ASIGNACIÓN DE TAREAS EN UN SISTEMA DISTRIBUIDO DE TIEMPO REAL DURO

Edgardo Ferro, Javier Orozco, Ricardo Cayssials
Dep. Ingeniería Eléctrica, Instituto de Ciencias e Ingeniería de Computación, CONICET.
Universidad Nacional del Sur, Av. Alem 1253, 8000 Bahía Blanca.
E-mail: ieferro@criba.edu.ar

Resumen

Se presenta un método que trata el problema de asignación de un conjunto de tareas apropiables, sobre un conjunto de procesadores heterogéneos distribuidos que deben trabajar en un entorno de tiempo real duro. Las tareas son cooperativas y utilizan como vía de comunicación una red. Sobre el sistema tareas-procesadores-red, existen restricciones de ubicación, diagramabilidad, memoria, comunicaciones y precedencia.

1. Introducción

En un Sistemas de Tiempo Real (STR) los resultados producidos deben ser siempre correctos desde los puntos de vista lógico y temporal. Alcanzar un resultado correcto temporalmente, se logra cumpliendo con las especificaciones temporales impuestas por el entorno de tiempo real. Sistemas de Control de Procesos, Fabricación Automática en Plantas mediante Robots, Multimedia y Telecomunicaciones son ejemplos concretos de STR. Las características básicas de un Sistema Multiprocesador es la existencia de varios procesadores acoplados por un canal de comunicaciones. Estos sistemas se emplean en tiempo real cuando los resultados esperados no pueden ser obtenidos por un solo procesador, o bien cuando se desea mejorar la tolerancia a las fallas.

El problema a resolver contempla un conjunto de m tareas de tiempo real apropiativas que deben ser ejecutadas en n procesadores, cada uno de los cuales posee una cantidad determinada de memoria. Si no hubiesen condicionamientos para la asignación, ésta podría realizarse de n^m formas diferentes. El objetivo es asignar m tareas a n procesadores con la intención de obtener una asignación, que cumpla con diferentes restricciones en un contexto de tiempo real duro. De las n^m asignaciones, es posible que sólo un subconjunto de ellas cumpla con las condiciones impuestas por el sistema. Es evidente que el problema puede no ser computable en tiempo razonable para valores de n^m considerables. Pueden utilizarse distintos mecanismos para seleccionar el par tarea-procesador que debe asignarse en un determinado momento, intentando minimizar el número de computaciones requerido para obtener una solución o para identificar la no resolución del problema. Investigaciones recientes cubren los siguientes tópicos: asignación de tareas en un sistema distribuido utilizando técnicas de Branch and Bound (Ma 1982), asignación y diagramación de tareas periódicas complejas (Ramamritham 1990), asignación y

diagramación de un conjunto de tareas no apropiables con constricciones de vencimiento, precedencia y exclusión sobre un sistema de procesadores idénticos (Xu 1993), asignación de tareas de tiempo real duro utilizando optimización por Recocido Simulado (Tindel 1992, Borriello 1994), asignación de tareas utilizando métodos heurísticos sintonizados (Ferro 1996) y asignación utilizando lógica difusa (Orozco 1996). Por otro lado el problema de partición de grafos (PPG) es una optimización combinatoria prácticamente intratable que tiene múltiples aplicaciones (diseño de VLSI, ruteo de información, etc.). A partir de un grafo con pesos en los arcos y nodos, el objetivo del PPG es encontrar una partición de los nodos del grafo, en k-subconjuntos disjuntos, tal que la suma de lo pesos de los arcos que no están en la misma partición es minimizada. El problema de partición de grafos es NP-completo (Garey and Johnson 1979). Existen diferentes enfoques para este problema: métodos heurísticos (Fiduccia and Mattheyses 1982, Dunlop 1985, Sarje and Sagar 1991), métodos basados en recocido simulado (Van den Bout 1990, Tindell 1991, Tao 1993) y métodos que utilizan algoritmos genéticos (Jones and Beltramo 1991, Laszewski 1991). La mayoría de estas técnicas intenta encontrar una solución inicial, y luego se buscan soluciones vecinas a la actual para obtener nuevas soluciones. La performance de estos métodos se ve degradada para valores no unitarios de los pesos. Las técnicas basadas en recocido simulado son superiores, pero consumen gran cantidad de tiempo para obtener una solución. Por otro lado la combinación del recocido con redes neuronales no trabaja con grafos pesados.
Los Algoritmos Genéticos (AGs) son métodos de búsqueda y optimización que imitan el proceso de selección natural, manipulando una población de individuos que contienen valores posibles de las variables de un problema específico (Goldberg 1989). Estos algoritmos de búsqueda están basados en la selección y genética natural, combinando el concepto de supervivencia del más apto, con un intercambio estructurado pero aleatorio de información. Al estar basados los AGs en la genética, la terminología utilizada se corresponde en forma directa con el vocabulario genético. Luego cada generacion esta compuesta por cromosomas, los cuales pueden ser implementados como vectores formados por genes.
Los expresiones anteriores involucran la presentación de las características de los mejores exponentes de una generación en la generación siguiente, introduciendo sobre la población cambios aleatorios utilizando los operadores genéticos de cruce y mutación. Antes de poder evaluar a un individuo es necesario decodificar la información que contiene, para utilizarla como entrada a la función de evaluación. Si utilizamos un enfoque basado en AGs que incorpore el espacio del problema a tratar, estaremos sacando provecho del paralelismo inherente aportado por el AG y del problema específico a resolver. Utilizando esta técnica (Ahmad 1995) la construcción del cromosoma estará basada en los datos del problema. Luego los operadores genéticos pueden ser aplicados directamente sobre el espacio del problema, sin existir necesidad de alterarlos al cambiar el contexto del mismo. La solución es obtenida utilizando una heurística simple y rápida que permite mapear del espacio del problema al

espacio de soluciones. En este trabajo se propone un método basado en la utilización de algoritmos genéticos para particionar grafos. Se intenta obtener mediante esta técnica, una asignación de tareas que cumpla con las constricciones de tiempo real.

2. Modelo

El modelo de tiempo real propuesto, contempla que cada tarea ♦ es apropiable y debe ser enteramente ejecutada en un procesador. Las tareas son periódicas, con un requerimiento de memoria específico y un tiempo de vencimiento y de ejecución predeterminado. Las tareas son parte de un *trabajo* ϑ, que es representado por un grafo dirigido y acíclico. En el grafo los nodos representan a las tareas, el valor asociado a los nodos indica el tiempo de ejecución de la tareas y el peso del arco que las une, indica la cantidad de información medida en bytes que intercambian las tareas.

Una tarea τ_i es denominada terminal, si y sólo si no existen arcos salientes de τ_i. Así mismo una tarea, τ_j es denominada fuente, si y sólo si no existen arcos entrantes a τ_j. Por otro lado τ_i y τ_j son predecesor y sucesor respectivamente ($\tau_i \prec \tau_j$) si y sólo si existe un arco desde τ_i a τ_j y no existe $\tau_k \mid \tau_i \prec \tau_k \prec \tau_j$.

A partir de la definición de las componentes del grafo podemos formalizar el modelo y definir la estrategia de PPG a adoptar. Se define al multigrafo $\vartheta = \{\Gamma, B, \Phi_1, \Phi_2\}$, donde $\Gamma = \{\tau_1, \tau_2, \ldots\ldots \tau_m\}$ es el conjunto de nodos, $B \subseteq \Gamma \times \Gamma$ es el conjunto de arcos, $\Phi_1 : B \rightarrow Z^+$ define los pesos de los arcos, $\Phi_2 : \Gamma \rightarrow Z^+$ define el peso de los nodos y Z^+ es el conjunto de los enteros positivos.

El problema de particionar en n-subconjuntos se reduce a encontrar una función de mapeo:

$$\Pi : \Gamma \rightarrow \{\pi_1, \pi_2, \ldots\ldots, \pi_n\} \text{ donde } \pi_i = \{\tau \in \Gamma \mid \Pi(\tau) = \pi_i\} \ \forall i \mid 1 \le i \le n \qquad (1)$$

tal que

$$\pi_i \cap \pi_j = \varnothing \text{ para } i \neq j \text{ y } \bigcup_{i=1}^{k} \pi_i = \Gamma \ \forall i \qquad (2)$$

$R(\pi_i)$ indica el tamaño del subconjunto π_i y se define como $\Sigma\Phi_2(\tau)\ \forall\tau\in\pi_i$. Además $X_{i,j}$ indica el peso originado al particionar entre dos subconjuntos π_i y π_j de Π y se define como $\blacklozenge\Phi_1(u,v)\ \forall(u,v)\in B \mid u\in\pi_i$ y $v\in\pi_j$. El objetivo es minimizar los pesos de Π:

$$Y(\Pi) = \sum X_{i,j} \ \ \forall i, j \mid 1 \le i \le j \le n \qquad (3)$$

tal que el desbalance W(Π) entre el tamaño de cada π_i sea mínimo.

$$W(\Pi) = \sum \left| R(\pi_i) - R(\pi_j) \right| \ \ \forall i, j \mid 1 \le i \le j \le n \qquad (4)$$

3. Restricciones

3.1 Restricciones temporales

En ambientes de tiempo real son comunes los sistemas en los que un conjunto de tareas comparten un único recurso. Debido a restricciones temporales, el uso del mismo debe ser garantizado a cada tarea

dentro de un cierto intervalo, contado a partir del instante en que se genera el requerimiento. Este intervalo es denominado *plazo de vencimiento* de la tarea. El tiempo se considera ranurado. La duración de una ranura se denomina *tiempo de ranura* y se simboliza T_r. Se supone además, que los requerimientos de las tareas se producen sincrónicamente con el comienzo de una ranura y que cada tarea los produce en forma periódica.

El conjunto de tareas Γ puede ser caracterizado por $S(m)=\{C_1, T_1, D_1, C_2, T_2,, D_2..., C_m, T_m, D_m\}$ donde C_i, T_i, D_i denotan el tiempo de ejecución, el mínimo periodo de generación y el vencimiento de la tarea τ_i respectivamente. El recurso permanece ocioso sólo cuando no existe requerimiento. Una ranura donde el recurso no es utilizado se denomina libre. Para asignar el recurso se establece una pila de prioridades, cuando la misma está organizada por períodos monotónicos crecientes se denomina PMC. En Liu and Layland (1973) se ha demostrado que la peor condición de carga del sistema, ocurre cuando todas los tareas producen requerimientos simultáneos. Santos *et al* (1993) han demostrado formalmente que un sistema S(m) es PMC diagramable si y sólo si para i=2,3,.....m:

$$\sum_{h=1}^{m} \frac{C_h}{T_h} \leq 1 \tag{5}$$

y

$$D_i \geq e_{C_i(i-1)} = \text{least } t \mid t = C_i + \sum_{h=1}^{m} C_h \left\lceil \frac{t}{T_h} \right\rceil \tag{6}$$

Cuando se comparten datos y/o recursos las tareas pueden sufrir inversiones de prioridad y ser bloqueadas por tareas de menor prioridad. El peor caso de bloqueo puede ser incorporado a la expresión de las ranuras vacías (Ferro 1996) agregando el bloqueo más largo K_i que puede recibir una tarea. Luego la expresión de diagramabilidad para i=1,2,3,.....m será:

$$D_i \geq e_{(C_i+K_i)(i-1)} = \text{least } t \mid t = K_i + C_i + \sum_{h=1}^{m} C_h \left\lceil \frac{t}{T_h} \right\rceil \tag{7}$$

3.2 Restricciones de ubicación

Algunas tareas deben ser ejecutadas en un procesador determinado, por lo que reciben el nombre de preasignadas. Estas deben ser ejecutas en un procesador particular, pues necesitan contar con recursos específicos. Estos pueden ser sensores, dispositivos de entrada/salida, bases de datos etc. Cuando es necesario definir réplicas de tareas para implementar tolerancia a fallas, la tarea original y la replica no podrán coexistir en un mismo procesador. Las tareas que no tienen procesador definido se las llama libres y pueden ser alojadas en cualquier procesador.

3.3 Restricciones de memoria

Cada tarea requiere de una determinada cantidad de memoria durante su ejecución. La misma es especificada como la suma de su código de ejecución y la memoria dinámica que va a necesitar durante la corrida. Antes de asignar una tarea a un procesador, es necesario verificar que el mismo disponga de una cantidad suficiente para recibirla. Luego debe cumplirse que:

$$M_h \leq M_n \tag{8}$$

donde M_h es la memoria requerida por la tarea τ_h y M_n es la memoria disponible en el procesador n.

3.4 Restricciones de comunicación

Las tareas pueden transferir información entre sí. Si las mismas se encuentran en distintos procesadores, una red de interconexión es necesaria. Generalmente esta función es realizada por una red local la cual debe operar con las mismas restricciones que el sistema de tiempo real. La red se supone que implementa una mecanismo de acceso que utiliza una disciplina de rueda cíclica. Esta es diagramable si cumple con:

$$\sum_{i=1}^{m} B_i \leq AB * D_{min} \tag{9}$$

donde B_i es la cantidad total de bytes que la tarea-i envía a través de la red a cada una de sus sucesoras, AB el ancho de banda útil y D_{min} el mínimo plazo de vencimiento del conjunto de tareas. El peor retardo de comunicaciones que puede recibir una tarea antes de enviar los datos a su sucesora, estará dado por:

$$\Delta = \left\lceil \frac{\sum_{i=1}^{m} B_i}{AB} \right\rceil \tag{10}$$

esto es deducido, teniendo en cuenta que una tarea para poder transmitir, debe esperar que todas las demás lo hagan.

3.5 Restricciones de Precedencia

Para un número importante de sistemas, es imprescindible que las tareas colaboren entre sí para alcanzar un objetivo predeterminado. Al definir una relación de precedencia se incorpora complejidad tanto para la asignación como para la diagramación. Los mejores resultados fueron obtenidos para subclases de la relación de precedencia, desarrollándose algoritmos polinómicos para relaciones con un solo predecesor y un solo sucesor. En nuestro enfoque, cuando una tarea τ_p debe enviar datos a otra que reside en un procesador distinto, el vencimiento de la tarea predecesora D_p debe ser corregido teniendo en cuenta el retardo de comunicaciones:

$$D_p^* = D_p - \Delta \tag{11}$$

La iniciación de la sucesora es manejada por eventos y es activada cuando el último byte transferido llega al nodo.

4. Algoritmo

El propósito del algoritmo es particionar un conjunto dado de tareas en un número determinado de procesadores. La técnica de agrupar, intenta solucionar el problema de distribuir *m* objetos en *n* grupos mediante la utilización de algún criterio de optimización. A pesar de no brindar la solución óptima al problema este algoritmo obtiene una solución casi óptima en un tiempo reducido, inclusive para problemas muy complejos donde la perfomance de los métodos tradicionales comienza a decaer.

Los pasos que comprende el Algoritmo son:

1. **Inicialización:** Se inicializa a: m^{ϑ}_{g} con el número de nodos del grafo ϑ, N_p con el tamaño deseado de la población inicial, N_g con el número de generaciones a evolucionar, P_c con la probabilidad de cruce, P_m con la probabilidad de mutación, n con el número de subconjuntos disjuntos (coincide con el número de procesadores del sistema), y α_1 y α_2 con los valores deseados para la influencia de cada peso dentro de la ecuación de costo.
2. **Generar la población inicial:** El primer cromosoma de la población inicial es construido a partir del grafo de cada trabajo, mientras que los restantes son obtenidos perturbando el cromosoma inicial. Cada cromosoma tiene asociado un arreglo de números reales que representan la *dominancia* de cada nodo del grafo que representarían los genes. La dominancia O de cada nodo i para el primer cromosoma se calcula como:

$$O_i^1 = \Phi_2\left(\tau_i\right) * \mathrm{INT}\left(random(0,1) * m_g^{\vartheta}\right) \tag{12}$$

   Los j-esimos cromosomas restantes se obtienen perturbando aleatoriamente la ecuación anterior:

$$O_i^j = O_i^1 + f(-\eta, \mu)\ \forall i \mid 1 \le i \le m_g^{\vartheta},\ \forall j \mid 2 \le j \le N_p \tag{13}$$

   donde $f(\eta,\mu)$ es un número generado con una distribución de probabilidad uniforme entre $-\eta$ y μ.
3. **Particionar:** Una vez obtenidos los N_p cromosomas y ordenados sus genes por valores decrecientes de dominancia, se ejecuta el algoritmo para particionar en n subconjuntos. El mismo selecciona el gen con el mayor valor de dominancia y lo asigna a el subconjunto que tenga el menor tamaño. El tamaño de cada subconjunto es la suma de los pesos de todos los nodos mapeados a ese subconjunto. La solución elaborada por el algoritmo para cada cromosoma calcula los valores resultantes de las ecuaciones (3) y (4) para ser aplicados en el paso siguiente.

4. **Calcular el Costo y la Aptitud:** La función costo es la llave que refleja el éxito de la optimización. Seleccionaremos la suma pesada de las ecuaciones (3) y (4) para calcular el costo de cada cromosoma *i*:

$$Costo(i) = \alpha_1 * \Upsilon(\Pi) + \alpha_2 * W(\Pi) \tag{14}$$

donde α_1 y α_2 son los pesos que controlan la función objetivo. Si el problema es duro en comunicaciones lo que se intentará controlar es α_1 para reducir indirectamente Δ y poder cumplir con las condiciones de precedencia y de diagramación. Si por el contrario el factor de utilización total de las tareas es alto se deberá controlar α_2. El valor de este costo fue utilizado para determinar la aptitud de cada cromosoma *i* en la siguiente forma:

$$A(i) = \frac{\left(CostoMaximo - Costo(i)\right)^{\theta}}{\sum_{k=1}^{N_p}\left(CostoMaximo - Costo(k)\right)^{\theta}} \tag{15}$$

donde *CostoMaximo* es el mayor costo de un cromosoma en la población y θ (cuyo rango va entre 1 y 5) es un parámetro usado para escalar la aptitud y balancear la convergencia..

5. **Operadores de Selección, Cruce y Mutación:** Mediante este operador se obtiene la nueva generación formada por una selección de cromosomas de la generación previa. Los cromosomas de mayor aptitud tendrán una probabilidad mayor de ser seleccionados, existiendo la posibilidad de contribuir con más de un descendiente en la nueva generación. Para representar la probabilidad de selección se utiliza la de una ruleta, en la cual cada cromosoma tiene asignada una cantidad de casilleros proporcional a su aptitud (Goldberg 1989). Una vez creada la nueva generación, se elige un par de cromosomas en forma aleatoria. Estos cromosomas son partidos en una posición *p* elegida al azar y se intercambian los primeros *p* genes de un cromosoma con los primeros *p* genes del otro, con una probabilidad P_c. La mutación es un operador secundario que cumple la función de evitar la perdida de información que puede darse por los dos operadores anteriores. En este proceso se recorren todos los genes uno a uno y se los muta con una probabilidad P_m, que generalmente es mucho menor que la probabilidad de cruce.

   Los pasos 3 al 5 son repetidos para un número predefinido de iteraciones de acuerdo a N_g.

6. **Obtener Solución:** La mejor solución para el grafo $\mathcal{G}$ es almacenada y se repiten los pasos 2 al 6 para cada uno de los grafos restantes.

7. **Evaluación:** Una vez obtenida la solución para cada grafo, se realiza el análisis de todas las particiones comunes encontradas. Sobre las *n* particiones resultantes se aplican las pruebas que verifican las restricciones. Si estos son satisfechos en su totalidad para cada

partición, se mapean las mismas a los procesadores correspondientes y una solución a sido alcanzada.

5. Ejemplo

Las tabla 2 y 3 del Apéndice muestran un Sistema de 8 procesadores y 43 tareas periódicas de tiempo real (ejemplo presentado en Tindell 1992) usado para evaluar el algoritmo presentado. La ancho de banda de la red seleccionado para el ejemplo fue de 4 Mbit/seg . La Tabla 2 muestra las características particulares de cada tarea, mientras que la Tabla 3 lista la capacidad de memoria de cada procesador. La simbolización 50/1 y 150/2 de la primera línea significa que la tarea τ_0 debe enviar 50 bytes a la tarea τ_1 y 150 bytes a la tarea τ_2. Los siguientes pares de tareas denominadas original y replica no pueden ser ejecutadas en el mismo procesador: (33,38), (34,39), (35,40), (36,41) y (37,42). La figura 1 muestra los 11 trabajos que deberán ser asignados cumpliendo las constricciones. El algoritmo fue programado en C y ejecutado en una Alpha AXP 3000. La primer solución obtenida detallada en la tabla 1 fue encontrada en un tiempo menor a 3 segundos y con los siguientes parámetros: n=8, pc=0.5, pm=0.001, λ=10, ω_1=1, ω_2=1, $\square$=3, N_p=150, N_g=100, $\eta = \mu = \max_i \{P_i^1\} / \lambda$

Procesador	Tareas	Factor de utilización	Memoria
P0	14;34;35;37;9;0;2	0.771429	970
P1	28;39;7;8;11;18;19;1;3	0.557143	990
P2	12;13;17;33;40;42;6	0.878572	760
P3	15;16;20;21;27;38;5	0.788095	1140
P4	4	0.033333	300
P5	22	0.071429	100
P6	25;26;29;31;36	0.528571	1130
P7	23;24;30;32;41;10	0.857143	810

Tabla 1

6. Conclusiones

El algoritmo utilizado es una primera aproximación al problema de resolver asignaciones en un Sistema Distribuido de Tiempo Real Duro, sobre el cual el número de investigaciones aún es muy escaso. Este método es sumamente ventajoso sobre el RS y sobre los AGs tradicionales. Lamentablemente en la solución obtenida por Tindell utilizando RS no se especifica el tiempo empleado en alcanzar la primer solución.

Luego es imposible poder comparar los rendimientos de ambos métodos, pero es conocido que una de las deficiencias del RS es la excesiva cantidad de tiempo utilizada para encontrar la solución. El algoritmo presentado pretende compararse con los distintos métodos utilizados para asignar en un escenario exclusivo de tiempo real. Actualmente se encuentra en fase de desarrollo una comparación con el método presentado por Rammarithan (100 ejemplos aleatorios con tres procesadores y 24 tareas), donde la técnica de diagramación utilizada es cíclica y no PMC como la presentada en este trabajo.

Los trabajos futuros contempla realizar un tratamiento conjunto entre asignación y diagramabilidad (Cayssials 1996) e incorporar lógica difusa (Orozco 1996) al algoritmo presentado, para poder determinar a priori cual parámetro del problema es crítico (utilización, memoria, comunicaciones) y poder inicializar con la mayor exactitud posible los pesos de la función costo.

7. Bibliografía

1. Ahmad, I. and Dhodhi, M. K. (1995). "On the m-Way Graph Partitioning Problem". *Computer Journal*, pp. 237-244.
2. Borriello, G. and D. Miles. (1994). "Task scheduling for real-time multiprocessor simulations". *11th Workshop on RTOSS*, pp. 70-73.
3. Cayssials, R., Orozco, J., Santos J. y Ferro, E., "Diagramación dinámica de tareas de Tiempo Real Duro con conscriciones de Precedencia en Sistemas Distribuidos Multitarea-Multiprocesador". *Actas del XXII CLEI*, pp.1101-1112.
4. Dunlop, A and Kernighan, B. (1985). "A procedure for placement of standard-cell VLSI circuit". *IEEE Trans. Computer-Aided Design of Integrated Circuits & Systems*, CAD-4, 92-98.
5. Ferro, E., Santos J., Orozco, J. and Cayssials, R. (1996). "Tuning Parameters to improve a Heuristic Method: More and Better Solutions to an NP-Hard Real Time Problem". *8th IEEE Euromicro Workshop on Real Time Systems*, pp. 47-51.
6. Ferro, E., Orozco, J., Santos J. y Cayssials, R. (1996). "Sincronización de tareas en Tiempo Real Duro utilizando el método de las Ranuras Vacias". *Información Tecnológica*. Vol 7, Nro 4, pp. 101-107.
7. Fiduccia, C. and Mattheyses, R. (1982). "A linear time heuristic for improving network partitions". *In Proc. IEEE/ACM 17th Design Automation Conf.*, pp. 175-181.
8. Garey, M. and Johnson, D. (1979). "Computers and Intractability: A guide to the Theory of NP Completeness". W.H. Freeman, San Francisco, CA.
9. Goldberg, D. (1989). "Genetic Algorithms in Search, Optimization and Machine Learning". Addison-Wesley, Reading, MA.
10. Jones, D. and Beltramo, M. (1991). "Solving partitioning problem with genetic algorithm". *In Proc. Int. Conf. on Genetic Algorithms*, pp. 442-449.
11. Laszewsky, G. (1991). "Intelligent structural operators for the K-way graph partitioning problem". *In Proc. Int. Conf. on Genetic Algorithms*, pp. 45-52.
12. Liu, C. and Layland, J. (1973). "Scheduling algorithms for multiprograming in Hard Real Time enviroments". *J. ACM*, Vol 20, Nro 1, pp.46-61.
13. Ma, P.-Yi R., Lee, E and Tsuchiya, M. (1982). "A task allocation model for distributed computing systems". *IEEE TC*, 31(1), pp. 41-47.
14. Orozco, J., Cayssials, R., Santos J. and Ferro, E. (1996). "Design of a Learning Fuzzy Production System to Solve an NP-Hard Real Time Assignment Problem". *8th IEEE Euromicro Workshop on Real Time Systems*, pp. 146-150.
15. Ramamritham, K. (1990). "Allocation and scheduling of complex periodic tasks". *Proc. 10th International Conference on Distributed Computing Systems*, pp. 108-115.

16. Santos, J. and Orozco, J. (1993). "Rate Monotonic Scheduling in Hard Real Time Systems". *Information Processing Letters*, Nro. 48, pp. 39-45.

17. Sarje, A. and Sagar, G. (1991). "Heuristic model for task allocation in Distributed Computer Systems". *Proc. IEEE*, 138, pp. 313-318.

18. Tao, L. and Zhao, Y. (1993). "Effective heuristic algorithms for VLSI circuits partition". *IEEE Proc.-G*, 140, 127-134.

19. Tindell, K., Burns, A. and Wellings, A. (1992). "Allocating Hard Real-Time tasks: An NP-Hard problem made easy". *Real-Time Systems*, 4, 2, pp. 145-165.

20. Van den Bout, D. and Miller, T. (1990). "Graph partitioning using annealed neural network". *IEEE Trans. Neural Network*, 1.

21. Xu, J. (1993). "Multiprocessor scheduling of processes with release times, deadlines, precedence and exclusion relations". *IEEE TSE*, 19,2, pp. 139-154.

8. Apéndice

Tarea	Período	Ejecución	Memoria	Mensajes	Ubicación
0	60	4	300	50/1; 150/2	0
1	60	4	150	60/3; 70/4; 30/5	
2	60	2	120	20/3	
3	60	2	170		1
4	60	2	300	60/6	
5	60	4	300	80/6	
6	60	6	110		2
7	35	2	50	40/8	1
8	35	2	70		1
9	35	8	90	90/11	0
10	35	14	220	250/11	
11	35	4	100		1
12	14	2	100	150/13; 150/14	2
13	14	2	150	50/15	
14	14	2	160	50/15	
15	14	2	130		3
16	14	2	110	50/17	3
17	14	2	100		2
18	35	1	100	50/19	1
19	35	1	160		1
20	14	1	190	40/21	
21	14	2	200		3
22	14	1	100	40/23	
23	14	1	200	40/24	
24	14	1	100	20/25	
25	14	1	200	20/26	
26	14	2	700	20/27; 20/28	
27	14	1	110	50/29	
28	14	1	90	30/29	
29	14	1	50		6
30	14	1	60	50/31	7
31	14	2	80	70/32	
32	14	2	130		7
33	20	3	100	50/35	2; 3
34	20	2	100	50/35	0; 1
35	20	2	100	60/36; 60/37	
36	20	2	100		6; 7
37	20	2	100		
38	20	3	100	50/40	2; 3
39	20	2	100	50/40	0; 1
40	20	2	100	60/41; 60/42	
41	20	2	100		6; 7
42	20	2	100		

Tabla 2. Período, Tiempo de ejecución, requerimientos de memoria, longitud del mensaje/tarea destino y ubicación inicial de cada tarea.

Procesador	Memoria
0	1000
1	1000
2	1200
3	1200
4	7,00
5	7,00
6	1200
7	1000

Tabla 3. Capacidad de memoria de cada procesador.

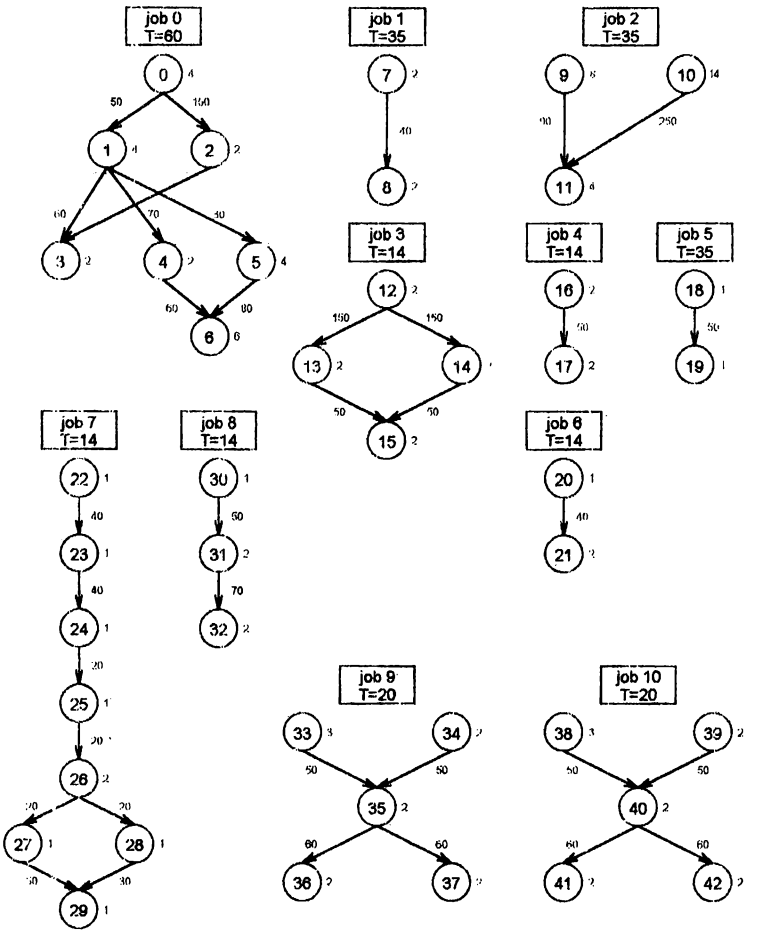

Figura 1 - Definición de los 11 trabajos.