UNIVERSIDAD NACIONAL DEL SUR
DTO. INGENIERÍA ELÉCTRICA
LABORATORIO DE SISTEMAS DIGITALES

ASIGNACION Y DIAGRAMABILIDAD DE SISTEMAS DISTRIBUIDOS DE TIEMPO REAL DURO

EDGARDO FERRO

Tesis presentada en cumplimiento parcial
de los requerimientos para el Doctorado en Ingeniería

INDICE

Prefacio

Los sistemas de computación de tiempo real son componentes críticos de la infraestructura tecnológica de cualquier país industrializado. Son de aplicación imprescindible en robótica, aviónica, manufactura automatizada, sistemas de control, etc. Constituyen, como consecuencia, una de las áreas de más rápido crecimiento dentro de la disciplina. Existen diferentes componentes (sistemas operativos, lenguajes de programación, arquitecturas) que caracterizan a dichos sistemas. Pero la diagramabilidad, entendida como el cumplimiento de todas las restricciones temporales del sistema y la asignación de un conjunto de tareas a un conjunto de procesadores cumpliendo cada una de ellas con las restricciones de recursos, de asignación, de comunicaciones y de precedencia aparece entre las más importantes.

Cada día más y mejores sistemas de tiempo real, dependen de sistemas basados en multiprocesadores. Desdichadamente se conoce poco acerca de la asignación y la diagramación basada en múltiples procesadores. Esto se debe por un lado a que la complejidad de los resultados muestra que la mayoría de estos sistemas son NP-duros y por otro a que la experiencia adquirida en el manejo de estos sistemas no es mucha. Como consecuencia hay un número reducido de heurísticas que emergen como una de las pocas herramientas disponibles para resolver el problema. Este consiste en asignar un conjunto de tareas de tiempo real duro, apropiables, sobre un conjunto de procesadores heterogéneos, cumpliendo restricciones de tiempo, ubicación, memoria, comunicaciones y precedencia.

Esta tesis tiene como base trabajos publicados en revistas y actas de congresos de arbitraje riguroso. El primero de ellos es publicado en el Real Time Systems Journal y se denomina “A Heuristic Approach to the Multitask-Multiprocessor Assignment Problem Using The Empty-Slots method and Rate Monotonic Scheduling”, el cual usando como premisa esencial todas la restricciones antes enumeradas, describe y compara con otros métodos la heurística de asignación implementada. En esta heurística fue utilizado por razones de velocidad y de mantenimiento en memoria de información útil, el cálculo de las ranuras libres disponibles en un determinado intervalo de tiempo. Estos resultados fueron publicados en el Journal Información Tecnológica bajo el titulo de “Sincronización de tareas en Tiempo Real Duro utilizando el método de las Ranuras Vacías”.

Sobre la base de estos resultados preliminares obtenidos, se refinaron los métodos heurísticos incorporando parámetros sintonizables que guían el proceso de asignación para obtener distintos tipos de soluciones (e.g. con red de comunicación interprocesadores más descargada). El diseño de parámetros que relacionen las restricciones de comunicaciones, tiempo y recursos que vayan sintonizándose durante el proceso heurístico de asignación permiten obtener soluciones orientadas. En el método sintonizado las asignaciones de tareas a procesadores se hacen desde una pila de tareas a una pila de procesadores, cada una ordenada de acuerdo a un cierto criterio. En lugar de usar criterios únicos, se pueden constituir varias pilas de acuerdo a distintos criterios. A partir de las mismas se forman polinomios cuyos coeficientes son los parámetros diseñados y cuyas variables son los ordinales asociados a la posición en cada una de las pilas. El valor de esos polinomios permite construir pilas de tareas

y de procesadores, con criterios combinados que posibilitan optimizar el proceso de asignación. Está mejora a la heurística base fue publicada en Proc 8th IEEE Euromicro Workshop on Real Time Systems: "Tuning Parameters to improve a Heuristic Method, More and Better Solutions to an NP-Hard Real Time Problem". Principalmente se confrontan los resultados obtenidos con los alcanzados por autores que utilizan Recocido Simulado (Simulated Annealing) o bien otro tipo de heurísticas para obtener una solución subóptima al problema de asignación.

Como soporte a los algoritmos se diseñó y construyó un generador aleatorio de problemas. El mismo permitió probar y comparar los métodos desarrollados. El generador produce aleatoriamente grafos acíclicos, orientados y con propiedad de clausura transitiva, que representan subconjuntos de tareas con relaciones de precedencia. Obtenido un grafo base, es posible generar aleatoriamente (dentro de cotas preestablecidas) períodos, tiempos de ejecución, requerimientos de recursos y longitud de mensajes intercambiados entre las distintas tareas que componen el grafo. La instanciación de diversos grafos, cada uno de los cuales se usa como base para la generación de problemas con distintos períodos, factores de utilización, memoria, etc., conduce al planteo de problemas en el orden de miles para poder validar conclusiones estadísticas experimentales. Con los datos obtenidos experimentalmente se construyeron curvas que caracterizan la relación de éxito obtenida por cada método para la resolución del problema atacado, utilizando al efecto los problemas generados mediante el dispositivo mencionado anteriormente.

La comparación con otros métodos se ve dificultada por el hecho de que no hay una norma de ensayo y cada autor elige la especificación del problema al cual aplica su

método. Sin embargo pequeñas variaciones en la especificación pueden producir grandes variaciones en el resultado.

Debe hacerse notar que el grupo de investigación en STR nucleado en el laboratorio de Sistemas Digitales de la Universidad Nacional del Sur, es relativamente reducido y de mucha interacción entre sus integrantes. Debido a esto último, los trabajos tienen, la mayoría de las veces, varios autores. Como regla, sin embargo, cada trabajo puede ser usado como apoyo de sólo una tesis y corresponde a alguno de los dos primeros autores.

La tesis esta presentada de la siguiente forma: un capítulo inicial en el que se realiza una introducción a los sistemas distribuidos de tiempo real, planteando el problema de asignación de un conjunto de tareas a procesadores distribuidos en un entorno de tiempo real duro y las técnicas de solución mas comúnmente usadas. En el segundo capítulo se describe el modelo utilizado y las restricciones a satisfacer para obtener una asignación. En el tercero se describe las heurísticas desarrolladas y el generador de problemas utilizado para los ensayos, comparando los algoritmos implementados, con otros existentes (eg: Recocido Simulado) a partir de lo cual se extraen conclusiones.

Capítulo 1

1 Introducción a los Sistemas de Tiempo Real

En los Sistemas de Tiempo Real (STR) los resultados no sólo deben ser correctos desde el punto de vista lógico-aritmético sino que deben ser obtenidos antes de un cierto tiempo predefinido [56]. Los Sistemas de control de procesos, fabricación automática en plantas mediante robots, multimedia y telecomunicaciones son ejemplos concretos de sistemas de tiempo real, donde la computadora cumple un papel esencial [12].

En los últimos tiempos, los STR han adquirido complejidad y refinamiento. El conocimiento requerido para implementarlos cubre tanto el diseño de hardware y software, de los cuales el último es el más complejo y el menos entendido. Para poder

obtener una base sólida con qué resolver los STR se debe tener en cuenta la especificación del sistema, la definición de los métodos de diseño, el soporte para lenguajes de alto nivel y la elaboración de buenas herramientas para el desarrollo. Las diferentes variantes de elementos de cálculo utilizados forman la parte integral de los sistemas de tiempo real.

Para comprender su uso se debe entender lo que tratan de lograr, para lo cual es necesario conocer dónde, cómo y porqué se presentan los problemas.

Típicamente un STR consiste en un sistema que controla y un sistema controlado. Por ejemplo en automatización de plantas el sistema controlado es el piso de la planta que incluye los robots, las líneas de ensamble, etc. El sistema que controla está conformado por la computadora y la interfaz hombre-máquina que administra y coordina la actividad en el piso de la planta. El sistema controlado puede ser visto entonces como el ambiente con el cual la computadora interactúa. El sistema que controla interactúa con su entorno a partir de la información que sobre el mismo dispone, la cual es suministrada por varios sensores. Para obtener resultados correctos es imperativo que el estado del entorno percibido por el sistema de control sea consistente con su estado actual, para lo cual es necesario obtener lecturas del mismo a intervalos periódicos. El dispositivo encargado de realizar los cálculos debe responder mediante el envío de señales a los actuadores. En todos los casos, hay un tiempo límite dentro del cual se debe producir la respuesta. La habilidad del dispositivo de cálculo para cumplir con estas demandas depende de su capacidad para realizar los cálculos necesarios en el tiempo especificado. Aun cuando un número grande de eventos ocurran muy próximos en el tiempo, el procesador debe diagramar

los cálculos de manera tal que la respuesta sea enviada a cada actuador dentro del límite de tiempo establecido. Puede suceder, que el sistema no sea capaz de cumplir con las demandas requeridas y en este caso se dice que el sistema no posee los recursos o la velocidad suficientes. Sólo un sistema con recursos ilimitados y capaz de procesar a una velocidad infinita podría satisfacer cualquier demanda. La imposibilidad de cumplir con las restricciones de tiempo para una respuesta determinada, puede producir diferentes consecuencias: en algunos casos, puede no haber efectos, en otros casos, los efectos pueden ser menores y corregidos, mientras que en algunos los resultados pueden ser catastróficos. Luego, podría decirse que un STR tiene en general recursos limitados, debe interactuar con un ambiente que posee propiedades variables en el tiempo y debe exhibir un comportamiento predecible también en función del tiempo.

En la mayoría de los STR las actividades críticas en tiempo deben coexistir con aquéllas que no son críticas en tiempo. Un almacenamiento de eventos (no crítico) en un medio magnético de una computadora que realiza la automatización (crítico) de una planta industrial es un ejemplo concreto de lo anterior. Se define a cualquiera de estas dos actividades como una *tarea* y cuando está debe cumplir con especificaciones temporales se la denomina *Tarea de Tiempo Real*. Idealmente, la computadora debería ejecutar un conjunto de tareas críticas en tiempo (las cuales deben cumplir con su *vencimiento*), de manera conjunta con tareas no críticas en tiempo, a las que debe minimizar sin embargo su tiempo de respuesta. Los STR convencionales son aquéllos en los cuales las tareas no cambian durante su ejecución ninguno de sus parámetros

temporales. Estas pueden ser esporádicas (e.g. alarmas, que son aperiódicas pero con un vencimiento definido) o periódicas (con vencimiento y periodo definidos).

Históricamente los STR eran relativamente sistemas simples y operaban sobre un entorno bien caracterizado. Sin embargo la emergente generación de sofisticados STR requiere manejar sistemas complejos, los cuales están incompletamente especificados y con entornos totalmente dinámicos. La resolución de este tipo de problema es definida como una búsqueda en el espacio del problema. A partir de esta noción, se define la resolución de problemas de STR como la búsqueda de soluciones con restricciones temporales en el espacio del problema. A partir de este enfoque es importante marcar la diferencia entre la *obtención de una solución en tiempo real* y la obtención de una solución que cumpla *con restricciones de tiempo real*. La primera intenta obtener una salida antes de un tiempo predefinido, usando técnicas heurísticas o bien de inteligencia artificial, mientras que la segunda obtiene una salida off-line que durante el tiempo de ejecución cumplirá con las restricciones temporales.

La mayor parte de la teoría clásica sobre diagramación de tareas en tiempo real utiliza una diagramación estática. Para realizar una evaluación de la diagramabilidad del sistema antes de la ejecución con la intención de poder garantizar los vencimientos, es necesario conocer de antemano, para cada tarea, su período, su vencimiento y su tiempo de ejecución. Esto es verdadero para la mayoría de los STR y además presenta una ventaja adicional, cual es que permite reducir el uso de preciosos recursos invertidos en la diagramabilidad y el cambio de contexto durante la ejecución [60].

El algoritmo estático de diagramación opera sobre el conjunto de tareas y produce una diagramación simple y fija en todo momento. En contraste, el algoritmo de

diagramación dinámica conoce completamente el conjunto de tareas activas en un determinado instante, pero pueden producirse en el futuro nuevos arribos de tareas, no conociéndose la diagramación al momento de otorgar el procesador al conjunto de tareas activo. Luego, la diagramación cambia a lo largo de todo el tiempo.

La diagramación off-line es definida a menudo como diagramación estática, pero esto es incorrecto. Si el diseñador del STR puede identificar el máximo conjunto de tareas con sus correspondientes peores casos puede aplicar un algoritmo de diagramación estático para producir una diagramación estática. Este diagramador es fijo y su procesamiento es on-line. Si todas las suposiciones son verdaderas, todas las tareas cumplirán con sus vencimientos.

En otros casos el análisis off-line produce un conjunto estático de prioridades que son usadas durante el tiempo de ejecución. El diagramador en sí mismo no es fijo, aunque las prioridades que maneja el diagramador lo son. Esto es común en la técnica de diagramación por periodos monotónicos crecientes. Generalmente un algoritmo de diagramación con algunas pequeñas modificaciones puede ser aplicado a diagramación estática o dinámica en forma off-line u on-line. La diferencia importante es conocer la perfomance del algoritmo utilizado en cada uno de esos casos. Por ejemplo si se considera el algoritmo Menor Tiempo al Vencimiento (Earliest Deadline First) veremos que al utilizar esta política de diagramación en forma estática resulta óptima para la mayoría de las situaciones antes enumeradas, pero cuando es aplicada a diagramación dinámica en multiprocesadores deja de serlo.

2 Sistemas Distribuidos de Tiempo Real

Una red de computadoras es un conjunto interconectado de procesadores, capaces de intercambiar información entre sí. En la industria, por ejemplo, el uso de computadoras Digitales aplicadas al control automático evolucionó desde un único computador supervisando algunos controladores analógicos, a complejos sistemas que correlacionan múltiples procesadores, integrados en una o varias redes de datos de tiempo real. Los nodos del sistema típicamente no comparten memoria y la comunicación es implementada por pasaje de mensajes. En [21] se especifican varias propiedades para definir un sistema distribuido (SD):

a) Multiplicidad de los recursos de propósito general, tanto físico como lógicos que son dinámicamente asignados a las tareas.

b) La distribución física de los recursos lógicos y físicos están acoplados por una red de comunicaciones.

c) Un sistema operativo de alto nivel que debe unificar e integrar el control de los componentes distribuidos.

d) Transparencia de los servicios, que permite acceder por su nombre, a todos los servicios distribuidos. Las motivaciones prácticas para implementar los SD incluyen alta perfomance, incremento de la confiabilidad y la seguridad, y el mejoramiento de acceso a datos dispersados.

Irónicamente, tratar de explotar estas características enumeradas, puede degrada a los SD. Por ejemplo, implementar tolerancia a fallas para mejorar la seguridad implica generar algoritmos de recuperación, los cuales pueden convertirse en una fuente seria de fallas potenciales y reducir la confiabilidad del sistema. Existen dos vertientes

importantes seguidas por los investigadores con diferentes motivaciones, la primera mira a la distribución de recursos como un fin en si mismo, que incluye el cómputo masivo multipropósito en paralelo, la tolerancia a la falla, la seguridad y la respuesta a la demanda. En la otra vertiente, el diseñador fuerza la computación distribuida pues la aplicación así lo requiere. Es decir, una aplicación que no es inherentemente distribuida, es implementada como tal. Esto puede abarcar a los sistemas de bases de datos, la automatización, el sensado y el control remoto.

Los sistemas distribuidos de tiempo real son cada vez más necesarios, siendo de mayor importancia en la fabricación automatizada con control distribuido. El mayor énfasis desde el punto de vista de Tiempo Real, está puesto en políticas eficientes de diagramación y asignación de tareas. Se obtiene entonces una alta proliferación de estos sistemas con una significante mejora en la tolerancia a la falla, la compartición de recursos y las comunicaciones.

Por otro lado los Sistemas Operativos (SO) para SD pueden ser categorizados en dos formas distintas:

1. Sistemas operativos de red, donde cada una de las computadoras enlazadas tiene un Sistema Operativo local que es independiente de la red. La interacción de todos estos Sistemas Operativos permite comunicarse y compartir recursos.
2. Sistemas Operativos inherentemente distribuidos donde existe un único sistema para todas las computadoras enlazadas.

Las técnicas y modelos utilizados en este trabajo están basados en esta última categorización, pues el diseño del Sistema Operativo nace con los requerimientos de red *in mente* y trata de manejar los recursos de la misma de una manera global,

permitiendo satisfacer una amplía variedad de requerimientos, en especial de Tiempo Real. Al tener un sistema operativo de tiempo real distribuido con un conjunto de tareas interactuando con el medio y comunicándose entre ellas a través de un canal de comunicaciones, existen suficientes motivos para establecer un proceso de asignación de tareas en forma automática y estática teniendo en cuenta todas las restricciones naturales impuestas por estos sistemas.

3 Algoritmos de Asignación

Aunque la capacidad científica de la humanidad parece haber alcanzado unos niveles que permiten en la actualidad dar respuestas a prácticamente cualquier reto que se le plantee, lamentablemente no es ésa la realidad. Es más, si pensamos en los problemas reales que suelen ser de interés, para los cuales el objetivo es encontrar la solución que optimiza algún tipo de criterio la situación es justamente la opuesta: tan sólo una pequeña parte de ellos pueden ser resueltos. En tales circunstancias nuevas herramientas están adquiriendo cada vez una mayor importancia. Centrémonos para aclarar estas ideas, por ejemplo, en lo que ocurre en la asignación de tareas a procesadores en un sistema distribuido con restricciones de tiempo real.

En los problemas de decisión que normalmente se presentan por lo general existe una serie de recursos escasos (procesadores, memoria, comunicación, etc.), o bien de requisitos mínimos que hay que cumplir (vencimientos, precedencias, etc.) que condicionan la elección de la estrategia más adecuada. Como por lo general el objetivo al tomar la decisión consiste en llevar a cabo el plan propuesto de una manera

óptima buscando el máximo beneficio, no es de extrañar, pues, que la resolución de este tipo de problemas haya atraído la atención de numerosos investigadores.

En el caso particular en que el objetivo y las restricciones son lineales apareció una disciplina denominada programación lineal (PL) y un algoritmo eficiente para su resolución denominado *simplex*. Este algoritmo minimiza una función lineal, sujeto a un conjunto de restricciones también lineales. El método es esencialmente una mecanización de un procedimiento que salta de una Nbfs (Non-zero basic factible solution, solución básica factible con coeficientes que no son ceros y tienen valor positivo) a otra Nbfs hasta reducir al mínimo a todos los coeficientes del costo que no son cero. Para poder aplicar la mecanización es necesario obtener el primer Nbsf que servirá como punto de partida para iterar. Para ello se construye una tabla con los coeficientes que son cero y con los que no lo son. Cuando son muchísimas las variables que intervienen en el modelo, este algoritmo no resulta eficiente dado que el tiempo de cálculo necesario es excesivamente largo, incluso para equipos con alto poder de cómputo. No es que el algoritmo no pueda llegar al óptimo buscado sino que su tiempo de respuesta no es operativo.

Existen diferentes tipos de algoritmo, que pueden definirse según el modo en que buscan y construyen sus soluciones. Una posible clasificación es la siguiente [54]:

a) Métodos Constructivos: Consisten en ir añadiendo paulatinamente componentes individuales a la solución hasta que se obtiene una solución factible. El método trabaja por estados, examinando en cada uno de ellos si la solución es factible y si es mejor o peor que la encontrada en el estado anterior. En el estado final es obtenida la mejor solución. Estos algoritmos no siempre

conducen a una buena solución y son mas comúnmente usados cuando los algoritmos óptimos de resolución de un problema se tornan intolerablemente largos. El más popular de estos métodos lo constituyen los algoritmos golosos o devoradores, los cuales construyen paso a paso la solución buscando el máximo beneficio en cada paso.

b) Métodos de Descomposición: Se trata de dividir el problema en subproblemas más pequeños, siendo la salida de uno la entrada del siguiente, de forma que al resolverlos todos obtengamos una solución para el problema global. Basados en la técnica del divide y triunfarás, este algoritmo es utilizado por ejemplo en el método de ordenación *mergesort*.

c) Métodos de Reducción: Tratan de identificar alguna característica que presumiblemente deba poseer la solución óptima y de ese modo simplificar el problema.

d) Manipulación del Modelo: Estas heurísticas modifican la estructura del modelo con el fin de hacerlo más sencillo de resolver, deduciendo, a partir de su solución, la solución del problema original. Pueden consistir en reducir el espacio de soluciones o en aumentarlo al eliminar restricciones del problema.

e) Programación Dinámica: Se utiliza cuando la única alternativa es enumerar todas las posibles configuraciones de un conjunto de datos y verificar cada una para encontrar si alguna de ellas es una solución. Una idea esencial es mantener una tabla que contenga todas las configuraciones previas realizadas y sus resultados. Si el total de configuraciones es grande entonces esta técnica requiere de mucho tiempo y espacio. Si hay un conjunto chico de

configuraciones distintas, el algoritmo evita recalcular la solución una sobre otra. Para determinar si existe un número chico de configuraciones distintas se utiliza un principio de optimalidad. Este principio determina qué decisiones contribuyen a que la solución final sea óptima con respecto al estado inicial. Un ejemplo común es la búsqueda optima en un árbol binario.

f) Recorrido y Búsqueda en Grafos: A menudo objetos complejos son almacenados usando una estructura de datos específica. La misma consiste en nodos que contienen campos asociados. Estos campos pueden tener datos asociados o bien punteros a otros nodos. Una particular instancia de algún objeto puede incluir muchos nodos conectados. Ejemplos típicos son árboles, listas y grafos. Este método se mueve a lo largo de la estructura de datos de un nodo a otro colectando información. Después que todos los nodos han sido expandidos puede ser conocida una respuesta final.

g) Backtraking: Es un algoritmo apropiado cuando la solución deseada puede expresarse en forma de vector. A menudo, el problema consiste en maximizar, minimizar o bien encontrar algún criterio de optimalidad. Todos los vectores son investigados. Cada componente del vector es evaluada usando funciones modificadoras que determinan si el vector que se está generando tiene chances de prosperar. Este método es eficiente cuando la función modificadora es capaz de eliminar un conjunto grande de posibles vectores.

h) Ramificación y Acotación Truncada: Su diseño es similar al backtraking y trata de encontrar uno o más vectores que satisfacen algún criterio. La diferencia con backtraking es que son generados y explotados todos los

elementos de una posible solución. En vez de generar un vector solución elemento por elemento, todos los posibles candidatos para la próxima entrada son producidos y cargados en un conjunto, cuyos elementos son denominados vivos. Una vez seleccionado uno para expandirse, se usa una función para eliminar elementos del conjunto. La función de acotamiento es seleccionada bajo algún criterio para poder rechazar cierto tipo de vectores. Con una buena función de acotamiento este método es muy eficiente.

i) Métodos de búsqueda por entornos: Dentro de esta última categoría es donde se encuadra la mayoría de las metaheurísticas. Estos métodos parten de una solución factible inicial y mediante alteraciones de esa solución, van pasando en forma iterativa, y mientras no se cumpla un determinado criterio de parada, a otras factibles de su entorno, almacenando como óptima la mejor de las soluciones alcanzadas. Un concepto clave es cómo realizar el paso de una solución factible a otra. El entorno de la solución es el conjunto de soluciones parecidas a ella. El mapeo de una solución factible a otra, debe entenderse como la posibilidad de obtener una solución a partir de otra obtenida inicialmente. Para ello se debe realizar una operación elemental que permita eliminar o añadir un elemento a la solución original. Estos métodos pretenden buscar de entre los elementos del entorno de la solución actual, aquél que tenga un mejor valor de acuerdo con algún criterio predefinido, moverse a él y repetir la operación hasta que se considere que no es posible hallar una mejor solución. Esto puede suceder cuando no hay algún elemento en el entorno de la solución actual, o bien porque se verifica algún criterio de parada. Una clase

especial dentro de este método lo constituye el método de búsqueda local o de descenso. En él, en cada iteración, el movimiento se produce desde la solución actual a una de su entorno que sea mejor que ella, finalizando la búsqueda cuando todas las de su entorno son peores. Es decir, la solución final será siempre un óptimo local. Uno de los mayores inconvenientes con los que se enfrentan estas técnicas es la existencia de óptimos locales que no sean óptimos absolutos. Si a lo largo de la búsqueda se cae en un óptimo local, en principio la heurística no sabrá continuar pues se queda ligada a ese punto.

Existe un tipo concreto de problemas de optimización que, en nuestro caso, resulta especialmente interesante. Lo forman los denominados problemas de optimización combinatoria. En ellos, las variables de decisión son enteras y el espacio de soluciones está formado por ordenaciones o subconjuntos de números.

Quizás los dos problemas combinatorios más conocidos sean el de la mochila (PM) y el del viajante (PV). El primero de ellos consiste en seleccionar de entre un conjunto de n productos, cada uno con un valor c_i y un volumen v_i, aquéllos que quepan en un recipiente con volumen V y que tengan el mayor valor posible. El otro problema, el PV, a pesar de ser muy sencillo de plantear, resulta de resolución compleja. Se trata de determinar, dado un mapa de caminos, en qué orden deben visitarse n ciudades del mapa para que la distancia sea mínima. En este caso, en lugar de tener que seleccionar un subconjunto de elementos, se trata de elegir una permutación de las n ciudades.

En los problemas de tipo combinatorio existe siempre un procedimiento elemental para determinar la solución óptima buscada: realizar una exploración exhaustiva del problema y obtener todo el conjunto de soluciones. Es decir, generar todas las

soluciones factibles que satisfacen las restricciones, calcular para cada una el coste asociado y elegir finalmente la que haya dado lugar al mejor de ellos. Aunque este método teóricamente nos lleva siempre a la solución óptima buscada, no es eficiente, pues el tiempo de cálculo necesario crece exponencialmente con el número de elementos del problema.

Consideremos el problema PM. El número de subconjuntos del conjunto {1,2, 3,… n} es 2^n. Luego, si una computadora pudiese, en tan sólo un segundo, generar un millón de esos subconjuntos y evaluar su valor, necesitaría solamente un segundo para hallar la solución de un problema con n=20 elementos, unas dos semanas para un problema con n=40 elementos y mas de 30 siglos de cálculo para analizar las posibles soluciones de un problema con n=60.

Por tanto, es evidente que existen problemas combinatorios para los que no se conocen algoritmos de resolución más que aquéllos en los cuales se produce una explosión del tiempo de cálculo al aumentar el tamaño del problema, siendo calificados como problemas computacionalmente intratables. Por el contrario, para otros problemas combinatorios como es el de asignación, sí existen algoritmos que sólo crecen polinomialmente en tiempo con el tamaño del problema. Sin embargo es imposible aseverar que en el caso de hallar una solución ésta sea óptima.

Matemáticamente se trató de caracterizar un tipo y otro de problemas [36]. De aquéllos para los cuales se conocen algoritmos que necesitan un tiempo polinomial para ofrecer la solución óptima se dice que pertenecen a la clase **P** (polinomial) y se considera que son resolubles eficientemente. Sin embargo, la mayoría de los principales problemas de optimización que aparecen en ingeniería pertenecen a otra

clase, la denominada **NP** (no deterministicamente polinomial), en la cual están incluidos aquéllos problemas para los que no se conoce un algoritmo polinomial de resolución, aunque sí es posible, dada una solución, comprobar en tiempo polinomial si su coste es mejor que un determinado valor.

Sólo los problemas en **P** son algorítmicamente resolubles en forma eficiente y es claro que **P** $\subseteq$ **NP**. Si ocurriera lo contrario, **P** $\supseteq$ **NP**, querría decir que para la mayoría de los problemas de interés existen algoritmos eficientes de resolución. Sin embargo, el hecho es que nadie hasta la fecha, ha podido demostrar ni que la igualdad **P=NP** sea cierta, ni tampoco que haya problemas en **NP** que no están en **P**. Esta es una de las más importantes cuestiones que actualmente tiene abiertas las matemática. Más aún, en [15] se demuestra que hay problemas en **NP** que son especialmente difíciles. Son los denominados NP-completos. Estos problemas, entre los que se incluye la mayoría de los de interés [36], son aquéllos que además son NP-duros, es decir, tienen la particularidad de que todos los problemas en **NP** pueden ser reducidos polinomialmente a ellos, o lo que es lo mismo, que si se puede dar una solución en tiempo polinomial para uno de ellos, se podría dar para todos los de **NP**. Nunca nadie ha podido encontrar algoritmos eficientes para problemas NP-completos y al detectar que un problema pertenece a esa clase, no resulta inteligente tratar de buscar algoritmos eficientes para él [38].

4 Heurísticas y Metaheurísticas en un contexto de TR.

A continuación describiremos brevemente las diferentes técnicas que más éxito han tenido en la resolución de problemas de tipo combinatorio en un contexto de tiempo real duro, con el fin de ofrecer una idea de sus bases teóricas.

4.1 Heurísticas

Existe una serie de importantes problemas combinatorios difíciles de resolver utilizando los métodos denominados simplex, ramificación y acotación o teoría de grafos. Debido a su interés práctico comenzaron a aparecer algoritmos que proporcionan soluciones factibles que satisfacen las restricciones del problema. Aunque las mismas no optimicen la función objetivo, se supone que se acercan al valor óptimo en un tiempo de cálculo razonable.

Una posible manera de describir esos algoritmos es como procedimientos simples, a menudo basados en el sentido común, que se supone ofrecerán de un modo fácil y rápido una buena solución a problemas difíciles. La solución obtenida no es necesariamente la óptima [61]. Este tipo de algoritmo es denominado heurística. Inicialmente, acusadas de escaso rigor matemático, no fueron bien vistas por los académicos [19]. Su interés práctico como herramienta útil que ofrece soluciones a problemas reales, fue lentamente abriendo las puertas de la aceptación generalizada.

Una importante ventaja de las heurísticas respecto a las técnicas que buscan soluciones exactas, es que permiten una mayor flexibilidad para el manejo de

las características del problema. Es decir, no resulta complejo diseñar algoritmos heurísticos que en lugar de considerar funciones lineales utilicen no linealidades. Generalmente las heurísticas ofrecen más de una solución, lo cual permite ampliar las posibilidades de elección, principalmente cuando existen factores no cuantificables que no han podido ser añadidos en el modelo, pero que también deben ser considerados. Además, suele ser más fácil de entender la fundamentación de las heurísticas que los complejos métodos matemáticos que utilizan la mayoría de las técnicas exactas.

El uso de heurísticas también presenta inconvenientes. Por lo general no es posible conocer la calidad de la solución, es decir, cuán cerca está del óptimo que nos ofrecen. Se ha trabajado al respecto y se desarrollaron métodos para realizar acotaciones que nos den una orientación respecto a la calidad de la solución obtenida.

Un procedimiento consiste en relajar el problema. Se pueden eliminar algunas de las restricciones al efectuar, una ramificación y acotación truncada, o bien una relajación. Bajo estas condiciones el problema es más fácil de resolver, ya que al eliminar restricciones aumenta el conjunto de soluciones y puede entonces aparecer un nuevo óptimo mejor que el original. De este modo, valores cercanos a la solución nos garantizan que la heurística está dando una buena aproximación. Cuando este tipo de procedimientos evaluadores de la calidad de la heurística no son posibles, siempre cabe utilizar métodos sencillos que detectan simplemente que la heurística no es buena. Una técnica sencilla consiste en generar aleatoriamente varias soluciones y si son similares

a la obtenida se puede poner en duda la efectividad de la heurística. No cabe duda, sin embargo, de que cuando esté disponible una técnica *exacta* debe ser preferida a cualquier tipo de heurística.

Una primera posibilidad para salvar esa dificultad consiste en reiniciar la búsqueda desde otra solución inicial y confiar en que, en esta ocasión, la exploración siga por otros caminos.

Ser independientes de la solución inicial de la que se parta, representa otro problema que tienen las heurísticas. El punto inicial tiene una gran influencia sobre la posibilidad de caer o no en un óptimo local.

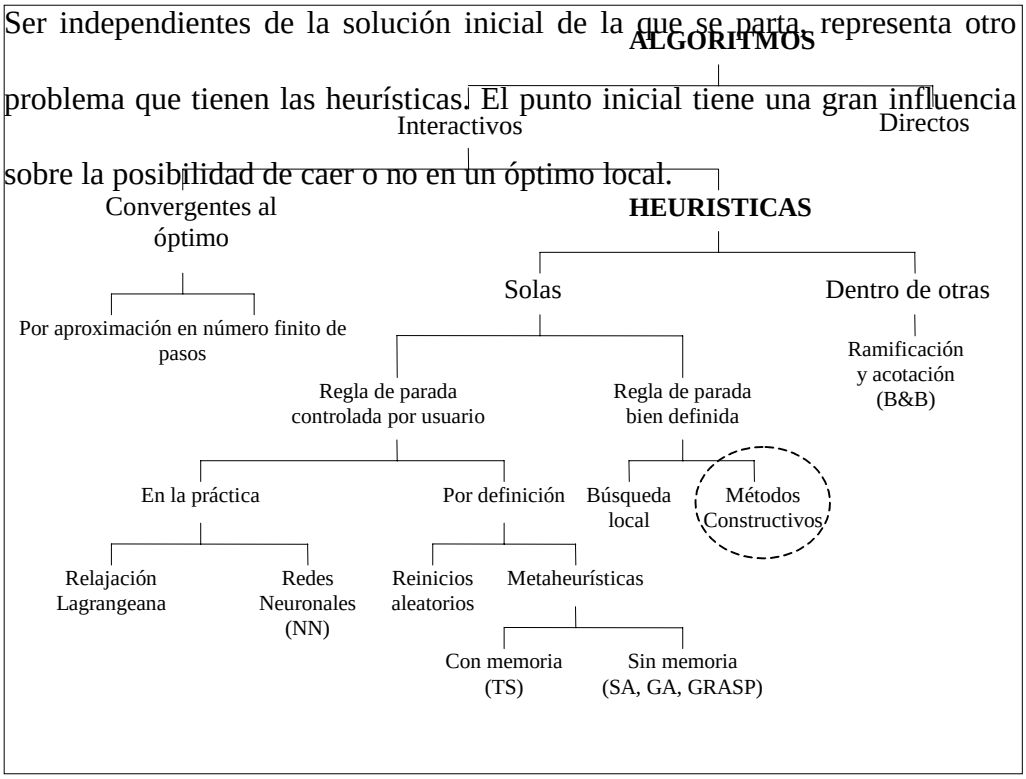

Figura 1

Una posible clasificación de las técnicas heurísticas [4], puede verse en la figura 1, donde el criterio fundamental es la regla de parada que se ha definido para el algoritmo.

La heurística desarrollada en esta tesis se ubica en el circulo punteado de la figura anterior. El propósito de las heurísticas constructivas es encontrar una posible solución que cumpla todas las restricciones usando una estrategia simple, evitando gastar mucho tiempo en optimizar la función objetivo.

Existen diferentes algoritmos basados en este tipo de heurísticas para atacar la asignación en un contexto de tiempo real duro. Las mismas pueden encontrarse en [48,1,31] y se caracterizan por utilizar unas listas globales de prioridad para decidir qué tarea y qué procesador estarán involucrados en la próxima asignación. Los métodos difieren principalmente en la función utilizada para dar prioridad a la asignación y construir las listas. Algunos de estos enfoques, para sobreponerse a malas soluciones originadas al tomar decisiones equivocadas, implementan un sistema de vuelta atrás [48,1]. La mayoría de estas técnicas usa la preasignación para reducir el problema de tamaños. Si la vuelta atrás no es utilizada, estos métodos son muy veloces. Sin embargo, al no utilizar esta técnica, se corre el riesgo de no alcanzar una solución factible o bien alcanzar una asignación con un valor pobre de la función objetivo. Las cosas mejoran para el método anterior si la evaluación se realiza utilizando como parámetro de medida la relación de éxito en alcanzar una solución, entendiéndose como relación de éxito a las veces que un algoritmo encuentra solución para un número predeterminado de ejemplos. El

procesamiento por listas es altamente sensitivo a la técnica utilizada para priorizar las listas, incrementándose el problema cuando se debe cumplir con diferentes criterios de asignación de manera concurrente, como podrían ser las restricciones temporales y de comunicaciones.

4.2 Algoritmos Genéticos

Los Algoritmos Genéticos (AG) son técnicas de búsqueda basadas en la mecánica de la selección natural y la genética [28]. Su estructura se ha diseñado basándose en una abstracción artificial del algoritmo de selección propio de la naturaleza, con la esperanza de que así se consigan éxitos similares en relación con la capacidad de adaptación a un amplio número de ambientes diferentes. En los organismos biológicos, la información hereditaria es pasada a través de los cromosomas. Estos están formados por los genes, a cada uno de los cuales esta asociado un valor conocido como alelo. Los alelos son binarios o no, representando valores de las variables de decisión, que se corresponderán con cada uno de los genes.

Estos algoritmos de búsqueda al estar basados en la selección y genética natural, combinan el concepto de supervivencia del más apto, con un intercambio estructurado pero aleatorio de información. Estos conceptos involucran la presentación de las características de los mejores exponentes de una generación en la generación siguiente, introduciendo sobre la población cambios aleatorios por medio de los operadores genéticos de cruce y mutación. Este componente aleatorio evita la caída en un máximo local del

cual no se pueda salir impidiendo que se llegue al máximo global. Al utilizar la formación histórica de la población los individuos pueden ser orientados hacia regiones del espacio de búsqueda mas favorables. Esto representa una de las principales ventajas de los Algoritmos Genéticos frente a los métodos tradicionales de búsqueda. Mientras la mayoría de las otras metaheurísticas (ver figura 1) generan una solución simple en cada iteración, los algoritmos genéticos trabajan sobre una población de soluciones, generando una nueva en cada iteración. Por lo general son algoritmos neutrales al contexto y no consideran ni explotan la información del problema dentro del proceso de búsqueda [2].

En la naturaleza, la aptitud se determina por la habilidad de un organismo para sobrevivir a obstáculos que le impiden llegar a la madurez y reproducirse. En los AGs, la función de evaluación mide la adaptación del individuo al medio ambiente que se enfrenta. La misma debe ser optimizada para encontrar los individuos que mejor la representan. La función de evaluación, es el único componente del AG dependiente del problema que se esta resolviendo; basta con cambiarla para cambiar el dominio del problema. Al estar basados los AGs en la genética natural, la terminología utilizada se corresponde en forma directa con el vocabulario genético.

Cada generación esta compuesta por cromosomas, los cuales pueden ser implementados como vectores formados por genes. Antes de poder evaluar un individuo, es necesario decodificar la información que contiene para encontrar el valor de las variables que se utilizan como entradas a la función

de evaluación. En un AG es necesario evaluar a todos los miembros de la población al principio de cada experimento. Durante la ejecución del algoritmo, sus operadores modifican las estructuras de la población creando nuevos miembros que deben ser evaluados antes de incorporarse a la población. Si el proceso de evaluación de cada individuo requiere de un tiempo considerable de procesamiento, el tiempo de respuesta del algoritmo será muy elevado.

En [42, 49, 7, 29] los AGs han sido utilizados para asignar tareas en un sistemas de tiempo real. En cada paso de iteración se seleccionan al azar dos de las mejores soluciones. Utilizando estos individuos se crean dos nuevos individuos usando el operador de cruce. En [7,29] la representación usada para aplicar los operadores está basada en una lista de diagramación. Para cada procesador existe un cromosoma que lo representa, donde los genes del mismo identifican a las tareas. El punto de cruce es seleccionado al azar considerando el vencimiento de las tareas, mientras que la mutación consiste simplemente en cambiar el lugar de asignación de la tarea seleccionada. En [49] el uso del cromosoma es diferente, la solución es representada por un único cromosoma que contiene toda la información de manera codificada. Esta representación es mas compacta que la anterior y resulta mas directo evaluar el punto de cruce, pero se gasta mucho tiempo en codificar/decodificar la información. Estos métodos son altamente sensitivos a sus parámetros (tamaño de la población, número de puntos de cruce, criterios de selección, etc.).

Los AGs poseen las mismas dos desventajas que otras búsquedas no guiadas:

1. La sensibilidad de los parámetros para encontrar soluciones para un amplio espectro de problemas de aplicación. Esta desventaja puede disminuirse utilizando una técnica introducida por Ahmad [2] que mapea del espacio del problema al espacio de soluciones codificando de manera particular el valor del alelo. Un método basado en esta técnica con buenos resultados puede encontrarse en [23]
2. Para la asignación en tiempo real cuando el contenido del alelo no guarda relación directa con datos del problema, el cruce entre cromosomas no es un intercambio estructurado de información. Por tal motivo es conveniente usar una representación del cromosoma tal que sus alelos estén vinculados en alguna forma con los datos del problema (e.g. factor de utilización de las tareas) y seleccionar un operador de cruce apropiado.
3. Cuando se trabaja con un ejemplo grande, el algoritmo que realiza el proceso de asignación debe manejar un volumen importante de información en cada iteración, excediendo generalmente la capacidad de memoria disponible. Esto puede salvarse mediante la utilización de mecanismos que adaptan dinámicamente la tasa de aplicación de los operadores, modificando las probabilidades de cruce y mutación durante la ejecución. A este efecto se usa un registro que indica la perfomance de los mismos [22].

La ventaja indiscutible de esta metaheurística es que escapa fácilmente de un mínimo local.

4.3 Recocido Simulado

La segunda de las metaheurísticas que veremos, hace uso de conceptos originalmente descritos por la mecánica estadística. El proceso físico de recocido primero reblandece el sólido mediante su calentamiento a una temperatura elevada, y luego lo va enfriando lentamente hasta que las partículas se van colocando por sí mismas en el estado fundamental del sólido. Para cada temperatura durante el proceso de recocido, el sólido puede alcanzar el equilibrio térmico si el enfriamiento se produce muy lentamente. Si el enfriamiento se efectúa demasiado rápido, el sólido puede llegar a estados metaestables en lugar del fundamental. En este último las partículas forman retículas perfectas y el sistema está en su más bajo nivel energético, mientras que en los meta - estables existen defectos en forma de estructuras de alta energía.

El algoritmo realiza el paso de un estado a otro según las siguientes reglas: si el estado generado posee una energía menor que el estado que actualmente se tiene, entonces se acepta el estado generado como el estado actual. En caso contrario, el estado generado se aceptará con una determinada probabilidad basada en la distribución de Boltzmann. Esta probabilidad de aceptación es función de la temperatura y de la diferencia entre los dos niveles de energía. Cuanto menor sea la temperatura, menor será la probabilidad de

transformación en un estado de mayor energía, y cuanto mayor sea la energía del nuevo estado, menor será la probabilidad de que sea aceptado. Por tanto, cada estado tiene una posibilidad de ser alcanzado, pero con diferente probabilidad a diferentes temperaturas.

Los estados del sistema se corresponden con las soluciones del problema de optimización combinatoria; la energía de los estados con el criterio de evaluación de la calidad de la solución; el estado fundamental con la solución óptima, mientras que los estados metaestables son los equivalentes de los óptimos locales. La temperatura no tiene correlación con el proceso de optimización, por lo tanto es un parámetro que debe ir ajustándose de acuerdo a un algoritmo predefinido.

Conceptualmente desde el punto de vista de la asignación, es un método de búsqueda por entornos, donde el algoritmo selecciona aleatoriamente un candidato de entre los que componen el entorno de la solución actual. Si el candidato es mejor que ella, en términos del criterio de evaluación, entonces es aceptado como solución actual. En caso contrario, será aceptado con una probabilidad que decrece según crezca la diferencia entre los costos de la solución candidata y actual. Cuando el candidato no es aceptado, el algoritmo selecciona aleatoriamente otro candidato y se repite el proceso.

La alteración en la selección de la siguiente solución se ha diseñado de forma que se reduzca la probabilidad de quedar atrapado en un óptimo local. Se ha llegado a demostrar que el recocido simulado es capaz de encontrar asintóticamente la solución óptima con probabilidad uno, pues aunque esté

garantizada la optimalidad, sólo se alcanzará sin embargo tras un número infinito de pasos en el peor de los casos. Sin embargo, la convergencia asintótica del algoritmo puede ser aproximada en la práctica y realizada en tiempo polinomial.

El recocido simulado se convirtió probablemente en el algoritmo más popular. En [6, 8, 58, 43, 17] ha sido usado para la asignación de tareas a procesadores en un entorno de tiempo real duro. En cada paso de la iteración una alternativa es evaluada. Las alternativas con buenas funciones objetivo son aceptadas, mientras que las alternativas con valores malos son incorporadas con una cierta probabilidad que depende del factor de temperatura. En [43,58] se trata exclusivamente el caso de asignación, mientras que en [6,17] tratan en forma conjunta el problema de asignación y el de diagramación, diferenciándose las contribuciones en el objetivo a optimizar: comunicaciones en [58,43], menor tiempo de respuesta en [6] y desplazamiento o desfasaje en [17].

El recocido simulado es muy sensitivo a los parámetros y al criterio de parada, siendo altamente dificultoso encontrar parámetros genéricos que funcionen para un número grande de ejemplos de aplicación. Trabajos de evaluación experimental muestran que recocido simulado obtiene buenos resultados al problema de asignación en tiempo real duro cuando se relaja el concepto estricto de calentamiento. La mayor ventaja del recocido es su habilidad para poder escapar de un mínimo local por saltos aleatorios, pero esta habilidad en muchos casos se transforma en una desventaja para valores grandes de n^m, donde n denota el número de procesadores y m el número de tareas. Esto se

debe principalmente a que el algoritmo salta de un pico a otro ocasionando tiempos de cálculo excesivamente grandes o generando soluciones que no cumplen todas las restricciones. El buen funcionamiento del recocido simulado va a depender en gran medida de la estructura del entorno, del patrón de enfriamiento y de las estructuras de datos seleccionadas.

4.4 Búsqueda tabú

La búsqueda tabú es un tipo de búsqueda por entornos que guía un procedimiento de búsqueda local para explorar el espacio de soluciones más allá del óptimo local [27]. La búsqueda tabú selecciona de modo agresivo el mejor de los movimientos posibles en cada paso, al contrario que en la búsqueda local que siempre se mueve al mejor de su entorno y finaliza con la llegada a un óptimo local.

La búsqueda tabú permite moverse a una solución del entorno aunque no sea tan buena como la actual, de modo que se pueda escapar de óptimos locales y continuar estratégicamente la búsqueda de soluciones aún mejores.

Mientras tanto, para evitar que el proceso vuelva a un viejo óptimo local, la búsqueda tabú clasifica un determinado número de los más recientes movimientos como movimientos *tabú*. Estos no son posibles de repetir durante un determinado tiempo. Por tanto, en este caso, el escape de los óptimos locales se produce de forma sistemática y no aleatoria. Es decir, la búsqueda tabú mantiene una memoria de eventos que sean de interés. Estos pueden ser soluciones o algún otro elemento propio de las soluciones

visitadas. Posteriormente, usa esa memoria para alterar el entorno de la solución actual y así influir en el proceso subsiguiente de búsqueda. Actualmente, el tema central del método consiste en tratar de explorar la memoria adaptiva para así controlar el proceso de búsqueda.

La memoria puede contener información respecto a la frecuencia de eventos y detalles de los últimos accesos. Un subconjunto de los movimientos contenidos en ella serán clasificados como tabú y así serán excluidos durante un corto período de tiempo al considerar, dentro de ese entorno, otras alternativas viables. A menos que sean soluciones de buena de calidad, un subconjunto de los movimientos contenidos en la memoria serán clasificados. Cuando el ritmo al cual se están encontrando nuevas soluciones mejores que las ya halladas decrece, entran en juego las estrategias de intensificación y diversificación. Las de intensificación enfatizan los atributos comunes a las buenas soluciones halladas hasta ese instante. Podría ser interesante volver a regiones atractivas ya exploradas para proceder a una búsqueda más intensiva en esa área. Por el contrario las estrategias de diversificación consideran atributos poco usados en el pasado, con el objeto de dirigir la búsqueda hacia nuevas regiones.

Mediante el uso de pesos asignados a diferentes atributos de las mejores soluciones, es posible en muchas ocasiones explorar regiones que resultan altamente interesantes.

Como se observa, la búsqueda tabú, al contrario que las dos anteriores, pone su énfasis en procedimientos determinísticos en lugar de aleatorios, es decir,

pretende usar la inteligencia y no la fuerza bruta. Aunque existe una variante del método que usa reglas probabilísticas para seleccionar el mejor movimiento en cada iteración, la filosofía se basa en la creencia de que la elección de una mala estrategia sistemática de búsqueda es más fructífera que la elección de una buena de tipo aleatorio. Por tanto, la búsqueda tabú hace uso de estructuras especiales de memoria y de estrategias de búsqueda dinámica cuidadosamente diseñadas para guiar el proceso de optimización de un modo eficiente.

La asignación de tareas a procesadores por un sistema de búsqueda no guiada consiste en considerar todas las posibles derivaciones de la solución corriente. De todas estas, se selecciona la que tenga el menor valor de la función objetivo para utilizarla en la iteración siguiente, hasta que este valor sea reducido lo más posible. La peor desventaja, es que no puede sobreponerse a un mínimos locales y la conducta del algoritmo durante el tiempo de corrida para aplicaciones grandes es pésima. Como alternativa de evolución a este método sencillo y con la intención de mejorar el tiempo de corrida fue que apareció la búsqueda tabú. En [46]fue usada la búsqueda tabú para la asignación pero no en un contexto exclusivo de tiempo real. Existe una alternativa definida en [3] usada para evaluar distintos métodos, donde se refleja la dificultad que tiene el método para definir la condición de parada en términos de “seguir hasta que no sea posible encontrar algo mejor”.

Capítulo 2

1. Modelo

Un conjunto de *m* tareas periódicas de tiempo real apropiativas deben ser ejecutadas en *n* procesadores, cada uno de los cuales posee una cantidad determinada de recursos. Si no hubiese condicionamientos podrían realizarse n^m asignaciones diferentes. El objetivo es obtener una asignación de las *m* tareas en los *n* procesadores cumpliendo con las diferentes restricciones de 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 de tiempo real. Es evidente que el problema puede ser no computable en tiempo razonable para valores de n^m considerables, si se pretende examinar todas las asignaciones posibles.

En la figura siguiente el conjunto A representa todas las posibles asignaciones de tareas a procesadores, el subconjunto C contiene las asignaciones que satisfacen todas las restricciones y por último el subconjunto M contiene todas las soluciones encontradas por el método utilizado. Obviamente, si C es vacío entonces el sistema es absolutamente no asignable. Si M es vacío pero C no, entonces el método utilizado no ha podido encontrar ninguna solución de las existentes.

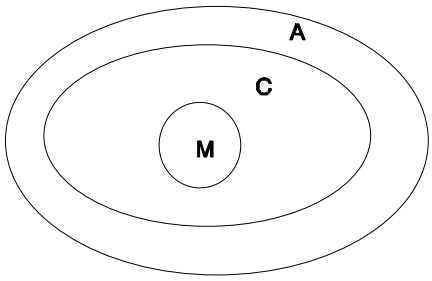

Figura 2

Para atacar el problema de asignación se considera que las m tareas τ , que modelan el sistema de tiempo real duro son parte de un *trabajo* ϑ que es representado por un grafo dirigido y acíclico con la propiedad de clausura transitiva para evitar arcos redundantes. En el grafo, los *nodos* representan las *tareas* y están identificadas por una dupla que indica su factor de utilización (U_h) y las unidades de *recursos* que necesita (R_h). El arco que une a los nodos corresponde a las restricciones de precedencia entre las distintas tareas, mientras que el peso de los arcos ($B_{h,i}$) indica la cantidad de información medida en bytes que intercambian las tareas. El periodo T de generación del trabajo es común a todas las tareas que componen el mismo. El subindice h,i indica que el arco sale del nodo predecesor h e incide en el nodo sucesor i. Luego τ_h y τ_i son predecesor y sucesor respectivamente (notado τ_h τ_i) si y solo si

existe un arco desde τ_h a τ_i y no existe $\tau_k \mid \tau_h \quad \tau_k \quad \tau_i$.

En el grafo un nodo que no tiene predecesor es llamado *nodo de entrada o nodo raíz,* mientras que uno que no tiene sucesor es llamado *nodo de salida o nodo hoja*. Un nodo no puede comenzar su ejecución antes de que todos los mensajes de sus nodos predecesores hayan sido recibidos. Se acepta que cada uno de los procesadores distribuidos tiene asociado un coprocesador de comunicaciones que maneja de forma independiente los datos recibidos desde la red de interconexión. Luego, la ejecución en los procesadores y el tráfico de información tiene lugar en forma simultánea.

La figura 3 es un subgrafo que muestra una dupla predecesora/sucesora con la información asociada, según fue definida anteriormente.

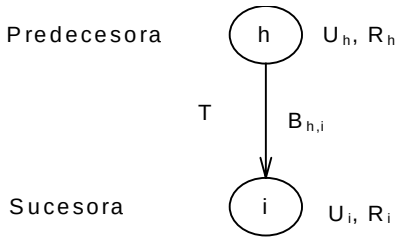

Figura 3

En [48] se define también un grafo para modelar el sistema de tiempo real, donde la designación de trabajo es reemplazada por la de tarea y la de tarea por la de subtarea.

2 Restricciones

2.1 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 de generación del requerimiento. Este intervalo es denominado *vencimiento* (en ingles "*deadline*", D) de la tarea. El tiempo se considera ranurado. La duración de una ranura se denomina *tiempo de ranura* y se simboliza T_{ran}.

La asignación del recurso es realizada por un mecanismo denominado diagramador. El tiempo de ranura es la suma del tiempo requerido por el diagramador para determinar a qué tarea se asignará el recurso, el tiempo necesario para efectivizar la asignación y una cantidad de tiempo para la utilización del recurso. En caso de requerimientos múltiples, el diagramador asignará el recurso de acuerdo a una disciplina de prioridades que establece una relación binaria de orden sobre el conjunto de tareas. Dicho conjunto es un orden total denominado *pila de prioridades*. Si el diagramador es capaz de asignar el recurso a todas y cada una de las tareas dentro de los intervalos requeridos, se dirá que el sistema es diagramable. Una vez adjudicado, la ranura no puede ser apropiada por otra tarea.

En lo que sigue, la unidad de tiempo es T_{ran}. Las ranuras se numeran 1, 2,.... *Instante t y comienzo de ranura t* son expresiones equivalentes. $\mathbf{S}(m)=\{ _1, _2, ..., _m\}$ y $\mathbf{D}(m)=\{D_1 ,D_2 ,...,D_m \}$ denotan el conjunto de m tareas y sus vencimientos, respectivamente. Se supone además que los requerimientos de la tarea se producen sincrónicamente con el comienzo de una ranura y que cada tarea los produce en forma periódica. $\mathbf{T}(m)=\{T_1 ,T_2 ,...,T_m\}$ denota el conjunto de períodos del sistema. Como es común en aplicaciones de tiempo real se supone que $\mathbf{T}(i)=\mathbf{D}(i)$. Se acepta además que ambos son múltiplos del

tiempo de ranura. **C**(m)={C_1 ,C_2 ,...,C_m } denota el conjunto de los tiempos de utilización del recurso de las tareas del sistema.

Al comienzo de cada ranura, el diagramador asigna el recurso a la tarea que posee mayor prioridad entre las que lo requieren. Sólo si no existe requerimiento alguno, el recurso permanecerá ocioso durante esa ranura. Una ranura ociosa o no usada se denomina libre. **E** denota el conjunto de ranuras libres y $e_{k(i)}$ la k-ésima ranura dejada libre por el sistema de i tareas.

La pila de prioridades puede ser dinámica o estática según que varíe o no en función del tiempo. En el caso de pila estática, la misma se arma en la inicialización del sistema y permanece invariante; se dice entonces que se trata de una disciplina de *Prioridades Fijas* (PF). Cuando la pila está organizada por períodos monotónicos crecientes, la disciplina se denomina PMC (o RMS, Rate Monotonic Scheduling). Liu and Layland [39] han demostrado que si un sistema es diagramable con una pila cualquiera de PF, también lo es con la pila PMC. Como la recíproca no es cierta, se sigue que PMC es la mejor variante de PF. Por otro lado, los mismos autores han demostrado que la peor condición de carga del sistema ocurre cuando todos los tareas producen requerimientos simultáneos. Si ello sucede en el instante t = 1,

$$R_m(t) = \sum_{h=1}^{m} \left\lceil \frac{t}{T_h} \right\rceil \quad (1) \qquad\qquad W_m(t) = \sum_{h=1}^{m} C_h \cdot \left\lceil \frac{t}{T_h} \right\rceil \quad (2)$$

$R_m(t)$ indica el número de requerimientos y $W_m(t)$ el número de ranuras solicitadas como consecuencia de los mismos en el intervalo (0,t]. $\lceil x \rceil$ indica el operador monádico techo, siendo x el menor entero mayor o igual a x.

2.1.1 El Método de las Ranuras Vacías con Bloqueo

Santos *et al* [52, 50] han demostrado que la k-ésima ranura que deja libre un sistema **S**(m) está dada por la expresión:

$$e_{k(m)} = \text{menor } t \mid t = k + W(t) \qquad (3)$$

y que un sistema **S**(m) será diagramable bajo la disciplina PMC sss

$$\forall i \in \{2,..., m\} \quad \sum_{h=1}^{i-1} \frac{C_i}{T_h} < 1 \quad y \quad T_i \geq e_{C_i (i-1)} \qquad (4)$$

donde la primera desigualdad indica la condición para que un subsistema **S**(i-1) admita la incorporación de una nueva tarea. La segunda expresión indica la condición que debe cumplir el período de la tarea i para que el mismo pueda utilizar el recurso antes de vencer el intervalo admisible.

Una de las mayores dificultades es encontrar una técnica generalizada para la evaluación de la diagramabilidad, con el propósito de predecir si el conjunto de tareas cumple con las constricciones de tiempo. El problema se complica cuando es posible que se produzcan inversiones de prioridad, fenómeno que consiste en que una tarea de baja prioridad bloquea la ejecución de una tarea de prioridad más alta. Un ejemplo común es el acceso a datos compartidos, que debe ser realizado en serie por las distintas tareas, para garantizar su consistencia. Si la tarea de baja prioridad accede y a posteriori pretende hacerlo la de mayor prioridad, ésta queda bloqueada hasta que aquélla abandona la zona compartida. Muchos diseñadores usan un ejecutivo cíclico para diagramar sus tareas en tiempo real. En el mismo las tareas son despachadas durante un quantum de

tiempo fijo, predefinido, en forma secuencial hasta completar la rueda [33]. Esto presenta algunos inconvenientes: primero, el programador debe usar herramientas adicionales para determinar los tiempos requeridos y la estructura a fijar en el programa, lo que generalmente lleva a cambiar varias veces el diseño para poder cumplir con las especificaciones temporales. Un segundo inconveniente es la fragilidad para la extensión o modificación futura de una tarea, lo que probablemente desemboca en una violación de la estructura de tiempos original [40]. Esto conduce a realizar el análisis utilizando un sistema determinístico, basado en prioridades fijas.

La viabilidad de diagramabilidad de un sistema de tiempo real por el método de ranuras vacías expuesto anteriormente considera que las m tareas comparten un único recurso (v.g. el procesador). En la práctica, sin embargo, lo más frecuente es que compartan otros recursos, e.g. hardware, datos o porciones de código, sobre los cuales deben utilizarse mecanismos de protección a fin de garantizar su consistencia e integridad. Son necesarios por lo tanto mecanismos de exclusión mutua (e.g. semáforos) que aseguren que las tareas accedan a una zona compartida en forma no simultánea. A este tipo de región se las denomina zonas críticas. Estas zonas no son apropiables, de modo que una vez que una tarea empezó a usar una de ellas no puede ser desplazada por otra tarea, aunque sea de mayor prioridad.

Todo lo anterior conduce a inversiones de prioridad que pueden ser largas en el tiempo y producir la caída del sistema por el no respeto de los

vencimientos. El ejemplo típico está dado por un sistema S(3) en el cual, en un instante dado, la tarea τ_0 ha completado un ciclo de ejecución. La tarea τ_2 que tiene ejecución pendiente accede entonces al procesador y a una zona de datos compartidos. Mientras se encuentra en ella, la tarea τ_0 debe ser ejecutada de nuevo y se apropia del procesador con lo cual se interrumpe la ejecución de τ_2. En ese momento despierta τ_1 que sólo usa el procesador pero no puede acceder a él por encontrarse en poder de τ_0. τ_0 requiere entonces el uso de los datos compartidos, pero su acceso está vedado porque τ_2 aún no los liberó, con lo cual la ejecución de τ_0 queda suspendida. τ_1 accede entonces al procesador y puede tenerlo por tiempo indefinido por cuanto tanto τ_0 como τ_2 tienen su ejecución suspendida. Se trata de una clara inversión de prioridades que puede ser tan larga como el tiempo de ejecución de τ_1. Una primera solución al problema consiste en negar la apropiación del procesador cuando una tarea de menor prioridad se encuentra en una zona crítica. Esto degrada la perfomance del sistema pues tareas en las que no se produce bloqueo pierden tiempo de ejecución.

Soluciones mejores son el protocolo de prioridad heredada (PPH) y el protocolo techo (PT) desarrollados en [53] a los cuales es aplicable el método de diagramación por ranuras vacías anteriormente descripto.

La idea básica del protocolo de prioridad heredada es que cuando una tarea bloquea otra u otras de mayor prioridad, durante el tiempo que dura el bloqueo su ejecución se realiza con la prioridad de la tarea de mayor

prioridad bloqueada. Se dice entonces que la tarea de menor prioridad heredó la prioridad mayor.

La expresión (2) debe modificarse para contemplar no sólo el tiempo de ejecución de τ_i y los posibles retardos por apropiaciones de tareas de mayor prioridad, sino también el tiempo que la tarea puede permanecer bloqueada por tareas de prioridad menor.

Como ya expresáramos un mecanismo de protección de las zonas críticas son los semáforos binarios. Los mismos, notados S, son cerrados o abiertos por operaciones atómicas P(S) y V(S) respectivamente. Cuando una tarea entra en una zona crítica, cierra el semáforo y sólo ella puede abrirlo, cosa que hace al abandonar la zona o bien al terminar su ejecución.

La porción de código de una tarea τ_i comprendida entre operaciones P(S) y V(S), notada $z_{i,j,k}$ representa la k-ésima sección crítica de la tarea τ_i controlada por el semáforo S_j , cuya duración es medida en términos de ranuras. En [53] Sha et al demostraron formalmente que con el protocolo de prioridades heredadas una tarea τ_i puede estar bloqueada a lo sumo por solo una sección critica de cada uno de los semáforos que comparte.

Como consecuencia de todo lo anterior, en el momento de la compilación es posible calcular el peor caso de bloqueo para cada tarea. Sin embargo, a pesar de ser mejor que el protocolo PMC puro, el protocolo de prioridades heredadas no evita las condiciones de bloqueo total o los bloqueos encadenados. El bloqueo total conduce a una paralización del sistema:

ocurre cuando una tarea está en posesión de un recurso que no puede liberar antes de apropiarse de otro; si éste está en poder de otra tarea que no puede liberarlo hasta no apropiarse del recurso en poder de la primera, se produce un bloqueo total, a veces denominado abrazo mortal, que paraliza el sistema. Los bloqueos encadenados se producen cuando una tarea de alta prioridad es bloqueada sucesivamente por las restantes tareas de menor prioridad. Aunque cada una de ellas pueda hacerlo sólo una vez, en el peor caso la tarea de mayor prioridad será bloqueada por la suma de las duraciones más largas de bloqueo de cada una de las tareas de menor prioridad, lo que es equivalente a incrementar su tiempo de ejecución en esa duración.

El protocolo techo extiende el protocolo de prioridades heredadas y corrige ambas deficiencias. La idea básica es asignar a cada semáforo la prioridad de la tarea de más alta prioridad que puede usarlo. Una tarea puede cerrar el semáforo que custodia una zona crítica si y solo si la prioridad de la tarea es mayor que la prioridad del semáforo cerrado de mayor prioridad.

En [47] se demuestra formalmente que el protocolo evita el bloqueo total y que cada tarea puede ser bloqueada a lo sumo durante la duración de una sola sección crítica. El análisis del peor caso se reduce entonces a determinar la sección crítica de mayor duración entre todas las que pueden bloquear esa tarea. En el caso de la tarea τ_i esa duración será notada β_i y corresponde al máximo bloqueo posible.

La inecuación (4) puede generalizarse para incorporar el tiempo que una tarea puede estar bloqueada por otras de menor prioridad.

El miembro de la derecha, además de tener en cuenta el propio tiempo de ejecución y el tiempo que puede perder por apropiaciones realizadas por tareas de mayor prioridad, incluye el tiempo que puede perder por inversiones de prioridad. En esencia, el sistema S(i-1) podrá ser expandido a S(i) por incorporación de τ_i si y solo si T_i es mayor que la $(C_i + \beta_i)$-ésima ranura que deja libre S(i-1). La expresión (4) se transforma entonces en:

$$\forall i \in \{2,...,m\} \quad \sum_{h=1}^{i-1} \frac{C_i}{T_h} < 1 \quad y$$
$$T_i \geq e_{(C_i + B_i)(i-1)} = menort \mid t = C_i + B_i + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil \qquad (5)$$

2.1.2 Una Alternativa al Método de las Ranuras Vacías

Se describe un método alternativo a los tradicionales que, en determinadas condiciones, reduce la complejidad de cálculo, permitiendo utilizar el mismo en problemas de asignación dinámica y estática en sistemas multitarea/multiprocesador[10] permitiendo esencialmente calcular el número de ranuras vacías en cualquier intervalo. En el mismo se adoptan las mismas hipótesis que en el método de las Ranuras Vacías [50].

A partir de la ecuación 4 se deduce que la última liberación de la tarea τ_i en el intervalo [1, t] estará dada por la expresión $\lfloor (t\text{-}1)/T_i \rfloor . T_i + 1$, donde $\lfloor (t\text{-}1)/T_i \rfloor$ determina el número de períodos T_i completos de la tarea τ_i en el intervalo [1, t], donde $\lfloor \ \rfloor$ representa el operador piso. Al

ser las tareas periódicas, cada liberación de τ_i se produce inmediatamente después de finalizado un período. Siendo $\lfloor (t\text{-}1)/T_i \rfloor . T_i$ un número entero de períodos de τ_i en [1, t], la próxima liberación de dicha tarea será en la ranura $\lfloor (t\text{-}1)/T_i \rfloor . T_i + 1$. Si la tarea τ_i es diagramable, en el intervalo [1, $\lfloor (t\text{-}1)/T_i \rfloor . T_i$] debe ejecutarse C_i ranuras por período T_i. Luego, en $\lfloor (t\text{-}1)/T_i \rfloor$ períodos, la tarea τ_j utilizará $\lfloor (t\text{-}1)/T_i \rfloor . C_i$ ranuras para su ejecución.

La cantidad de ranuras libres que tendrá la tarea τ_i, en el intervalo [$\lfloor (t\text{-}1)/T_i \rfloor . T_i + 1$, t], simbolizada $RL(\tau_i, t)$, es:

$$RL(\tau_i, t) = t - \left\lfloor \frac{t-1}{T_i} \right\rfloor \cdot T_i - \sum_{\forall h \mid \tau_h \in \mathbf{S}(i-1)} \left[R(\tau_h, t) - R(\tau_h, \left\lfloor \frac{(t-1)}{T_i} \right\rfloor \cdot T_i) \right] \quad (6)$$

donde $R(\tau_h, t)$ simboliza la cantidad de ranuras utilizadas por la tarea τ_h en el intervalo [1, t].

El número de ranuras libres que deja el sistema $\mathbf{S}(m)$ en el intervalo [1, t], simbolizado $E(\mathbf{S}(m), t)$ es:

$$E(\mathbf{S}(m), t) = t - \sum_{\tau_x \in \mathbf{S}(m)} R(\tau_x, t) \quad (7)$$

El número de ranuras libres que deja el sistema $\mathbf{S}(m)$ en el intervalo [1, t] es el número total de ranuras menos la cantidad de ranuras que ocupan las tareas que pertenecen a dicho sistema. Luego, el sistema $\mathbf{S}(m)$ es diagramable si:

$$\forall \tau_i \in \mathbf{S}(m) \qquad T_i - \sum_{\forall x \mid \tau_x \in \mathbf{S}(i-1)} R(\tau_x, T_i) \geq C_i \quad (8)$$

Si toda tarea τ_i integrante del sistema **S**(m), tiene por lo menos C_i ranuras libres en el intervalo [1, T_i], podrá ejecutarse completamente sin entrar en crisis.

En el cálculo de E(**S**(m), t) el conjunto de tareas está ordenado por prioridades crecientes. La tarea de mayor prioridad en consecuencia será τ_1, y la de menor prioridad τ_m.

Para calcular la diagramabilidad del sistema **S**(m) se debe evaluar la función 7 para cada tarea. En [10] se demuestra que la complejidad de este algoritmo (entendida como su costo computacional [20]) es $O(2^m-1)$. Como la complejidad del algoritmo de las ranuras vacías es $O(mT_m)$ [10], el método alternativo puede resultar mejor en casos de m chicos y T_m grandes.

2.2 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 que 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 réplica no podrán coexistir en un mismo procesador. A las tareas que no tienen procesador definido se las llama libres y pueden ser alojadas en cualquier procesador.

2.3 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:

$$\sum_{1}^{j} M_h \leq M_p \qquad (10)$$

donde M_h es la memoria requerida por cada una de las j tarea y M_p es la memoria disponible en el procesador *p*.

2.4 Restricciones de Comunicaciones

Si el trabajo lo requiere, las tareas pueden transferirse información entre sí. En algunos casos los datos fluyen de una tarea a otra a través de un canal de comunicaciones. Si las mismas se encuentran en distintos procesadores, es necesaria una red de interconexión. Generalmente esta función es realizada por una red local la cual debe operar también con restricciones de tiempo real. Si las tareas se encuentran en el mismo procesador, la comunicación es realizada utilizando un mecanismo interno al procesador y no cargan la red. Aunque el tráfico entre procesadores es siempre el costo mas alto y el menos fiable no siempre es considerado como un factor importante durante el proceso de asignación. Cuando un mensaje es generado en un nodo debe transcurrir un cierto tiempo hasta que el nodo obtiene el acceso al medio de transmisión.

Luego el mensaje debe ser transmitido y propagado hasta la estación destino. El mensaje consiste en una cantidad de bytes de sobrecarga (preámbulo, delimitadores, direcciones, control, detección de errores, etc.) y un campo efectivo para la transferencia de datos. Producto de la sobrecarga, el acceso y la propagación el ancho de banda para la transferencia de datos es solamente una fracción del ancho de banda especificado en el nivel físico del protocolo. Por ejemplo una red con 1Mb/s (125 bytes/ms) de ancho de banda puede tener solamente un ancho de banda efectivo para transferencia de datos de 0.8 Mb/s (100 bytes/ms).

Si utilizamos una red que implementa un mecanismo de acceso utilizando una disciplina de Rueda Cíclica, por ejemplo 802.5 o FDDI con igual prioridad en todos sus mensaje, la misma es diagramable si cumple con:

$$\sum_{i=1}^{m} B_i \leq AB * D_{min} \qquad (11)$$

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:

$$\Delta = \left\lceil \frac{\sum_{i=1}^{m} B_i}{AB} \right\rceil \qquad (12)$$

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

El *factor de utilización de la red* es definido como la relación entre el número de bytes actualmente transmitidos y el número máximo de bytes que es posible transmitir de acuerdo al ancho de banda efectivo del canal. Es fácil concluir que para una asignación determinada de tareas a procesadores este factor decrece cuando la velocidad de la red aumenta.

La *carga relativa de la red* es definida como la relación entre el número de bytes realmente transmitido a través de la red y el número total de bytes intercambiados por las tareas antes de producir la asignación. Una carga relativa baja indica una buena agrupación en idénticos procesadores de tareas que intercambian datos.

2.5 Restricciones de Precedencia

Tareas relacionadas con restricciones de precedencia son comúnmente encontradas en complejos sistemas de tiempo real. Esta relación proviene de restricciones externas impuestas por las características físicas de los procesos a controlar y de las condiciones impuestas por el diseño modular de estas aplicaciones. Las imposiciones por precedencia son usualmente presentadas en forma de grafos acíclicos y dirigidos. Para un número importante de sistemas, es imprescindible que las tareas colaboren entre sí para alcanzar un objetivo predeterminado. En un diseño modular del software la tendencia es crear grupos de tareas seudo-independientes, bien definidas e interrelacionadas solamente a través de restricciones de precedencia pre-especificadas. Estas tareas seudo-independientes ejecutan fuera de su espacio de direcciones y

contexto y son diagramadas como parte de un grupo de tareas que conforman un determinado trabajo, cuyo único interés es cumplir con la precedencia. Por ejemplo un grupo de tres tareas que realizan la adquisición, la visualización y el almacenamiento de datos, con la única restricción que la adquisición debe terminar antes de que las otras dos tareas arranquen, podemos deducir fácilmente que desde el punto de vista de la diagramación este trabajo no presenta otra restricción que no sea la precedencia y que es convenientes que se lo ejecute en un sistema de procesadores distribuidos. El paralelismo inherente de este trabajo puede ser explotado, siempre y cuando existan procesadores disponibles. Además del paralelismo, el diseño modular tiene otras características importantes. Por ejemplo, en ciertas ocasiones el trabajo puede ser parcialmente ejecutado y de existir un error no es necesario volver a ejecutar todo el trabajo. El trabajo que permite esta ejecución parcial de las tareas es denominado *trabajo no atómico* .

Por otro lado, los *trabajos atómicos* demandan que el grupo de tareas termine su ejecución. Este tipo de trabajos requiere que las tareas sean diagramadas de un extremo a otro del grafo y no pueden ser rediagramadas sin rediagramar todo el trabajo en forma completa. Si alguna tarea es abortada, todo el trabajo debe ejecutarse nuevamente. Este tipo de trabajos es utilizado en procesos monolíticos, los cuales son separados en tareas para mejorar la utilización de los recursos disponibles. En contraste con las *trabajos no-atómicos* este grupo de tareas no es fácilmente diagramable en procesadores distribuidos, debido a los efectos de la separación. Este tipo de tareas son comúnmente encontradas

en sistemas críticos de tiempo real y es una parte obligatoria de la computación imprecisa. En contraste con los *trabajos no-atómicos, los atómicos* no son fácilmente diagramables en procesadores separados, principalmente por los efectos de la fragmentación del trabajo, originado por tareas que se ejecutan en un espacio de direcciones y de contexto que no es el definido por el *trabajo atómico*.

Son claras las diferencias entre trabajos *atómicos* y *no-atómicos*. Desde el punto de vista de la diagramación se ve claramente que los *no-atómicos* son inherentemente distribuidos y flexibles para diagramar, pues no imponen otra constricción que no sea la precedencia.

Al definir una relación de precedencia se incorpora complejidad tanto para la asignación como para la diagramación. Se han desarrollado algoritmos polinómicos obteniendo resultados para subclases de la relación de precedencia. Estas subclases de precedencia están compuestas por cadenas de tareas, las cuales tienen un solo predecesor y un solo sucesor, excepto en el principio y final de la misma. En [13] define una técnica para modificar vencimientos y diagramar múltiples trabajos en un entorno monoprocesador utilizando Menor Tiempo al Vencimiento (EDF, Earlier Deadline First). Aunque no existe una diferenciación explicita, la literatura actual trata con dos tipos de precedencias que se denominan *Precedencia-Blanda* y *Precedencia-Dura* respectivamente.

2.5.1 Precedencia Blanda

Es el tipo de restricción utilizada en Tindell [58]. En este caso, cuando las tareas se ejecutan en diferentes procesadores, es tácitamente asumido que en el arranque del sistema, la iniciación de las tareas sucesoras debe ser diferido un periodo respecto de la predecesora. Con este desfasaje inicial, el peor caso ocurre cuando el último bit del mensaje es recibido al final del periodo del predecesor y el sucesor es la primera tarea a ser ejecutada en el procesador correspondiente en el próximo período, según ilustra la figura 4. A partir de este punto ambas tareas son ejecutadas periódicamente en forma independiente del cumplimiento de sus respectivos vencimientos.

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 (12):

$$D_p^* = D_p - \Delta \quad (13)$$

La iniciación de la sucesora es manejada por eventos y es activada cuando el último byte transferido llega al nodo. En este caso el sucesor y el predecesor pueden ser vistos como tareas independientes.

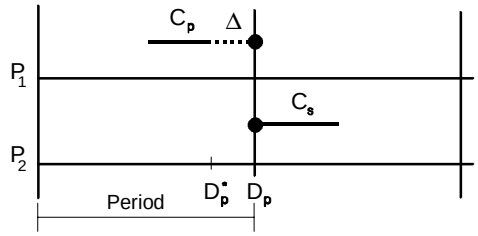

Figura 4

Están blandamente acopladas debido a que una requiere sólo datos producidos por la otra y no puede ser considerada como parte integral de un trabajo con el mismo vencimiento total.

2.5.2 Precedencia Dura

Si las tareas que se comunican son consideradas como parte del trabajo con un período igual para todas las tareas y el desfasaje inicial no esta permitido, y suponiendo que las tareas pueden arrancar en cualquier momento, el análisis a realizar resulta mas complejo. Para que el trabajo sea diagramable, todas las tareas que lo forman deben ser ejecutadas dentro del periodo. Este tipo de precedencia se lo denomina "*Duras*" y son tratadas en [11], donde se analiza la diagramación de tareas apropiativas y cooperativas en un ambiente distribuido Multitarea-Multiprocesador. El peor estado de carga para un sistema Monoprocesador con tareas independientes es el arribo simultáneo de todas las tareas [50]. Sin embargo, en los casos de sistemas multiprocesador, cuando las tareas cooperan en su ejecución al formar parte de un mismo trabajo y son asignadas a distintos procesadores, la verificación local de la diagramabilidad puede encontrar el inconveniente de que los tiempos entre arribos de una tarea no conservan. Entre los pares de tareas con relación de precedencia que se transfieren datos, la llegada de los mismo al nodo donde se aloja la sucesora define el instante de arribo de la tarea sucesora al contexto de diagramación local. En este caso, los tiempos de arribo no

sólo no son conocidos con exactitud sino que pueden llegar a ser menores que el período mínimo del trabajo al cual pertenecen. En consecuencia, no son aplicables los métodos de análisis de la diagramación para sistemas monoprocesadores en forma directa, por lo cual el método de las ranuras vacías es modificado para obtener un diagramador dinámico local a cada nodo, estableciéndose las condiciones de suficiencia para que el sistema sea factible mediante este mecanismo. El mismo define el máximo tiempo de respuesta de la tarea (MTR) y un contador asociado a cada una de ellas. El contenido de este último, y de acuerdo a regla predefinidas, permite al diagramador activar o no las tareas que esperaban la llegada de datos desde su predecesora. Este mecanismo trabaja sobre una asignación de tareas a procesadores preestablecida por alguno de los métodos aquí desarrollados. Para todos los métodos de asignación propuestos en este trabajo, se utilizara para permitir la comparación con otros métodos, las precedencias blandas.

Capítulo 3

1 Heurística de Asignación

Se presenta en este capítulo una heurística de asignación, constructiva y con regla de parada bien definida [51] para resolver el problema de asignación de un conjunto de tareas de tiempo real sobre un conjunto de procesadores, con diversas restricciones. Los argumentos racionales que la respaldan se exponen en la página 59. Luego se describen los pasos que la constituyen, diseñados sobre la base de dichos argumentos. Se consignan los resultados experimentales al aplicarlos a problemas cuyos datos fueron extraídos de trabajos similares publicados por otros investigadores. El método, por supuesto, no tendría mayor valor si no mejorara en alguna forma resultados previos. Se muestra finalmente, que esto es así para el tipo de problemas estudiado.

Un sistema se dice absolutamente no asignable si ninguna de las n^m posibles asignaciones (donde n y m denotan el número de procesadores y de tareas, respectivamente) cumple las restricciones enumeradas en el capítulo anterior. La verificación de ubicación, tiempo, memoria y comunicaciones es denominado test de asignación. Si las tareas pasan el test de asignación, se verificada a continuación la restricción de precedencia sobre el conjunto de tareas para validar una asignación tentativa.

Para el problema de asignación con restricciones temporales los métodos heurísticos constituyen uno de los pocos enfoques válidos para atacar el problema. La utilización de algoritmos heurísticos son interesantes como enfoque cuando se dan algunas de las siguientes circunstancias: a) No existe un método exacto de resolución, b) Los métodos exactos requieren mucho tiempo de cálculo o memoria, c) Es aceptable una solución subóptima o no optima, d) Las limitaciones de tiempo y espacio conducen a aceptar métodos de rápida respuesta, aun a costa de la precisión.

Los aspectos racionales del método base propuesto en esta tesis, para asignar tareas en un entorno de tiempo real, pueden definirse de la siguiente manera:

1. Primero se ubican las tareas preasignadas. De esta manera es posible incorporar durante el proceso de asignación tareas vinculadas en el mismo procesador. Una detección temprana de un sistema absolutamente no diagramable por definición permite evitar tiempos de búsqueda estériles.
2. Asignar al mismo procesador tareas que se comunican entre si, ubicando primero los pares de tareas que mayor tráfico intercambian. De esta forma las restricciones de precedencia pueden ser más fácilmente superadas y la carga

sobre la red disminuye notablemente lo cual permite mejorar aun más las chances de cumplir con las restricciones temporales. En un sistema de tiempo real una determinada asignación será solución al problema sólo si verifica las condiciones de diagramabilidad. Luego la factibilidad de dicha asignación dependerá de que se satisfaga la diagramabilidad de la red de comunicaciones y de las tareas en los procesadores. En el dominio temporal dichas restricciones se encuentran fuertemente vinculadas. Reducir la carga sobre la red en cada instancia de la asignación no sólo tiende a mejorar la diagramabilidad de las comunicaciones sino que además reduce los retardos de comunicación predecesor/sucesor y por ende, produce un aumento proporcional de los vencimientos corregidos por precedencia de las tareas comunicantes. Esto último facilita la diagramación de los trabajos intervinientes en el problema. Por lo tanto, la búsqueda de una solución deberá orientarse a las particiones que minimicen el tráfico en la red y no a aquellas que resulten óptimas según criterios usuales en sistemas que no son de tiempo real como el balance de carga en la red, balance de carga en los procesadores.

3. El conjunto de tareas remanentes es asignado de acuerdo al factor de utilización, enviando las de mayor coeficiente a los procesadores más descargados. De esta manera las tareas más críticas que no fueron asignadas en la primer pasada, tienen mas probabilidad de ser asignadas.

En el algoritmo propuesto, los conjuntos de tareas, procesadores y pares de tareas comunicantes se denominan pila de tareas, procesadores y comunicaciones respectivamente.

Los pasos del método desarrollado son detallados a continuación:

Paso 1: El test de asignación es ejecutado sobre todas las tareas preasignadas. Si alguno de estos tests falla se dice que el sistema es absolutamente no asignable, si no sigue el paso 2.

Paso 2: La pila de procesadores es ensamblada mediante un ordenamiento aleatorio de los procesadores.

Paso 3: La pila de comunicaciones es ensamblada ordenando los pares de tareas comunicantes por valores decrecientes de sus tiempos de comunicación. La pila de tareas es ordenada por valores decrecientes de su factor de utilización.

Paso 4: Para cada procesador 1,2,…,n de la pila de procesadores, la pila de comunicaciones es inspeccionada desde arriba hacia abajo hasta encontrar el primer par que contiene una tarea que esta asignada al procesador. El test de asignación es llevado a cabo sobre la tarea compañera; si el test pasa, la tarea es asignada al procesador y el par es borrado de la pila de comunicaciones. Cada vez que una tarea es asignada, se repite la encuesta sobre la pila de comunicaciones y la tarea compañera es también evaluada. Este procedimiento es realizado hasta que todas las candidatas son asignadas o hasta encontrar que no es posible incorporar una tarea mas al procesador.

Después de alcanzar el último procesador queda remanente sólo un subconjunto de tareas libres (no asignadas).

Paso 5: La pila de procesadores es reordenada por valores crecientes de su factor de utilización. El empate entre elementos con igual valor es roto de manera aleatoria. El efecto combinado de ordenar procesadores y tareas tiene un doble efecto: tareas que quedaron libres con requerimientos de tiempo fuertes, son tratadas sobre los

procesadores mas descargados, facilitando la diagramación. Además, la posibilidad de incorporar una tarea comunicante es mejorada sustancialmente.

Paso 6: El test de asignación es realizado sobre la primer tarea libre para asignarla a algun procesador. Si la asignación no es posible, sigue el paso 9, si no la tarea es borrada de la pila y sigue el paso 7.

Paso 7: La pila de comunicaciones remanente es encuestada desde arriba hacia abajo hasta encontrar el primer par que contiene una tarea que esta asignada al procesador bajo consideración. El test de asignación es llevado a cabo sobre la tarea compañera; si el test pasa, la tarea es asignada al procesador y el par es borrado de la pila de comunicaciones. Cada vez que una tarea es asignada, se repite la encuesta sobre la pila de comunicaciones y la tarea compañera es también evaluada. Si la pila de tareas no está vacía, sigue el paso 5.

Los tests son realizados en forma fácil y rápida a partir de cálculos anteriores. Por ejemplo la información acerca de la memoria residual en cada procesador, la carga total de la red de comunicaciones, el número de ranuras libres en un intervalo, son conocidos.

Cuando la pila de tareas esta vacía, se ha obtenido una asignación tentativa. Debe hacerse notar claramente que las comunicaciones tienen una influencia negativa sobre las restricciones de precedencia. Puede suceder que los requerimiento temporales sobre la red han sido satisfechos, pero al sumar los retardos de comunicaciones particulares de la asignación, se caiga en una violación de las restricciones de precedencia. Esto puede suceder cuando se verifican las precedencias sobre la asignación tentativa en el paso 9.

Paso 8: La diagramación de cada procesador que contienen tareas que envían mensajes a otras, es reverificada con los vencimientos corregidos de acuerdo a la ecuación 13 que evalúa las precedencias blandas. Si el test es satisfecho, se ha encontrado una solución al problema de asignación. Si esto no es verdadero, el sistema se dice no diagramable para esa permutación particular de la pila de procesadores.

Paso 9: Una nueva permutación a la pila de procesadores es generada y el proceso repetido desde el paso 3. Si todas las n! permutaciones de la pila de procesadores han sido tratadas sin éxito, el sistema se dice absolutamente no asignable por este método.

Nótese que la naturaleza de los problemas aquí tratados (cada tarea ejecutándose completamente en un procesador y tareas estructuradas secuencialmente constituyendo un trabajo) favorece, en la búsqueda de una solución óptima, la tendencia a agrupar tareas que se comunican en un mismo procesador. La reducción de los retardos de comunicaciones tiene un doble efecto: se pueden resolver más fácilmente las restricciones de comunicaciones y las restricciones de precedencia. Dado que la búsqueda se aleja de soluciones no óptimas o no cercanas a la óptima, para algunos problemas el número de soluciones diferentes logradas con el método será bajo.

Un criterio de optimización es reducir el volumen de comunicación de tiempo real vía red. Esto deja ancho de banda disponible para comunicaciones que no son de tiempo real, con lo cual la calidad de este servicio mejora por disminución de retardos en los tiempos de respuesta. Para reducir el volumen de comunicaciones, el método tiende a formar constelaciones de tareas con restricciones de precedencia, y por ende con carga

de comunicaciones, en un mismo procesador. En este sentido la asignación tiende a ser óptima o cercana a la óptima. Puede haber casos en que por disponer de ancho de banda elevado el sistema tolere una dispersión grande de tareas en distintos procesadores. Sin embargo, por estar estas soluciones alejadas de la óptima o cercanas a la óptima el método no las va a determinar. Esto se comprueba experimentalmente en el ejemplo de la página 63 y siguientes.

A continuación se describe el algoritmo desarrollado, utilizando un seudo-código.

```
Program Heuristic_ Algorithm (tasks, processors, communications)
begin
  If Allocability_test(processors ← pre-allocated tasks) =TRUE Then
  Begin
    Assemble _Processor-stack_RND              ; randomly ordering
    While all permutations have not been tried do
    begin
    Assemble_Task-stack_DUF             ; decreasing utilization factor
    Assemble _Communication-stack_DCT          ; decreasing communication
    times
    For each processor ∈ processor_stack following stack order
    begin
      While ∃ task_pair:= first pair ((τa, companion_task) or (companion_task,
      τa))∈ communication-stack | τa ∈ {tasks allocated to processor} and
      Allocability_test(processor ← companion_task)=TRUE do
      begin
      Allocate(processor ← companion_task)
      Delete_pair(task_pair)
      end
    end
    Sort_Processor-stack_IUF                   ; increasing utilization
    factor
    While ∃ proc_place:= first processor ∈ processor-stack |
    Allocability_test(processor ← first task in the task-stack)=TRUE do
    begin
      Allocate(proc_place ←first task in the task-stack)
      Delete_first_task(tasks-stack)
```

```
      While ∃ task_pair:= first pair ((τa, companion_task) or (companion_task,
      τa))∈ communication-stack | τa ∈ {tasks allocated to proc_place } and
      Allocability_test(proc_place ←companion_task)=TRUE do
        begin
        Allocate(proc_place ←companion_task)
        Delete_pair(task_pair)
        end
      Sort_Processor-stack_IUF
    end
    If task-stack = ∅ and Precedence_test(processors)=TRUE then solution++
    Permutation(processor-stack)
    end
    Return(the system is not schedulable for this method)
  end
  else Return(the system is absolutely non schedulable)
end
```

Ejemplo: La figura 6 muestra un conjunto de 6 grafos representando tres trabajos de dos tareas cada uno y tres trabajos de cuatro, cinco y seis tareas, respectivamente. Los trabajos deben ser ejecutados en 6 procesadores interconectados mediante una red local de 1 Mb/s.

Tarea	Periodo	Vencimiento	FU	Memoria	Ubicación
0	46	46	0.196	300	
1	46	46	0.196	150	
2	46	46	0.196	120	0
3	46	46	0.196	170	
4	92	92	0.098	300	
5	92	92	0.098	300	1
6	92	92	0.098	110	
7	92	92	0.098	50	2
8	92	92	0.098	70	
9	92	92	0.098	90	
10	92	92	0.098	220	
11	92	92	0.098	100	
12	138	138	0.065	100	
13	138	138	0.065	150	
14	138	138	0.065	160	3
15	138	138	0.065	130	4
16	138	138	0.065	110	

17	138	138	0.065	100	
18	138	138	0.065	100	5
19	138	138	0.065	160	
20	138	138	0.065	190	
21	138	138	0.065	200	
22	138	138	0.065	100	1
23	138	138	0.065	200	

Tabla 1

En la tabla 1 se muestran los factores de utilización, requerimientos de memoria y preasignaciones. Como se utilizará sólo con fines ilustrativos, el problema fue simplificado manteniendo factores de utilización bajos y eliminando restricciones de ubicación debidas a tolerancia a fallas. Cada una de las 720 permutaciones conducen a una solución. Pese a esto sólo tres de ellas eran diferentes.

Proc	**Tareas**		
	Solución 1	**Solución 2**	**Solución 3**
0	**2**; 3	**2**; 3	**2**; 3
1	**5**; 16; 17; 19; 20; 21; **22**; 23	**5**; 16; 17; 19; 20; 21; **22**; 23	4; **5**; **22**
2	6; **7**; 8; 9	6; **7**; 8; 9	6; **7**; 8; 9
3	10; 11; 12; 13; **14**	10; 11; 12; 13; **14**	10; 11; 12; 13; **14**
4	0; 1; **15**	4; **15**	0; 1; **15**
5	4; **18**	0; 1; **18**	16; 17; **18**; 19; 20; 21; 23

Tabla 2

Las soluciones se muestran en la tabla 2. La solución 2 será utilizada como ejemplo para una ejecución, mostrando los sucesivos direccionamientos, la evolución de las pilas, etc., para cada paso del algoritmo.

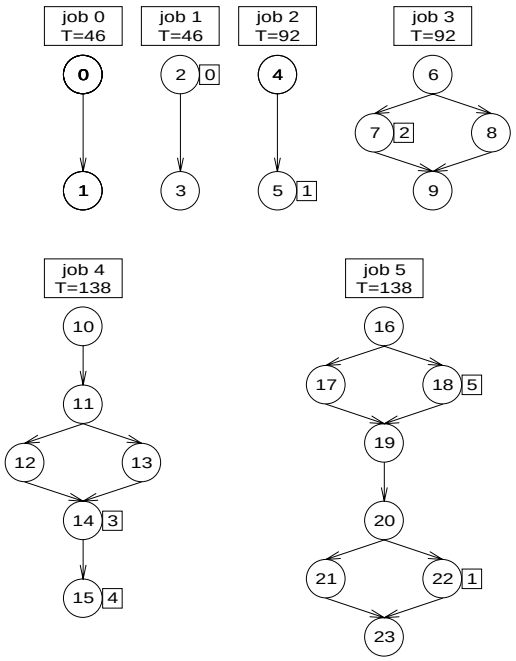

Figura 6

Paso 1: El test de asignación es efectuado sobre las tareas 2,5,7,14,15 y 18, preasignadas a los procesadores 0,1,2,3,4 y 5 respectivamente y la tarea 22 preasignada al procesador 1. Como ninguno de los tests falla, las asignaciones son definitivas y pasamos al paso 2.

Paso 2: La pila de procesadores, obtenida por la permutación de la primera pila, es <0, 1, 2, 3,5, 4>.

Paso 3: La pila de comunicaciones es <11/13, 20/21, 21/23, 11/12, 19/20, 13/14, 18/19, 6/7, 0/1, 12/14, 8/9, 10/11, 14/15, 22/23, 20/22, 17/19, 7/9, 16/17, 16/18, 2/3, 4/5, 6/8>.

Paso 4: De acuerdo a la pila de procesadores el procesador 0 es considerado primero. El test de asignación efectuado en la tarea 3, compañera de la tarea 2 ya preasignada, permite su incorporación. El par 2/3 es eliminado de la pila de comunicaciones. Como ésta ya no contiene ninguna tarea asignada al procesador 0, pasa a consideración el

procesador 1. Los tests de asignación efectuados sobre las tareas 23 y 20, compañeras de la tarea 22, permiten su incorporación, por lo que los pares 20/22 y 22/23 son eliminados de la pila de comunicaciones y se efectúa el test sobre la tarea 21, compañera de las tareas 20 y 23. Esta tarea puede ser asignada y los pares 20/21 y 21/23 son consecuentemente eliminados de la pila de comunicaciones. Tests sucesivos asignan las tareas 19, 17 y 16, en ese orden, y se eliminan los pares 19/20, 17/19 y 16/17. Al intentar asignar la tarea 4, compañera de la tarea preasignada 5 en el penúltimo par, encontramos que no es posible asignarla porque la memoria remanente en el procesador no es suficiente. Siguiendo el procedimiento, las tareas 6, 9 y 8 se incorporan en el procesador 2 y las tareas 3, 11, 12 y 10 al procesador 3, eliminando en consecuencia los pares 6/7, 8/9, 6/8, 11/13, 11/12, 13/14, 12/14 y 10/11. Las tareas compañeras de las tareas 15 y 18, preasignadas a los procesadores 4 y 5 respectivamente, ya están asignadas. Al finalizar este paso, el conjunto de tareas libres es {0, 1, 4}.

Paso 5: Los procesadores 4 y 5 tienen el mismo factor de utilización. El empate se rompe aleatoriamente y la pila de procesadores resulta ser <5, 4, 0, 1, 2, 3> . La pila de tareas es <0, 1, 4>, con un ordenamiento aleatorio de las tareas 0 y 1, que tienen el mismo factor de utilización.

Paso 6: La tarea 0 es asignada al procesador 5.

Paso 7: La pila de comunicaciones es ahora <18/19, 0/1, 14/15, 16/18, 4/5>. El primer par que contiene una tarea asignada al procesador considerado es 0/1. El test de asignación aplicado a la tarea 1 permite su incorporación. El par 0/1 es eliminado de

la pila. Como los pares restantes no contienen tareas asociadas a tareas asignadas al procesador 5, proseguimos con el paso 5.

Paso 5: Las pilas de procesadores y tareas son ahora, <4, 5, 0, 2, 1, 3> y <4>, respectivamente.

Paso 6: La tarea 4 puede ser asignada al procesador 4.

Paso 7: La pila de tareas ahora está vacía. Las tareas comunicándose por medio de la red local son {18/19, 14/15, 16/18 y 4/5}. El proceso está completo y se encontró una solución tentativa.

Paso 6: El test muestra que se puede satisfacer las precedencias de tiempo. Por lo tanto, la solución es válida.

Muchas otras soluciones son posibles. El problema es tan sencillo que muchas de ellas pueden ser encontradas incluso manualmente. El método, sin embargo, ofrece sólo tres; esto se debe a que optimiza las comunicaciones, con un agrupamiento efectivo en un mismo procesador de las tareas que se comunican entre sí. Las comunicaciones por medio de la red local se limitan a los pares 4/5, 14/15, 16/18 y 18/19 en las primeras dos soluciones y a los pares 14/15,20/22 y 22/23 en la tercera.

Debido a que las tareas 14 y 15 son preasignadas a distintos procesadores, deben comunicarse por la red local. La tarea 22 es preasignada al procesador 1 y, debido a comunicaciones transitivas, las tareas 16, 17, 19, 20, 21 y 23 son asignadas al mismo procesador que las tareas 1 y 2. Como las tareas 18 y 22 están preasignadas a diferentes procesadores, se deduce que las comunicaciones 16/18 y 18/19 deben necesariamente realizarse por medio de la red local. En la tercera solución las tareas 16, 17, 19, 20, 21 y 23 son asignadas, por comunicaciones transitivas, al procesador

5, el cual ya tiene preasignada la tarea 18. Al estar las tareas 18 y 22 preasignadas a diferentes procesadores, se sigue que las comunicaciones 20/22 y 20/23 sólo pueden efectuarse por intermedio de la red local.

En las primeras dos soluciones la comunicación entre las tareas 4 y 5 también aparece por intermedio de la red. Esto es porque la pila inicial de procesadores es <0, 1, 2, 3, 4, 5> en el primer caso y <0, 1, 2, 3, 5, 4> en el segundo. Las tareas 5 y 22, preasignadas al procesador 1 son asignadas definitivamente en el paso 1 del método. En el paso 4, las tareas 16, 17, 19, 20, 21 y 23 son asignadas al mismo procesador por tener una carga de comunicaciones mayor que el par 4/5. Al intentar asignar la tarea 4 al procesador 1 el método encuentra que no hay suficiente memoria disponible y, en consecuencia, no puede efectuar la asignación. En el paso 6, la tarea 4 es asignada a los procesadores 5 y 4 en la primera y segunda solución respectivamente, porque son los menos cargados. En la tercera solución la pila inicial de procesadores es <0, 2, 3, 4, 5, 1, >. Todos los procesadores son llenados, en ese orden, en los pasos 1 a 4. La mayor parte del trabajo 5 es asignada al procesador 5, el penúltimo en ser considerado. Finalmente, la tarea pendiente, 4, es asignada al procesador 1, el último en ser considerado.

El ejemplo muestra lo que se podía esperar: siendo guiadas activamente hasta soluciones óptimas o casi óptimas, el método no produce soluciones que, aunque posibles, disten de ser óptimas. De ahí el pequeño número de soluciones (0,41% del total de permutaciones ensayadas). Por otro lado, otros problemas pueden producir un gran número de soluciones diferentes. En algunos casos, pueden hallarse n! soluciones

o bien ninguna. Por ejemplo, tomemos el mismo problema anterior, pero sin preasignaciones. La primera solución se presenta en la tabla 3.

Procesador	Tareas
0	0; 1
1	2; 3
2	4; 5
3	6; 7; 8; 9
4	10; 11; 12; 13; 14; 15
5	16; 17; 18; 19; 20; 21; 22; 23

Tabla 3

Fue obtenida para la primera pila de procesadores (<0, 1, 2, 3, 4, 5>). Es obvio que desde que no se impusieron preasignaciones y todos los procesadores se suponen iguales, existen n!=720 soluciones diferentes, las que son fácilmente obtenibles manteniendo invariables las filas de las tareas en la tabla y colocando en la columna de procesadores todas las posibles permutaciones de los procesadores (<0, 1, 2, 5,3,4>, < 0, 1, 2, 4, 3, 5>, etc.) Si todas las tareas son preasignadas podría haber una sola solución o ninguna.

2 La perfomance del método: Resultados y análisis

2.1 Número de soluciones vs. APUF para diferentes NBWs.

La perfomance del método fue determinada para diferentes factores de utilización promedio de la red (Average Processor's Utilization Factor, APUF) y anchos de banda de la red (Network Bandwidth, NBW). APUF es, por supuesto, la suma de los factores de utilización de todas las tareas dividida por el número de procesadores. Nótese que Liu and Layland [39] encontraron el techo teórico para el factor de utilización $U \leq m(2^{1/m} -1)$ para el caso de m

tareas independientes diagramables en un único procesador. Este valor fue incrementado posteriormente por [37, 35, 50) y puede alcanzar la unidad.

Para determinar la perfomance del método, se realizaron tests sistemáticos en 920 problemas basados en tres familias de grafos, cada grafo representando un trabajo. El número de tareas iniciales (sin predecesor) de cada trabajo y el número de sucesores para cada tarea, se determina aleatoriamente entre 1 y 3 con probabilidades decrecientes en ese orden. Todas las tareas de un trabajo tienen por lo menos un sucesor, excepto la última.

Seis tareas (el 25% del total) se seleccionan aleatoriamente para ser preasignadas a alguno de los tres primeros procesadores. Dos sucesoras a una predecesora se seleccionan al azar para ser ejecutadas en diferentes

procesadores, para hacerlas tolerantes a fallas. La longitud de los mensajes a transmitir entre tareas que se comuniquen es asignada aleatoriamente para cada par en el rango de 50 a 450 bytes. Se suponen restricciones de precedencia blandas.

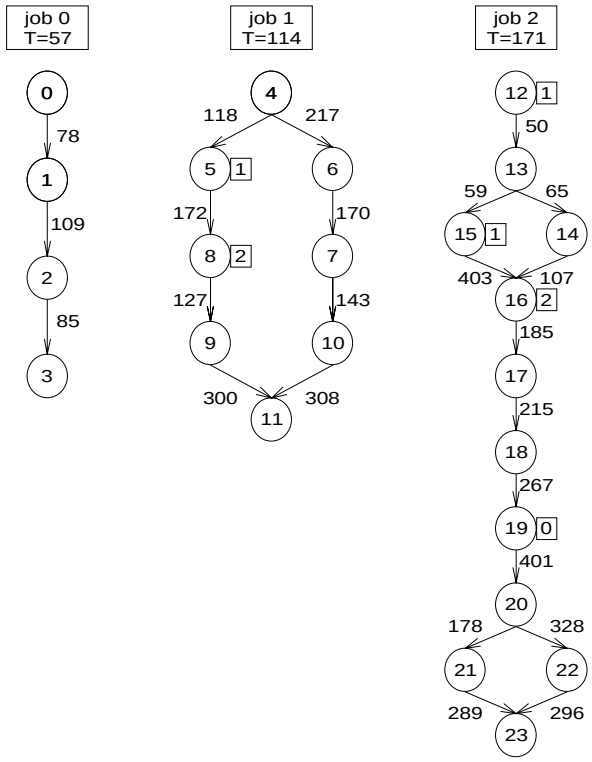

Figura 7

Los grafos pertenecientes a la primera, segunda y tercera familias tienen 4, 8 y 12 nodos respectivamente. Cinco conjuntos subyacentes de tres grafos, representando tres trabajos, se generan por instanciación de un grafo de cada familia. Las únicas excepciones a estas reglas están en el primer trabajo en el primer conjunto de grafos y en el quinto conjunto de grafos. En ambos casos los grafos fueron forzados a ser cadenas, con el objeto de endurecer las restricciones de precedencia. Los cinco conjuntos se describen en las figuras 7 a 11 poniendo de manifiesto períodos, requerimientos de comunicaciones y preasignaciones (el procesador preasignado se indica por un número dentro de un cuadrado a la derecha del nodo). Como se puede ver, los períodos del segundo y tercer trabajos son del doble y el triple del período del primer

trabajo, respectivamente. El período de generación es común a todas las tareas de un trabajo. Los tiempos de ejecución se ajustan para producir APUFs entre 0.1 y 0.8 en pasos de 0.1. El NBW varía entre 0.1 y 100 Mbps, produciendo un ancho de banda útil de 10 y 10000 bytes/ms. Se utilizó un factor de utilización promedio de memoria de 0.5. Variando los tiempos de ejecución se obtienen distintos factores de utilización de los trabajos, que darán como resultado diferentes APUFs.

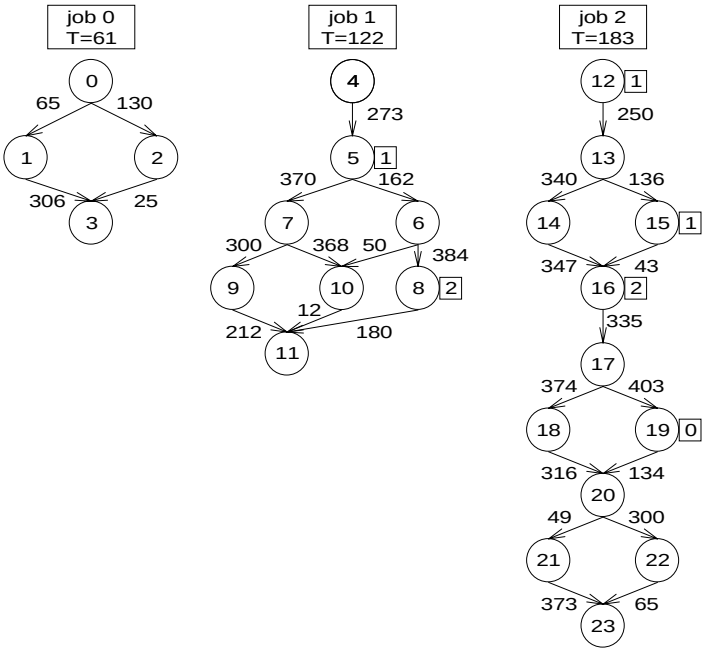

Figura 8

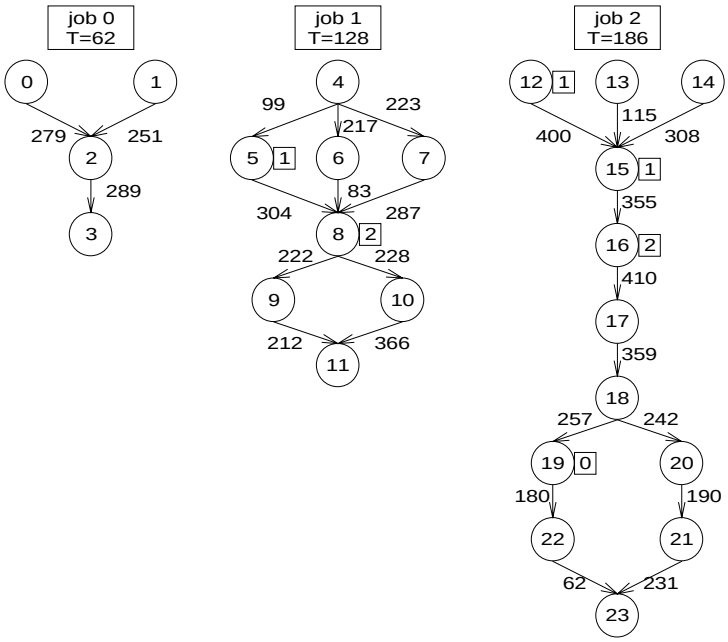

Figura 9

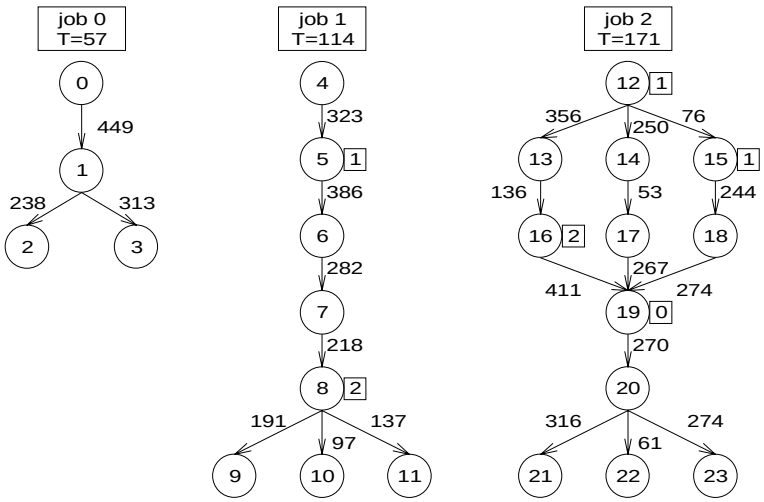

Figura 10

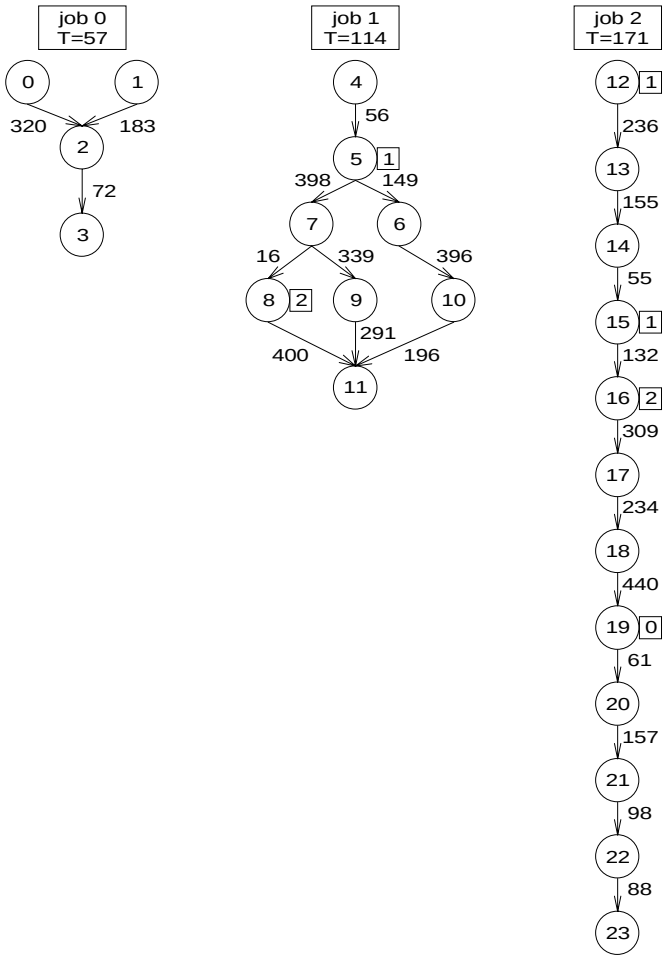

Figura 11

Cada uno de los cinco conjuntos subyacentes se utiliza para encontrar la perfomance del método para ocho factores de utilización de las tareas (0.1 a 0.8) y 23 anchos de banda diferentes (entre 0.1 y 100 Mbps). Cada uno de los 390 ternas (grafo, factor de utilización y ancho de banda) define un problema. Todas las soluciones diferentes obtenidos por cada par (APUF y NBW) son finalmente promediadas. Los resultados se presentan en la figura 12.

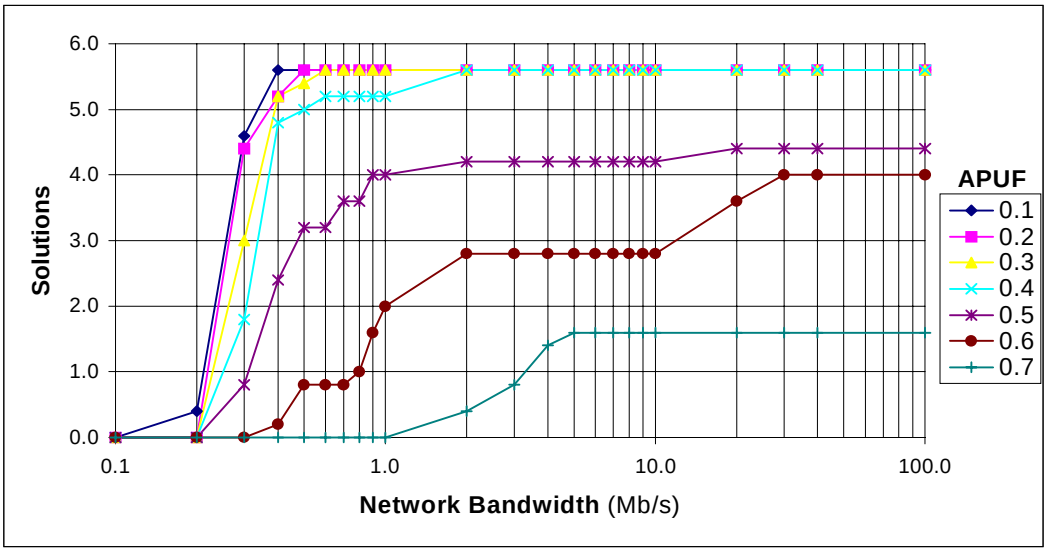

Figura 12

En la figura 13 se representan los retardos de comunicaciones promedio (en ranuras) vs. APUF para diferentes valores de NBW.

Del análisis de los resultados expuestos en las figuras se deduce que:

a) Para un factor de utilización promedio APUF dado, el número de soluciones diferentes encontrado se incrementa con el ancho de banda de la red. Esto ocurre como consecuencia de los menores retardos de comunicaciones que facilitan el cumplimiento de las restricciones de precedencia fuerte.

b) A medida que se incrementa el APUF, se pueden encontrar soluciones sólo en menor número y con anchos de banda superiores. Esto se explica porque el problema es más difícil de resolver: al aumentar los factores de utilización de los procesadores por medio de incrementos en los tiempos de ejecución de las tareas, queda menos tiempo disponible para las comunicaciones, lo que a su vez provoca la necesidad de un mayor ancho de banda.

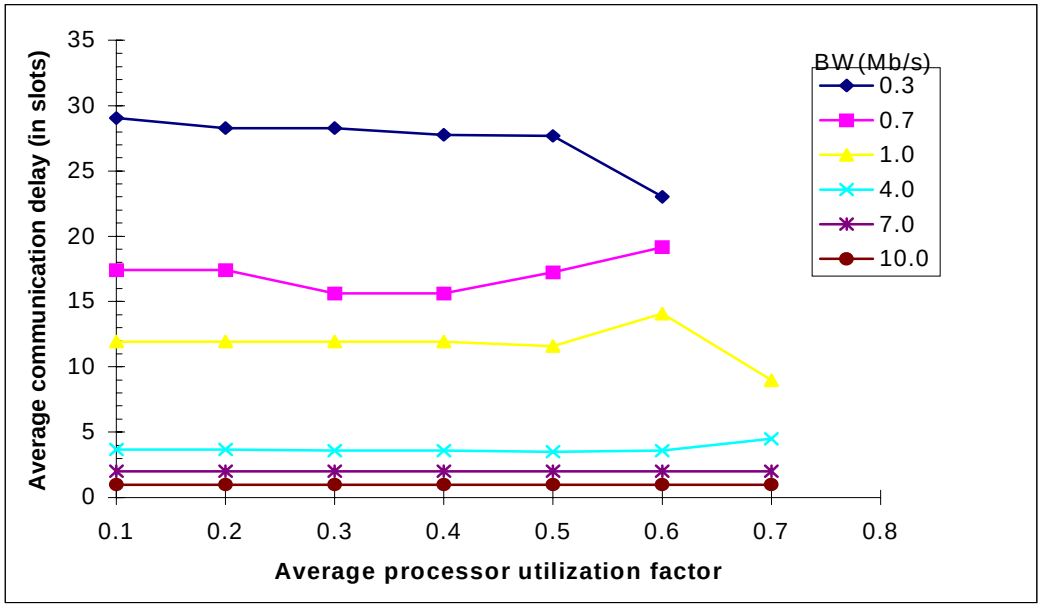

Figura 13

c) A partir de un determinado ancho de banda, no se pueden encontrar soluciones diferentes adicionales. Esto se explica porque el número de soluciones tentativas diferentes encontradas al finalizar el paso 7 antes de la verificación de precedencias es siempre el mismo. Con menores anchos de banda, algunas de ellas se descartan por no cumplir las restricciones de precedencias. Aumentar el ancho de banda y, por lo tanto, disminuir los retardos de comunicaciones permite la incorporación de soluciones previamente descartadas. Cuando todas las soluciones fueron incorporadas, no se puede ganar nada más aumentando más el ancho de banda.

d) Para un ancho de banda dado, el número de soluciones decrece a medida que APUF aumenta. Esto se debe a dos motivos. En primer lugar, algunas de las soluciones encontradas al final del paso 7^{mo} pueden cumplir las restricciones de precedencia porque el incremento en los tiempos de ejecución

permite menos tiempo para las comunicaciones. En segundo lugar los incrementos en los factores de utilización de los trabajos reducen el número de tareas que se pueden ejecutar en un procesador dado y eventualmente todos los procesadores están saturados, pero no todas las tareas han sido asignadas.

e) Los retardos promedio de comunicaciones no siempre están monotónicamente correlacionados con respecto a los factores de utilización promedio de los procesadores. Esto es debido a que diferentes factores de utilización conducen a diferentes asignaciones y la redistribución de las tareas provoca diferentes necesidades de comunicaciones y, por lo tanto, diferentes retardos.

f) No se encontró ninguna solución para un APUF de 0.8. Dadas las características del método es imposible saber si la solución no existe o si el mismo no puede encontrarla. Cabe aclarar que los grafos no son completamente aleatorios sino que fueron manipulados arbitrariamente forzando dos trabajos para que sean cadenas, imponiendo por lo tanto una fuerte carga en las precedencias. Como se verá más adelante, las restricciones de precedencia, según se definen en este trabajo, juegan un papel decisivo en la factibilidad de la asignación.

2.2 Relación de Éxito vs. APUF para diferentes NBWs

La relación de éxito (Success Ratio, SR) es la relación entre el número de veces que se encontró al menos una solución para un problema y el número de problemas considerado. Se determinaron las curvas que representen SR vs.

APUF para diferentes NBWs y seis procesadores. Los APUFs variaron entre 0.1 y 0.8 y los NBWs entre 0.2 y 20 Mbps. 100 conjuntos de tres grafos representando trabajos de 4, 8 y 12 tareas fueron generados al azar. En este caso, el número de tareas iniciales, en el primer nivel, es elegido aleatoriamente dentro de los intervalos [1,3], [1,7] y [1,11] respectivamente. El número de tareas en niveles sucesivos es generado de manera similar aunque el intervalo es, en cada caso, [1, número de tareas asignadas en el nivel anterior]. La conectividad entre las tareas en un nivel y los siguientes niveles está establecida, en cada caso, con una probabilidad del 20%. La carga de comunicaciones, en bytes, asociada a cada conexión se obtuvo multiplicando por 300 un número generado aleatoriamente entre 1 y 10. Se preasignaron aleatoriamente seis tareas y dos pares de tareas en el mismo nivel de un trabajo dado se seleccionaron al azar para ejecutarse en procesadores diferentes. En la figura 14 se ilustra un conjunto típico de grafos con las cargas de comunicaciones y restricciones de ubicación asociadas.

Cada uno de los 100 conjuntos de grafos subyacentes se usó para generar diferentes problemas, correspondientes a distintas duplas (APUF, NBW). Para poder evaluar la influencia de las restricciones de precedencia en el SR , se planteó el problema con precedencias blandas. Los resultados se representan en la figura 15. En la misma, se procesaron 100 conjuntos de tareas para cada punto. Esto hace un total de 4800 problemas. Los resultados, como eran de esperar, son los siguientes:

a) Para una NBW dada, el SR decrece a medida que se incrementa APUF.

b) Para un APUF dado, SR se incrementa a medida que crece NBW.

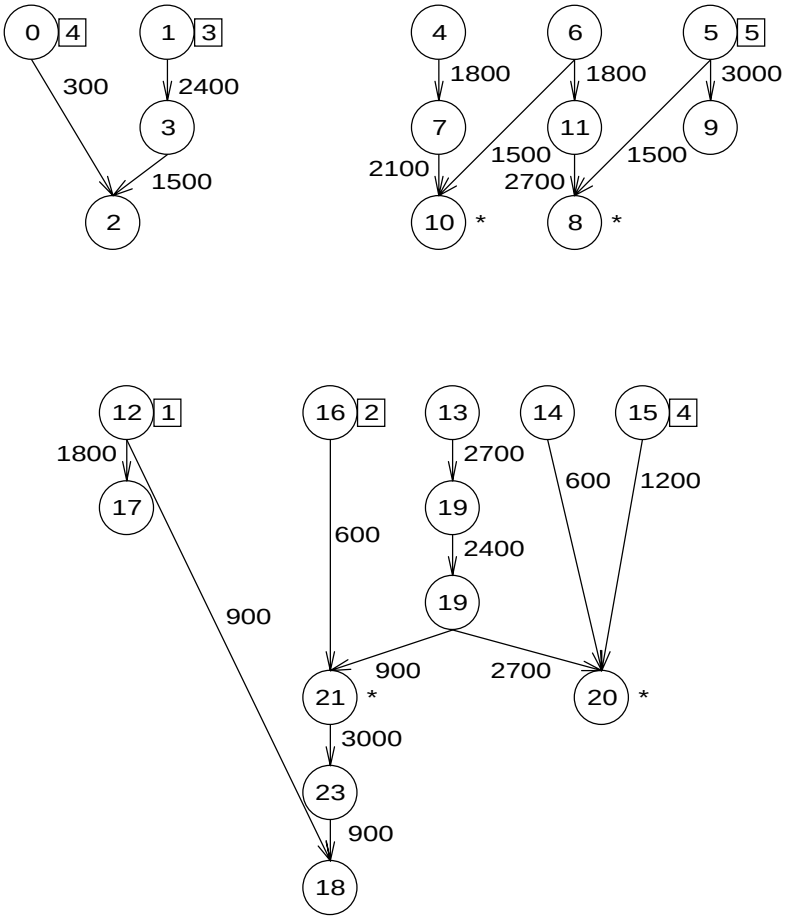

Figura 14

La gran influencia de la restricción de precedencia se puede ver claramente: con precedencia blanda, se alcanza un SR del 98% con una NBW de 2Mb/s (con un APUF de 0.4). Para la precedencia blanda, el intervalo de confianza usualmente aceptado (95%) produjo el menor espacio entre los límites superior e inferior del intervalo, 6.98, para una NBW de 2Mb/s y APUF de 0.4. El mayor intervalo, 20.2, se produjo con NBW igual a 0.6Mb/s y APUF igual a 0.8. Como los SRs están referidos al número total de problemas estudiados y no al número de problemas para el cual existe al menos una solución, los

resultados muestran una visión pesimista de la efectividad del método: a medida que los problemas se vuelven más difíciles debido al aumento de las APUFs y/o disminución de NBWs, puede que algunos de ellos no tengan ninguna solución.

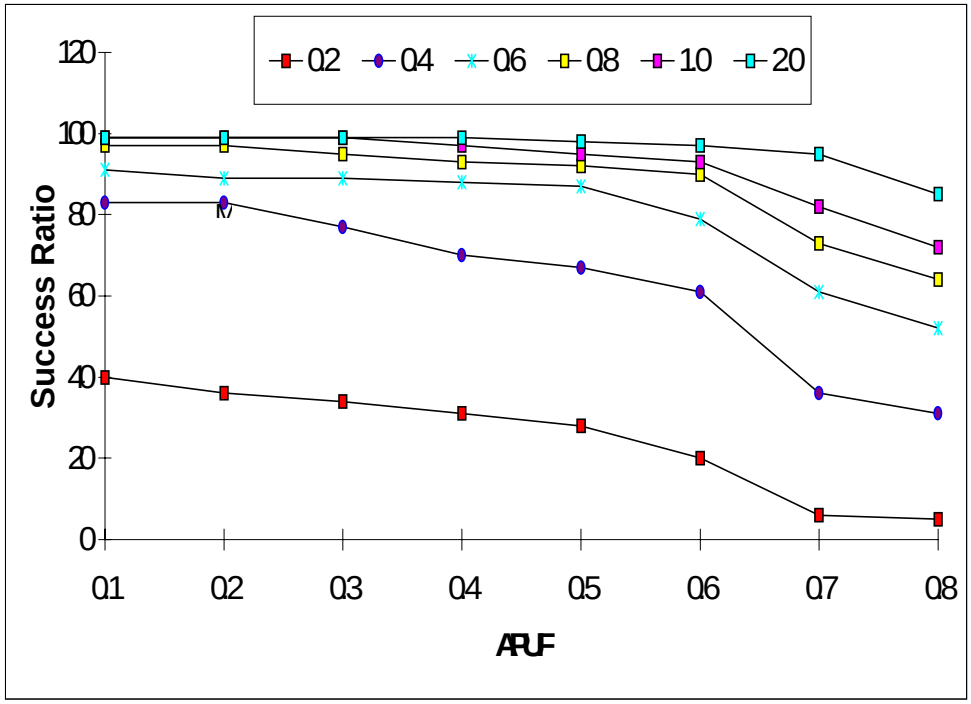

Figura 15

3 Comparación con otros métodos

La piedra de toque de un método heurístico es la comparación con otras heurísticas o simulaciones. La mejor comparación se hace cuando exactamente el mismo conjunto de problemas bien especificado, usados como referencias, son estudiados por los diferentes métodos. Luego de eliminar las diferencias que surgen de, por ejemplo, el potencial del equipo de cómputo o el lenguaje utilizado, los tiempos que se tarde en hallar la primera o la "mejor" solución pueden utilizarse como una medida conveniente. Desafortunadamente, esa información detallada no está normalmente

incluida en los trabajos. Además del tiempo, otros indicadores (por ejemplo, SRs, tolerancia a fallas, robustez, etc.) pueden utilizarse como parámetros, pero siempre se deben poder reproducir los experimentos y verificar los resultados obtenidos.

El ejemplo de Tindell (1992) está completamente especificado (con tiempos de ejecución, períodos, preasignaciones, requerimientos de memoria y comunicaciones para cada tarea, NBW y capacidad de memoria en cada procesador). Por lo tanto, este ejemplo se consideró suficiente para iniciar las comparaciones.

Las tablas 4 y 5 muestran el sistema de 8 procesadores y 43 tareas periódicas de tiempo real usado en [58] como ejemplo ilustrativo. La tabla 4 lista la capacidad de memoria de cada procesador. En la tabla 5 se muestra el período, tiempo de ejecución para el peor caso, requerimientos de memoria, carga de comunicaciones y, en caso de existir, el procesador preasignado o restringido para cada tarea. 50/1 y 150/2 en la primera línea significa que la tarea τ_0 debe enviar mensajes de 50 bytes a la tarea 1 y de 150 bytes a la tarea 2. Los siguientes pares son original y réplica y no pueden ser ejecutados en el mismo procesador: (33, 38), (34, 39), (35, 40), (36,41) y (37, 42). El método de asignación utilizado es el recocido simulado. El mismo comienza con una energía y una temperatura arbitrarias asociadas a un sistema de asignación arbitrario.

La energía es una medida de la no diagramabilidad y altas temperaturas están asociadas a altas energías. Se efectúan saltos desde una asignación a otra tratando de bajar la energía. Se efectúan centenares de saltos desde un punto de asignación a otro para cada temperatura hasta alcanzar una solución. La tabla 6 muestra la solución presentada en Tindell, con las tareas asociadas a procesadores y los factores de utilización y memoria de cada procesador. El ancho de banda útil de la red Round

Robin es de 90 bytes/ms, correspondientes aproximadamente a una red local Token Ring 802.5 de 0.9Mb/s. Se consideran sólo restricciones de precedencia blanda.

Processor	Memory
0	10,000
1	10,000
2	10,000
3	12,000
4	7,000
5	7,000
6	12,000
7	10,000

Tabla 4

Task	Period	WCET	Memory	Messages	Location
0	60	4	3000	50/1; 150/2	0
1	60	4	1500	60/3; 70/4; 30/5	
2	60	2	1200	20/3	
3	60	2	1700		1
4	60	2	3000	60/6	
5	60	4	3000	80/6	
6	60	6	1100		2
7	35	2	500	40/8	1
8	35	2	700		1
9	35	8	900	90/11	0
10	35	14	2200	250/11	
11	35	4	1000		1
12	14	2	1000	150/13; 150/14	2
13	14	2	1500	50/15	
14	14	2	1600	50/15	
15	14	2	1300		3
16	14	2	1100	50/17	3
17	14	2	1000		2
18	35	1	1000	50/19	1
19	35	1	1600		1
20	14	1	1900	40/21	
21	14	2	2000		3
22	14	1	1000	40/23	
23	14	1	2000	40/24	
24	14	1	1000	20/25	
25	14	1	2000	20/26	
26	14	2	7000	20/27; 20/28	
27	14	1	1100	50/29	
28	14	1	900	30/29	
29	14	1	500		6
30	14	1	600	50/31	7
31	14	2	800	70/32	
32	14	2	1300		7
33	20	3	1000	50/35	2; 3
34	20	2	1000	50/35	0; 1
35	20	2	1000	60/36; 60/37	
36	20	2	1000		6; 7
37	20	2	1000		
38	20	3	1000	50/40	2; 3
39	20	2	1000	50/40	0; 1
40	20	2	1000	60/41; 60/42	
41	20	2	1000		6; 7
42	20	2	1000		

Tabla 5

El retardo de comunicaciones, Δ, encontrado con la ecuación (12) se usa para verificar las precedencias blandas de acuerdo con la ecuación (13).

Processors	Tasks	PU%	MU%
0	0; 1; 2; 4; 9; 34; 35; 37	72.9	127.0
1	3; 7; 8; 10; 11; 18; 19; 39	81.9	97.0
2	6; 12; 13; 14; 17; 33	82.1	72.0
3	5; 15; 16; 20; 21; 38	71.7	85.8
4	22; 23; 24; 25	28.6	85.7
5		00.0	00.0
6	26; 27; 28; 29; 36	45.7	87.5
7	30; 31; 32; 40; 41; 42	65.7	57.0

Tabla 6

La asignación, de todas formas, es incorrecta, dado que la capacidad de memoria del procesador 0 (10000) se ve excedida por los requerimientos de memoria del conjunto de tareas asignadas a ese procesador (12600). Probablemente se trate de un error de tipeo, dado que la solución sería correcta si los requerimientos de memoria de la tarea 0 fueran de 300 en lugar de 3000 como indica la tabla 1 en el trabajo de Tindell.

El método heurístico propuesto aquí produjo la primera solución en el primer intento. La misma se muestra en la tabla 7. La utilización promedio del procesador es del 56%, la utilización promedio de memoria es del 79.7%, el factor de utilización del canal es del 61% y la carga relativa del canal es del 40%.

Processors	Tasks	PU%	MU%
0	0; 1; 2; 9; 34; 35; 37	70	96
1	3; 7; 8; 10; 11; 18; 19; 39	82	97
2	6; 12; 13; 14; 17; 33	82	72
3	15; 16; 20; 21; 38; 40	69	75
4	42; 4; 5	20	100
5	22; 23; 24; 25	29	86
6	26; 27; 28; 29; 36	46	88
7	30; 31; 32; 41	46	37

Tabla 7

Evaluar las 40320 permutaciones en una Alfa AXP 3000 tomó un tiempo aproximado de 28 minutos. 38389 de las permutaciones produjeron asignaciones diagramables, aunque sólo 26 de ellas fueron diferentes. Por ejemplo, la solución de la tabla 7 fue

producida también por la siguiente permutación generada en la cual el orden de los procesadores fue <0, 1, 2, 3, 4, 5, 7, 6>. La solución lograda por el método de recocido simulado no coincidió con ninguna de las soluciones encontradas por el método heurístico. Este hecho nos pareció tan curioso que nos motivó a revisar la solución lograda por el método de recocido simulado, encontrando el defecto que mencionamos más arriba.

El método, programado en lenguaje C, tardó menos de 42 mseg. en encontrar la primera solución. Desafortunadamente, el tiempo tomado por el método de recocido simulado, indispensable para comparar los dos métodos, no se menciona en (Tindell, 1992) y no pudo ser obtenido por otros medios. De todas maneras, dada la complejidad del problema propuesto, el orden de tiempo tardado sugiere por sí solo una buena perfomance del método heurístico. Dado que 38389 de las 40320 permutaciones posibles de los procesadores condujeron a alguna de las 26 asignaciones válidas, la probabilidad de hallar una buena solución en el primer intento fue mayor a un 95%. A pesar de que la probabilidad de ocurrencia es prácticamente nula, en el peor caso la primera solución se habría encontrado en el 1931^{o} intento, lo que da un tiempo menor a 82 segundos.

Dada la ausencia de un benchmark, la comparación con otras heurísticas también es difícil. De todas maneras, como el de Ramamritham [48] es todavía uno de los trabajos relevantes en materia de heurísticas para resolver el problema aquí tratado, será utilizado como un punto de referencia. Debe hacerse notar que la comparación es solo aproximada puesto que en Ramamritham no se permiten bloqueos, las únicas restricciones de recursos se refieren a la CPU y a las comunicaciones, se utiliza un

Ejecutivo Cíclico en lugar de un diagramador por PMC y las cargas de comunicaciones se miden en unidades de tiempo en lugar de unidades de datos. Además se determina la relación de éxito vs. el factor de relajamiento para diferentes factores de comunicación (una variable y un parámetro definido *ad hoc* en el trabajo) y no se suponen preasignaciones. Finalmente, se usó un generador de tareas de acuerdo a lo descripto en la página 77; el mismo coincide con el presentado por Ramamritham en que forma 3 grafos de 4, 8 y 12 tareas respectivamente. La probabilidad respecto del número de tareas iniciales, la cantidad de información intercambiada, etc., pueden diferir de las utilizadas por Ramamritham, pues estos datos no fueron incorporados en su trabajo.

APUF y NBW, utilizadas como variable/parámetro y viceversa, se prestaron para una presentación y evaluación de perfomance más clara, como se muestra en las figuras 15. Con propósitos comparativos, los APUFs pueden ser traducidas a Factores de Relajamiento, esencialmente una función de los factores de utilización de las tareas y el número de procesadores, que Ramamritham llama Laxity Factor (LF). En nuestro caso, como en Ramamritham (1990), LF = 1/2.2 APUF. Esto significa que barremos en un rango de LF entre [0.56, 4.5], alcanzando un valor de SR no nulo incluso para APUFs de 0.8 y 0.7 para precedencias blandas, mientras que en Ramamritham (1990) SR se grafica sólo para un APUF de 0.5 y, extrapolando las mejores curvas, SR cae a cero antes de un APUF de 0.6.

4 Método Sintonizado

La filosofía del método sintonizado se puede definir como un ajuste dinámico de la heurística de asignación, al considerar antes de cada asignación el estado del sistema. El método presentado aquí es una alternativa al método desarrollado previamente. En los pasos 4 y 6 del método base solo se toman en cuenta las consideraciones sobre comunicaciones y precedencias. El método puede ser mejorado si en los pasos apropiados, muchas restricciones más son evaluadas. Se presentan dos alternativas a la heurística base que denominaremos versión 2 y 3 respectivamente. Para la versión 2 se debe modificar el paso 3 de la siguiente manera: Se obtienen tres pilas formadas por las tareas, en la primera de ellas se ordena a las tareas que se comunican con las tareas ya asignadas al procesador por valores decrecientes del factor de utilización, en la segunda se ordena a las mismas tareas por valores decrecientes de la memoria requerida y en la tercera por valores decrecientes de la cantidad de información que intercambian con otras tareas. En todos los casos las tareas tienen asociado un ordinal que indica su posición en las pilas. Una cuarta pila es obtenida utilizando el siguiente polinomio:

$$P_\tau = \alpha_1 X_{\tau,u} + \alpha_2 X_{\tau,m} + \alpha_3 X_{\tau,c} \qquad (14)$$

donde $X_{\tau,u}$, $X_{\tau,m}$, y $X_{\tau,c}$ indican la posición de la tarea τ en la pila de utilización, memoria y comunicaciones respectivamente. Los parámetros α son calculados reflejando el estado del factor de comunicaciones, de utilización y de memoria disponible respectivamente, en un instante determinado del proceso de asignación.

Cálculo de α_1 :

$$\alpha_1 = D_{min}.W / (\sum L_{ij} - \sum L_{hk}) \quad (15)$$

donde L_{ij} determina la longitud del mensaje desde la tarea i a la tarea j cuando ambas no están asignadas al mismo procesador. La sumatoria indica la carga total de la red, si todas las comunicaciones entre las tareas no asignadas al mismo procesador, se realizaran a través de la red. L_{hk} determina la longitud del mensaje desde la tarea h a la tarea k cuando ambas tareas están asignadas al mismo procesador.

D_{min} determina el menor vencimiento de todas las tareas y W el ancho de banda de la red. Si D_{min} .W, el total transmisible, es mayor que $\sum L_{ij} - \sum L_{hk}$ entonces la red no presenta problemas de comunicaciones y α_1 se hace igual a uno. Luego la incorporación de tareas compañeras dependerá del estado local del factor de utilización y memoria. Al estar mas solicitada la red, menor será α_1 y mayor será la influencia sobre la sintonización.

Cálculo de α_2 :

$$\alpha_2 = UF_p \quad (16)$$

donde UF_p indica el factor de utilización del procesador que esta siendo cargado y se define como la sumatoria de la relación entre el tiempo de ejecución y el periodo de todas las tareas ya asignadas al mismo.

Cálculo de α_3 :

$$\alpha_3 = MF_p \quad (17)$$

donde UM_p indica el factor de memoria del procesador que esta siendo cargado y se define como la sumatoria de la relación entre la cantidad de memoria usada por todas las tareas ya asignadas al mismo y la cantidad de memoria física disponible en el procesador.

Cada vez que una tarea es asignada al procesador, los parámetros α son recalculados. Obviamente, al ir evolucionando en cada instancia el proceso de asignación los parámetros varían su influencia dinámicamente.

En el paso 5 del algoritmo la pila de procesadores es construida de manera polinómica al combinar dos pilas particulares. En la primera de ellas los procesadores son ordenados por factores crecientes de sus factores de utilización. En la segunda son ordenados por valores decrecientes de disponibilidad de memoria. La pila resultante es obtenida utilizando el siguiente polinomio $P_p = \beta_1 X_{p,u} + \beta_2 X_{p,m}$ donde $X_{p,u}$ y $X_{p,m}$ son el ordinal asociado a la posición del procesador p en las pilas de utilización y memoria respectivamente. β_1 y β_2 reflejan el estado del factor de utilización y de la memoria disponible en un instante determinado del proceso de asignación.

En el paso 7, la pila de tareas es ensamblada de manera polinomial y similar a las descriptas para el paso 3. En este punto, cada vez que una tarea es asignada al procesador, los parámetros α y β son recalculados.

La versión 3 del algoritmo es idéntica, desde el punto de vista lógico, a la versión 2, con la salvedad que las dos primeras pilas del paso 3 son ordenadas de manera inversa, es decir por valores decrecientes de sus factores de utilización y memoria respectivamente. De esta manera un mayor número de tareas comunicantes pueden ingresar en el mismo procesador.

5 Perfomance, Resultados y Análisis

Para obtener resultados experimentales se uso el ejemplo de Tindell descripto anteriormente, con restricciones de precedencia blandas y 90 bytes/mseg de ancho de

banda. La heurística base (versión 1) alcanza 26 soluciones diferentes, mientras que la versiones 2 y 3 producen 34 y 36 soluciones respectivamente.

Para realizar un método sistemático de verificación, los grafos del problema citado son tomados como base y a partir de un generador aleatorio de problemas los requerimientos de ejecución, vencimiento, generación, memoria y comunicaciones son modificados para producir 100 problemas distintos.

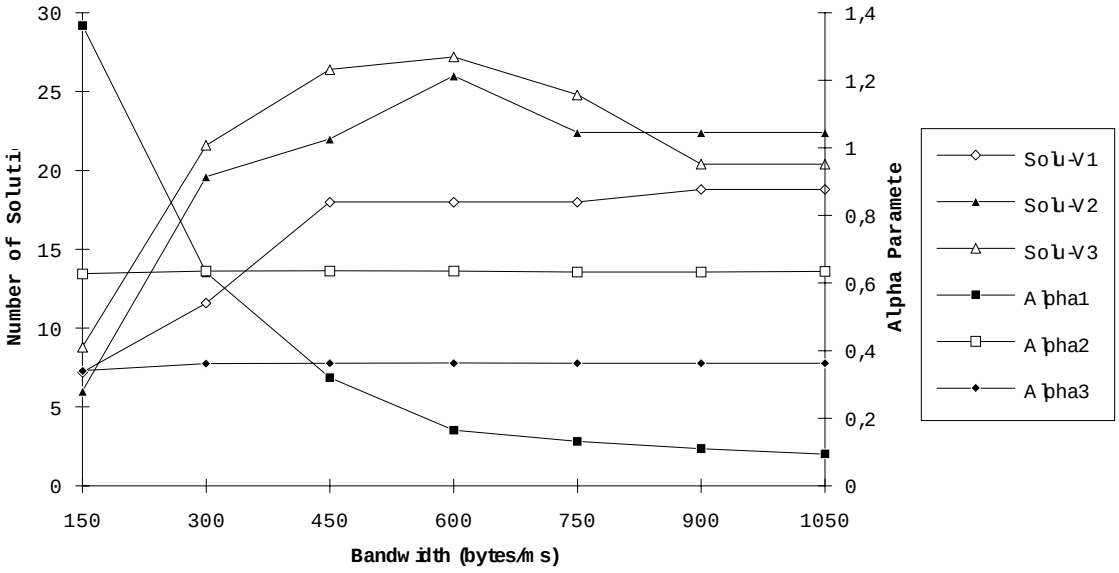

Figura 16

Se definió un factor de utilización promedio para las tareas de 0,5 y una red cuyo ancho de banda varia entre 150 y 1050 bytes/mseg en pasos de 150.

El número promedio de soluciones diferentes encontrados por cada versión y el valor promedio de los alpha para la versión 3 en función del ancho de banda se muestran en la figura 16. El análisis relevante es el siguiente:

a) α_1 es un parámetro global relacionado con las comunicaciones. Su influencia disminuye cuando el NBW aumenta. α_2 y α_3 son parámetros locales

de cada procesador e indican el promedio del factor de utilización y memoria para las n! permutaciones. Luego, al aumentar el ancho de banda pueden esperarse variaciones pequeñas.

b) El número de soluciones diferentes encontradas por la versión 1 aumenta monotónicamente con el ancho de banda. Esto se debe que la disminución en el retardo de comunicaciones, permite incorporar un número de soluciones previamente descartadas por restricciones de precedencia. Cuando todas ellas han sido incorporadas, nada puede ganarse aumentando el ancho de banda disponible.

c) Con las versiones 2 y 3 el número de soluciones es mayor que en la versión 1. Esto se debe a que el efecto de sintonización mejora el método base.

d) El número de soluciones encontradas por el método 2 y 3 aumenta, llega a un máximo y luego decrece. El incremento inicial se debe principalmente al incremento del ancho de banda que reduce el retardo de comunicaciones, luego mas soluciones pueden ser alcanzadas. Como el valor de α_1 disminuye a partir un cierto punto y pierde influencia sobre los pesos locales al procesador, determinados por α_2 y α_3, no contribuye a la búsqueda de una solución, a pesar de incrementar el ancho de banda.

e) Desde el punto de vista de las comunicaciones (menor retardo y menor tráfico sobre el canal), la versión 3 produce levemente mejores soluciones que la versión 2, pero ambas producen mejores soluciones que la versión 1. Menor tráfico sobre el canal de comunicaciones mejora la disponibilidad de ancho de

banda para tráfico que no es de tiempo real, mejorando la calidad de servicio de cada usuario.

f) Las comparaciones alcanzadas para la heurística base con el recocido simulado también son aplicables a las versiones sintonizadas.

6 Resumen y Conclusiones

En [56] se enumeran diferentes temas de investigación como de desarrollo necesario para elaborar una ciencia sobre grandes sistemas de tiempo real. Entre ellos, la Teoría de la Diagramabilidad, esencialmente la asignación de recursos de acuerdo a buenos algoritmos de forma que se satisfagan todos los vencimientos, permanece como una de las más importantes. Que los requerimientos de tiempo deben ser comprensibles, predecibles y mantenerse, es un requerimiento particularmente significativos en el caso de sistemas multitarea-multiprocesador. Los métodos propuestos para resolver estos problemas se remontan a los años setenta, pero la mayor parte de la literatura sobre el tema se publicó en las dos últimas décadas. Desafortunadamente, la mayor parte de los métodos no pueden ser comparados entre sí debido simplemente a que los problemas a los que están orientados están definidos en términos diferentes. Algunas veces, diferencias pequeñas en esos términos derivan en grandes diferencias en los resultados de las evaluaciones. Por ejemplo, si en este trabajo se utilizara un Ejecutivo Cíclico en lugar de PMC, los resultados probablemente no habrían sido tan buenos. Incluso si se utilizara la misma disciplina de prioridades, un método diferente para aplicarlas (por ejemplo Lehozky vs. Liu) habría conducido a resultados diferentes.

La ausencia de una base comparativa es un fenómeno común en los tiempos iniciales de una disciplina. La aparición de normas es un signo de madurez. En consecuencia, probablemente deba transcurrir un tiempo antes de poder definir un patrón de referencia para evaluar las simulaciones o heurísticas multitarea/multiprocesador propuestas. No es una tarea difícil a causa de todos los atributos y parámetros que caracterizan a los sistemas distribuidos pero, sin duda, sería un buen intento.

La heurística presentada es una búsqueda guiada activamente hacia soluciones óptimas o casi óptimas al problema de diagramar un conjunto de tareas de tiempo real apropiables, que comparten recursos en un conjunto de procesadores distintos. Se utilizan el método de las Ranuras Vacías y la disciplina de los Períodos Monotónicos Crecientes para diagramar las tareas en los procesadores. Asimismo se utiliza Rueda Cíclica para diagramar la red de comunicaciones. Primeramente se evalúan las tareas preasignadas, permitiendo la detección precoz de sistemas absolutamente no diagramables. Si cada vez que es posible, las tareas que se comunican entre sí son asignadas al mismo procesador, no sólo está menos cargada la red de comunicaciones sino que, lo que es más importante, las restricciones de precedencia se satisfacen más fácilmente. El método es general en el sentido que maneja todo tipo de restricciones: de ubicación, (por motivos de asignaciones y tolerancia a fallas), de comunicaciones, de tiempo y de precedencia. Se ofrece una aproximación formal a los aspectos teóricos de precedencia blanda. Se evalúa la perfomance del método a través de miles de simulaciones. Los resultados se expresan y analizan como el número de soluciones en función del ancho de banda para factores de utilización diferentes y como relación de éxito en función de factores de utilización promedio para distintos anchos de banda.

Para verificar la eficacia del método se realizaron comparaciones con otros métodos. El método sintonizado permite definir un ajuste dinámico de la heurística de asignación, al considerar antes de cada asignación el estado del sistema. Mediante este mecanismo se pudo obtener una diversificación mayor del tipo y número de soluciones obtenidas y mejorar aun mas la relación de éxito de la heurística base. Las mejoras siempre están basadas en la premisa de trabajar en pos de reducir al mínimo la influencia de las restricciones de precedencia.

Como trabajos futuros desarrollaremos métodos metaheurísticos basados en Algoritmos Genéticos y una alternativa híbrida heurística/genética para resolver el mismo tipo de problema planteado en esta tesis, siguiendo la técnica de representación de la información planteada en [2].

REFERENCIAS

1. Adan, J. Magalhaes, F. and Ramamrithan K. (1995). “Meeting Hard Real Time Constraints Using a Client-Server Model of Interaction”. In 7th IEEE Euromicro Workshop on Real Time Systems, pp. 190-195.
2. Ahmad, I. and Dhodhi, M. K. (1995). “On the m-Way Graph Partitioning Problem”. Computer Journal, pp. 237-244.
3. Altenbernd, P. (1996). “Multiprocessor Allocation of Periodic Hard Real-Time Task”. C-Lab Report 10/96. Pp. 1-37.
4. Barr, R., Golden, B., at all (1995). “Designing and Reporting on Computational Experiments with Heuristic Method”. Journal of Heuristic, Vol. 1, Nro 1.
5. Bellman, R. and Zadeh, L. (1977). “Local and Fuzzy Logic in Modern Uses of Multiple Valued Logic”. Reidel Publ. Dordrecht, Netherlands.
6. Blazewicz, J. and Ecker, K. (1994). ¨Multiprocessor task scheduling with Resource Requirements¨, Journal of Real Time Systems, 6(1).
7. Boccouche, L. (1995). “Efficient Static Allocatión of Real-Time Task Using Genetic Algorithms”. Imag Institute, Laboratoire de Genie Informatique.
8. Borriello, G. and D. Miles. (1994). “Task scheduling for real-time multiprocessor simulations”. 11th Workshop on RTOSS, pp. 70-73.
9. Caldeira, C. (1996). “Neural network versus Max-Flow Algorithms for Multiprocessor Real Time Scheduling”. 8th IEEE Euromicro Workshop on Real Time Systems, pp. 175-180.

10. Cayssials, R, Ferro, E. y Orozco, J. (1997). “Análisis de diagramabilidad PMC en sistemas multitarea-monoprocesador. una alternativa a los métodos tradicionales”, XXVI Jornadas Argentinas de Informática e Investigación Operativa. Buenos Aires.
11. Cayssials, R., Orozco, J., Santos, J., and Ferro, E. (1997). “Precedence constraints in hard real-time distributed systems”. Proceedings Third IEEE International Conference on Engineering of Complex Computer Systems. Como, Italy. pp. 33-38.
12. Cheng, A. M. K., Browne J., Mok A. and Wang R. 1993. Analysis of real-time rule-based systems with behavioral constraint assertions specified in Stella. *IEEE Trans. Software Eng.*, 19(9), pp. 863-885.
13. Chetto, H., Silly, M. and Bouchentouf, T. 1990. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems, 2*, pp. 181-194.
14. Coli, M and Palazzari, P. (1995). “A New Method for Optimization of Allocation and Scheduling in Real-Time Applications”. In 7th IEEE Euromicro Workshop on Real Time Systems, pp. 262-269.
15. Cook, S.A. (1971). “The complexity of theorem-proving procedures”, Proceedings of the third annual ACM symposium on the theory of computing, Association for Computing Machinery.
16. Cooling, J. (1991). “Software Design for Real-time Systems”, Chapman and Hall.
17. Di Natale, M. and Stankovic, J. (1995). “Applicability of Simulated Anneling Method to Real-Time Scheduling and Jitter Control. ”. In Proceedings of IEEE Real Time Systems Simposum.

18. Dubois, D. and Prade, H. (1980). “Fuzzy Sets and Systems, Theory and Aplications”. Academic press, New York.

19. Eilon, S. (1977). “More against Optimization”, Omega, Vol. 5.

20. Encyclopedia of Computer Science. (1993). *Edit by a Raeston and Reilly*, 3d edition, Van Nostrand Reinhold, N York. pp 112.

21. Enslow, P. (1978). “What is a Distributed Data Processing Systems”, Computer, Vol 11, pp .

22. Esquivel, S, Gallard, R (1997). “Computación Evolutiva: Conceptos y Aplicaciones”. Universidad Nacional de San Luis.

23. Ferro, E., Cayssials, R., Orozco, J. (1999). “Tuning the Cost Function in a Genetic/Heuristic Approach to the Hard Real-Time Multitask-Multiprocessor Assignment Problem”. 5th International Conference on Information Systems Analysis and Synthesis”, Orlando, USA.

24. 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.

25. Ferro, E., Santos J., Orozco, J. y 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.

26. Garey, M. and Johnson, D. (1979). “Computers and Intractability: A guide to the Theory of NP Completeness”. W.H. Freeman, San Francisco, CA.

27. Glover, F., taillard E. and deWarra, D. (1993). ¨User’s guide to Tabu Search¨, Annals of OR, pp. 3-28.

28. Goldberg, D. (1989). “Genetic Algorithms in Search, Optimization and Machine Learning”. Addison-Wesley, Reading, MA.

29. Greenwood, G., Lang, C. and Hurley, S. (1995). “An Evolutionary Strategy for Scheduling Periodic Task in Real-Time Systems”. In Applied Decision Technologies, pp. 171-188.

30. Jones, D. and Beltramo, M. (1991). “Solving partitioning problem with genetic algorithm”. In Proc. Int. Conf. on Genetic Algorithms, pp. 442-449.

31. Jonsson, J., Olsson, A. (1995). “Predicting Real Time Behaviors for data-Flow Computations”. In 7th IEEE Euromicro Workshop on Real Time Systems, pp. 270-275.

32. Katcher I.D., Arakawa H. (1993). “Engineering and Analisis of Fixed Priority Schedulers”. IEEE Trans. on Software Engineering, Vol.: 19 N^{o} 9.

33. Laplante P. (1992). “Real-Time systems design and analisys”, IEEE Press.

34. Lawler, L. (1983). ¨Recent Results in the Theory of Machinge Scheduling¨, Springer Verlag, pp. 202-233.

35. Lehoczky J P, Sha L and Ding Y. (1987). "The Rate Monotonic Scheduling Algorithm. Exact characterization and average case behaviour", Proc. IEEE Real-Time Systems Symp.

36. Lenstra, J. K., Rinnooy Kan, A.H.G., Shmoys, D. (1985). “The travelling Salesman Problem: A Guide tour of Combinatorial Optimization”, Wiley-Interscience Chichester.

37. Leung J and Whitehead J. (1982). "On the Complexity of Fixed Priority scheduling of periodic real-time tasks", Performance Evaluation, Vol. 2, No. 4, pp. 237-250.

38. Lewis, H.R., Papadimitriou, Ch. (1978). “The efficiency of Algorithms”, Scientific American, Vol. 238, Nro. 1.

39. Liu, C L and Layland J W (1973) “Scheduling algorithms for multiprogramming in hard real time environments”, JACM Vol 20 No 1 pp 46-61.

40. Locke, Douglas C. (1992). "Software Architecture for Hard Real-Time Applications: Ciclic Executives vs. Fixed Priorities Executives", Real-Time Systems, Vol 4 No 1 pp 37-53.

41. 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.

42. Mansour, N. and Fox G. (1991) “A Hybrid Genetic Algorithm for Task Allocation in Multicomputers”. Proceedings of the fourth International Conference on Genetic Algorithms, pp. 466-473.

43. Nicholson, M. (1993). ¨Allocating and Scheduling Hard Real Time Task on a point to point Distributed Systems¨. In Proceeding of Workshop on Parallel and Distributed Real Time Systems.

44. 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.

45. Peng, D.and Shin, K.(1989). “Static Allocation of Periodic Task with Precedence Constraints in Distributed Real Time Systems”. In Proceedings of the 9th International Conference on Distributed Computing Systems, pages 190-198.

46. Porto, S.C., Ribeiro, C. (1993). “A tabu Search Approach to task scheduling on Heterogeneous processors under precedence constraints”, Monografia Nro. 03/93, Pontificia Universidade Católica do Rio de Janeiro.

47. Rajkumar, R. (1991). “Synchronization in Real-Time Systems”, Kluwer Academic Publishers.

48. Ramamritham, K. (1990). “Allocation and scheduling of complex periodic tasks”. Proc. 10th International Conference on Distributed Computing Systems, pp. 108-115.

49. Sandnes, F. (1995). “A hybrid Genetic Algorithm Applied to Automatic Parallel Controller Code Generation”. In 8th IEEE Euromicro Workshop on Real Time Systems, pp. 70-75.

50. Santos, J. and Orozco, J. (1993). “Rate Monotonic Scheduling in Hard Real Time Systems”. Information Processing Letters, No. 48, pp. 39-45.

51. Santos, J., Ferro, E., Orozco, J. and Cayssials, R (1997). ¨A Heuristic Approach to the Multitask-Multiprocessor Assignment Problem Using The Empty-Slots method and Rate Monotonic Scheduling”. Real-Time Systems Journal, Vol 13. Número 2, pp. 167-199.

52. Santos, J., Gastaminza, M., Orozco, J., Picardi, D. and Alimenti, O. (1991). “Priorities and protocols in hard real-time LANs”, Computer Communications Vol 14 No 9, pp 507-514.

53. Sha L, Rajkumar R y Lehoczky J. (1990). "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Trans on Computers, Vol 39 No 9 pp 1175-1185.

54. Silver, E.A., Vidal, D., Werra, D. (1980). “A Tutorial on heuristic method”, European Journal of Operational Research”, Vol. 5.

55. Stankovic, J A "A serious problem for next-generation systems", IEEE Computer, Vol 21 No 10 (Octubre 1988) pp 10-19.

56. Stankovic, J. (1988). "A serious problem for next-generation systems", IEEE Computer, Vol 21 No 10 pp 10-19.

57. Terano, T., and Suseno, M. (1991). “Fuzzy System Theory and its Aplications”. Academic press Inc. San Diego C. A.

58. 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.

59. Warren, C. (1991). “Rate Monotonic Scheduling”, IEEE Micro, Vol 11 No 3 pp 34-38, 102.

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

61. Zanakis, S.H., Evans, (1981). “Heuristic Optimization: Why, When and How to use it”, Interfaces, Vol. 11, Nro. 5.