esquemas de asignación de tareas [8, 11], siendo los test exactos los que mejores resultados ofrecen a costa de un incremento notable en el tiempo de computo necesario para determinar la planificabilidad de un subconjunto de tareas. Con el objetivo de reducir el tiempo de ejecución de los test de planificabilidad es habitual el uso de los test de condición suficiente en primera instancia, aplicando los test exactos sólo en caso de que los anteriores indiquen que no se puede asegurar la planificabilidad de un subconjunto de tareas.

Recientemente se han publicado optimizaciones que mejoran el rendimiento temporal de los test de planificabilidad exactos, tanto de los basados en prioridades fijas [12] como los basados en prioridades dinámicas [13]. Sin embargo, en ninguno de ellos se tiene en cuenta la naturaleza, habitualmente secuencial o incremental, del proceso de asignación de tareas. Aunque el algoritmo de asignación puede considerar todas las tareas del sistema simultáneamente e intentar conseguir una partición del mismo mediante la

aplicación de algún algoritmo de optimización [14], lo habitual es aplicar algoritmos heurísticos más simples utilizados en los problemas de *bin packing*, como el *first-fit* o el *best-fit* [8]. En ambos heurísticos, cada tarea del sistema se intenta asignar de forma secuencial a uno de los M subconjuntos de tareas que representan los procesadores del sistema. Cada vez que se intenta asignar una nueva tarea a uno de los M subconjuntos, se debe aplicar el test de planificabilidad para comprobar si el nuevo subconjunto de tareas sigue siendo planificable tras asignar la nueva tarea.

El presente trabajo propone modificar los test de planificabilidad para tener en cuenta que lo que se quiere comprobar es si una nueva tarea mantiene planificable un conjunto de tareas que previamente ya lo era. Para ello se intentará aprovechar el conocimiento de cuál es la nueva tarea, así como los cálculos realizados para el resto de las tareas en comprobaciones anteriores.

2. Modelo del sistema

Dada una aplicación de tiempo real compuesta por múltiples tareas, denominaremos $\mathcal{T}$ al conjunto de tareas que la componen, siendo N la cardinalidad del mismo ($N = |\mathcal{T}|$). Cada tarea T_i vendrá especificada por la tupla (C_i, D_i, P_i), donde C_i es el tiempo de ejecución de peor caso de la tarea, D_i es su plazo de entrega relativo al instante de activación y P_i es el periodo de activación de la misma. Se denomina factor de utilización de una tarea T_i a la fracción $U_i = C_i/P_i$ y densidad a la fracción $\delta_i = C_i/D_i$.

La plataforma de ejecución está formada por un sistema con M procesadores idénticos, cada uno de los cuales ejecuta el algoritmo de planificación local S. Se denominará $\mathcal{M}$ al conjunto de M procesadores del sistema.

Se desea realizar una partición $\mathcal{P} = \{\mathcal{P}_j : \forall j \in \mathcal{M}\}$ del conjunto de tareas $\mathcal{T}$ de manera que cada subconjunto de tareas $\mathcal{P}_j$ sea planificable bajo el algoritmo de planificación S y que, por lo tanto, pueda ser ejecutado en el procesador j del sistema. Al ser $\mathcal{P}$ una partición de $\mathcal{T}$ se debe cumplir que:

$$\bigcup_{j \in \mathcal{M}} \mathcal{P}_j = \mathcal{T} \tag{1}$$

$$\mathcal{P}_j \cap \mathcal{P}_k = \emptyset, \forall j, k \in \mathcal{M} : j \neq k \tag{2}$$

es decir, cada tarea del sistema T_i debe pertenecer a uno y sólo uno de los subconjunto $\mathcal{P}_j$ de la partición $\mathcal{P}$[1].

Para construir la partición $\mathcal{P}$ se va a utilizar un algoritmo heurístico $\mathcal{A}$ de asignación incremental de tareas que intentará construir la partición $\mathcal{P}$ añadiendo las tareas de $\mathcal{T}$ de una en una. Así pues, el algoritmo de asignación realizará un máximo de N pasos, en cada uno de los cuales intentará asignar una de las tareas de $\mathcal{T}$ a uno de los subconjuntos $\mathcal{P}_j$. Sin pérdida de generalidad se supondrá que el conjunto de tareas $\mathcal{T}$ se ha ordenado con algún tipo de criterio[2] y que el subíndice i de la tarea T_i indica su posición en dicho orden. El estado de un subconjunto de tareas $\mathcal{P}_j$ se denominará $\mathcal{P}_j^k$ tras la asignación de T_k a uno de los procesadores de $\mathcal{M}$. Tengase en cuenta que $\mathcal{P}_j^k \neq \mathcal{P}_j^{k-1}$ si y solo si $T_k \in \mathcal{P}_j^k$ y que si el algoritmo $\mathcal{A}$ ha alcanzado el paso k, entonces todos los subconjuntos $\mathcal{P}_j^{k-1}$ son planificables bajo el algoritmo S. Esta nomenclatura

[1] Nótese que se ha eliminado la restricción $\forall j \in \mathcal{M}, \mathcal{P}_j \neq \emptyset$ por ser innecesaria en un sistema multiprocesador.

[2] Un criterio de ordenación habitual es en orden decreciente de factores de utilización ($U_i = C_i/P_i$).

se utilizará en general en todas las formulas implicadas en los test de planificabilidad, aplicando el superíndice k para los valores de las mismas tras la asignación de la tarea T_k.

En el presente trabajo se va a suponer que las tareas de $\mathcal{T}$ cumplen que $C_i \leq D_i \leq P_i$ y que no sufren tiempos de bloqueo por el acceso a recursos compartidos. El uso de recursos compartidos, locales o globales, se deja como una futura extensión al presente estudio.

Dado que el objetivo del trabajo es reducir el tiempo de ejecución del proceso de asignación de tareas, se va suponer que el algoritmo de asignación siempre comprueba si un conjunto de tareas es planificable utilizando los test de planificabilidad de condición suficiente y si éstos no son capaces de determinar la planificabilidad del sistema, entonces se aplicarán los test exactos. En todos los casos, si la utilización U_j^k del conjunto $\mathcal{P}_j^k$ es superior a 1, el procesador j se descartará como candidato.

3. Test Basados en Prioridades Fijas

En esta sección se van a tratar de optimizar los cálculos necesarios para llevar a cabo un test de planificabilidad de condición suficiente y un test de planificabilidad exacto para un conjunto de tareas planificadas bajo un algoritmo de prioridades fijas. En general, los resultados que a continuación se muestran son independientes del algoritmo de asignación de prioridades utilizado.

Se supone que el sistema dispone de un número de prioridades igual o mayor al máximo número de tareas asignadas a un procesador y, por lo tanto, cada tarea T_i tiene una prioridad distinta al resto de tareas asignadas al mismo procesador.

Los test de planificabilidad que se presentan a continuación se centran en un único procesador y, por lo tanto, el conjunto de tareas al que se refieren, así como sus subíndices utilizados en los atributos de las tareas (C_i, D_i, etc), hacen referencia a las tareas asignadas a dicho procesador, es decir al conjunto $\mathcal{P}_j^k$, siendo $n = |\mathcal{P}_j^k|$.

3.1. Test basados en la utilización

En el trabajo [2] se propone un test de planificabilidad suficiente basado en la utilización de un conjunto de tareas cuyas prioridades se han asignado mediante el criterio *Rate Monotonic* y con los plazos de entrega igual a los periodos. Bajo ese supuesto un conjunto de tareas es planificable si cumple la condición:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(2^{1/n} - 1\right) \tag{3}$$

Para un sistema con plazos de entrega restringidos ($D_i \leq P_i$) cuyas prioridades se han asignado mediante el criterio *Deadline Monotonic* se puede utilizar la densidad de las tareas para realizar una comprobación similar:

$$\delta = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n\left(2^{1/n} - 1\right) \tag{4}$$

Aunque este test de planificabilidad tiene un coste temporal muy bajo ($O(n)$), se puede realizar una pequeña optimización si los resultados del paso de asignación $k-1$, U^{k-1} y δ^{k-1}, se almacenan en una estructura asociada a cada procesador. En ese caso el test quedaría como sigue:

$$\delta^{k-1} + \frac{C_k}{D_k} \leq n\left(2^{1/n} - 1\right) \tag{5}$$

siendo este test de coste $O(1)$. El valor de δ^k se actualizará a su nuevo valor $\delta^{k-1} + C_k/D_k$ sólo

si finalmente la tarea T_k se asigna al procesador bajo estudio. Este criterio de actualización se aplicará a todos los datos intermedios que se necesiten en cada paso de asignación.

3.2. Test de análisis del tiempo de respuesta

El test de Liu y Layland tiende a un límite en la utilización de un procesador aproximadamente del 69 % ($\ln 2$). El uso de un test exacto permite eliminar este límite obteniendo así un mejor comportamiento de los algoritmos de asignación de tareas. En el trabajo [9] se propone el mecanismo para calcular el tiempo de respuesta de peor caso de cada tarea del sistema mediante la formula:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j \qquad (6)$$

siendo $hp(i)$ el conjunto de tareas más prioritarias que T_i. Está formula se puede resolver mediante la relación de recurrencia:

$$r_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i^n}{P_j} \right\rceil C_j \qquad (7)$$

que termina cuando se cumple que $r_i^{n+1} = r_i^n$, siendo así el tiempo de respuesta en el peor caso $R_i = r_i^{n+1}$, siempre que se cumpla $r_i^{n+1} \leq D_i$. En caso contrario el conjunto de tareas no es planificable, ya que T_i podría perder su plazo de entrega.

Prácticamente todas las optimizaciones que se han realizado sobre este test de planificabilidad se basan en utilizar como valor inicial de la relación de recurrencia, r_i^0, una cota inferior de R_i lo más ajustada posible, para así reducir el número de iteraciones necesarias para resolver R_i. Desde el caso básico $r_i^0 = C_i$, pasando por $r_i^0 = R_{i-1} + C_i$ [15, 16] suponiendo que T_{i-1} es la tarea con una prioridad inmediatamente superior a T_i, hasta algunas cotas más elaboradas presentadas recientemente [12].

Las optimizaciones que a continuación se presentan se pueden aplicar independientemente del valor inicial utilizado en la relación de recurrencia. Al valor inicial utilizado, que combinaremos con la propuesta que a continuación se presenta, lo denominaremos r_i^{-1}.

Optimización 1

La primera optimización es trivial e inherente a la naturaleza del esquema de prioridades fijas. Dado que el conjunto de tareas $\mathcal{P}_j^{k-1}$ se ha comprobado que es planificable, todas las tareas de dicho conjunto tienen un tiempo de respuesta que cumple la restricción $R_i \leq D_i$. Por la definición de R_i se puede ver que al añadir la nueva tarea T_k al conjunto de tareas sólo habrá que calcular el tiempo de respuesta de aquellas tareas con una prioridad inferior a T_k, $lp(k) = \{T_i : T_k \in hp(i)\}$, ya que las tareas de $hp(k)$ no cambiarán su tiempo de respuesta[3].

Optimización 2

Al añadir una nueva tarea T_k a un conjunto de tareas, las tareas con menor prioridad, $lp(k)$, verán aumentado su tiempo de respuesta al menos al tiempo de respuesta que ya tenían más la interferencia producida por la nueva tarea. Se plantea utilizar dicho valor como cota inferior del tiempo de respuesta:

[3]Esto no se mantendría en el caso del uso de recursos compartidos, a no ser que se hubiera tomado como tiempo de bloqueo de las tareas de $hp(k)$ el tiempo de bloqueo para un recurso global.

$$LR_i = R_i^{k-1} + \left\lceil \frac{R_i^{k-1}}{P_k} \right\rceil C_k \tag{8}$$

Este tiempo no está garantizado que sea una cota más ajustada que ninguna de las propuestas anteriores, de ahí que la nueva propuesta de tiempo de respuesta inicial sea:

$$r_i^0 = \text{máx}(r_i^{-1}, LR_i) \tag{9}$$

Dado que el nuevo r_i^0 cumple que $r_i^{-1} \leq r_i^0 \leq R_i^k$, la propuesta sólo puede mantener o reducir el tiempo necesario para calcular el tiempo de respuesta a costa de un par de operaciones sencillas.

4. Test Basados en Prioridades Dinámicas

Al igual que en el caso de los test de planificabilidad para algoritmos basados en prioridades fijas, en esta sección se van a presentar algunas optimizaciones que reduzcan los cálculos necesarios en un test de planificabilidad de condición suficiente y un test de planificabilidad exacto para un conjunto de tareas planificadas bajo el algoritmo *Earliest Deadline First* (EDF).

Al igual que en el apartado anterior los test de planificabilidad que se presentan se aplican a un único procesador y, por lo tanto, el conjunto de tareas al que se refieren, así como sus subíndices utilizados en los atributos de las tareas hacen referencia a las tareas asignadas a dicho procesador, es decir al conjunto $\mathcal{P}_j^k$, siendo $n = |\mathcal{P}_j^k|$.

4.1. Test basados en la utilización

Al ser el algoritmo EDF un algoritmo óptimo, para un conjunto de tareas con los plazos de entrega igual a los periodos, una condición suficiente y necesaria basada en la utilización del conjunto de tareas podría ser:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{10}$$

Para un sistema con plazos de entrega restringidos ($D_i \leq P_i$) se puede utilizar la condición suficiente basada en la densidad de las tareas:

$$\delta = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1 \tag{11}$$

Al igual que en el caso de las prioridades fijas se puede realizar una pequeña optimización si los resultados del paso de asignación $k-1$ se han almacenado en una estructura asociada a cada procesador. El test quedaría como sigue:

$$\delta^{k-1} + \frac{C_k}{D_k} \leq 1 \tag{12}$$

Aunque este test está claramente limitado como test de planificabilidad, puede ser de gran ayuda en los primeros pasos del algoritmo de asignación en los que los conjuntos $\mathcal{P}_j$ se encuentran prácticamente vacíos.

4.2. Test basados en el cómputo solicitado

Al contrario que en el caso de los test basados en el análisis del tiempo de respuesta, los test de planificabilidad exactos para EDF se basan en analizar el comportamiento de todas las tareas en su conjunto, y no en el peor tiempo de respuesta de cada una. Lamentablemente ello va a reducir las posibilidades de optimización al añadir una única tarea al conjunto, ya que hay que analizar el comportamiento del nuevo conjunto completo. Estos test se agrupan bajo la

denominación de *análisis de los requisitos de procesador solicitados* (PDA)[4].

Los test del tipo PDA se basan en comprobar que la función $H(t)$:

$$H(t) = \sum_{i=1}^{n} \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor \tag{13}$$

cumple la condición abajo expresada para todos los valores entre 0 y un límite L dado:

$$\forall t < L, H(t) \leq t \tag{14}$$

Dado que la función $H(t)$ representa la cantidad de cómputo que se debe haber servido para que ninguna tarea del conjunto pierda un plazo de entrega, la condición 14 índica simplemente que ese tiempo no debe exceder en ningún momento la capacidad del procesador.

Las optimizaciones que se encuentran en la bibliografía han ido fundamentalmente encaminadas a reducir dos parámetros clave del test de planificabilidad: (a) el valor L, límite superior del intervalo en el que debe comprobarse la función $H(t)$ [17, 18, 10], y (b) el número de puntos o instantes de tiempo en los que es necesario comprobar la función $H(t)$ en el intervalo $[0, L]$ [18, 13]. A continuación se muestra como se pueden optimizar los test de planificabilidad en ambos parámetros.

4.2.1. Optimizando el límite de comprobación L

Para el modelo de tareas analizado en este trabajo, el valor de L utilizado [18, 10] es el siguiente:

$$L = \text{mín}(L_a, L_b) \tag{15}$$

[4]Processor Demand Analysis

donde L_a y L_b son límites que no están relacionados, de ahí el uso de la función mín.

El límite L_a se calcula a partir de la utilización de las tareas:

$$L_a = \frac{\sum_{i=1}^{n}(P_i - D_i)U_i}{1 - U} \tag{16}$$

representando una recta que limita la cantidad de computo solicitado en cualquier momento. En el instante L_a dicha recta cruza la recta que indica la capacidad del procesador ($L_a = t$) y se sabe que a partir de ese punto no se superará nunca dicha capacidad.

Por otro lado, el valor L_b indica la duración del *intervalo ocupado síncrono* o *instante crítico inicial* y se calcula con la relación de recurrencia:

$$w^0 = \sum_{i=1}^{n} C_i \tag{17}$$

$$w^{m+1} = \sum_{i=1}^{n} \left\lceil \frac{w^m}{P_i} \right\rceil C_i \tag{18}$$

siendo $L_b = w^{m+1}$ cuando $w^{m+1} = w^m$. Este valor representa el instante en que se debe haber servido todo el tiempo de cómputo solicitado cuando todas se activan simultáneamente.

De forma similar al caso de las prioridades fijas, las optimizaciones que se proponen van encaminadas a reducir el tiempo de computo necesario para calcular los valores de L_a y L_b.

Optimización 1

Si dado un conjunto de tareas $\mathcal{P}_j^{k-1}$ planificable, almacenamos el numerador y el denominador de la fracción L_a ($L_{a'num}^{k-1}$ y $L_{a'den}^{k-1}$), al añadir la tarea T_k al conjunto el valor de L_a podría calcularse como:

$$L_a = \frac{L_{a'num}^{k-1} + (P_k - D_k)U_k}{L_{a'den}^{k-1} - U_k} \quad (19)$$

reduciendo el coste temporal de $O(n)$ a $O(1)$.

Optimización 2

Por otro lado, al añadir una nueva tarea T_k al conjunto $\mathcal{P}_j^{k-1}$ el valor de L_b^k será al menos el valor de L_b^{k-1} más todo el cómputo solicitado por T_k en dicho intervalo. Dicho valor podría utilizarse como valor inicial para la relación de recurrencia, reduciendo así el número de iteraciones de la misma. El cálculo de L_b^k sería idéntico pero utilizando el valor w^0 como se define a continuación:

$$w^0 = L_b^{k-1} + \left\lceil \frac{L_b^{k-1}}{P_k} \right\rceil C_k \quad (20)$$

4.2.2. Optimizando los puntos de comprobación

En [18] los autores presentaron una de las primeras optimizaciones que se hicieron para reducir el número de puntos en los que hay que comprobar la condición $H(t) < t$. En dicho trabajo se delimitan los puntos de comprobación únicamente a los instantes en que vencen los plazos absolutos de las tareas, siendo éstos los únicos instantes en los que la función $H(t)$ puede cambiar. Esto delimita el cálculo de la función $H(t)$ sólo a dichos puntos de interes y no a todos los instantes de tiempo posibles entre 0 y L. El test de planificabilidad quedaría como sigue:

$$\forall t \in P, H(t) \leq t \quad (21)$$

siendo $P = \{d_k | d_k = kP_i + D_i \wedge d_k < L, k \in \mathbb{N}\}$.

Sin embargo, en trabajos recientes se han reducido drásticamente el número de puntos a comprobar mediante el algoritmo *Quick Processor demand Analysis* (QPA) [13] que se muestra en el listado 1.

Listado 1: Test de planificación QPA

```
function test_QPA
   t := d_min(L);
   loop
      s := H(t);
      if s <= Dmin then return Schedulable;
      if s > t then return Unschedulable;
      if s = t then
         t := d_min(s);
      else
         t := s;
      end if;
   end loop;
```

En el pseudo-código del test QPA el valor $Dmin$ es el menor plazo de entrega relativo, es decir, $Dmin = \min_{i=1}^{n} D_i$, y la función d_min($t$) devuelve el máximo plazo de entrega absoluto estrictamente menor que t. Se definiría como sigue:

$$\text{d\_min}(t) = \max_{i=1}^{n} \left\lfloor \frac{t + P_i - D_i - 1}{P_i} \right\rfloor P_i + D_i \quad (22)$$

Este algoritmo reduce de forma exponencial el número de puntos en que se debe comprobar la función $H(t)$.

Optimización 1

En este caso, dado que se analiza el conjunto de tareas completo mediante la función $H(t)$, la única optimización posible debida al uso de incremental del algoritmo es la sustitución de $Dmin$ por D_k. Dado que el conjunto $\mathcal{P}_j^{k-1}$ era

planificable, la función $H(t)$ está garantizado que cumplirá la condición 14 entre los valores $[0, D_k]$ y como $D_k \geq Dmin$, esta modificación puede reducir sensiblemente el número de iteraciones del algoritmo QPA.

Optimización 2

Esta optimización no está relacionada con el uso incremental del test QPA, sino con el uso de la función d_min(t). El uso de está función se introduce para determinar el siguiente punto de comprobación en los casos en que se cumple que $H(t) = t$. Sin embargo, comprobando en su lugar el valor de $H(t-1)$, el cumplimiento de dicha condición ($H(t-1) = t$) pasa a marcar el conjunto de tareas como *no planificable*. Así se podría evitar el uso de la función d_min(t) eliminando del test de planificabilidad el coste de calcularla.

Los autores del presente trabajo utilizaron en [11] una versión previa al algoritmo QPA que utiliza las condiciones descritas, aunque los detalles del mismo no fueron publicados. El algoritmo que denominaremos *Fast Processor-demand Analysis* (FPA) se muestra en el listado 2.

Listado 2: Test de planificación FPA

```
function test_FPA
   t := L;
   loop
      s := H(t−1);
      if s <= Dmin then return Schedulable;
      if s >= t then
         return Unschedulable;
      else
         t := s;
      end if;
   end loop;
```

5. Evaluación de Prestaciones

En esta sección se va a presentar un estudio experimental preliminar de las optimizaciones propuestas. Los experimentos realizados se han limitado al uso del algoritmo *First-Fit* ordenando las tareas por factor de utilización decreciente (FF-DUF). Para los test basados en prioridades estáticas el esquema de asignación de prioridades ha sido el *Deadline Monotonic*, es decir, máxima prioridad a la tarea con el plazo de entrega relativo más corto.

De formas similar a los trabajos [12, 13], y dado que las mejoras propuestas se centran fundamentalmente en reducir el número de iteraciones de las formulas implicadas en los test de planificabilidad, será dicho valor el que se utilice como métrica para ponderar las mejoras. Ya que prácticamente todas las formulas realizan divisiones y/o operaciones de suelo o techo, fusionar las iteraciones realizadas en diferentes formulas no se considera que introduzca una desviación muy notable en la evaluación del coste.

Se han realizado comparaciones entre los test para el algoritmo de prioridades fijas en su versión convencional (RTA) e incremental (iRTA), así como entre los test de planificabilidad para EDF con el algoritmo QPA [13] y la propuesta del algoritmo FPA en su versión incremental (iFPA). Las versiones incrementales de los test incluyen las mejoras, tanto en los test exactos en si mismos, como en los test de condición suficiente que se aplican previamente.

5.1. Generación de la carga

El mecanismo de generación de carga es similar al presentado en [13], utilizando el algoritmo UUniFast para la generación de las utilizaciones de las tareas. Sin embargo, al ser la utilización

total del sistema superior a 1, para generar el conjunto de tareas final, se han generado M conjuntos de tareas con una utilización cada uno del 80 %, siendo M el número de procesadores. El número de tareas de cada subconjunto se ha escogido con un algoritmo similar a UUniFast pero para números naturales, asegurando un mínimo de $N/2M$ tareas por conjunto, siendo N el número de tareas totales. Los M subconjuntos, una vez generados, se han fusionado y ordenado bajo el criterio requerido por el algoritmo FF-DUF. Igualmente, los periodos y plazos de entrega de las tareas se han generado siguiendo los parámetros utilizados en [13], variando los periodos en el rango $[10, 10000]$. Los experimentos se han realizado con 4, 8 y 16 procesadores. Para cada configuración se han generado conjuntos de tareas con un número medio de tareas por procesador ($\frac{N}{M}$) entre 10 y 30.

5.2. Resultados

Los resultados de la comparación experimental realizada se muestran en las figuras 1 y 2. Las gráficas muestran la aceleración obtenida por las nuevas propuestas en relación al número de operaciones necesarias para calcular los test.

Se puede observar que ambas propuestas introducen una mejora en el coste de los test de planificabilidad. Sin embargo, como muestra la figura 1, en los test de planificabilidad de prioridades fijas, la versión incremental mejora con el número de tareas totales y con el número de procesadores. La gráfica muestra como las optimizaciones propuestas reducen notablemente el número de operaciones necesarias entre un factor de 1.5 y 4.

Por otro lado, como era de esperar, las mejoras propuestas para los test de planificabilidad basados en prioridades dinámicas ofrecen una mejora

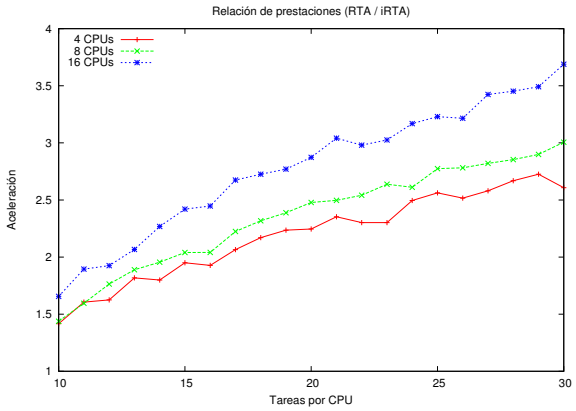

Figura 1: Comparación entre los algoritmos de prioridades fijas

menos contunde, situándose entre un 30 % y un 40 % respecto al algoritmo QPA. En este caso, los resultados mostrados en la figura 2 son independientes del número de tareas del conjunto, pero no así del número de procesadores utilizado, siendo los resultados para 16 procesadores más reducidos que para 4.

6. Conclusiones

El presente trabajo ha presentado un sencillo conjunto de optimizaciones de los test de planificabilidad para algoritmos basados tanto en prioridades fijas como en prioridades dinámicas. Dichas optimizaciones van encaminadas a reducir el número de cálculos a realizar cuando los test se utilizan dentro de un algoritmo de asignación de tareas incremental. Los resultados han mostrado que con unas pequeñas modificaciones en los test de planificabilidad se pueden obtener importantes mejoras en los tiempo de evaluación de dichos test.

Los mismos resultados se pueden extrapolar para cualquier test de planificabilidad que modi-

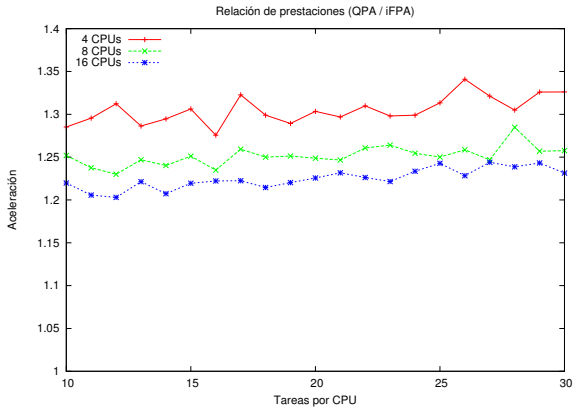

Figura 2: Comparación entre los algoritmos de prioridades dinámicas

fique un conjunto de tareas de manera incremental, como podría ser un algoritmo de aceptación de tareas *on-line*.

Como trabajo futuro se plantea la evaluación más exhaustiva del comportamiento de las propuestas realizadas, utilizando diferentes algoritmos de asignación y test de planificabilidad más completos que tengan en cuenta el uso de recursos compartidos.

English Summary

On Improving Schedulability Tests with Incremental Task Allocation in Real-Time Multiprocessor Systems

Abstract

During the design of a Real-Time Multiprocessor System, schedulability tests are a key component of the task allocation algorithms. Using exact schedulability tests increases the efficiency of these allocation algorithms, but the execution cost to validate a task partition is also greatly increased. Although several improvement to these schedulability test have been recently published, their use in the multiprocessor context is still unaddressed. This work presents several improvements to execution costs of the schedulability test when they are used by task allocation algorithms taking advantage of the incremental nature of this allocation process.

Keywords:

Multiprocessor Systems, Schedulability Analysis, Real-Time Systems

Agradecimientos

Este trabajo se encuentra parcialmente financiado por el proyecto MUSES (PAID/20101002) del Vicerrectorado de Investigación de la Universidad Politécnica de Valencia, el proyecto HI-PartES (TIN2011-28567-C03-03) del Ministerio de Ciencia e Innovación y el proyecto MultiPARTES (FP7-ICT-287702) de la Unión Europea.

Referencias

[1] IEEE, *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) System Interfaces, Issue 6*. 2001.

[2] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 40–61, 1973.

[3] J.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[4] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," in *Handbook on Scheduling Algorithms, Methods, and Models*, Chapman Hall/CRC, Boca, 2004.

[5] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, pp. 35:1–35:44, Oct. 2011.

[6] J. F. Ruiz, "Towards a ravenscar extension for multi-processor systems," *Ada Lett.*, vol. 30, pp. 86–90, May 2010.

[7] S. ISO/IEC JTC 1, *Ada 2012 Reference Manual: Language and Standard Libraries*, 2012. ISO/IEC 8652:201z(E) (Submission Draft [Draft 18]).

[8] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "New strategies for assigning realtime tasks to multiprocessor systems," *IEEE Transactions On Computers*, vol. 44, no. 12, pp. 1429–1442, 1995.

[9] M. Joseph and P. K. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, 1986.

[10] I. Ripoll, A. Crespo, and A. K. Mok, "Improvement in feasibility testing for real-time tasks," *Real-Time Systems*, vol. 11, no. 1, pp. 19–39, 1996.

[11] S. Sáez, J. Vila, and A. Crespo, "Using exact feasibility tests for allocating real-time tasks in multiprocessor systems," in *Proceedings of 10th Euromicro Workshop on Real-Time Systems*, pp. 53 –60, June 1998.

[12] R. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *Computers, IEEE Transactions on*, vol. 57, no. 9, pp. 1261 –1276, 2008.

[13] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *Computers, IEEE Transactions on*, vol. 58, no. 9, pp. 1250 –1258, 2009.

[14] K. W. Tindell, A. Burns, and A. J. Wellings, "Allocating hard real-time tasks: an NP-hard problem made easy," *Real-Time Systems*, vol. 4, pp. 145–165, May 1992.

[15] N. Audsley, *Flexible Scheduling of Hard Real-Time Systems*. PhD thesis, Dept. of Computer Science, Univ. of York, 1993.

[16] M. Sjödin and H. Hansson, "Improved response-time analysis calculations," in *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '98, (Washington, DC, USA), pp. 399–, IEEE Computer Society, 1998.

[17] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *In Proceedings of the 11th Real-Time Systems Symposium*, pp. 182–190, IEEE Computer Society Press, 1990.

[18] S. K. Baruah, R. R. Howell, and L. E. Rosier, "Feasibility problems for recurring tasks on one processor," *Theoretical Computer Science*, vol. 118, pp. 3–20, September 1993.