Optimización de Recursos en Sistemas Distribuidos Adaptivos de Tiempo Real

[1]Celeste Jiménez, [2]Ricardo Garibay,[3]Flavio Reyes, [4]Klaus Ecker, [5]Jacek Blazewicz, [6]Andrei Tchernykh

[1]Instituto Tecnológico Superior de Poza Rica, México, etselec18@hotmail.com
[2]Instituto Tecnológico de Morelia, Morelia, Mich. México, fokenene@hotmail.com
[3]Universidad del Caribe, Cancún, México, freyes@ucaribe.edu.mx
[4]Ohio University Athens, OH, USA, ecker@ohio.edu
[5]Poznan University of Technology, Poland, jblazewicz@cs.put.poznan.pl
[6]CICESE Research Center, Ensenada, B.C. México, chernykh@cicese.mx

RESUMEN

El uso de tecnología computacional distribuida en sistemas de tiempo real aumenta rápidamente. Las aplicaciones empotradas se encuentran presentes en diferentes áreas críticas tales como sistemas de defensa, control de vehículos, control robótico o en posicionamiento satélital. Los ejemplos de aplicaciones que tenemos presente consisten en sistemas técnicos operando autónomamente tales como satélites, o sistemas dinámicos (o adaptivos) tales como aplicaciones de mecatrónica, que son capaces de ajustarse a condiciones cambiantes de operación. Para controlar tales ambientes, se emplean *parámetros extrínsecos* que informan permanentemente acerca del estado o condiciones de trabajo del ambiente. Las aplicaciones dinámicas las cuales requieren un control con alto grado de flexibilidad tienen que tratar con *parámetros extrínsecos* variantes que implican cambios en las condiciones de frecuencia y tiempos de ejecución para los cómputos de punta a punta. Como consecuencia se hace necesaria la reasignación (recalendarización) de tareas entre los anfitriones. Se presenta una función de costo de reasignación la cual mide el esfuerzo de realizar la reasignación de tareas computacionales entre los anfitriones. En este artículo proponemos optimizar la reasignación de tareas minimizando la suma compensada de las migraciones de tareas. Enfocándonos en el caso particular de anfitriones conectados mediante un bus común, podemos encontrar una solución óptima en tiempo polinomial.

I INTRODUCCION

Los sistemas de tiempo real son usados para controlar ambientes técnicos construidos para servir algún propósito específico. Debido a su naturaleza principalmente no cooperativa, ellos no se comportarán automáticamente como es esperado. La ingeniería común de tiempo real trata con ambientes cuyas condiciones operativas son fijas o por lo menos no cambian durante algún lapso de tiempo relativamente largo. En tal situación, el sistema controlador encuentra condiciones de tiempo bien definidas para los cómputos de punta a punta periódicamente ejecutados. Han sido publicados muchos resultados de investigaciones relacionadas con la especificación y diseño de tales sistemas [1, 2], verificación y confiabilidad [3, 4] y calendarización (por ejemplo, [5-13]). La desventaja de estos enfoques es que limitan innecesariamente las funciones que pueden ser realizadas por el ambiente y limitan las opciones que están disponibles para manejar eventos y anomalías no previstos, tales como la sobrecarga de los recursos del sistema. Como se indicó en [14-17], esto puede llevar a una pobre utilización de recursos y esta fuera de lugar para aplicaciones que deban ejecutarse en entornos sumamente dinámicos.

Para la ejecución de sus actividades de cómputo, un sistema complejo de tiempo real se encuentra equipado con varias unidades

digitales del control que son procesadores, anfitriones o computadoras. Los enfoques comunes de la ingeniería de tiempo real utilizan tiempos de ejecución del peor-caso para caracterizar las cargas de trabajo por tarea *a priori* y asignar a los procesos recursos de cómputo y de red en tiempo de diseño. En contraste las aplicaciones dinámicas, las cuales requieren un control con mayor flexibilidad tienen que tratar con *parámetros extrínsecos* variables que llevan a condiciones cambiantes para los componentes de software en el sistema de tiempo real. Esto puede tener consecuencias para la frecuencia y el tiempo de ejecución de los cómputos de punta a punta. Como consecuencia, los anfitriones pueden llegar a estar sobre o descargados, y las tareas computacionales quizás deban ser redistribuidas entre los servidores.

Es obvio que tales reasignaciones no son libres de costo puesto que el movimiento de una tarea (es decir código y datos) hacia otro anfitrión consume tiempo, poder de cómputo y capacidad de red. Una función del costo de reasignación mide el esfuerzo de realizar tal reasignación de las tareas computacionales hacia los anfitriones.

Tradicionalmente se ha considerado que un sistema computacional funciona adecuadamente cuando la solución que obtiene es correcta desde el punto de vista lógico. Pero esto no es suficiente cuando estamos considerando sistemas que interactúan fuertemente con un entorno físico. Dada la naturaleza de interacción y cooperación entre diferentes recursos se obliga a que no sólo cada operación sea correcta, sino a que se realice en los instantes oportunos. A este tipo de sistemas es preciso imponerles restricciones temporales y se denominan sistemas de tiempo real (STR) [18].

"Un sistema de tiempo real es aquel en el que la exactitud del cómputo no solo depende de la exactitud de la lógica del cómputo, sino también del tiempo dentro del cuál el resultado es producido. Si los límites temporales del sistema no son alcanzados, se puede decir que una falla del sistema ha ocurrido" [19]

Los STR interactúan con su entorno (objeto o medio que controla), reciben estímulos de este y actúan en consecuencia para producir cambios en ese entorno. En estos casos se les denominan sistemas reactivos, ya que reaccionan ante estímulos del medio ambiente. Si bien el tiempo es el factor determinante para definir el correcto funcionamiento de un STR, erróneamente se tiende a afirmar que cuanto más rápido se realiza una tarea, mejor funciona el STR. Para evaluar el correcto funcionamiento de un STR hay que tener en cuenta los límites según una línea de tiempo, que pueden estar expresados en milisegundos, en algunos casos y en segundos en otros [20].

La estructura de este documento es el siguiente: en la sección 2 se describen los fundamentos de la calendarización y la calendarización con fechas límites. En la sección 3 se discute la cooperación entre el administrador de recursos y el administrador de asignaciones. Se describe el modelo de asignación así como el cálculo de la matriz de costos de asignaciones posibles y la elección de una solución optima con el método Húngaro. La sección 4 describe brevemente las conclusiones y trabajos futuros.

II DESARROLLO

2.1 CALENDARIZACION DE TAREAS PARALELAS

Un problema de calendarización emerge siempre que haya una elección del orden en el cual un número de tareas pueden ser procesadas y/o asignadas a servidores para ser procesadas. La meta de la calendarización es determinar la asignación de las tareas hacia los elementos de procesamiento y el orden en el cual las tareas son ejecutadas para optimizar algunas medidas de desempeño. [21]

En general, los problemas de calendarización asumen una serie de recursos y una serie de consumidores, que son servidos por estos recursos de acuerdo a una cierta política. Basados en la naturaleza de las restricciones de los consumidores y de los recursos, el problema es encontrar una política eficiente para manejar el acceso y uso de los recursos por varios clientes optimizando algunas consideraciones de rendimiento, como es la longitud del calendario. Se puede considerar que un calendario consiste en un conjunto de consumidores, un conjunto de servidores y una política de calendarización. Una tarea en un programa de cómputo, un trabajo en alguna fábrica, o un cliente en un banco son ejemplos de consumidores. Un elemento de proceso en un sistema de computadoras, una

maquina en una fabrica, o una caja en un banco son ejemplos de recursos. El primero-en llegar - primero-en ser servido (*first-in, first-out*) es un ejemplo de una política de calendarización [21].

En el siguiente tema usamos una descripción de los problemas de calendarización a base de notaciones en tres campos α | β | γ (alfa, beta, gama), donde α describe el ambiente del procesador, β describe las características de las tareas y los recursos, y finalmente γ denota un criterio de optimización (una medida de rendimiento) [21].

2.1.1 Calendarización con fechas limite para procesadores identicos

Los problemas de procesadores con criterios de optimización de fechas limite (due date) son NP-hard (difícil) (no determinísticos de tiempo plinomial) en la mayor parte de los casos. Sin embargo en algunos casos dentro de este tipo de calendarización existe una regla EDD-rule (regla de Jackson de fechas limites mas tempranas) la cual produce calendarios óptimos. [5]

Problema $P|\ |L_{\max}$. Se comienza primero con la planificación non-preemptive de tareas independientes. Tomando en cuenta transformaciones simples entre la planificación de problemas y la relación entre los criterios $C_{\max}$ y $L_{\max}$, se nota que todos los problemas que son NP-hard conforme al criterio $C_{\max}$ dejan de ser NP-hard conforme al criterio $L_{\max}$. De ahí, por ejemplo, $P2\|L_{\max}$ es NP-hard. Por otra parte, una sola unidad procesando por tiempos las tareas hace el problema $P|p_j=1, r_j|L_{\max}$ fácil. Además, el problema $P|p_j=p, r_j|L_{\max}$ puede ser solucionado en tiempo polinomial por una extensión del algoritmo para procesador único. Desafortunadamente se conoce muy poco sobre el comportamiento del peor caso de algoritmos de aproximación para los problemas NP-hard en cuestión.

Problema $P|pmtn, r_j|L_{\max}$. El modo preventivo de procesamiento hace la solución del problema de planificación mucho más fácil. El enfoque fundamental en esta área prueba la viabilidad del problema $P|pmtn, r_j, \tilde{d}_j|$ vía el método de red de flujo. Usando este enfoque repetidamente, entonces uno puede solucionar el problema original $P|pmtn, r_j|L_{\max}$ cambiando las fechas limite (due dates) y fechas limite estrictas de a cuerdo al procedimiento de búsqueda binaria.

Para probar la viabilidad del problema $P|pmtn, r_j, \tilde{d}_j|$ se puede utilizar el enfoque Horn. Por ejemplo decidir si existe realmente o no para un conjunto de tiempos dados y fechas limite, un calendario con una tarea no tardía. Supongamos que los valores de tiempos y fechas límites estrictas en un caso $P|pmtn, r_j, \tilde{d}_j|$` - sean ordenados en una lista de tal modo que $e_0 < e_1 <,...,< e_k$, $k < 2n$, donde e_i significa algún r_j o $\tilde{d}_j$. Podemos construir una red que tiene dos conjuntos de nodos, además de una entrada final y una salida (Figura 2.1). El primer conjunto corresponde a intervalos de tiempo en un calendario, es decir el nodo w_i corresponde al intervalo $[e_{i-1}, e_i], i=1,2,...,k$. El segundo conjunto corresponde al conjunto de tareas. La capacidad de un arco que une la fuente de la red al nodo w_i es igual a $m(e_i - e_{i-1})$ y así corresponde al total de la capacidad procesada de m procesadores en ese intervalo. Si la tarea T_j pudiera ser procesada en el intervalo $[e_{i-1}, e_i]$ debido a su tiempo y fechas límite estrictas entonces w_i es unido a T_j por un arco de capacidad $[e_i, e_{i-1}]$.

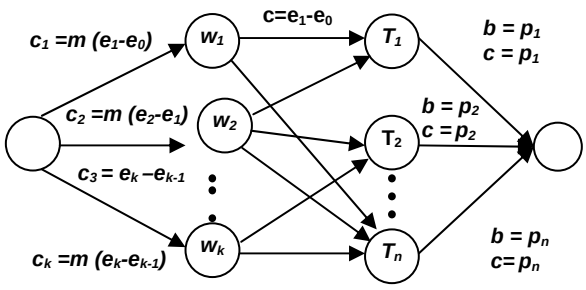

Figura 2.1. *Red correspondiente al problema* $P|pmtn, r_j, d_j|$.

El nodo T_j es unido al destino de la red por un arco con la capacidad igual a p_j y con una cota inferior sobre el flujo del arco que es

también igual a p_j. Hay que hacer notar que encontrar un modelo de flujo factible corresponde a la construcción de un calendario factible y esta prueba puede ser hecha en tiempo $O(n^3)$. Un calendario es construido sobre la base de los valores de flujo, sobre los arcos entre el intervalo y los nodos de tarea.

Hasta ahora hemos considerado problemas estáticos: donde las tareas son no-periódicas, y los requerimientos de los recursos (tiempos de procesamiento) son fijos. En el siguiente capitulo consideramos el problema de reasignación de tareas. La reasignación solamente tiene sentido si las tareas son ejecutadas *repetidamente* como es el caso de tareas periódicas.

Una tarea periódica debe ser procesada en los intervalos regulares de una longitud dada y que es establecida por los requerimientos de control. Puesto que las tareas periódicas son procesadas repetidamente, pueden ser vistas como una secuencia infinita de tareas, nosotros preferiríamos verlas como trabajos en vez de tareas periódicas. Con un trabajo J_j usualmente existen asociados tres parámetros: El periodo π_j definiendo la longitud del intervalo, el tiempo de procesamiento p_j de cada tarea del trabajo, y la fecha límite estricta d_j (deadline) dentro de un intervalo, medido relativamente en el punto de comienzo del intervalo. Asumiendo en general el punto de comienzo en el tiempo 0, la i-th ejecución (o ejemplificación) del trabajo J_j es una tarea denotada por T_j^i. El tiempo de listo de la tarea T_j^i en el intervalo i^{th} es $r_j^i := (i-1)\pi_j (i = 1,2,...)$; y su tiempo limite estricto (deadline) es $d_j^i \le (i-1)\pi_j + d_j$. Por lo tanto la condición de viabilidad para los trabajos periódicos es $p_j \le d_j$.

2.2. MINIMIZANDO EL COSTO DE REASIGNACION DE RECURSOS EN SISTEMAS DISTRIBUIDOS DE TIEMPO REAL

2.2.1 ADMINISTRADOR DE RECURSOS Y ASIGNACIONES

En el campo de la administración adaptable de recursos (ARM) se incluyen modelos estáticos y modelos dinámicos para la asignación de recursos de sistemas. Las aplicaciones de los modelos dinámicos han mostrado su eficacia para la administración adaptable de recursos.

La ejecución de tareas y el comportamiento de un sistema son determinados por valores de varios atributos. Existen dos tipos de atributos, los atributos extrínsecos y los atributos de servicio. Los atributos extrínsecos expresan las condiciones o requerimientos funcionales. Ellos son ajustados por el ambiente, o por la posición de los componentes del sistema, y no pueden ser cambiados por el sistema controlador. Un ejemplo de un atributo extrínseco “externo” es el periodo dictado por el ambiente en que una acción controladora se tiene que realizar. Un ejemplo de un atributo extrínseco "interno", definido por el sistema, es la disponibilidad actual de procesadores, de los búferes, o del ancho de banda interno de una red. En contraste a los atributos extrínsecos, los atributos de servicio son entidades que pueden ser alteradas en cualquier momento por el sistema controlador. Si las condiciones externas (representadas por los atributos extrínsecos) cambian, el ajuste apropiado de los atributos de servicio permite la adaptación a las nuevas condiciones.

Uno de los objetivos principales es encontrar una reasignación óptima de las aplicaciones a servidores tal que la operación segura del sistema controlador quede garantizada. Si los valores extrínsecos actuales del atributo cambian y exceden los límites operacionales de la asignación actual, el ARM puede resolver el problema reconfigurando dinámicamente la manera en la que los recursos de cómputo y de red son asignados a procesos. Para ello tendrá que escoger e instalar una nueva asignación que sea capaz de manejar los nuevos requisitos. Tal asignación puede ser determinada aplicando una heurística en línea, o escoger de una lista de asignaciones predeterminadas mediante una estrategia de consulta. Un administrador de recursos también mantiene el estado actual de los anfitriones, es decir su CPU y utilización de memoria a través de las actualizaciones periódicas enviadas por un proceso demonio por cada anfitrión.

El administrador de recursos recoge y mantiene toda la información de la aplicación tal como la medida de funcionamiento dinámicamente actualizada.

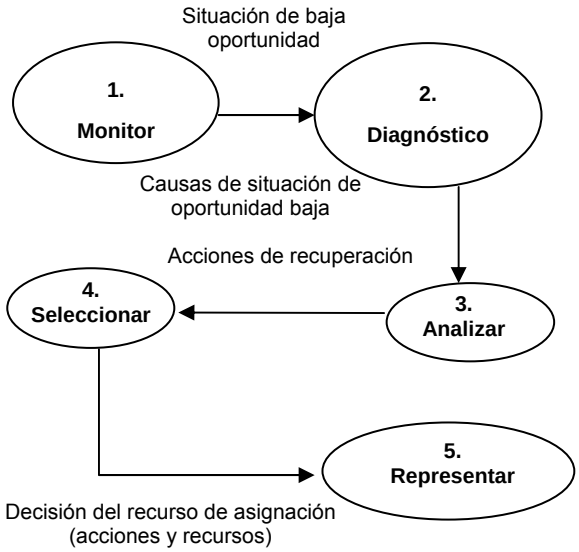

Figura 3.1. *Los procesos de la administración del recurso adaptable.* [21]

El proceso de la administración adaptable de recursos es ilustrado en Figura 3.1.

Los diferentes pasos en el proceso de la administración adaptable de recursos son explicados a continuación:

1. Monitor: El administrador de recursos recibe etiquetas de acontecimientos fechadas desde los programas de aplicación, los transforma en latencias de tarea, y compara las latencias contra fechas límites de tarea para descubrir situaciones de oportunidad de degradación.

2. Diagnóstico: Cuando una situación de oportunidad baja surge, el manejador del recurso emprende la acción diagnóstico. Esta acción encontraría la causa de la situación de oportunidad baja. La razón podría ser la alta latencia de ejecución de un programa de tarea auxiliar, un embotellamiento de la aplicación, o un aumento de la latencia de comunicación entre un par de programas de tarea auxiliar.

3. Analizar: Después de que la causa es determinada, digamos que una tarea auxiliar de aplicación es encontrada por tener una alta latencia de ejecución, causando que la tarea rebase su tiempo límite, un análisis es realizado puesto que las réplicas de aquella tarea auxiliar tienen que ser reproducidas de modo que la tarea de aplicación se reponga de la situación de oportunidad baja.

4. Seleccionar: Después de que el número de réplicas de los programas de tarea auxiliar del embotellamiento han sido determinados, el administrador de recursos selecciona al anfitrión (o anfitriones) sobre el cual aquellas réplicas deben ser ejecutadas. La selección del anfitrión podría estar basada en el criterio simple como el anfitrión con el menor índice de utilización de CPU, el menor índice de utilización de memoria, etc.

5. Representar: Finalmente, después de que la decisión es hecha sobre los programas de tarea auxiliar y los anfitriones, el administrador de recursos envía comandos a los demonios del sistema para comenzar el número requerido de programas de réplica sobre los anfitriones seleccionados.

El administrador de recursos refresca su tabla de anfitriones periódicamente. Si este no recibe una actualización de un demonio para 10 ciclos consecutivos periódicos, esto suprime la entrada para aquel anfitrión. Si otra vez llega una actualización de aquel demonio, el administrador de recursos actualiza su tabla de anfitrión como corresponde.

Por recursos pensamos en procesadores o computadoras, anfitriones llamados H, y una red conectada a los servidores. Las tareas son asignadas a los anfitriones para la ejecución. Puesto que los parámetros extrínsecos determinan la conducta del tiempo de ejecución de las tareas, la reasignación de las tareas puede ser requerida de vez en cuando. Hay dos módulos en el sistema controlador que deciden el inicio de la reasignación: El administrador de recursos (RM) el cual verifica si la reasignación del software de aplicaciones a anfitriones es necesaria, trata de optimizar el beneficio total, y, si es necesario debido a cambios mayores de requerimientos extrínsecos, proporciona al administrador de asignaciones con la información requerida para escoger una nueva asignación.

El administrador de asignaciones (AM) el cual inicia una nueva asignación, y realiza las migraciones requeridas de tareas de la nueva asignación. Este último paso (realizando las migraciones de la tarea) no es considerado en este documento. Ambos administradores, el de recursos y el de asignaciones pueden ser ejecutados en uno de los anfitriones de H, o en un procesador separado.

En todo momento el administrador de recursos debe proporcionar una asignación dentro de las limitaciones del sistema. El marco de trabajo sostiene dos limitaciones.

Primero, el administrador de recursos debe asegurar que cada aplicación sea asignada a un anfitrión válido, es decir, uno que sea capaz de ejecutar la aplicación. Segundo, para cada aplicación, sus condiciones del tiempo de ejecución (procesamiento de tiempos, fechas limite, requerimientos de memoria, etc.) están también garantizados.
La responsabilidad mínima del administrador de recursos consiste en escoger una asignación de aplicaciones a anfitriones tal que estas dos limitaciones se satisfagan para una colocación dada de atributos extrínsecos y de servicio, procesamiento de tiempos, y períodos. Una solución posible es la especificación de una función ***alloc*:** $T \to H$ de aplicaciones (tareas) a computadoras anfitriones de modo que satisfagan todas las limitaciones de la asignación. Tal asignación tiene que cumplir las condiciones del tiempo de ejecución y las limitaciones de memoria en los anfitriones.
Si los valores extrínsecos actuales de atributos cambian y rebasan los límites operacionales de la asignación actual, el RM tiene que decidir rápidamente si otra asignación se debería de instalar, o si la asignación actual debe ser mantenida al costo de una menor calidad de servicio. Si el RM escoge una nueva asignación el provoca que el AM lo instale (Figura 3.2).
Un objetivo adicional es encontrar una asignación óptima: El administrador de recursos debe tener la habilidad de realizar varias optimizaciones de la asignación. El objetivo es encontrar una asignación y un ajuste de los valores del parámetro tal que todas las aplicaciones pueden ser calendarizadas posiblemente y una función predefinida de la utilidad es llevada al máximo. El ajuste de los niveles del servicio de una aplicación es una perilla que el administrador de recursos puede utilizar para ajustar el uso del recurso y la utilidad general.

2.2.2 MODELOS DE ASIGNACION

El punto importante de todo esto es el problema de calendarización en donde se requiere encontrar una asignación: $T \to H$ de tareas hacia servidores tal que cada servidor procese la tarea asignada dentro de un marco de tiempo dado.
Existen diferentes ejemplos de problemas de asignación que nos ayudan a entender un poco mas acerca del funcionamiento de este problema, por ejemplo supongamos que tenemos un conjunto de personas *P* y un conjunto de trabajos *J*, donde todas las personas no son convenientes para todos los trabajos y esto se puede mostrar a través de grafos, graficas que nos ayudan ver un poco mas cerca de forma grafica como es la relación existente entre un conjunto de personas y trabajos, es decir por medio del grafo podemos ver un conjunto de vértices $P + J$. Esto quiere decir que si una persona P_i es conveniente para un cierto trabajo J_i hay una arista entre P_i y J_i en el grafo, hay una relación que los une, y de esta manera podemos realizar las asignaciones correctas de una persona hacia un trabajo o también de tareas hacia servidores.

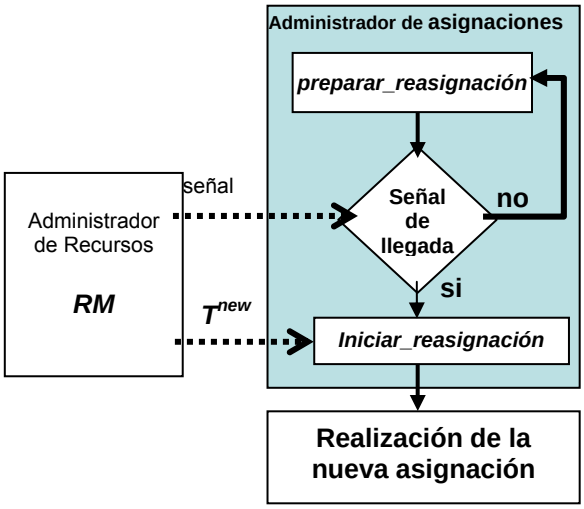

Figura 3.2. *Estructura del administrador de asignaciones.*

Dado un arreglo lineal de $m = 3$ servidores (ver figura 3.3) y 9 tareas, con la asignación actual: $\{T_1, T_2, T_3\}$ en H_1, $\{T_4, T_5\}$ en H_2, y $\{T_6, T_7, T_8, T_9\}$ en H_3.
Los tiempos de transmisión son:

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
6	4	2	5	8	3	1	5	4

Dada una nueva partición $\{T_1^{new} = \{T_1, T_6, T_7\}, T_2^{new} = \{T_2, T_4, T_8\}, T_3^{new} = \{T_3, T_5, T_9\}\}$, el costo mínimo de asignación debe ser determinado.

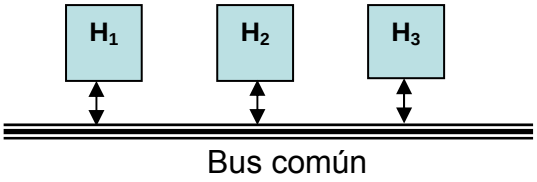

Figura 3.3 Ejemplo de una red de procesadores

Los costos de transmisión de los elementos de T_i^{new} en el H_k para *i* = 1, 2, 3 y *k* = 1, 2, 3 son:

	T_1^{new}	T_2^{new}	T_3^{new}
H_1	3 + 1	5 + 5	8 + 4
H_2	6 + 3 + 1	4 + 5	2 + 4
H_3	6	4 + 5	2 + 8

Matriz de costo *C*= $\begin{pmatrix} 4 & 10 & 12 \\ 10 & 9 & 6 \\ 6 & 9 & 10 \end{pmatrix}$

El costo mínimo de asignación es obtenido si las posiciones acentuadas en *C* son escogidas, por ejemplo C_{11}, C_{23}, C_{32}, con un costo total 4 + 9 + 6 = 19.
La nueva asignación correspondiente es:

$$\alpha^{new}(T_1^{new}) = H_1, \alpha^{new}(T_2^{new}) = H_3, \alpha^{new}(T_3^{new}) = H_2$$

2.2.3 ALGORITMO HUNGARO

El algoritmo húngaro es un algoritmo de optimización combinatoria que soluciona problemas de asignación en tiempo $O(n^3)$.
La primera versión, conocida como el método húngaro, fue inventada y publicada por Harold Kuhn en 1955. Esta fue revisada por James Munkres en 1957, y ha sido conocido como el algoritmo húngaro, el algoritmo de asignación Munkres, o el algoritmo de Kuhn-Munkres.
El algoritmo modela un problema de asignación como una matriz de costo $n \times m$, donde cada elemento representa el costo de asignar el n^{th} trabajador al m^{th} trabajo. El algoritmo realiza la minimización sobre los elementos de la matriz como en el caso de un problema de minimización de precios. Se utiliza el método de eliminación Gaussiana para hacer aparecer ceros (al menos un cero por línea y por columna). Sin embargo, en el caso de un problema de maximización de beneficio, el costo de la matriz necesita ser modificada de modo que la minimización de sus elementos resulte maximizar los valores de costo originales. En un problema de costo infinito, la matriz de costo inicial puede ser remodelada restando cada elemento de cada línea del valor máximo del elemento de esa línea (o la columna respectivamente). En un problema de costo finito, todos los elementos son restados del valor máximo de la matriz entera.

2.2.4. ANFITRIONES UNIFORMES

En este caso se asume que los anfitriones tienen capacidades uniformes en el sentido de que las tareas pueden ser procesadas en cualquier anfitrión H_i, pero posiblemente a diferentes velocidades $s_i \geq 1$, $i = 1, \ldots, m$. Los tamaños de las memorias locales pueden también variar. Sea $\mathcal{A} = \{\mathcal{T}^1, \ldots, \mathcal{T}^L\}$ un conjunto dado de particiones posibles de tareas, y considérese la partición de tareas $\mathcal{T}^k = \{\mathcal{T}_1^k, \ldots, \mathcal{T}_m^k\} \in \mathcal{A}$. Puesto que lo sanfitriones pueden tener diferentes capacidades de procesamiento, expresadas por diferentes velocidades y capacidades de memoria, para algunas $i \in 1 \ldots m$, puede suceder que no todos los anfitriones son capaces de procesar las tareas de un subconjunto $\mathcal{T}_i^k$. Por lo tanto, las posibles asignaciones quedarán restringidas y no cada función 1-1 α^{new} podrá definir una asignación viable.

Si en tiempo de ejecución la asignación actual $\alpha^{cur}(\mathcal{T}^{cur})$ debe ser cambiada por una nueva asignación a partir de una partición $\mathcal{T}^{new}$, debemos verificar cuales anfitriones son capaces de procesar los elementos de $\mathcal{T}^{new}$. Es entonces importante tener información sobre posibles asignaciones. Definamos con $\mathcal{H}^i$ el subconjunto de procesadores capaces de desempeñar las tareas de $\mathcal{T}_i^{new}$, $i = 1, \ldots, m$. A partir de nuestra consideración general de que existe una asignación realizable $\mathcal{T}^{new}$ concluimos que los conjuntos $\mathcal{H}^i$ son no vacíos. Aun mas, puesto que cada anfitrión tiene que procesar exactamente un subconjunto de $\mathcal{T}^{new}$, i.e., la función de nuevas asignaciones α^{new} es 1-1, debemos tener $\bigcup_{i=1}^{m} \mathcal{H}^i = \mathcal{H}$.

Tales restricciones pueden ser fácilmente modeladas en la matriz de costo C excluyendo de toda consideración ciertos elementos. Para esto introducimos un valor "no possible" (representado por el símbolo •) en las posiciones correspondientes de C.
For example, if H_1 is not able to process $\mathcal{T}_1^{new}$ in the above example, then $C_{11} = \bullet$. Si no hay mas restricciones la matriz de costos del ejemplo anterior es:

$$C = \begin{pmatrix} \bullet & \mathbf{10} & 12 \\ 10 & 9 & \mathbf{6} \\ \mathbf{6} & 9 & 10 \end{pmatrix} .$$

La asignación optima en este caso es α^{new} $(\mathcal{T}_1^{new}) = H_3$, $\alpha^{new}(\mathcal{T}_2^{new}) = H_1$, y $\alpha^{new}(\mathcal{T}_3^{new}) = H_2$, con un costo total de 22.

Es importante notar que el grafo bipartito no está completo. El algoritmo de máximo ajuste discutido en 4.1 puede ser utilizado también con pequeñas modificaciones, siendo la complejidad del tiempo de nuevo $O(m^3)$.

III CONCLUSIONES

Un sistema de tiempo real funciona como un sistema donde el tiempo en que se producen sus acciones es significativo. En este documento presentamos una problema de reasignación de tareas a procesadores en un sistema de mecatrónica que es capaz de ajustarse a condiciones de cambio de los condiciones de su funcionamiento. Presentamos un modelo del costo de la reasignación y proponemos un método para optimizar la reasignación de tareas. Enfocándonos en el caso particular de conexión de servidores tipo bus común, donde podemos encontrar una solución optima en tiempo polinomial minimizando la suma compensada de migraciones de tareas. También presentamos en forma general la función principal de un sistema de tiempo real, una introducción de la teoría de calendarización poniendo principal atención a los problemas de calendarización con fechas límite por ser estos de mayor interés en el área de los sistemas de tiempo real. Discutimos la cooperación del adminstrador de recursos y el administrador de asignaciones, y presentamos el algoritmo Húngaro para solucionar el problema.

En trabajos futuros se trabajará con el desarrollo del algoritmo considerando servidores uniformes y considerando una topología entre servidores para la migración de tareas, que permita medir los retardos inherentes a la misma.

IV REFERENCIAS

1. R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, *Hybrid Systems*, LNCS, vol. 73, 1993.

2. P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, *Hybrid Systems II*, LNCS, vol. 999. Springer, 1993.

3. J. Rushby, Formal Methods and their Role in the Certification of Critical Systems, Computer Science Laboratory, SRI International, Menlo Park, CA, 1999.

4. A. Arora, M. Gouda, Closure and convergence: A foundation for fault-tolerant computing, *IEEE Trans. on Comp.* 19, 1993.

5. J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Scheduling Computer and Manufacturing Processes*, Springer-Verlag, 2001

6. J. Blazewicz, K. Ecker, B. Plateau, and D. Trystram, Handbook on Parallel and Distributed Processing, International Handbooks on Information Systems, Springer-Verlag Berlin, Heidelberg, 2000.

7. C. J. Hou and K. G. Shin, Load sharing with consideration of future task arrivals in heterogeneous distributed real-time systems, IEEE Transactions on Computers 43, 1994.

8. J. Huang and D. Z. Du, Resource management for continuous multimedia database applications, Proceedings of the IEEE Real-Time Systems Symposium, 46-54, IEEE, 1994.

9. T. F. Abdelzaher and K. G. Shin, Combined task and message scheduling in distributed real-time systems, IEEE Trans. on Parallel and Distributed Systems 10, 1999.

10. D. T. Peng, K. G. Shin, and T. F. Abdelzaher, Assignment and scheduling of communicating periodic tasks in distributed real-time systems, IEEE Trans. on Software Engineering 23, 1997.

11. A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, New strategies for assigning real-time tasks to multiprocessor systems, IEEE Trans. on Computers 44, 1995.

12. R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek, A QoS Based Resource Allocations Model, Proceedings of the IEEE Real-Time Systems Symp., 1997.

13. R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek, Practical Solutions for QoS-Based Resource Allocation Problems, Proceedings of the IEEE Real-Time Systems Symposium, 1998.

14. V. Kalogeraki, P. Melliar-Smith, and L. Moser, Dynamic Scheduling for Soft Real-Time Distributed Object Systems, Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, California, 2000.

15. T. Kuo, and A. Mok, "Incremental reconfiguration and load adjustment in adaptive real-time systems," IEEE Transactions on Computers 46, 1997.

16. L. Sha, M. Klein, and J. Goodenough, Rate monotonic analysis for real-time systems, in Scheduling and Resource Management, Kluwer, ed. A. M. van Tilborg and G. M. Koob, 1999.

17. D. Stewart, D. and P. Khosla, Mechanisms for detecting and handling timing errors, CACM 40, 1997.

18. STANKOVIC J.A. y RAMAMRITHAM K.: "Hard Real-Time Systems". IEEE, 1998.

19.polaris.dit.upm.es/~jpuente/gstr/str.html

20. http://arco.inf-cr.uclm.es/docs/DSC/TrabajoSOTR.pdf

21. Hesham El-Rewini,Theodore G. Lewis, Hesham H. Ali,Task scheduling in parallel and distributed systems Prentice-Hall Series In Innovative Technology, pp 290, 1994,

22.http://citeseer.ist.psu.edu/cache/papers/cs/31493/http:zSzzSzscholar.lib.vt.eduzSztheseszSzavailablezSzetd0213200142541zSzunrestrictedzSzravithesis.pdf/real-time-distributed-systems.pdf

CURRICULUM VITAE

Andrei Tchernykh es investigador del Departamento de Ciencias de la Computación, en el CICESE, Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada, Baja California, México desde 1995. Gradúa de Sevastopol Technical University (titulado con honor) en 1975. Recibió su grado de doctor en Ciencias Computacionales del Institute of Precise Mechanics and Computer Technology de Academia Ciencia Rusa (Moscú, Rusia), en 1986. Sus intereses principales incluyen clusters y computación paralela, calendarización en línea, sistemas de tiempo real, y la optimización de recursos en GRID computacional.

Klaus Ecker recibió su grado de Dr. en Física Teórica de la University of Graz, Austria, y su habilitation en Ciencias de la Computación de la University of Bonn. Desde 1978 el es profesor en el Departamento de Ciencias Computacionales de la Clausthal University of Technology, Alemania, y desde el 2005 es profesor visitante de la Ohio University. Sus intereses principales son el procesamiento paralelo y la teoría de calendarización, especialmente en sistemas de tiempo real, y bioinformática

Jacek Blazewicz recibió su grado de Dr. en Ciencias Computacionales en 1977, y su habilitation en 1980. Es profesor en el Institute of Computing Science at the Poznań University of Technology, Polonia. Es autor y co-autor de 19 libros y 186 artículos de investigación a nivel internacional. Su campo de investigación incluye: Diseño de algoritmos y análisis de complejidad de algoritmos, especialmente en teoría de calendarización; problemas globales de calendarización en paralelo y procesadores dedicados, como son sistemas flexibles de manufactura; y problemas combinatorios en biología molecular.

Ricardo Garibay actualmente es estudiante de la carrera de Ingeniería en Sistemas Computacionales en el Instituto Tecnológico de Morelia, en Michoacán, México, Participante del XI Verano de la Investigación Científica del Pacifico en el Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada, Baja California, México, bajo la asesoria de Dr. Andrei Tchernykh.

Celeste Jiménez es estudiante de la carrera de Ingeniería en Sistemas Computacionales del Instituto Tecnológico Superior de Poza Rica, en Poza Rica, Veracruz, México, Participante del Verano Científico patrocinado por la Academia Mexicana de Ciencias, en el Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada, Baja California, México, bajo la asesoria de Dr. Andrei Tchernykh

Flavio Reyes es profesor/investigador del Departamento de Ciencias Básicas e Ingenierías de la Universidad del Caribe, en Cancún, Quintana Roo, México. Recibió el grado de Maestro en Ciencias en Ingeniería Eléctrica en el Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional en la Ciudad de México en 1993 realizando trabajos relacionados con calendarización de procesos para una máquina de visión computacional. Sus áreas de interés son la calendarización de procesos en sistemas distribuidos para entornos de tiempo real y sus aplicaciones a la robótica y a vehículos autónomos, así como los clusters y otras arquitecturas de procesamiento paralelo.