TESIS DEFENDIDA POR

Ricardo Garibay Martínez

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Andrei Tchernykh
Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez
Miembro del Comité

Dr. Mikhail Shlyagin
Miembro del Comité

Dr. Klaus H. Ecker
Miembro del Comité

Dra. Ana Isabel Martínez García
Coordinador del programa de posgrado en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales
Director de Estudios de Posgrado

7 de *septiembre* de 2009

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS

DE LA COMPUTACIÓN

ADMINISTRACIÓN ADAPTATIVA DE RECURSOS EN SISTEMAS DISTRIBUIDOS DINÁMICOS DE TIEMPO REAL

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de MAESTRO EN CIENCIAS

Presenta:

RICARDO GARIBAY MARTÍNEZ

Ensenada, Baja California, México, septiembre de 2009

RESUMEN de la tesis de **Ricardo Garibay Martínez,** presentada como requisito parcial para la obtención del grado de MAESTRO EN CIENCIAS en Ciencias de la Computación. Ensenada, Baja California. Septiembre de 2009.

ADMINISTRACIÓN ADAPTATIVA DE RECURSOS EN SISTEMAS DISTRIBUIDOS DINÁMICOS DE TIEMPO REAL

Resumen aprobado por:

Dr. Andrei Tchernykh

Director de Tesis

Los sistemas técnicos de operación autónoma y los sistemas adaptativos tales como los sistemas mecatrónicos son aplicaciones dinámicas capaces de ajustarse a condiciones de operación cambiantes. Típicamente, estos se encuentran en sistemas distribuidos de tiempo real y requieren de un alto grado de flexibilidad en control. Además tratan con la variación de parámetros extrínsecos que causan condiciones de cambio para los periodos y los tiempos de ejecución de los cómputos de fin-a-fin. Como consecuencia la reasignación de tareas a procesadores puede ser necesaria.

En la presente tesis, se propone la optimización del costo de reasignación de tareas para diferentes configuraciones del sistema. Se consideran procesadores idénticos y uniformes, la arquitectura de bus común y la arquitectura de arreglo lineal de procesadores. Se introducen funciones que miden el esfuerzo de realizar la reasignación de tareas a procesadores para cada modelo. Se proponen y analizan algoritmos óptimos y heurísticas. Además, se concluye que las heurísticas propuestas encuentran soluciones factibles para sistemas críticos de tiempo real y presentan un equilibrio entre el tiempo de ejecución y la calidad de la solución.

Palabras Clave: reasignación, calendarización de tiempo real, bus común, arreglo lineal de procesadores, *bin packing*.

ABSTRACT of the thesis presented by **Ricardo Garibay Martínez** as a partial requirement to obtain the MASTER OF SCIENCE degree in Computer Science. Ensenada, Baja California, México. September 2009.

ADAPTIVE RESOURCE MANAGEMENT IN DISTRIBUTED DINAMIC REAL-TIME SYSTEMS

Autonomously operating technical and adaptive systems as in mechatronics are dynamic applications that are able to adjust to changed operation conditions. Typically they are realized as distributed real-time systems requiring high flexibility in control. They have to deal with varying extrinsic parameters that cause changing conditions for the frequency and the run-times of the end-to-end computations. As a consequence, re-allocations of tasks to hosts will become necessary.

In this thesis, we focus on the optimization of the task re-allocation cost for different system configurations. Identical and uniform hosts, the common bus architecture and linear array of hosts are considered. Re-allocation cost functions that measures the effort of realizing a re-allocation of computational tasks to hosts are introduced for each model. Optimal algorithms and heuristics are proposed and analyzed. And we conclude that the proposed heuristics presents feasible solutions for critical real time systems and tradeoff between the execution time and suitable cost can be found.

Keywords: Reallocation, real-time scheduling, common bus, linear array of processors, bin packing.

Dedicatorias

A mi mamá y mi papá.
Son un gran ejemplo para mí.
Los quiero mucho.

Agradecimientos

A mi asesor y director de tesis el Dr. Andrei Tchernykh por introducirme en la ciencia, por sus sabios consejos y dedicación, pero sobre todo por su apoyo, paciencia y amistad.

A los miembros de mi comité de tesis, Dr. Carlos Alberto Brizuela Rodríguez, Dr. Mikhail Shlyagin y Dr. Klaus H. Ecker por su tiempo, dedicación y sus buenos consejos durante el desarrollo de esta tesis.

A mis padres, hermanos, tíos, primos y abuelos por su apoyo y palabras de aliento. ¡Vamos familia, uno más!

A mi novia Anna, a su mamá y su papá por su apoyo y por hacerme sentir como en casa.

Al departamento de ciencias de la computación. En especial al Dr. Andrei, Dr. Gilberto, Dr. Favela, Dr. Fernández, Dr. Brizuela, Dra. Ana, Dra. Josefina y Dr. Antonio, por contribuir en mi formación académica, pero sobre todo por tener una excelente calidad humana.

A los "charolastras" del CICESE por su invaluable amistad. A la generación 2007 por todo el tiempo compartido a lo largo de la maestría.

Al Centro de Investigación Científica y de Educación Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología por su apoyo económico.

CONTENIDO

CONTENIDO (continuación)

Página

CONTENIDO (continuación)

Página

LISTA DE FIGURAS

LISTA DE FIGURAS (continuación)

LISTA DE FIGURAS (continuación)

LISTA DE FIGURAS (continuación)

LISTA DE TABLAS

Capítulo I

Introducción

I.1 Antecedentes y motivación

Las aplicaciones de los sistemas de tiempo real distribuidos y empotrados son diversas. Estas pueden encontrarse en muchas áreas críticas, como en sistemas de defensa, control automotriz y de robots, o en posicionamiento satelital, entre otras. Los sistemas de tiempo real se usan para controlar ambientes técnicos, que fueron construidos para cumplir algún propósito específico. Dada su naturaleza no predecible no se comportan automáticamente como se espera. Para asegurar cierto comportamiento, algunos sensores informan acerca del comportamiento y estado del ambiente. Este proceso es conocido como computación fin-a-fin (*end-to-end* en inglés (Gerber, 1995)). En el caso que los datos medidos se desvíen de las especificaciones técnicas, se tienen nuevas configuraciones para los dispositivos del sistema. La lectura de sensores, su evaluación y la generación de señales para los dispositivos se debe hacer de manera periódica, con periodos dependientes de los requerimientos de control particulares.

Algunos ejemplos de esto son sistemas técnicos operando de manera autónoma, tal como satélites, o sistemas dinámicos adaptivos como aplicaciones mecatrónicas que son capaces de adaptarse a condiciones cambiantes (Ecker *et. al*., 2003 y Schomann, 2006). Para controlar estos ambientes, los parámetros extrínsecos informan permanentemente acerca del estado o condiciones de

trabajo del ambiente. El sistema controlador de los sistemas de tiempo real tiene que reaccionar dentro de intervalos de tiempo dados, para garantizar los parámetros de operación, o garantizar alguna calidad de servicio deseada.
Para caracterizar la carga de las tareas, los esquemas comunes de ingeniería de tiempo real usan tiempos de ejecución del peor caso, para después asignar los recursos de cómputo y de red, todo esto en tiempo de diseño. En contraste, las aplicaciones dinámicas requieren un alto grado de flexibilidad en control y tratan los parámetros extrínsecos que llevan a condiciones de cambio para los componentes de software de los sistemas de tiempo real. Esto puede tener consecuencias en las frecuencias y tiempos de ejecución de las tareas. Como consecuencia, los procesadores se pueden encontrar sobre- o sub-ocupados, y las tareas en el sistema tienen que ser re-distribuidas (reasignadas) entre los procesadores. Es obvio que tal reasignación entre procesadores no está libre de costo (por ejemplo, la transferencia de código y datos consume tiempo, poder de computo y capacidad de red). Además, estos pueden introducir retrasos adicionales, que impactan fuertemente en la factibilidad de calendarizar una tarea en un procesador dado (Ecker *et al*., 2004).

I.2 Planteamiento del Problema

La administración de recursos es un problema bien estudiado en sistemas tradicionales. Se han publicado muchos resultados de investigaciones relacionadas con la especificación y diseño de tales sistemas (Antsaklis *et. al*., 1993 y Grossman *et. al*., 1993), verificación y confiabilidad (Arora y Gouda, 1993) y calendarización (por ejemplo, (Blazewicz *et. al*., 2000 y Blazewicz *et. al*., 2007)).

Sin embargo, las estrategias utilizadas en estos enfoques no se pueden aplicar directamente a sistemas de calendarización adaptativos (Kuo *et. al*., 1997 y

Kalogeraki *et.al*., 2000), y en particular a sistemas adaptativos distribuidos de tiempo real.

Los sistemas de tiempo real interactúan con el ambiente, recibiendo impulsos del ambiente controlador y respondiendo con una acción que compense las necesidades del ambiente o sistemas controladores. Para imponer una cierta conducta, los sensores que informan acerca de la posición del ambiente se leen y procesan regularmente por los recursos computacionales. En este caso los cómputos de punta a punta pueden tener como resultado nuevos escenarios que necesitan ser controlados.

Los problemas mencionados anteriormente se pueden resolver por las supercomputadoras actuales, pero tienen el gran inconveniente de ser excesivamente costosas. De esta manera, se puede ver la importancia de la necesidad de desarrollar heurísticas para la asignación de tareas a recursos. Así el enfoque que se toma en la presente tesis es hacia los sistemas de tiempo real distribuido, presentando especial atención a los problemas de optimización en reasignación de tareas.

I.3 Objetivos de la investigación

I.3.1 Objetivo general

Diseñar heurísticas para la minimización de costos en la reasignación de tareas a procesadores, para diferentes modelos de sistemas de tiempo real. Tomando en cuenta las características de los procesadores y la topología de la red. Hacer un análisis de la calidad de las heurísticas diseñadas.

I.3.2 Objetivos específicos

1. Entender el comportamiento de los sistemas de tiempo real, calendarización de tareas, de los sistemas de tiempo real bajo las restricciones impuestas en ambientes distribuidos y altamente dinámicos y de los modelos relacionados con el problema de reasignación de tareas.
2. Proponer y analizar nuevas heurísticas para la solución del problema de reasignación de tareas.
3. Mostrar los métodos y los procedimientos (heurísticas) utilizados para llevar acabo la solución de la minimización del costo de reasignación de tareas.
4. Codificación e implementación de un simulador que ayude al cálculo de la métricas de optimización para cada modelo particular, en la reasignación de tareas hacia procesadores en sistemas distribuidos de tiempo real, el cual debe contar con las siguientes funcionalidades:
    a. Generador de cargas. Esto es con el objetivo de controlar y comparar los experimentos del simulador.
    b. Implementación de métodos y heurísticas. Se busca codificar varios métodos y heurísticas para la reasignación de tareas.
    c. Núcleo del simulador. El cual se encarga de la ejecución de los métodos y procedimientos propuestos.
5. Análisis de los resultados obtenidos durante la simulación. Evaluar las diferentes métricas de los métodos y heurísticas propuestas.
6. Encontrar resultados y conclusiones del análisis de las simulaciones.

I.4 Metodología de investigación

Para cumplir con los objetivos planteados se ha definido una metodología que consta de las siguientes etapas:

1. Análisis y revisión de la literatura.

   Se realiza una revisión bibliográfica de los sistemas de tiempo real, de algoritmos aplicados a sistemas de tiempo real, de calendarización y optimización combinatoria relacionada con el problema de reasignación, con el objetivo de conocer los diversos algoritmos y heurísticas existentes en la actualidad. Esto con el fin de comprender los modelos existentes en el área de reasignación de tareas y a partir de ello proponer nuevos modelos y heurísticas.

2. Estudio e implementación de métodos y algoritmos existentes.

   Con el fin de encontrar un punto de comparación se implementan algoritmos y heurísticas existentes (si es el caso). Con esto se busca aprovechar diversas técnicas ya existentes para tomar aquellos aspectos de utilidad para nuestro trabajo. Además, se busca tratar las cargas de trabajo, buscando así estimar la calidad de los resultados obtenidos.

3. Propuesta de modelos y heurísticas.

   Con la información obtenida en las etapas de revisión bibliográfica y el análisis de métodos existentes, se proponen nuevos modelos y heurísticas. Se busca desarrollar las métricas específicas para la evolución del desempeño de los algoritmos propuestos para el problema de reasignación de tareas.

4. Diseño del sistema y experimentación.

   Haciendo uso de las cargas de trabajo y de las métricas desarrolladas, se diseña el sistema de experimentación, para los diferentes modelos de reasignación de tareas.

5. Análisis experimental y presentación de conclusiones.

   Se realizan pruebas a través de simulación por computadora con el fin de concluir el comportamiento de las heurísticas propuestas para determinar su desempeño. El criterio de evaluación al que se someten las heurísticas propuestas incluyen la comparación con algoritmos conocidos (si es el caso), y el tratamiento estadístico de los datos: tratamiento descriptivo y estimación de características numéricas. Esto con el fin de presentar las conclusiones del presente trabajo.

I.5 Organización de la tesis

La presente tesis consta de siete capítulos organizados de la manera siguiente:

En el Capítulo II se presenta el marco teórico que servirá para entender todos los conceptos relacionados con la investigación, se introducen los conceptos de sistemas de tiempo real y su modelado matemático. Así como los conceptos de calendarización necesarios para comprender lo expuesto en este trabajo de investigación.

El Capítulo III presenta el objetivo del modelo de reasignación. También se introduce el modelo matemático que se utiliza en los modelos que se describen en los capítulos IV, V y VI.

El Capítulo IV presenta el modelo de arquitectura de bus común, donde las tareas se transmiten de manera secuenciada y la transmisión no depende de un procesador objetivo en particular. Así el objetivo es minimizar la suma ponderada de migración de tareas. También se describe el modelo particular, las heurísticas propuestas y los resultados de la experimentación.

El Capítulo V presenta el modelo de arquitectura de arreglo lineal de procesadores, la transmisión puede verse como un conjunto de actividades de transmisión con dependencias en cadena, sobre un conjunto de enlaces. Así el objetivo es minimizar la carga máxima sobre el conjunto de enlaces de transmisión. También se describe su modelo particular, las heurísticas propuestas y los resultados de la experimentación.

El Capítulo VI también describe la reasignación mediante técnicas de *bin packing*, el problema puede verse como un conjunto de tareas que se deben reasignar a procesadores, tomando en cuenta cambios en sus periodos de ejecución, y por tanto los requerimientos de procesamiento cambian. Así el objetivo es minimizar el número de fallas al realizar la reasignación de tareas a procesadores. También se describe las heurísticas propuestas y los resultados de la experimentación.

El Capítulo VII presenta las conclusiones y algunas ideas para la continuación de este trabajo de investigación.

Capítulo II

Sistemas de Tiempo Real

En este capítulo se introduce brevemente a los conceptos y requerimientos de los sistemas de tiempo real con especial atención a los aspectos de calendarización en plataformas multiprocesador y distribuidas.

Los sistemas de tiempo real se usan para el control industrial, médico, científico, de consumo, ambiental, entre otros. Éstos operan muy de cerca con sistemas técnicos llamados *ambientes* o *sistema externo*, estos pueden ser plantas de producción, plantas de energía, etc. El propósito de los sistemas de tiempo real, también llamado *sistema interno*, en contraste con el sistema externo, es asegurar algún comportamiento específico del ambiente (ver figura 1). Este comportamiento, se mantiene a través de las reacciones con restricciones temporales estrictas propias del ambiente, en el caso que el estado del ambiente se desvíe de la especificación requerida. Dependiendo de la naturaleza de la aplicación, el sistema interno tiene muchos procesos que se ejecutan repetidamente.

Algunos ejemplos son: la medición periódica de instrumentos que informan acerca del estado del ambiente controlado, la evaluación de los datos medidos, el cálculo de nuevos datos derivados de lo anterior, o la generación de señales para controlar el ambiente. Y por lo tanto, asegurar el funcionamiento esperado o requerido para el ambiente (Blazewics *et.al*., 2007).

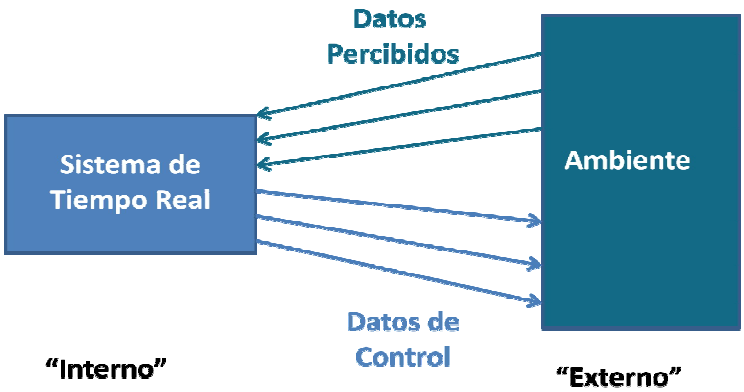

Figura 1. Comunicación entre el ambiente controlado y el sistema de tiempo real.

Además, una de las características principales de los sistemas de tiempo real es la fuerte interacción con su ambiente. Los sistemas técnicos demandan capacidad de cómputo y control de procesamiento por parte de sus sistemas de control, así como garantía de predictibilidad, confiabilidad y capacidad de operar dentro de sus limitaciones temporales.

II.1 Definición de los sistemas de tiempo real

La definición de sistemas de tiempo real dada por Koymans (Koymans *et al*., 1988)) abarca las características de un sistema de tiempo real de manera acertada:

Definición (Sistemas de Tiempo Real (Koymans *et al*., 1988) Un sistema de tiempo real se define como un sistema de interacción que mantiene una relación continua con un ambiente asíncrono, por ejemplo, un ambiente que trabaja sin relación alguna con el sistema de tiempo real en una manera no cooperativa. El sistema de tiempo real es responsable por completo de la sincronización adecuada de su operación con respecto al ambiente.

II.2 Características, requerimientos y estructura de los sistemas de tiempo real

A continuación se describen las características y requerimientos de los sistemas de tiempo real, estos dirigen el comportamiento especial de estos sistemas y los distinguen de los sistemas de cómputo tradicionales. También se introduce la estructura en la que se basa su funcionamiento.

II.2.1 Características de los sistemas de tiempo real

En los sistemas de tiempo real el comportamiento temporal es la cuestión más importante. Además, no solo el comportamiento temporal de los programas tiene que ser preciso, sino que también la teoría de calendarización se vuelve una disciplina clave. Es por esto que, cuestiones como el modo de operación computacional del sistema de tiempo real, la administración de los diferentes tipos de procesos, las primitivas de sincronización, las operaciones de calendarización de tareas, y la ejecución de tareas con restricciones temporales es fundamental. Por otro lado, la utilización del procesador es menos relevante, porque el costo de cualquier procesador involucrado en la falla de un proceso externo es insignificante comparado con el costo de los daños causados por la falla de un proceso. Esto siempre es verdad, incluso en ambientes relativamente no costosos. Es mas, es preferible cumplir con los requerimientos de predictibilidad y confiabilidad. Si una computadora no es capaz de garantizar los tiempos de reacción, puede ser que ésta no pueda hacer frente a situaciones excepcionales o de emergencia provenientes del ambiente y por tanto violar los requerimientos de seguridad. La dinámica de los sistemas físicos bajo control, impone restricciones temporales que pueden ser cumplidas, de ser así se puede dictar como alcanzar el comportamiento temporal deseado. Por lo tanto, el asegurar que la corrección

(funcional y temporal) del los sistemas de tiempo real es posible en especial para los sistemas altamente críticos (Blazewics *et.al*., 2007).

Además, existen ambientes con restricciones temporales suaves y rígidas. Estas se distinguen por las consecuencias que conlleva violar los requerimientos de puntualidad. Los ambientes de tiempo real suave se caracterizan por el costo creciente con respecto al retraso de los resultados. En los ambientes de tiempo real rígido tal retraso no se puede permitir bajo ninguna circunstancia, porque las reacciones de los cómputos tardíos, son inútiles o peligrosas. En otras palabas, el costo de no alcanzar las fechas límite en ambientes de tiempo real rígido son, desde el punto de la aplicación, considerados como infinitamente altos.

II.2.2 Requerimientos funcionales de los sistemas de tiempo real

Los sistemas de tiempo real tienen que cumplir con tres requerimientos básicos (Blazewics *et.al*., 2007), estos se explican a continuación:

- *Puntualidad:* Existen dos tiempos generales de requerimientos de puntualidad: restricciones temporales *relativas*, estas acciones se pueden ejecutar dentro de un intervalo de tiempo relativo con respecto a la ocurrencia de un evento interno o externo. *Absoluto*, las restricciones temporales especifican el comportamiento global del sistema para puntos específicos en el tiempo.

- *Predictibilidad:* el sistema controlador de tiempo real debe manejar cada evento externo de manera predecible con sus restricciones temporales asociadas. Además, las respuestas mostradas por la computadora se deben planear precisamente con la finalidad de conseguir un comportamiento completamente predecible.

- *Confiabilidad:* ésta se refiere a los requerimientos generales de seguridad de un trabajo correcto. El sistema tiene que producir las señales de control correctas en el tiempo correcto (corrección). Además, se debe contar con robustez en el sistema. La robustez hace referencia a que los requerimientos del sistema se mantengan en un estado predecible, incluso si el ambiente no responde a las especificaciones, por ejemplo, si las entradas no se encuentran dentro de un rango establecido. Otra condición es la tolerancia a fallas, lo que significa que un sistema no puede terminar (por ejemplo, como el resultado de una falla), y debe tolerar fallas de software o hardware.

II.3 Ejemplos de sistemas de tiempo real

Para un mejor entendimiento de los sistemas en cuestión, se presentan algunos ejemplos específicos de sistemas de tiempo real. Existen diferencias fundamentales entre las aplicaciones que se consideran de tiempo de real y las que no. Por ejemplo, podemos ver las diferencias entre un compilador y un programa para el control de un proceso químico, la compilación de un programa puede ser rápida o lenta, dependiendo de la velocidad de la maquina, el lenguaje utilizado, y el tamaño de programa. Un usuario puede tolerar más o menos el tiempo de compilación. En contraste, un programa para el control de un proceso químico, controla un proceso externo. Los pasos del programa se deben sincronizar con los eventos externos del proceso. Un diseño que no sea cuidadoso puede causar daños peligrosos en el ambiente, como en el caso del control de calor en un reactor químico. Algunos otros ejemplos son:

- En general, un ambiente como el de una fábrica de químicos tiene muchos componentes los cuales necesitan controlarse. Un ejemplo simple puede ser un solo componente en un sistema de gran tamaño, como el control de

flujo de líquido de una válvula a través de una tubería. El cálculo de un nuevo ángulo en la válvula puede ser ligeramente complejo.

- Usualmente los sistemas mecatrónicos requieren de una cooperación interdisciplinaria. Por ejemplo, los sistemas mecánicos clásicos proveídos con componentes electrónicos permiten la creación de nuevos productos de gran funcionalidad y adaptabilidad. Típicamente, el sistema electrónico monitorea el estado del sistema mecánico a través de sensores que transforman las señales de los actuadores para asegurar un rendimiento óptimo.

- Los sistemas de control de tráfico, sistemas complejos de control automotriz, sistemas de transporte y sistemas de manufactura asistidos por computadora pertenecen al amplio espectro de las aplicaciones de tiempo real.

La corrección de los sistemas de tiempo real no solamente depende de la lógica de los resultados obtenidos, sino también del tiempo en que estos resultados se producen (Blazewics *et.al*., 2007). Los requerimientos temporales pueden ir desde los milisegundos hasta las horas, días,…, incluso dentro de las mismas aplicaciones.

Otro punto importante es atender la necesidad de acciones de control. En el control de sistemas de tiempo real rígido, las fechas límite tienen que ser observadas, y cumplidas dentro de las limitaciones temporales específicas. En contraste, en los sistemas de tiempo real suave las acciones pueden retrasarse (fechas límite suaves), o los tiempos de respuesta establecidos, aunque rígidos, pueden no cumplirse ocasionalmente.

II.4 Estructura de los sistemas de tiempo real

A continuación se presenta el conjunto de actividades que se ejecutan dentro de un sistema real (ver figura 2). La estructura de cinco etapas distingue:

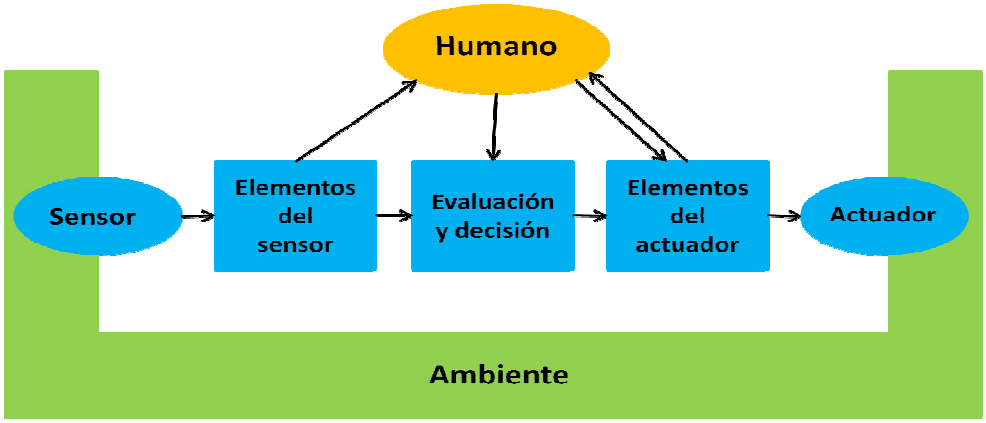

Figura 2. Estructura de cinco etapas de un sistema de tiempo real.

- *sensores*, los cuales informan acerca del estado del ambiente,
- *elementos del sensor*, los cuales consisten en módulos de software que ejecutan algunas acciones de pre-proceso sobre los datos sensados,
- la etapa de *evaluación y decisión*, es donde se verifica la validez del pre-proceso de los datos y donde se encuentra la relación con otros datos tales como datos históricos, además se calculan los requerimientos para los cambios en el ambiente. En sistemas grandes puede ser necesaria una base de datos o una base de conocimiento para la ejecución de las funciones requeridas,
- *elementos del actuador*, los cuales convierten los cambios requeridos en datos de control para los actuadores, y
- *actuadores,* los cuales son los dispositivos finales para cambiar el comportamiento del ambiente.

En general un sistema de tiempo real tiene muchos procesos de control. Se puede diferenciar entre muchos tipos de procesos (también llamados, rutas de fin-a-fin (Gerber, 1995)). Un proceso *periódico* se ejecuta repetidamente, una vez en un periodo de longitud fija, y la finalización de la ejecución no debe exceder su fecha límite. Un ejemplo típico es la lectura de datos de un sensor y la actualización del estado de las variables internas. Un proceso *asíncrono* o *aperiódico* responde a eventos internos y externos, pero se requiere que se conozcan con antelación los tiempos de petición para su ejecución. Usualmente está disponible la cantidad mínima de tiempo entre dos peticiones consecutivas y la fecha límite para una tarea que ha completado su ejecución.

II.5 Aspectos de calendarización

En general, la calendarización trata la asignación de tareas hacia recursos del sistema. Existen dos tipos de recursos del sistema, llamados procesadores y recursos adicionales. Los procesadores ejecutan las tareas de manera activa, los recursos adicionales, como por ejemplo la memoria, complementan la ejecución de las tareas en una manera pasiva (Blazewics *et. al.*, 2001, Liu, 2000).

Es en esta sección se presentan algunas definiciones básicas con el objetivo de permitir una descripción formal de los problemas de calendarización. Ésta descripción se enfoca en las definiciones necesarias para el presente trabajo. Información adicional puede encontrarse en (Blazewics *et. al.*, 2007, Liu, 2000, Pinedo, 2002).

Los siguientes tres conjuntos son necesarios para describir los problemas de calendarización:

- un conjunto de procesadores $\mathcal{P} = \{P_1, \dots, P_k\}$

- un conjunto de tareas $\mathcal{T} = \{T_1, \dots, T_n\}$
- un conjunto de recursos $R = \{R_1, \dots, R_s\}$

Además, en este trabajo se toman en cuenta procesadores *idénticos* (procesadores que ejecutan tareas con velocidades iguales) y *uniformes* (procesadores con iguales capacidades de procesamiento pero posiblemente a diferentes velocidades, además la velocidad no depende del tipo de tarea). Para información adicional puede consultarse (Blazewics *et. al*., 2007, Liu, 2000, Liu *et. al* 1973, Pinedo, 2002, Rajkumar, 1991).

II.5.1 Propiedades de las tareas

Una tarea T_j se describe con la ayuda de las siguientes características:

- el tiempo de procesamiento p_j es el tiempo de ejecución del peor caso,
- el tiempo de llegada r_j es el tiempo en que una tarea T_j es elegible para su ejecución,
- la fecha límite d_j es el instante de tiempo en el cual una tarea T_j debe completar su ejecución. Una tarea no puede comenzar de nuevo sin terminar antes su ejecución,
- el periodo π_j de una tarea T_j que se caracteriza por los intervalos iguales entre sus tiempos de llegada r_j,
- entre dos periodos consecutivos de T_j puede haber una variación positiva o negativa con límites de j^+ y j^-, respectivamente. Esto define la siguiente

restricción: Si el tiempo de inicio de la $i-esima$ instancia de T_j es $s_i \in [(i-1)\pi_j, i\pi_j]$, la siguiente instancia debe comenzar en el intervalo $[\max\{i\pi_j, i\pi_j + s_i - j^-\}, \min\{i\pi_j, i\pi_j + s_i + j^+\}]$,

- el peso w_j expresa la prioridad de la tarea T_j. En cualquier momento una tarea T_h prioridad más alta que la tarea ejecutándose actualmente T_l llega, T_l se interrumpe inmediatamente y T_h se inicia. Esto quiere decir que se supone que todas las tareas permiten interrupción,

- una tarea T_j puede tener restricciones de precedencia a las cuales se les llama tareas dependientes, de otra manera se les llama independientes. Para especificar restricciones de precedencia, se usa la relación de precedencia " $\prec$ " el cual se define sobre un conjunto de tareas $\mathcal{T}$. Si $T_i \prec T_j$ es verdad, T_j no puede inicializar antes de que T_j se ejecute por completo. En este caso, T_i es sucesor de T_j y T_j es el predecesor de T_i.

- un retraso de comunicación $c(i,j,s,t) \geq 0$ se puede dar por las tareas T_i y T_j ejecutados en sus respectivos procesadores P_s y P_t con $T_i \prec T_j$.

- una tarea T_j puede requerir recursos adicionales $R(T_j)$.

En todos los modelos que se describen con posterioridad, se supone que toda la información sobre las características de los procesadores y tareas se conoce a priori.

Existen tres tipos diferentes de tareas (Blazewics *et. al.*, 2007, Liu, 2000, Liu *et. al* 1973, Pinedo, 2002, Rajkumar, 1991):

- tareas periódicas π_j que tienen un periodo tal que la $i-esima$ instancia de T_j se procesa completamente en el intervalo $\left[(i-1)\pi_j, i\pi_j\right], i = 1, 2, 3, \ldots$

- tareas aperiódicas, las cuales ocurren en intervalos irregulares.

- tareas esporádicas que se ejecutan repetidamente, pero en diferentes instantes de tiempo. En lugar de un periodo se tiene un tiempo mínimo de llegada entre sus instancias. Si éste tiempo es conocido, la tarea se puede tratar como una tarea periódica con un periodo igual a su tiempo de llegada mínimo.

Dependiendo de cual sea el tipo de sistema de tiempo real, las tareas se pueden clasificar en tareas de tiempo real suave y tareas de tiempo real rígido. Para una tarea de tiempo real suave T_j la fecha límite d_j es vista como una fecha límite suave, en la cual, el tiempo para cumplir la ejecución de T_j no es tan estricto como las limitaciones impuestas por una fecha límite rígida. Además, los sistemas de tiempo real pueden contener ambas clases de tareas.

II.5.2 Propiedades de los recursos

Los recursos que proveen la misma función pueden ser vistos como recursos de un tipo singular. Puede haber muchos tipos de recursos. Los recursos $R = \{R_1, \ldots, R_s\}$ están disponibles en sus respectivas unidades $m_1, \ldots, m_s$. El vector de recursos requeridos

$$R(T_j) = \left[R_1(T_j), R_2(T_j), \ldots, R_s(T_j)\right]$$

especifican los recursos para la tarea T_j, donde

$$R_l(T_j), \tag{1}$$

con $0 \leq R_l(T_j) \leq m_l, l = 1, \ldots, s$, los cuales denotan el número de unidades requeridas. En los sistemas de tiempo real solo los recursos de uso exclusivo y no retirable son relevantes, debido a que solo este tipo de recursos son capaces de impedir la ejecución total de las tareas. Los recursos no retirables no son tomados en cuenta en el caso que las tareas se interrumpan por una tarea de más alta prioridad. Algunos ejemplos de recursos para el caso de aplicaciones de computadoras son: la capacidad de memoria primaria, la capacidad de almacenamiento en disco y los dispositivos de entrada/salida (Blazewics *et. al*., 2007, Liu, 2000).

II.5.3 Calendarios

A continuación se presenta la definición de calendario y calendario factible.

Definición (Calendario) *Un calendario $\tilde{S}$ es una asignación de procesadores $P \in \mathcal{P}$ y, si es necesario, los recursos $R \in \mathcal{R}$, a las tareas $T \in \mathcal{T}$ tal que las condiciones*

- *en cualquier momento cada tarea es procesada por a lo mas un procesador,*
- *en cualquier momento cada procesador es asignado a lo mas a una tarea,*
- *el procesamiento de una tarea T_j no puede comenzar antes de r_j,*
- *el número de interrupciones para una tarea es finito,*
- *la cantidad total de tiempo de procesador asignado a cada tarea es igual a su tiempo de procesamiento,*
- *si $T_i \prec T_j$ para cada tarea T_i y T_j se mantiene, T_j no puede iniciar antes de la ejecución total de T_i,*

- *posiblemente si existen restricciones de recursos, se tienen que satisfacer,*

se mantienen.

Definición (Calendario factible) *Un calendario $\tilde{S}$ se llama* ***factible*** *si cada tarea $T \in \mathcal{T}$ satisface sus restricciones temporales.*

En los sistemas de tiempo real el objetivo típico es encontrar un calendario factible (Blazewics *et. al*., 2007, Liu, 2000, Rajkumar, 1991).

II.6 Sistemas de tiempo real multiprocesador y distribuidos

Los sistemas de tiempo real de la actualidad, usualmente están formados por más de un procesador. La existencia de múltiples procesadores conlleva a nuevos problemas de calendarización, como la necesidad de asignación, la sincronización de procesos y retrasos de comunicación. La distribución de tareas hacia los procesadores se puede hacer estáticamente o dinámicamente. Estáticamente quiere decir que una vez que las tareas han sido asignadas siempre permanecen en el mismo procesador. Con distribución dinámica la distribución puede cambiar durante el transcurso del tiempo. Esto requiere estrategias de calendarización más sofisticadas o la separación de calendarización y asignación de tareas. (Liu, 2000, Rajkumar, 1991).

Los sistemas de tiempo real con varios procesadores pueden dividirse en sistemas multiprocesador y sistemas distribuidos. En ambos casos los procesadores se encuentran conectados por una red de comunicación. Los sistemas multiprocesador cuentan con una red de comunicaciones con una latencia baja, acceso a memoria compartida y emplean un calendarizador global. Por lo tanto, a estos se les llama fuertemente acoplados. Por otro lado, la red de comunicación

en los sistemas distribuidos tiene una latencia alta y el sistema no hace uso de memoria compartida. Cada nodo en el sistema tiene su propio calendarizador, así a estos se les llama débilmente acoplados. En un sistema distribuido un nodo es la unidad de procesamiento, el controlador de la comunicación y la interface de comunicación (Kopetz, 1997, Liu, 2000, Rajkumar, 1991).

Los sistemas distribuidos tienen numerosas ventajas respecto a los sistemas centralizados (Kopetz, 1997):

1. *Tolerancia a fallas*: Es más fácil asegurar la tolerancia a fallas en un sistema distribuido. Si un nodo falla, otros nodos son capaces de procesar su trabajo.

2. *Modularidad*: Frecuentemente los sistemas grandes están formados por subsistemas. Estos subsistemas se prueban detalladamente de acuerdo a sus propiedades, por ejemplo, no se debe perder la puntualidad durante la integración de una tarea dentro del sistema. A esta arquitectura del sistema se le llama de módulos o componentes. En un sistema distribuido la integración se alcanza con la ayuda de una red de comunicaciones. Por lo tanto un sistema distribuido es usualmente orientado a componentes.

3. *Escalabilidad*: Durante el transcurso del tiempo, el ambiente de un sistema de tiempo real puede requerir nuevas funcionalidades. Una arquitectura que permita y dé soporte a estas posibles extensiones se le llama escalable. Los sistemas distribuidos aseguran esta propiedad puesto que son capaces de admitir la agregación de nuevos nodos.

4. *Confiabilidad*: Con la propiedad de replicación, la confiabilidad se asegura fácilmente en los sistemas distribuidos. Debido a que un nodo se puede ver

como un sistema autónomo, el sistema ofrece más posibilidades para la detección de errores que un sistema centralizado.

5. *Bajo costo*: Los procesadores en los sistemas distribuidos son normalmente menos sofisticados comparados con los sistemas centralizados y además son más baratos. Por otro lado, la certificación de los sistemas de software es frecuentemente menos costoso. La certificación de software de seguridad crítica cuesta aproximadamente de dos a diez veces más que el software de seguridad no crítica. En los sistemas centralizados el software completo se tiene que certificar, en contraste, en los sistemas distribuidos se puede certificar solo las partes de seguridad crítica y las interfaces necesarias hacia las partes de seguridad no crítica.

Este trabajo presenta un enfoque a los sistemas distribuidos de tiempo real, donde cada nodo tiene su propio calendarizador y donde se observa de manera particular los retrasos causados por la comunicación entre nodos.

II.7 Estrategias de calendarización para sistemas distribuidos de tiempo real

En este trabajo la asignación de tareas se trata de manera separada a la calendarización de tareas. Por tanto ésta sección se enfoca a la calendarización de tareas periódicas en procesadores unitarios, con interrupción. En la siguiente sección se describe de manera breve las estrategias de calendarización para multiprocesadores.

A continuación se introducen las nociones básicas de utilización de procesador y los conceptos de prioridades estáticas y dinámicas, así como las de calendarización en línea *“on-line”* y fuera de línea *“off-line”*. También se introduce

brevemente a las estrategias de calendarización de regla de tasa monotónica (RM (*"Rate Monotonic"*), por sus siglas en inglés) y la fecha límite más tempranas primero (EDF (*"Earliest Deadline Fist"*), por sus siglas en inglés). La regla RM y EDF pertenecen a las estrategias mejor conocidas para la calendarización de tareas periódicas dentro de un procesador unitario.

II.7.1 La utilización del procesador

Para tener una medida básica del algoritmo de calendarización, se define la utilización del procesador.

Definición (Utilización del Procesador) *La **utilización del procesador** de una tarea T_j se define por*

$$u_j = \frac{p_j}{\pi_j} \tag{2}$$

para cada tarea T_j.

Definición (Utilización Total del Procesador) *Dado un conjunto de tareas $\mathcal{T}$ de n tareas,*

$$U = \sum_{j=1}^{n} \left(\frac{p_j}{\pi_j}\right) \tag{3}$$

*Es la **utilización total del procesador** de $\mathcal{T}$.*

La utilización total del procesador describe la fracción de tiempo que se gasta para la ejecución de un conjunto de tareas $\mathcal{T}$. Un calendario factible puede existir solo si

$U \leq 1$ se mantiene verdadero (Liu *et. al.*, 1973). Un límite superior de la utilización total se puede denotar por U^{max}, así el límite natural superior esta dado por U^{max} = 1. Dependiendo de la estrategia de calendarización aplicada, este límite puede ser más ajustado.

II.7.2 Prioridades estáticas vs. dinámicas

Los algoritmos de calendarización para sistemas de tiempo real están basados en asignación de prioridades dinámicas o estáticas.

Una prioridad estática w_j para las tareas $T_j \in \mathcal{T}$ se asigna a la tarea durante la etapa de diseño y no cambia durante la ejecución. Las prioridades estáticas pueden depender de las propiedades de una tarea, ya sea el periodo o el tiempo de ejecución. Además se supone que para cualquier par de tareas la prioridad no es igual.
Si diferentes valores w_j se asignan a las instancias de una tarea T_j, entonces se trata de prioridades dinámicas. El valor de w_j típicamente depende de algún parámetro de ejecución, por ejemplo, la fecha límite d_j, y puede cambiar durante el tiempo de ejecución (Blazewics *et. al.*, 2007, Liu *et. al.*, 1973).

II.7.3 Calendarización en línea vs. calendarización fuera de línea

Una decisión importante es escoger qué paradigma de calendarización utilizar, en línea o fuera de línea.

En calendarización en línea, el calendarizador decide que tarea se debe ejecutar en tiempo de ejecución. Este esquema presenta gran flexibilidad, especialmente en circunstancias imprevistas. Por otro lado, la ejecución del algoritmo de calendarización es consumidor de tiempo, y éste incrementa la carga de trabajo

del sistema. Frecuentemente, un calendario en línea se calcula a expensas de la calidad de la solución o incluso a expensas de su factibilidad, porque se cuenta tiempo disponible limitado. También es mucho más difícil satisfacer restricciones adicionales, como restricciones de precedencia, ya que estas son restricciones de decisión en las cuales una tarea que espere el término de la ejecución de otra, vuelve el caso más complicado.

En calendarización fuera de línea el calendario se calcula con anterioridad, por ejemplo, durante el diseño del sistema de tiempo real. Como sea, se le da más importancia a la calidad de la solución que al comportamiento de ejecución del algoritmo de calendarización. La flexibilidad no es tan alta como en la calendarización en línea, pero es fácil asegurar la observancia del tiempo, de los recursos y las restricciones de precedencia.

Las estrategias de calendarización que se introducen en la siguiente sección, llamadas regla RM y estrategia EDF, pueden ser usadas en ambos paradigmas (Ecker *et. al.*, 2004, Liu, 2000).

II.7.4 Estrategia de calendarización de tasa monotónica (*"RM Scheduling"*)

La regla RM es una estrategia de prioridades estáticas. La tarea T_j se procesa con una prioridad $w_j := \frac{1}{\pi_j}$. Si cualquier tarea con una prioridad más alta llega al sistema, interrumpe a una tarea con más baja prioridad y así puede comenzar su ejecución (Blazewics *et. al.*, 2007, Ecker *et. al.*, 2004, Liu, 2000, Liu *et. al.*, 1973, Sha *et. al.*, 1991).

Teorema. *Dado un conjunto* $\mathcal{T}$ *de* n *tareas con orden de prioridad RM, el límite superior para la utilización del procesador está dado por*

$$U = n\left(2^{\frac{1}{n}} - 1\right) \tag{4}$$

Demostración. (Liu *et. al.*, 1973)

Para una n suficientemente grande el límite se aproxima a $\ln 2$. En la práctica se puede alcanzar una utilización de cerca del 90% (Sha *et. al.*, 1991).

II.7.5 Estrategia de calendarización de fechas límite más tempranas primero (*"EDF Scheduling"*)

Aplicando la estrategia de fechas limite mas tempranas (EDF por las siglas en inglés de *Earliest deadline first*), las prioridades se asignan a las tareas de acuerdo a sus fechas límite respectivas. La tarea con la fecha límite más cercana tiene la más alta prioridad. En cualquier momento la tarea con alta prioridad interrumpe a una tarea de más baja prioridad y comienza su ejecución.

Debido a que las prioridades cambian constantemente durante el tiempo de ejecución, se considera a EDF como una estrategia dinámica (Blazewics *et. al.*, 2007, Liu, 2000, Liu *et. al.*, 1973).

El siguiente teorema tiene por objetivo mostrar que si existe un algoritmo de asignación basado en prioridades dinámicas que encuentra un calendario factible, la estrategia EDF también pude encontrar un calendario factible bajo el límite de utilización superior $U \leq 1$.

Teorema. *Dado un conjunto* $\mathcal{T}$ *de* n *tareas, la estrategia de calendarización EDF es factible si y solo si*

$$U \leq 1 \tag{5}$$

se mantiene.
Demostración. (Liu *et. al.*, 1973)

Se debe observar que EDF es óptimo, debido a que la regla de EDF puede encontrar un calendario factible, si es que éste existe (Blazewics *et. al.*, 2007, Lopez *et. al.*, 2004).

II.7.6 Otras estrategias de calendarización para procesadores unitarios

Otras estrategias de prioridades dinámicas son la del algoritmo de holgura mínima (ML (*"minimum laxity"*), por sus siglas en inglés) y el algoritmo de último tiempo de llegada (LRT (*"least release time"*), por sus siglas en inglés). El algoritmo ML, también conocido como el algoritmo de última holgura (*"least laxity algorithm"*, en inglés), calcula para cada tarea T_j la holgura l_j con $l_j = d_j - p_j$. Entonces la tarea con la holgura más pequeña es calendarizada con la más alta prioridad (Kalogeraki *et. al.*, 1999, Kalogerati *et. al.*, 2000, Kopetz, 1997, Liu, 2000, Kurose *et. al.*, 1991). El algoritmo LRT calendariza las tareas de manera inversa y consecuentemente trata a las fechas límite como tiempos de llegada y viceversa. Así los tiempos de llegada tardíos conllevan a una prioridad más alta en el calendario inverso. La ocurrencia de tiempos vacios es posible (Liu, 2000).

La regla de fechas límite monotónicas (DM (*"deadline monotonic"*), por sus siglas en inglés) es otra estrategia de calendarización de prioridades estáticas. A una

tarea T_j se le asigna una prioridad $w_j \coloneqq \frac{1}{d_j}$, esto quiere decir que la tarea con una fecha límite más corta es calendarizada con la más alta prioridad (Liu, 2000).

También existe una estrategia que mezcla prioridades estáticas y dinámicas. Aplicando esta estrategia, las k tareas con los periodos más cortos de un conjunto de tareas $\mathcal{T}$ con n tareas se calendarizan de acuerdo a la regla de RM, el resto de las tareas $n - k$ se insertan en los espacios vacios usando calendarización EDF (Liu, 1973).

II.7.7 Estrategias de calendarización para sistemas paralelos de tiempo real

Como se menciona en la Sección II.7.2 la asignación de tareas se puede hacer de manera estática y dinámica. En el caso de las asignaciones dinámicas los retrasos de comunicación de tareas tiene que ser considerado (Blazewics *et. al.*, 2007). Frecuentemente a una estrategia de calendarización multiprocesador con comunicación dinámica se le llama estrategia global, dado que el calendario para todo el sistema se obtiene de manera centralizada. La conexión de una estrategia de calendarización para procesadores unitarios con la comunicación dinámica conlleva a dos fases, llamadas la fase de asignación y la calendarización en cada procesador. A este tipo de calendarización se le conoce como estrategia de calendarización local.

La calendarización global tiene muchas desventajas. Las estrategias globales causan un sobre flujo muy alto comparado con la conexión de una estrategia de asignación con una estrategia de calendarización uniprocesador. La extensión de la estrategia EDF para multiprocesadores no es óptima, en contraste con la estrategia EDF clásica, especialmente el límite de utilización del procesador no es alta comparada con la versión de EDF para procesadores unitarios. El caso es el mismo para el límite de utilización para la versión multiprocesador de RM.

Además, el análisis para las estrategias globales es mucho más complejo. Además, los algoritmos multiprocesador con prioridades estáticas, no importando si son aplicados con asignación dinámica o estática, no pueden garantizar una utilización más alta que $\frac{(k+1)}{2}$ para k procesadores (Andersson *et. al.*, 2001). Es decir, para una k suficientemente grande se tiene una utilización $U \cong \frac{1}{2}$ por cada procesador.

Ahora se presenta una breve descripción de varias estrategias de calendarización para sistemas multiprocesador y consecuentemente se muestra la distinción entre estrategias locales y globales.

Existen muchas aproximaciones de calendarización global. En (Huo y Shin, 1994) se considera la probabilidad de llegada para las futuras tareas. Las tareas llegan a diferentes nodos y migran en algunos casos. El algoritmo de ahorro en (Qiao *et. al.*, 2001) trata de retrasar los comienzos de las tareas tanto como sea posible para incrementar la factibilidad de las tareas no calendarizadas. La estrategia de RM conectada con el algoritmo de primer-tamaño (*"fist-fit"*, en inglés), RMFF presentado por Dhall y Liu (Dall y Liu, 1978). En este caso las tareas son asignadas con la ayuda del algoritmo de *"first-fit"* y luego calendarizadas en cada nodo de acuerdo a la regla de RM. Incluso, existen aproximaciones basadas en estrategias de empaquetado en recipientes (*"bin packing"*, en inglés) multidimensional (Huang y Du, 1994), y en estrategias de ramificación y acotamiento (*"branch and bound"*) como en (Peng *et. al*,1997). Además, se ha estudiado ampliamente la regla de EDF basada en estrategias de *"bin packing"* (Lopez *et. al,* 2004).

Capítulo III

Modelo de reasignación

Como se menciona en la Sección I.2, la administración de recursos es un problema bien estudiado en sistemas tradicionales (Antsaklis *et. al*., 1993, Grossman *et. al*., 1993, Arora y Gouda, 1993, Blazewicz *et. al*., 2000 y Blazewicz *et. al*., 2007). Sin embargo, las estrategias utilizadas en estos enfoques no se pueden aplicar directamente a sistemas de calendarización adaptativos (Kuo *et. al*., 1997 y Kalogeraki *et.al*., 2000), y en particular a sistemas adaptativos distribuidos de tiempo real.

En este capítulo, en la Sección III.1 se describe la operación y los requerimientos de los sistemas distribuidos dinámicos de tiempo real. Posteriormente en la Sección III.2 se define la notación formal y la definición de costo para los modelos desarrollados en la presente tesis.

III.1 Reasignación de tareas en sistemas distribuidos dinámicos de tiempo real

La ingeniería común en tiempo real trata con ambientes los cuales operan en condiciones que no cambian durante un lapso de tiempo dado. En tal situación, el sistema de control encuentra condiciones de cronometraje bien definidas para los cómputos de punta a punta ejecutados periódicamente.

Ejemplos típicos son sistemas mecatrónicos, los cuales son entendidos como sistemas técnicos equipados con dispositivos digitales de control local o

distribuido. Además, una interfaz de usuario define las condiciones de funcionamiento para el sistema. Teniendo una perspectiva más detallada, el usuario pone direcciones para algún comportamiento específico del sistema. Estas direcciones son traducidas dentro de los requerimientos de control para el sistema de tiempo real, que toma la responsabilidad de la realización de las órdenes humanas de manera correcta.

El cambio de las exigencias de control conduce a los nuevos ajustes de los parámetros usados en las rutinas de control, y a nuevas condiciones de cronometraje para sus cómputos. Como consecuencia, los tiempos de cómputo de los procesos de control y parámetros de cronometraje como períodos y fechas límites son cambiados. Además, se supone que el sistema técnico trabaja en varios modos de operación diferentes, que tienen la influencia sobre los requerimientos del control en tiempo real (Schomann, 2006).

Un ejemplo es el proyecto de manejo automático de automóviles de la *Technical University of Clausthal*, donde variables extrínsecas son puestas por condiciones de operación como la velocidad del automóvil y la velocidad de las ruedas. Obviamente existen dependencias entre ellas. Las condiciones de funcionamiento pueden cambiar rápidamente, y el sistema de tiempo real tiene que adaptarse continuamente a modos de cambio. Como consecuencia, los modos de cambio de operación pueden conducir a la necesidad de migrar de un procesador a otro y así las acciones de control se distribuyen entre los procesadores.

Frecuentemente los ajustes de los parámetros extrínsecos no fijan completamente las condiciones de funcionamiento del sistema técnico y permiten variaciones conductuales. Por lo tanto se puede definir el concepto de calidad del servicio (QoS, *Quality of Service* por sus sigas en inglés). También, se usa como un criterio para seleccionar la siguiente mejor asignación.

El objetivo no es sólo hacer cumplir el sistema técnico para encontrar los requisitos de funcionamiento, sino también hacerlo de manera eficiente. Este concepto está basado en los parámetros de servicio, que se pueden variar para alcanzar la utilidad máxima posible del sistema (Schomann, 2006).
La ejecución de tareas, y consecuentemente el comportamiento del sistema es determinado por los valores de dos atributos, los atributos extrínsecos y los atributos del servicio. Los atributos extrínsecos expresan las condiciones o requerimientos funcionales. Ellos son colocados 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" dictado por el ambiente es el período de la tarea. 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 del servicio son entidades que pueden ser alteradas en algún tiempo por el sistema controlador. Si las condiciones externas (representadas por los atributos extrínsecos) cambian, los escenarios apropiados de los atributos del servicio permiten la adaptación a las nuevas condiciones de cambio.

III.2 Definición del modelo de reasignación

En esta sección se introduce el modelo general de reasignación y se introduce la definición de costo. En los Capítulos IV, V y VI, se describe el modelo de bus común, el modelo de arreglo lineal de procesadores y de reasignación mediante técnicas de *bin packing*, respectivamente. Al describir la notación en los capítulos IV, V y VI, siempre debe tomarse en cuenta la siguiente definición formal para el modelo de reasignación de tareas a procesadores.

Es claro, que la reasignación de tareas no esta libre de costo. El costo puede verse como: *el movimiento de las tareas dentro del sistema de tiempo real y su impacto en la factibilidad de calendarización de los mismos*. Por ejemplo, el mover

código y datos de un procesador a otro consume tiempo, poder de computo y capacidad de red. Además, se debe notar que las métricas de optimización dependen de la manera en que la red se encuentre conectada y/o de los requerimientos particulares del modelo estudiado.

El modelo que se introduce en esta sección sigue el modelo clásico de tareas periódicas (Bate y Burns, 2003 y Liu y Layland, 1973). Se supone la existencia de un conjunto de tareas $\mathcal{T} = \{T_1, \dots, T_n\}$; las cuales se ejecutan repetidamente dentro de un periodo dado $\pi_j := \pi(T_j)$. Esto es que cada tarea se debe ejecutar completamente en el intervalo $[(i-1)\pi_i, i\pi_i]$, $i = 1, 2, 3, \dots, n$. Cada tarea tiene un tiempo de procesamiento conocido p_j, que usualmente es el tiempo de ejecución del peor caso y un período asociado π_j. Entonces podemos decir que la tarea está caracterizada por una dupla de requerimientos de procesamiento $(p_j\ ,\ \pi_j)$. También se supone que las tareas en $\mathcal{T}$ son independientes entre si. Para el procesamiento de las tareas se usa un conjunto de procesadores $\mathcal{H} = \{H_1, \dots, H_m\}$. Una partición de tareas $(\mathcal{T}_1, \dots, \mathcal{T}_m)$ se define como una descomposición del conjunto de tareas $\mathcal{T}$ en subconjuntos $\mathcal{T}_1, \dots, \mathcal{T}_m$.Se busca asignar la partición de tareas $(\mathcal{T}_1, \dots, \mathcal{T}_m)$ debido a un cambio en los parámetros del sistema. Una asignación se especifica por una función de 1-1 $\alpha = (\mathcal{T}_1, \dots, \mathcal{T}_m) \rightarrow \{H_1, \dots, H_m\}$. Inicialmente, se supone una partición inicial $\mathcal{T}^{init}$ y una asignación inicial $\alpha^{init}(\mathcal{T}^{init})$. Entonces, durante el transcurso del tiempo una asignación puede cambiar. Así que podemos decir que en el tiempo actual se puede realizar una asignación actual $alloc^{cur} = \alpha^{cur}(\mathcal{T}^{cur})$ definida por alguna partición $\mathcal{T}^{cur}$ y mapeada por α^{cur}. También llamamos a la partición $(\mathcal{T}_1, \dots, \mathcal{T}_m)$ factible si existe una función 1-1 que cree una asignación factible. Nuestro punto de comienzo es por lo tanto una asignación factible $alloc^{cur} = \alpha^{cur}(\mathcal{T}^{cur})$. También, se supone que dados algunos cambios en los parámetros extrínsecos, se debe instalar una nueva asignación $\mathcal{T}^{new} = \{\mathcal{T}_1^{new}, \dots, \mathcal{T}_m^{new}\}$ para responder a los nuevos requerimientos del sistema de tiempo real. Esto significa que, las tareas de un mismo subconjunto

$\mathcal{T}_i^{cur}$ (y por tanto asignadas actualmente al mismo procesador) pueden encontrarse ahora en diferentes subconjuntos de $\mathcal{T}^{new}$ (y por lo tanto deben ser puestos en diferentes procesadores). Como consecuencia, algunas tareas tienen que migrar a otros procesadores. El objetivo es determinar el mapeo de $\alpha^{new}: \mathcal{T}^{new} \rightarrow \mathcal{H}$ tal que el calendario de transmisión sea minimizado. Mover una tarea T_j del procesador $H_i(\mathcal{T}_i^{cur})$ al procesador $H_k(\mathcal{T}_k^{new})$ incurre en un costo $c(T_j, H_i, H_k)$ que depende del peso de la tarea y de la estructura de la red. Si T_j permanece en el mismo procesador, entonces $c(T_j, H_i, H_i) = 0$.

Capítulo IV

Modelo I: Arquitectura de Bus Común

En este capítulo, en la Sección IV.1 se presenta la arquitectura de bus común, así como su definición formal. También se describe la correspondencia con el problema de maximización del peso total en una relación bipartita perfecta en la Sección IV.2. Así como las heurísticas propuestas y los resultados de la experimentación, en las secciones IV.4 y IV.5 correspondientemente. Finalmente, en la Sección IV.6 se observan las conclusiones relacionadas con este modelo.

IV. 1 Arquitectura de Bus Común

Para la comunicación entre los procesadores se supone un sistema de bus común, el cual usa el método de acceso al medio múltiple por detección de portadora, con detección de colisiones y arbitraje por prioridad de mensaje (*CSMA/CD+AMP*, por la abreviación en inglés de *Carrier Sense Multiple Access with Collision Detection and Arbitration Message Priority*). De acuerdo con este método, los nodos en la red que necesitan transmitir información deben esperar a que el bus esté libre (detección de portadora); cuando se cumple esta condición, dichos nodos transmiten un bit de inicio (acceso múltiple). En la figura 3 se muestra la arquitectura de bus común. Debido a que se supone un sistema de bus común, el costo de transmisión no es independiente de un recurso en particular o de un procesador objetivo (Ecker *et. al.*, 2004).

Se busca que el costo de reasignación sea minimizado, la optimización significa la minimización de la suma del peso de las tareas a migrar, donde los pesos representan el tamaño del código y de los datos a ser transmitidos.

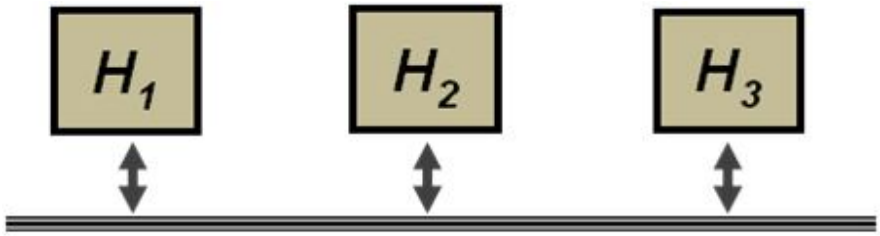

Figura 3. Arquitectura de bus común.

Las suposiciones de transmisión para la arquitectura de bus común mantienen las siguientes características:

- La técnica de transmisión es conmutación de paquetes,
- un nodo puede: enviar un mensaje, y
- recibir un mensaje simultáneamente,
- capacidad de buffer ilimitada en los nodos.

Entonces, si cada enlace tiene a lo más un paquete que transmitir, su costo total de transmisión está dado por la suma de los pesos de las tareas hacia los procesadores.

Para el caso de topología de bus común (CAN bus), realizar una nueva asignación $\propto^{new}: \mathcal{T} \rightarrow \mathcal{H}$ supone lo siguiente: mover una tarea T_j del procesador $H_i(\mathcal{T}_i^{cur})$ al procesador $H_k(\mathcal{T}_k^{new})$ incurre en un costo $c_w(j,i,k) = c(T_j, H_i, H_k)$. Así en términos de $\mathcal{T}^{cur}$, $\mathcal{T}^{new}$ y mapeados por α^{new}, la matriz de costos se puede calcular como sigue:

$$C(i,k) = \sum_{j=1}^{n} \{c_w(j,i,k) \mid (T_j \notin \mathcal{T}_i^{cur} \cap \mathcal{T}_k^{new} y\ \alpha^{new}(\mathcal{T}_k^{new}) = H_k)\}, \qquad (6)$$

para toda $i = 1, \ldots, m$ y $k = 1, \ldots m$, se define la matriz C.

El problema de minimizar el costo de reasignación puede ser formulado como sigue: dado $\mathcal{T}^{cur}$, α^{cur} y $\mathcal{T}^{new}$ encontrar α^{new} tal que el costo de reasignación sea minimizado. Entonces, el problema de reasignación para el caso de arquitectura de conexión con bus común se puede formular encontrar el costo de reasignación $c(r)$, tomando en consideración lo siguiente:

$$Dada\ la\ matriz\ de\ costos\ encontrar\ la\ matriz\ de\ permutación\ Q^r(i,k)$$
$$tal\ que, c(r) = \sum_{i=1}^{m}\sum_{k=1}^{m} C(i,k) \cdot Q^r(i,k)\ \ sea\ minimizado, \qquad (7)$$

para $r = 1, \ldots m!$

IV.1.1 Ejemplo: arquitectura de bus común

Para mostrar el comportamiento de transmisión para el caso de bus común se presenta el siguiente ejemplo. Suponer $m = 3$ procesadores, $n = 9$ tareas, una asignación actual $\mathcal{T}_1^{cur} = \{T_1, T_2, T_3\}$ en H_1, $\mathcal{T}_2^{cur} = \{T_4, T_5\}$ en H_2 y $\mathcal{T}_3^{cur} = \{T_6, T_7, T_8, T_9\}$ en H_3. Los costos de transmisión se observan en la tabla I:

Tabla I. Costos de transmisión para tareas para la arquitectura de bus común.

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

Suponer también que se necesita asignar los nuevos subconjuntos $\mathcal{T}_1^{new}$ = $\{T_1, T_6, T_7\}$, $\mathcal{T}_2^{new}$ = $\{T_2, T_4, T_8\}$ y $\mathcal{T}_3^{new}$ = $\{T_3, T_5, T_9\}$. Los costos de transmisión de los elementos de $\mathcal{T}_i^{new}$ en H_k para i = 1,2,3 y k = 1,2,3 se muestran en la figura 4:

	$\mathcal{T}_1^{new}$	$\mathcal{T}_2^{new}$	$\mathcal{T}_3^{new}$
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 Costos

$$\begin{pmatrix} \mathbf{4} & 10 & 12 \\ 10 & 9 & \mathbf{6} \\ 6 & \mathbf{9} & 10 \end{pmatrix}$$

Figura 4. Ejemplo del cálculo de la matriz de costos para arquitectura bus común.

Entonces, instalar T_1^{new} en H_1 implica: mover T_6 y T_7 a H_1 con un costo de 3 + 1 = 4. El cual ocupa la posición C_{11} en la matriz de costos correspondiente. Instalar T_2^{new} en H_1 implica: mover T_4 y T_8 a H_1 con un costo de 5 + 5 = 10. Y de la misma manera, instalar T_1^{new} en H_2 implica: mover T_1, T_6 y T_7 a H_2 con un costo de 6 + 3 + 1 = 10. 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, que para el anterior ejemplo representan una solución óptima. La nueva asignación correspondiente es:

$$\alpha^{new}(T_1^{new}) = H_1, \alpha^{new}(T_2^{new}) = H_3, \alpha^{new}(T_3^{new}) = H_2. \quad (8)$$

IV.2 El problema de asignación

El problema de reasignación en donde se requiere encontrar una asignación: de tareas hacia procesadores tal que cada procesador procese la tarea asignada dentro de un intervalo de tiempo dado.

Existen diferentes ejemplos de problemas de asignación que nos ayudan a entender un poco más 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. Esto se puede modelar a través de grafos, por medio del grafo podemos ver un conjunto de aristas $P \cup J$. Esto quiere decir que si una persona P_i es conveniente para un cierto trabajo J_i, entonces existe una arista entre P_i y J_i en el grafo. Esto es, existe una relación que los une, de manera análoga se pueden realizar asignaciones correctas de tareas a procesadores.
El modelo de reasignación depende de manera directa de la topología que se use para la transmisión de las tareas entre procesadores. Así que podemos modelar el problema de conmutación de paquetes en sistemas de tiempo real con la ayuda de grafos.

IV.2.1 Grafos bipartitos

En el campo de la teoría de grafos, un grafo bipartito es un grafo especial, también llamado bi-grafo, el cual es un conjunto de vértices que puede ser dividido en dos conjuntos no-conexos V_1 y V_2 tal que cada arista tenga un punto final en V_1 y un punto final en V_2.

Un grafo bipartito es un caso especial de un grafo $k-partito$ (un grafo $k-partito$ es un grafo cuyos vértices de grafo pueden ser divididos en conjuntos k no-conexos de modo que ningún par de vértices dentro del mismo conjunto sean adyacentes) con k = 2. En la figura 5 se muestra un ejemplo de un grafo bipartito.

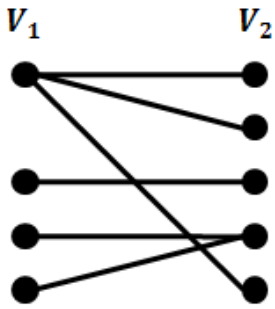

Figura 5. Ejemplo de un grafo bipartito.

Un grafo dirigido $G \coloneqq (V, E)$ es llamado bipartito si allí existe una partición del conjunto de vértice $V = V_1 \cup V_2$ de modo que tanto V_1 como V_2 sean conjuntos disjuntos. A menudo se escribe $G \coloneqq (V_1 + V_2, E)$ para denotar un grafo bipartito cuya partición tiene los conjuntos V_1 y V_2. Si $|V_1| = |V_2|$, esto es, si el número de elementos en V_1 es igual al número de elementos en V_2, entonces G es llamado un grafo bipartito balanceado.

IV.2.2 Grafo bipartito completo

En el campo de la teoría de grafos, un grafo bipartito completo es una clase especial de grafo bipartito donde cada vértice del primer conjunto es conectado a cada vértice del segundo conjunto a través de una arista. La figura 6 muestra dos ejemplos de grafos bipartitos $K_{m,n}$.

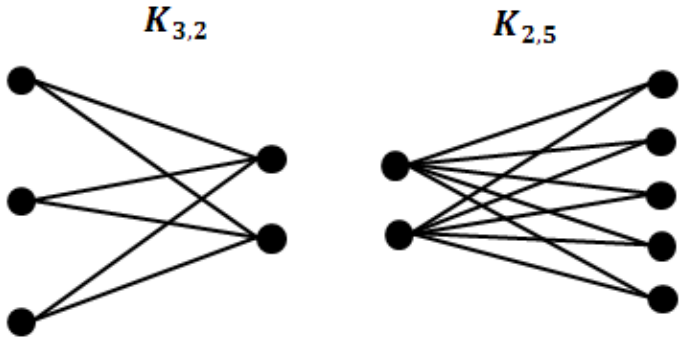

Figura 6. Ejemplos de un grafo bipartito completo $K_{m,n}$.

Un grafo bipartito completo $G \coloneqq (V_1 + V_2, E)$ es un grafo bipartito tal que para cualquiera de los dos vértices $v_1 \in V_1$ y $v_2 \in V_2$ existe una arista en G. El grafo bipartito completo con conjuntos de tamaño $|V_1| = m$ y $|V_2| = m$ se denota como $K_{m,m}$. La figura 7 muestra dos ejemplos de grafos bipartitos $K_{m,m}$.

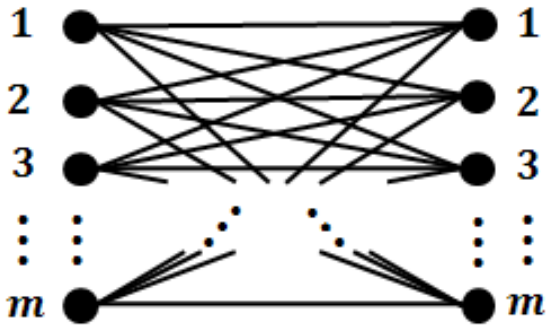

Figura 7. Ejemplo de un grafo bipartito completo $K_{m,m}$.

IV.2.3 Procesadores idénticos

Si los procesadores son idénticos, es decir, tienes las mismas capacidades de procesamiento a mismas velocidades, entonces cualquier función 1-1 $\alpha^{new}: \mathcal{T} \rightarrow \mathcal{H}$ define una asignación factible, pero posiblemente a un costo diferente. En orden de resolver el problema de optimización se transforma el problema de relación máxima de pesos en grafos bipartitos.

Dado un grafo bipartito con un peso en las aristas, encontrar una relación perfecta o emparejamiento perfecto, para la cual la suma de los pesos de los arcos escogidos sea el máximo. Una relación perfecta es una relación en la cual se relacionan todos los vértices del grafo. Esto es, cada vértice del grafo es incidente exactamente por una arista durante el emparejamiento.

Para demostrar la equivalencia entre el problema de asignación de tareas a procesadores y el problema de encontrar una relación perfecta en un grafo G, se

debe notar que el grafo bipartito completo es G = ($S, T, S\times$T), donde $|S| = |T| = m$ corresponden a una matriz cuadrada $m\times m$ con $C = (C_{ij})$ donde C_{ij} corresponde a los costos de las aristas apropiadas. Como se busca solucionar el problema de relación máxima de pesos, se tiene que hacer un ajuste para encontrar la relación mínima de pesos (como corresponde al problema en cuestión) como sigue; se supone que $K - C_{ij}, i = 1, \dots, m, j = 1, \dots, m$ para una $K >$ max $\{C_{ij}\}$.

Se puede observar que la minimización del costo de reasignación para arquitectura de comunicación de bus común corresponde al problema de maximización del peso total en una relación bipartita perfecta.

IV.2.4 Procesadores uniformes

Si los procesadores se suponen uniformes, esto significa que todas las tareas se pueden procesar en cualquier procesador H_i, pero posiblemente a diferentes velocidades. Además, las capacidades de memoria de los procesadores también pueden diferir. Entonces, si se supone que las capacidades de procesamiento pueden ser diferentes para los H_i para $i \in 1, \dots, m$, puede ocurrir que no todos los procesadores son capaces de procesar los subconjuntos de tareas $\mathcal{T}_j^k$.

Por lo tanto las asignaciones posibles se pueden ver restringidas, esto es que no todas las asignaciones 1-1 $\alpha^{new}: \mathcal{T} \to \mathcal{H}$ se mantienen factibles. Si el tiempo de ejecución de una asignación actual $\alpha^{cur}(\mathcal{T}^{cur})$ cambia se debe realizar una nueva asignación $\mathcal{T}^{new}$, pero se debe verificar cuáles procesadores són capaces de procesar los elementos de $\mathcal{T}^{new}$. Es así qué se debe definir que subconjunto de procesadores H_i son capaces de ejecutar las tareas en $\mathcal{T}_j^{new}$, $j = 1, \dots, m$.

Si se parte de la suposición general de que existe una asignación factible para $\mathcal{T}^{new}$, se puede concluir que el conjunto H_i no es vacío. Es mas, debido a que

cada procesador tiene que procesar exactamente un subconjunto de $\mathcal{T}^{new}$, se debe tener $\bigcup_{i=1}^{m} H_i = H$. Tal restricción se puede modelar en la matriz de costos C excluyendo ciertos elementos considerados de la matriz de costos. Para esto se introducen valores "no-posibles" en las posiciones correspondientes de C. Se debe notar que el grafo bipartito correspondiente no es completo. El algoritmo del problema de maximización del peso total (que se presenta en la Sección IV.2.3), se puede usar introduciendo valores suficientemente grandes (representando los valores "no posibles") a los elementos de la matriz de costos para así excluirlos de la solución del problema de minimización.

IV.3 Algoritmo Húngaro

El algoritmo húngaro es un algoritmo de optimización combinatorio que soluciona problemas de asignación en tiempo $O(m^3)$. Para resolver el problema anterior se puede usar el método húngaro propuesto por W. Kuhn, Lawler (Kuhn, 1956) el cual muestra que de acuerdo a las estructuras de datos se puede reducir la complejidad de este método en tiempo $O(m^3)$, y ha sido conocido como el algoritmo húngaro, el algoritmo de asignación Munkres, o el algoritmo de Kuhn-Munkres.

El algoritmo realiza la minimización sobre los elementos de la matriz. Los pasos que sigue el algoritmo húngaro son los siguientes (Peltola, 2004). Un ejemplo de ejecución del algoritmo Húngaro se muestra en el Apéndice B.

Algoritmo Húngaro

1. **Inicio**
2. **para cada** fila $i = 1,..,m$ **hacer**
3. restar las entradas de la fila por el mínimo de la fila.
   (a. cada fila tiene al menos un cero)
   (b. todas las entradas son positivas o cero)
4. **fin para**
5. **para cada** columna $j = 1,..,m$ **hacer**
6. restar las entradas de la columna por el mínimo de la columna.
   (a. cada fila y cada columna tienen al menos un cero)
7. **fin para**
8. **asignar a bandera el valor 0**
9. **hacer**
10. seleccionar las filas y columnas a través de las cuales se dibujen líneas, de tal
    modo que todos los ceros sean cubiertos y que no mas líneas necesarias hayan sido
    dibujadas.
11. **si** (el número de las líneas es igual a *m*) **entonces**
12. escoger una combinación modificada para el costo de la matriz de tal
    manera que la suma sea cero.
13. bandera = 1.
14. **fin si**
15. **si** (el número de las líneas es < *m*) **entonces**
16. encontrar el elemento más pequeño el cual no es cubierto por ninguna
    de las líneas.
17. restárselo a cada entrada que no ha sido cubierta por las líneas.
18. sumarlo a cada entrada la cual está cubierta por una línea vertical y
    horizontal
19. **fin si**
20. **mientras** (bandera igual a 0)
21. **Fin**

IV.4 Heurísticas para la minimización del costo de transmisión: bus común

La complejidad en tiempo del algoritmo húngaro, el cual consigue obtener soluciones óptimas es $O(m^3)$. Desde el punto de vista de los sistemas de tiempo real, y en especial de los sistemas críticos esta complejidad en tiempo no es necesariamente satisfactoria. Es por esto que se propone una serie de heurísticas

que mejoran la complejidad en tiempo a costa de calidad en la solución. El algoritmo óptimo y diferentes métodos heurísticos para la reasignación de tareas se describen a continuación:

Húngaro: el algoritmo húngaro es optimó al resolver el problema de asignación lineal. La complejidad en tiempo es $O(m^3)$. Para más detalles ver Sección IV.3.

Min: el cual para una matriz de costos dada C *Min* escoge el valor más pequeño para cada renglón $i = 1,..,m$, considerando las columnas que no estén seleccionas. Esto es, al escoger un elemento de un renglón, la respectiva columna para $j = 1,..,m$ se marca como seleccionada y los elementos en la columna no se pueden considerar para las siguientes iteraciones de la heurística. El pseudocódigo se muestra a continuación. La complejidad en tiempo es $O(m^2)$.

Heurística Min

```
1.  Inicio
2.      para cada fila i = 1,..,m hacer
3.          encontrar el valor mínimo de la fila (considerando las columnas que no estén
            seleccionadas)
4.          asignarla el mínimo a la solución
5.          marcar el renglón y columna del elemento seleccionado (para que no pueda
            asignarse en otra iteración)
6.      fin para
7.  Fin
```

La figura 8 muestra el comportamiento de la heurística *Min* en (a) se observa la matriz sin cambios, al inicio de la heurística. En (b) se muestra con el valor mínimo del renglón uno se encuentra, se asigna a la solución y se marca el renglón y la columna. En (c) se muestra el mismo proceso mostrado en (b), para la segunda

iteración de la heurística y en (d) se muestra la tercera iteración de la heurística *Min*.

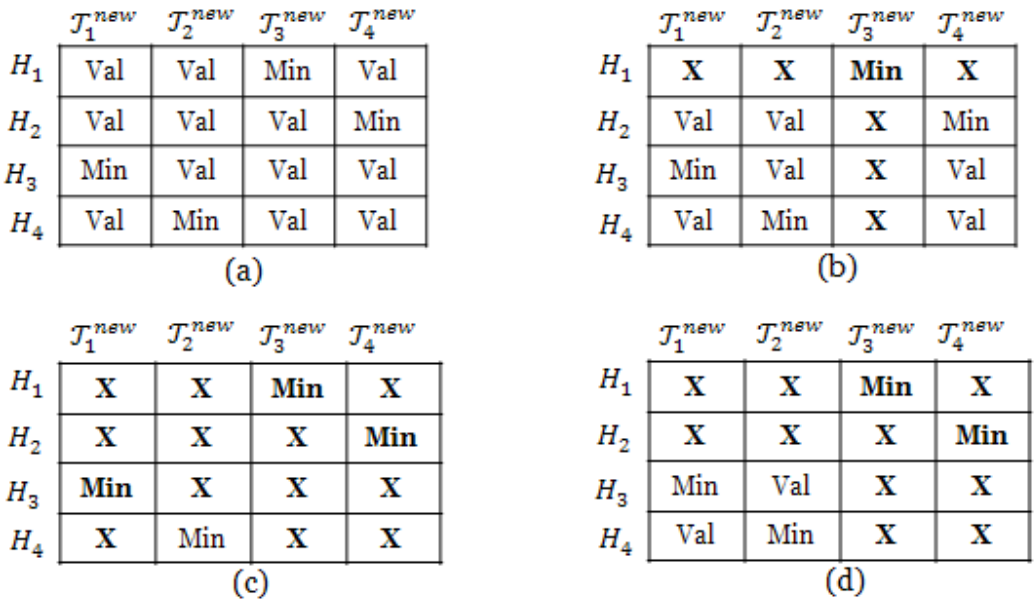

***Figura 8. Ejemplo heurística Min, (a) matriz C_{ij} al inicio de la ejecución, (b) matriz C_{ij} durante la primera iteración, (c) matriz C_{ij} durante la segunda iteración, (d) matriz C_{ij} durante la tercera iteración*.**

Rand: el cual para una matriz de costos dada C *Rand* escoge un procesador aleatoriamente el valor más pequeño para cada $i = 1,..,m$, considerando las columnas que no estén seleccionas. Esto es, al escoger un elemento de un renglón, la respectiva columna para $j = 1,..,m$ se marca como seleccionada y los elementos en la columna no se pueden considerar para las siguientes iteraciones de la heurística. El pseudocódigo se muestra a continuación. La complejidad en tiempo es $O(m^2)$.

La figura 9 muestra el comportamiento de la heurística *Rand* en (a) se observa la matriz al inicio de la heurística, esto es sin cambios. En (b) se muestra qué un valor aleatorio del renglón uno, se encuentra, se asigna a la solución y se marca el

renglón y la columna. En (c) se muestra el mismo proceso mostrado en (b), para la segunda iteración de la heurística y en (d) se muestra la tercera iteración de la heurística *Rand*.

Heurística Rand

1. **Inicio**
2. **para cada** fila $i = 1,..,m$ **hacer**
3. encontrar un valor de manera aleatoria de la fila (considerando las columnas que no estén seleccionadas)
4. asignarla a la solución
5. marcar el renglón y columna del elemento seleccionado (para que no pueda asignarse en otra iteración)
6. **fin para**
7. **Fin**

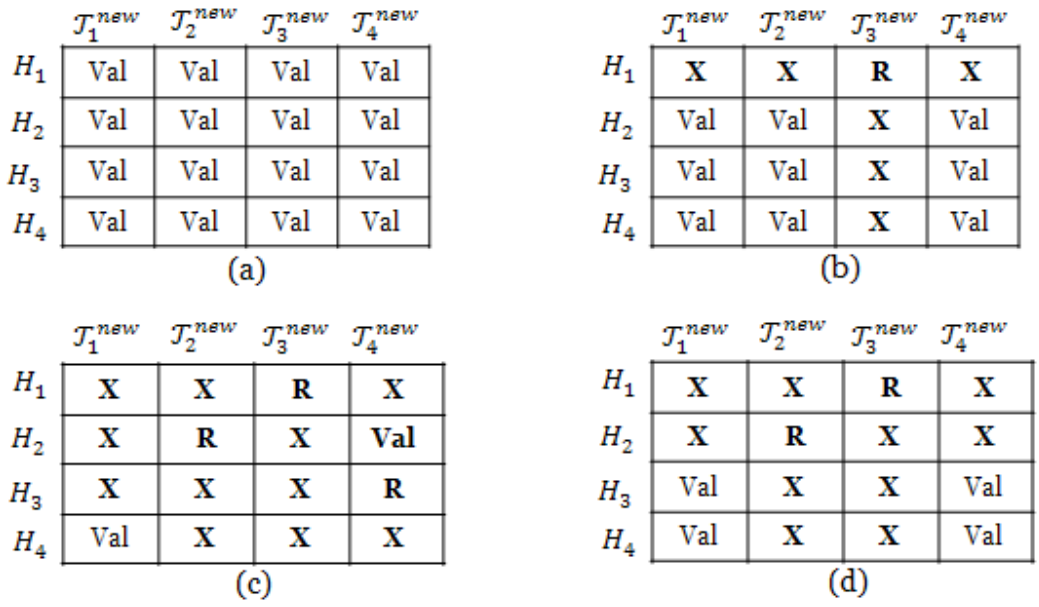

***Figura 9. Ejemplo heurística Rand, (a) matriz C_{ij} al inicio de la ejecución, (b) matriz C_{ij} durante la primera iteración, (c) matriz C_{ij} durante la segunda iteración, (d) matriz C_{ij} durante la tercera iteración*.**

Diag: el cual selecciona los elementos de la diagonal de la matriz de costos C. El pseudocódigo se muestra a continuación. La complejidad en tiempo es $O(m)$.La figura 10 muestra el comportamiento de la heurística *Diag* en (a) se observa la matriz al inicio de la heurística y en (b) se muestra qué la diagonal seccionada por la heurística *Diag*.

Heurística Diag

```
1.  Inicio
2.      para cada fila i = 1,..,m hacer
3.          asignarla a la solución la celda con numero de fila y columna iguales (asignar la
            diagonal)
4.      fin para
5.  Fin
```

	T_1^{new}	T_2^{new}	T_3^{new}	T_4^{new}
H_1	Val	Val	Val	Val
H_2	Val	Val	Val	Val
H_3	Val	Val	Val	Val
H_4	Val	Val	Val	Val

(a)

	T_1^{new}	T_2^{new}	T_3^{new}	T_4^{new}
H_1	**D**	Val	Val	Val
H_2	Val	**D**	Val	Val
H_3	Val	Val	**D**	Val
H_4	Val	Val	Val	**D**

(b)

Figura 10. Ejemplo heurística Diag, (a) matriz C_{ij} al inicio de la ejecución, (b) diagonal seleccionada de la matriz C_{ij}.

Diag_swap: selecciona los elementos de la diagonal en C como una solución inicial, entendiendo que la diagonal en C representa una asignación valida (para el caso de procesadores idénticos). Entonces la operación "swap" ampliamente conocida en la literatura se aplica m veces. Los *swaps* se realizan utilizando un vector de posiciones de tamaño m, donde el subíndice i representa el valor del renglón y el valor contenido en la celda representa el valor de la columna j. La complejidad en tiempo es $O(m)$.

La figura 11 muestra el comportamiento de la heurística *Diag_swap* en (a) se observa la matriz al inicio de la heurística. En (b) se muestra qué se toma la diagonal como solución inicial. En (c) se muestra qué se generan los valores R1=2 y R2=3 y se realiza el intercambio en el vector_posición y en (d) se muestra un ejemplo de la asignación final después de aplicar el operador *swap*.

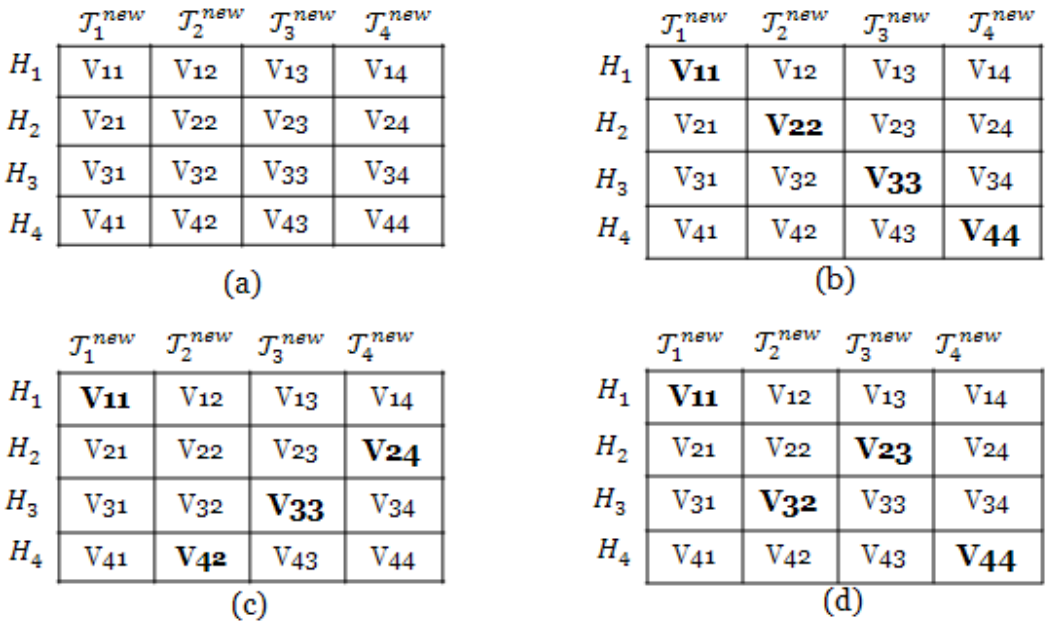

***Figura 11. Ejemplo heurística Diag_swap, (a) matriz C_{ij} al inicio de la ejecución, (b) selección de la matriz C_{ij}, (c) matriz C_{ij} durante la primera iteración, (d) matriz C_{ij} al final de la ejecución de la heurística Diag_swap*.**

El pseudocódigo se muestra a continuación.

Heurística Diag_swap

1. **Inicio**
2. tomar los elementos de la diagonal como solución inicial y asignarlos al vector de posición
   (esto es *i* = *j*)
3. **para** i = 1 hasta m **hacer**
4. generar dos números aleatoriamente R1 y R2
5. Intercambiar los valores contenidos en vector_posición[R1] por vector_posición[R2]
6. **fin para**
7. asignar las posiciones en vector_posición a la solución, donde el subíndice i es el valor del
   renglón y el valor contenido en la celda representa el valor de la columna j.
8. **Fin**

Eval_swap: es una modificación de Diag_swap. El operador swap se aplica sólo si el costo se reduce. La complejidad en tiempo es $O(m + k)$, donde k es el número de swaps que se ejecutan. El pseudocódigo se muestra a continuación.

Heurística Eval_swap

1. **Inicio**
2. tomar los elementos de la diagonal como solución inicial y asignarlos al vector de posición
   (esto es *i* = *j*)
3. Asignar un valor escogido k
4. **para** i = 1 hasta k **hacer**
5. generar dos números aleatoriamente R1 y R2
6. **si** (costo encontrado por el intercambio entre vector_posición[R1] y vector_posición[R2] es menor) **entonces**
7. Intercambiar los valores contenidos en vector_posición[R1] por vector_posición[R2]
8. **fin si**
9. **fin para**
10. asignar las posiciones en vector_posición a la solución, donde el subíndice i es el valor del
    renglón y el valor contenido en la celda representa el valor de la columna j.
1. **Fin**

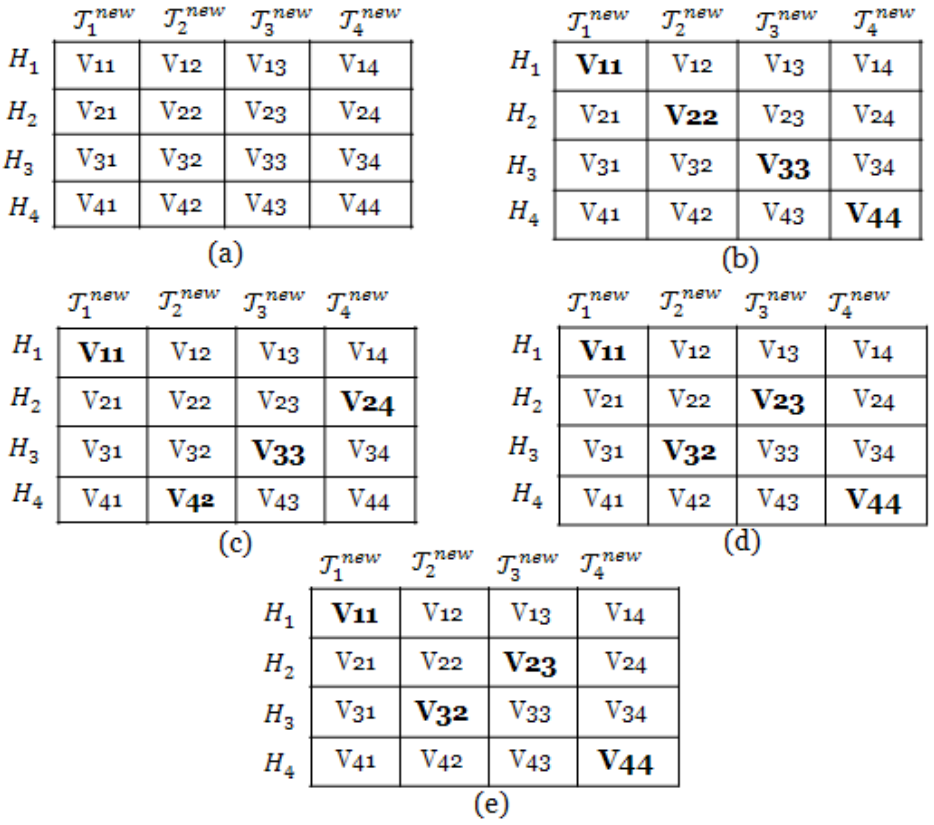

Figura 12. Ejemplo heurística Eval_swap, (a) matriz C_{ij} al inicio de la ejecución, (b) selección de la matriz C_{ij}, (c) matriz C_{ij} durante la primera iteración, (d) matriz C_{ij} después de varias iteraciones con cambio y (e) matriz C_{ij} sin cambio.

La figura 12 muestra el comportamiento de la heurística *Eval_swap* en (a) se observa la matriz al inicio de la heurística. En (b) se muestra qué se toma la diagonal como solución inicial. De manera similar a la figura #, en (c) se muestra qué se generan los valores R1=2 y R2=4, además, se realiza el intercambio ya que el costo de encontrado por el intercambio de celdas es menor. En (d) se muestra el estado de la matriz de costos después de aplicar el operador *swap*. En (e) se observa que la matriz en (d) no cambia, esto es, porque el intercambio en el vector_posición para los valores generados R1 y R2 no disminuye el costo obtenido.

IV.5 Resultados y análisis experimental: bus común

La figura 13 muestra la relación entre el número de swaps y la calidad de la solución. Se muestra el promedio (sobre 30 experimentos) de la reducción del costo versus el incremento del número porcentual de swaps y su diferencia absoluta. El número máximo de swaps a probar es m^2 (100%). Los elementos de la diagonal se escogen como solución inicial y su costo es considerado como el 100%.

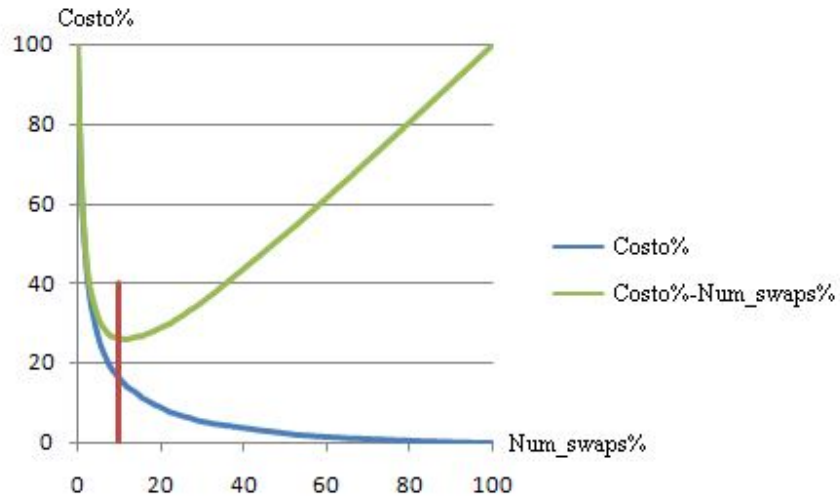

Figura 13. Reducción del costo versus incremento del número de swaps para arquitectura de bus común aplicando la heurística Eval_swap.

Se puede observar que al aplicar Eval_swap la calidad de la solución mejora. El 10% del incremento en el número de swaps consigue mejorar la solución inicial cerca del 80% en la reducción del costo. La reducción del costo disminuye con el incremento del número de *swaps*. Así, podemos encontrar una relación entre el número de *swaps* probados y la calidad de la solución.

Por otro lado, es importante considerar que los tiempos de ejecución de las heurísticas propuestas. En la figura 14 se muestran los tiempos de ejecución de

cada una de las heurísticas sobre el tiempo de ejecución del algoritmo húngaro. Esta es la razón de cuantas veces las heurísticas son mas rápidas en ejecución comparadas con el algoritmo óptimo.

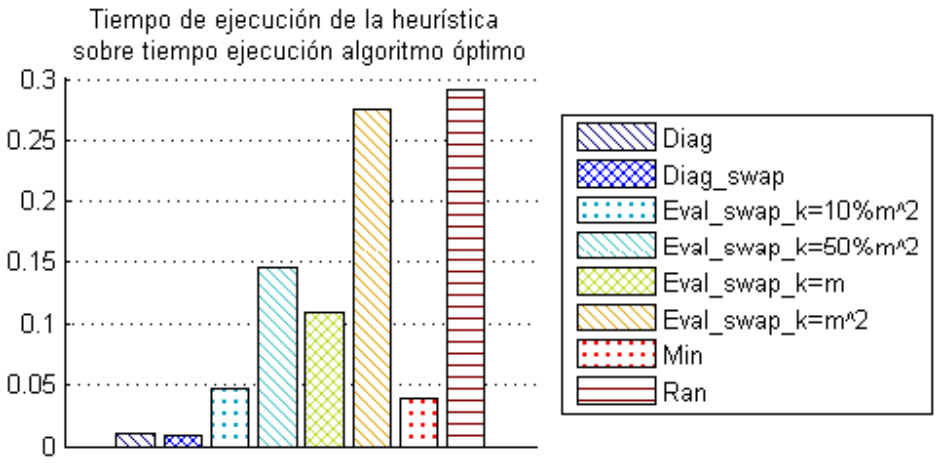

Figura 14. Tiempo de ejecución de las heurísticas sobre el tiempo de ejecución del algoritmo óptimo.

IV.5.1 Procesadores idénticos

Si los hosts son idénticos, es decir, que sus capacidades de procesamiento son las mismas, entonces cualquier función $\propto^{new}$ con relación de 1-1 puede definir una asignación factible, pero posiblemente con costos diferentes. Para la realización de los experimentos se tomaron en cuenta los siguientes parámetros: El algoritmo húngaro y ocho heurísticas sub-optimas Min, Rand, Diag, Diag_swap, Eval_swap con $k = m$, $k = m^2$, $k = 10\%m^2$ y $k = 50\%m^2$.

Los siguientes parámetros se generaron para la simulación: diferentes subconjuntos $\mathcal{T}_i^{cur}$ y $\mathcal{T}_i^{new}$, con una distribución uniforme para $m = 100$ representando el número de procesadores y $n = 1000$ representando el número de tareas que se encuentran distribuidas en los diferentes subconjuntos. Se realizaron 30 experimentos diferentes, los resultados de la simulación son presentados a continuación.

La figura 15 muestra el costo promedio absoluto obtenido por el algoritmo óptimo y las diferentes heurísticas sub-optimas, para el caso de procesadores idénticos.

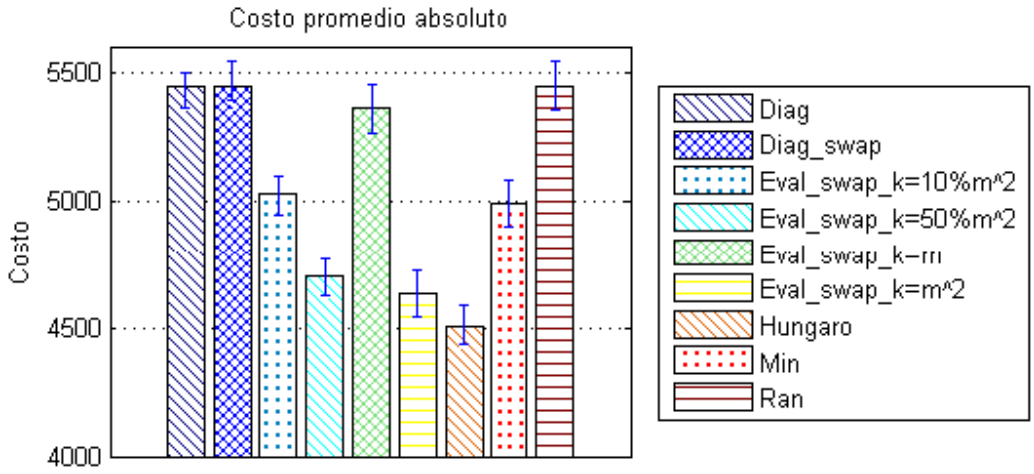

Figura 15. Costo promedio absoluto con m=100 para procesadores idénticos para arquitectura de bus común.

La figura 16 muestra el costo promedio porcentual obtenido durante la experimentación, para el caso de procesadores idénticos. Se puede observar que el algoritmo óptimo representa la mejor solución de todas, éste se pondera con un 100%. También se muestra el rendimiento porcentual del conjunto de heurísticas sub-óptimas.

La figura 17 muestra costo de las diferentes heurísticas sobre costo del algoritmo óptimo para procesadores idénticos. Esta es una métrica para comparar cuantas veces es peor la heurística propuesta comparada con el algoritmo óptimo.

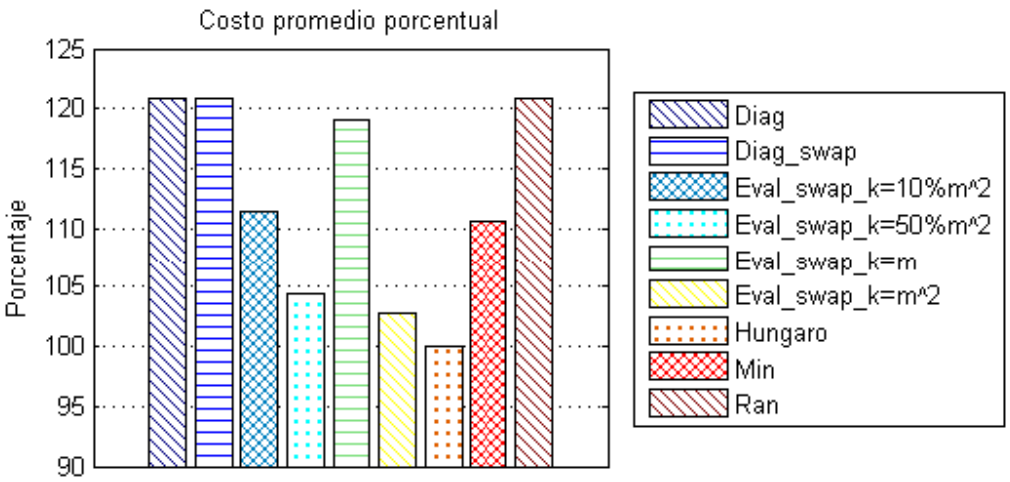

Figura 16. Costo porcentual con m=100 para procesadores idénticos para arquitectura de bus común.

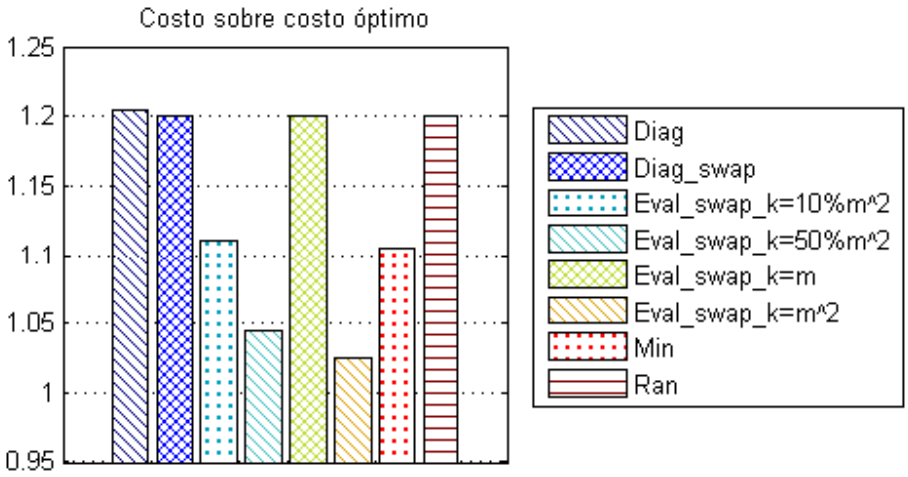

Figura 17. Costo sobre costo óptimo para procesadores idénticos para arquitectura de bus común.

IV.5.2. Procesadores Uniformes

Al igual que para el caso de procesadores idénticos se generaron para la simulación: diferentes subconjuntos $\mathcal{T}_i^{cur}$ y $\mathcal{T}_i^{new}$, con distribución uniforme para m = 100 representando el número de procesadores y n = 1000 representando el número de tareas que se encuentran distribuidas en los diferentes subconjuntos.

Dadas las características de los procesadores uniformes, los procesadores se ordenan decrecientemente de acuerdo a sus velocidades. Se realizaron 30 experimentos diferentes, en los cuales se generaron m valores no posibles de manera aleatoria, tomando en cuenta el 30% de los procesadores más lentos. Simulando así la incapacidad de algunos procesadores para procesar un subconjunto de tareas. Los resultados se presentan a continuación.

La tabla III muestra el costo promedio absoluto obtenido por el algoritmo óptimo y las diferentes heurísticas, para el caso de procesadores uniformes. Se puede apreciar en algunos de los métodos heurísticos que el promedio es muy grande. Esto representa soluciones no factibles.

Tabla II. Costo promedio con m=100 para procesadores uniformes para arquitectura de bus común.

Diag	*Diag_swap*	*Húngaro*	*Eval_swap_k=10%m^2*	*Rand*
$\mathbf{90.5 \cdot 10^4}$	$\mathbf{73.9 \cdot 10^4}$	$\mathbf{0.451 \cdot 10^4}$	$\mathbf{0.5 \cdot 10^4}$	$\mathbf{147.2 \cdot 10^4}$
Eval_swap_k=50%m^2	*Eval_swap_k=m^2*	*Min*	*Eval_swap_k=m*	
$\mathbf{0.472 \cdot 10^4}$	$\mathbf{0.463 \cdot 10^4}$	$\mathbf{3.83 \cdot 10^4}$	$\mathbf{3.86 \cdot 10^4}$	

En la figura 18 se muestra el costo promedio absoluto pero tomando en cuenta sólo las heurísticas que obtienen soluciones factibles para el caso de procesadores uniformes.

La figura 19 muestra el costo promedio porcentual obtenido durante la experimentación, para el caso de procesadores uniformes. Se puede observar que el algoritmo óptimo representa un 100%. También se muestra el rendimiento porcentual del conjunto de heurísticas sub-optimas. De manera similar la figura 20 el costo promedio porcentual, tomando en cuenta sólo las heurísticas que obtienen soluciones factibles.

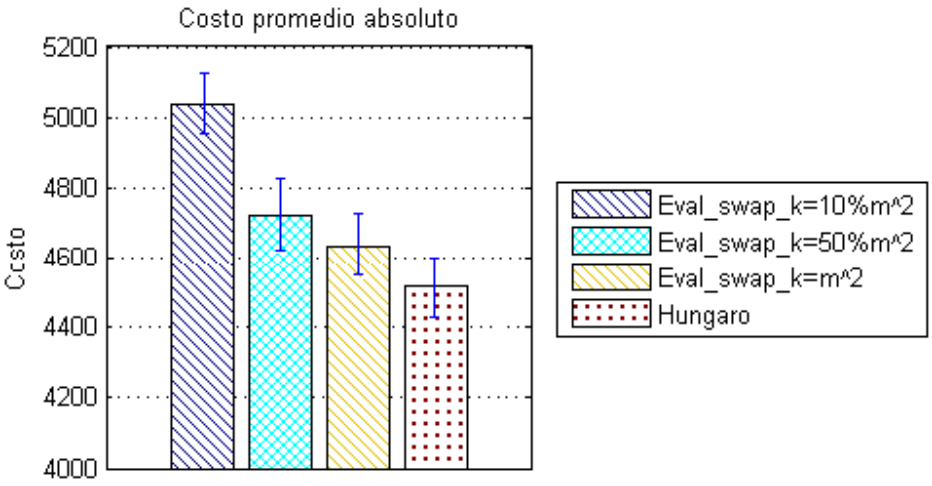

Figura 18. Costo promedio absoluto con m=100 para procesadores uniformes para arquitectura de bus común.

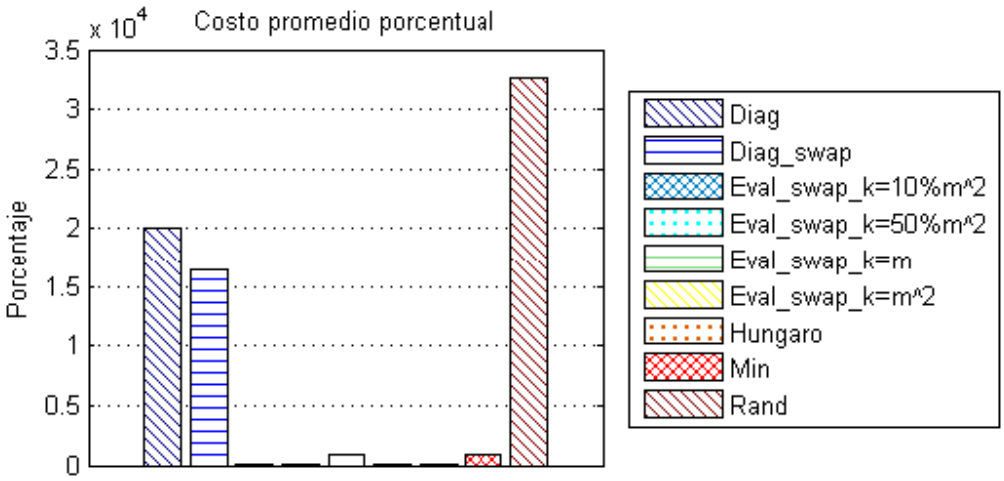

Figura 19. Costo porcentual 1 con m=100 para procesadores uniformes para arquitectura de bus común

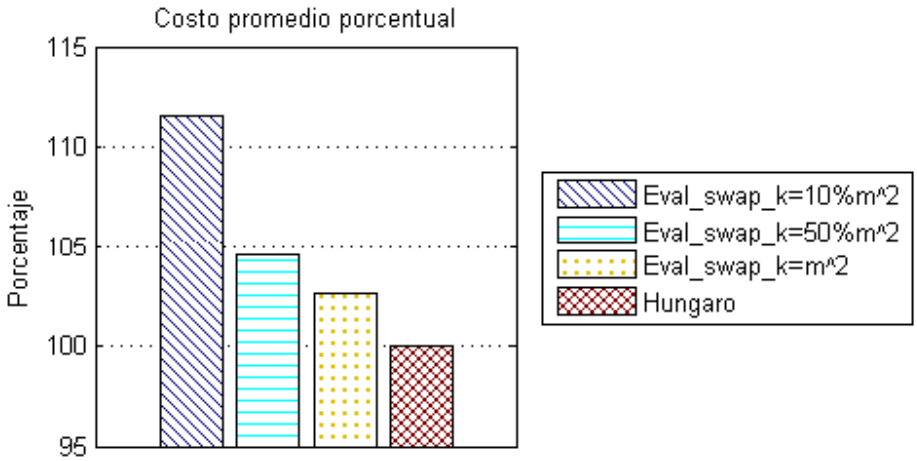

Figura 20. Costo porcentual 2 con m=100 para procesadores uniformes para arquitectura de bus común.

La figura 21 muestra el costo de las diferentes heurísticas sobre costo del algoritmo óptimo para procesadores uniformes. Esta es una métrica para comparar cuantas veces es peor la heurística propuesta comparada con el algoritmo óptimo. La figura 22 muestra esta misma métrica excluyendo las heurísticas que no obtienen soluciones factibles para el caso de procesadores uniformes.

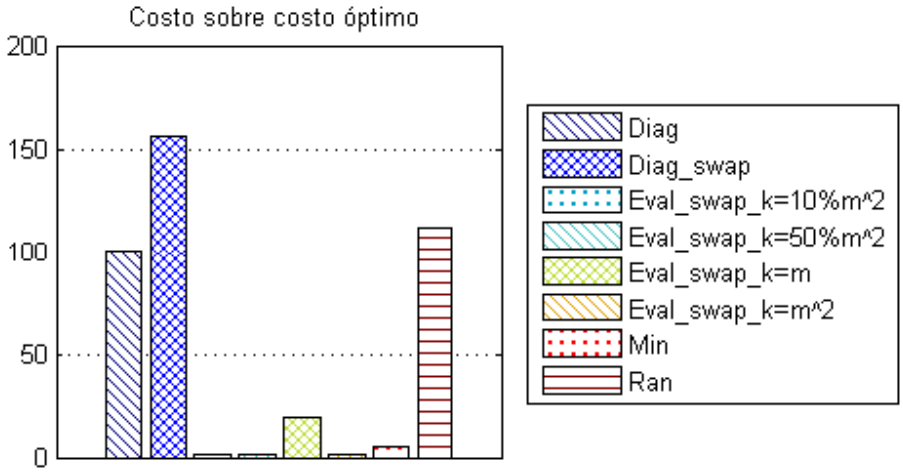

Figura 21. ***Costo sobre costo óptimo 1 para procesadores uniformes para arquitectura de bus común***.

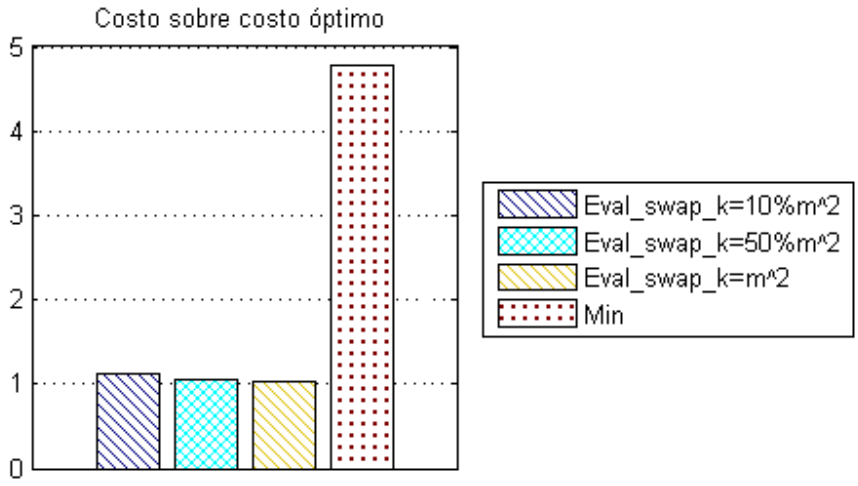

Figura 22. ***Costo sobre costo óptimo 2 para procesadores idénticos para arquitectura de bus común***.

IV.6 Conclusiones: bus común

Como sabemos el algoritmo óptimo no es necesariamente satisfactorio, para los requerimientos de los sistemas de tiempo real. Por lo tanto, podemos encontrar heurísticas que presentan resultados de manera más rápida pero disminuyendo la calidad de la solución.

La heurística *Eval_swap* consigue mejorar la solución inicial cerca del 80% en la reducción del costo con el 10% del incremento en el número de swaps. Por otro lado, es importante considerar que los tiempos de ejecución de las heurísticas propuestas son más rápidos en ejecución que el algoritmo óptimo. Además, la heurística *Eval_swap* presenta soluciones cercanas al algoritmo óptimo y consume sólo el 25% del tiempo de ejecución comparado con el algoritmo óptimo.

Para el caso de procesadores idénticos, las heurísticas propuestas muestran un rendimiento satisfactorio en la relación tiempo versus calidad de la solución. Las diferentes heurísticas pueden ser aplicadas tomando en cuenta las

aproximaciones a la solución óptima según sea el caso. Sin embargo todas estas heurísticas consiguen obtener una solución factible para el problema de reasignación de tareas con procesadores idénticos y con topología de bus común.

Por otro lado en el caso de procesadores uniformes, no todas las heurísticas presentan buenos resultados en las simulaciones. Como sabemos los valores de "no-posibles" representan una inconsistencia para las restricciones temporales del sistema. Por tanto, no puede tomarse como factibles aquellas en las cuales no se prevea la manera de evitar estos valores de "no-posible". Ahora bien, el algoritmo húngaro, y las heurísticas de Eval_swap con $k \geq 10\% m^2$ mostraron buena eficiencia para encontrar soluciones factibles para el caso de procesadores uniformes con topología de bus común.

Capítulo V

Modelo II: Arquitectura de Arreglo Lineal Procesadores

En la Sección V.1 se presenta el modelo de arquitectura de arreglo lineal de procesadores, así como su definición formal. También se describe el algoritmo descendente como estrategia de búsqueda para la minimización del calendario de transmisión en la arquitectura de arreglo lineal en la Sección V.2. Así como las heurísticas propuestas y los resultados de la experimentación, en las secciones V.4 y V.5, correspondientemente. Finalmente, en la Sección V.6 se observan las conclusiones relacionadas con este modelo.

V.1 Arquitectura de arreglo lineal de procesadores

El problema de transmisión sobre un arreglo lineal de procesadores se puede ver como: encontrar un calendario para un conjunto de actividades de transmisión con dependencias en cadena, sobre un conjunto de enlaces; buscando la minimización de la longitud del calendario. Mas detalladamente en la topología de arreglo lineal de procesadores, para el caso de procesadores idénticos se tienen las siguientes condiciones de transmisión de tareas:

- Enlaces bidireccionales con capacidad de transmisión unitaria,
- La técnica de transmisión es conmutación de paquetes,
- Un procesador puede enviar y recibir mensajes simultáneamente en cada uno de sus enlaces,
- Capacidad de buffer ilimitada en los procesadores.

La figura 23 muestra la topología de arreglo lineal procesadores para m = 3 y $|\mathcal{L}|$ = 2.

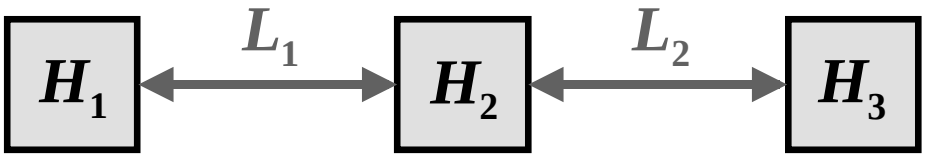

Figura 23. Topología de arreglo lineal de procesadores para $m = 3$ y $|\mathcal{L}| = 2$.

Si se considera la topología de un arreglo lineal de procesadores, la comunicación de los procesadores se realiza a través de un subconjunto $L_j^t \subseteq \mathcal{L}$ de enlaces, donde $\mathcal{L} = \{L_1, \dots, L_{m-1}\}$ es el conjunto total de enlaces. Mover una tarea T_j del procesador $H_i(\mathcal{T}_i^{cur})$ a través del enlace L_h.al procesador $H_k(\mathcal{T}_k^{new})$ incurre en un costo $c_w(j,i,h,k) = c(T_j, H_i, L_h, H_k)$.

$$c(i,h,k) = \sum_{j=1}^{n} \{c_w(j,i,h,k) \mid (T_j \notin \mathcal{T}_i^{cur} \cap \mathcal{T}_k^{new} \; y \; \alpha^{new}(\mathcal{T}_k^{new}) = H_k)\}. \qquad (9)$$

Además una operación pueden ser vista como: una tarea moviéndose a través de los enlaces sucesivos hasta alcanzar su destino. l_h denota la suma de los tiempos de transmisión de las tareas a través del enlace L_h.

Así en términos de $\mathcal{T}^{cur}$, $\mathcal{T}^{new}$, y mapeados por α^{new}, se puede definir el problema de minimizar la longitud del calendario de transmisión como:

$$Dada\ la\ matriz\ de\ costos\ encontrar\ la\ matriz\ de\ permutación$$
$$Q^r(i,h,k)\ sujeta\ a\ minimizar\ max\{l_1,\dots,l_h\},\ donde$$
$$(l_1,\dots,l_h) = \sum_{i=1}^{m}\sum_{k=1}^{m} C(i,h,k)\cdot Q^r(i,h,k) \mid L_h \in H_i \cap H_k\ sea\ minimizado, \quad (10)$$

para $r = 1,\dots m!$

V.1.1 Ejemplo: arquitectura de arreglo lineal de procesadores

Para mostrar el comportamiento de transmisión presentamos el siguiente ejemplo. Suponer $m = 3$ procesadores, $n = 9$ tareas, una asignación actual $\mathcal{T}_1^{cur} = \{T_1, T_2, T_3\}$ en H_1, $\mathcal{T}_2^{cur} = \{T_4, T_5\}$ en H_2 y $\mathcal{T}_3^{cur} = \{T_6, T_7, T_8, T_9\}$ en H_3. Los costos de transmisión para las tareas son los siguientes:

Tabla III. Costos de transmisión para tareas para la arquitectura de arreglo lineal de procesadores.

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

Suponer también que se necesita asignar los nuevos subconjuntos $\mathcal{T}_1^{new} = \{T_1, T_6, T_7\}$, $\mathcal{T}_2^{new} = \{T_2, T_4, T_8\}$ y $\mathcal{T}_3^{new} = \{T_3, T_5, T_9\}$.

Entonces, instalar T_1^{new} en H_1 implica: mover T_6 y T_7 de H_3 a H_2 con costo = 3 + 1 sobre el enlace l_2, y T_6 y T_7 de H_2 a H_1 con costo = 3 + 1 = 4 sobre el enlace l_1, T_1 ya se encuentra en H_1 por lo tanto el costo es cero. Este movimiento incurre en un costo de C_{11} = (4,4). De la misma manera, instalar T_2^{new} en H_3 implica: mover T_2 de H_1 a H_2 con costo = 4, T_2 y T_4 de H_2 a H_3 con costo = 4 + 5 = 9, T_8 ya se encuentra en H_3 por lo tanto el costo es cero. Este movimiento incurre en un costo de C_{32} = (4,9). en la figura 24 se muestra el cálculo de todos los posibles movimientos de $\mathcal{T}^{new}$ y sus respectivos costos:

	$\mathcal{T}_1^{new}$	$\mathcal{T}_2^{new}$	$\mathcal{T}_3^{new}$
H_1	(3+1, 3+1)	(5+5, 5)	(8+4, 4)
H_2	(6, 3+1)	(4, 5)	(2+4)
H_3	(6, 6)	(4, 4+5)	(2, 2+8)

Matriz de Costos

$$C = \begin{pmatrix} (\mathbf{4},\mathbf{4}) & (10,5) & (12,4) \\ (6,4) & (4,5) & (\mathbf{2},\mathbf{4}) \\ (6,6) & (\mathbf{4},\mathbf{9}) & (2,10) \end{pmatrix}$$

Figura 24. Ejemplo del cálculo de la matriz de costos para arquitectura de arreglo lineal de procesadores.

El objetivo es determinar el mapeo de $\alpha^{new}: \mathcal{T}^{new} \rightarrow \mathcal{H}$ tal que el calendario de transmisión sea minimizado. Podemos calcular el costo para ambos enlaces, un ejemplo puede ser C_{11} + C_{32} + C_{23} = (10,17). La longitud del calendario de transmisión está dado por max{10,17} = 17. Se puede verificar que la nueva asignación minimiza la carga máxima sobre los enlaces y además es óptima.

Se debe notar que el grafo que representa los enlaces en el arreglo lineal de procesadores no es un grafo bipartito completo. Por esta razón no es posible hacer uso del algoritmo húngaro para encontrar una solución. Es por esto que encontrar la matriz de permutación (Q_{ij}) sujeta a $minimizar\ max\ \{q_1, \dots, q_s\}$, puede ser visto como un problema combinatorio donde los métodos de búsqueda local ofrecen soluciones factibles para el problema mencionado. Por tanto, se debe hacer uso de heurísticas que permitan encontrar una solución factible, buscando minimizar la longitud del calendario de transmisión. En la Sección V.3 se proponen heurísticas efectivas para encontrar una solución factible. Las heurísticas están basadas en técnicas de búsqueda local.

V.2 Algoritmo descendente

Esencialmente, el método de búsqueda local consiste en moverse de una solución a otra dentro de su vecindario de acuerdo a reglas bien definidas (Pirlot, 1996). Si consideramos el problema de minimizar una función $F(x)$ en un conjunto finito de puntos X. Éste se puede considerar como el enunciado general de los problemas de optimización combinatoria. Una estrategia de búsqueda local comienza desde una solución arbitraria $x_1 \epsilon X$ y en cada paso n, se escoge una nueva solución x_{n+1} dentro del vecindario (x_n) de la solución actual x_n. Esto presupone la definición de *vecindario* en X; para cada $x \in X$ se asocia un subconjunto $V(x) \subseteq X$ al que se llama vecindario de x. Convencionalmente, suponemos que esa solución nunca pertenece a su propio vecindario, por ejemplo, $x \notin V(x)$, $\forall$ x ∈ X. Alternativamente, podemos decir que los vecinos de x son las soluciones obtenidas de x por un movimiento elemental. La evolución de la solución actual x_n, n = 1, 2, ..., dibuja una trayectoria en el espacio de búsqueda X. El criterio más común para encontrar una nueva solución x_{n+1} es escoger la mejor solución dentro del vecindario de x_n, por ejemplo, una solución $x_{n+1} \in V(x_n)$ con:

$$F(x_{n+1}) \leq F(x) \ \forall \ \text{x} \in \ V(x). \qquad (11)$$

Entonces, x_{n+1} se convierte en la nueva solución actual sabiendo que no es peor que x_n, por ejemplo, $F(x_{n+1}) \leq F(x_n)$. De otra forma, la búsqueda se detiene. Esta estrategia se conoce como *descendente* o de *etapas descendentes*. Para propósitos de comparación a continuación se presenta una descripción formal. F_n^* denota la mejor solución de F hacia el paso n y x_n^* tal que $F(x_n^*) = F_n^*$. A continuación se muestran los pasos del algoritmo descendente:

1. Inicialización: seleccionar $x_1 \in X$.
2. Paso n = 1, 2, … ; x_n denota la mayor solución.
    a) Encontrar la mejor $\bar{x}$ en el vecindario $V(x_n)$.
    b) Si $F(\bar{x}) \leq F(x_n)$, entonces $\bar{x}$ se vuelve la nueva solución actual x_{n+1} en el paso n + 1 y el mejor valor F_n^* de F asi también debe actualizarse x_n^*.
    c) Si no: fin

V.3 Heurísticas para la minimización del calendario de transmisión: arreglo lineal de procesadores

Las heurísticas propuestas, se basan en los principios de búsqueda y hacen uso de la metodología del algoritmo descendente, esto con el objetivo de encontrar una solución mejor durante las iteraciones de las heurísticas. La descripción de las heurísticas es la siguiente:

Linear_diag: para una matriz de costos C, linear_diag selecciona los elementos de la diagonal de la matriz de costos, la complejidad en tiempo es $O(m)$. El pseudocódigo se muestra a continuación.

Heurística Linear_diag

```
1.  Inicio
2.      para cada fila  i = 1,..,m hacer
3.          asignarla a la solución la celda con numero de fila y columna iguales (asignar la
            diagonal)
4.      fin para
5.  Fin
```

Linear_local_swap: para una matriz de costos C, linear_local_swap selecciona los elementos de la diagonal como una solución inicial, y el operador *swap* se aplica si el intercambio implica una reducción del costo para el par de valores involucrados en el *swap*, la complejidad en tiempo es $O(m + k)$ donde k es el número de *swaps* ejecutados. Los *swaps* se realizan utilizando un vector de posiciones de tamaño m, donde el subíndice i representa el valor del renglón y el valor contenido en la celda representa el valor de la columna j. El pseudocódigo se muestra a continuación.

Heurística Linear_local_swap

```
1.  Inicio
2.      tomar los elementos de la diagonal como solución inicial y asignarlos al vector de posición
        (esto es i = j)
3.      Asignar un valor escogido k
4.      para i = 1 hasta k hacer
5.          generar dos números aleatoriamente R1 y R2
6.          si (costo encontrado por el intercambio individual entre vector_posición[R1] y
            vector_posición[R2] es menor) entonces
7.              Intercambiar los valores contenidos en vector_posición[R1] por
                vector_posición[R2]
8.          fin si
9.      fin para
10.     asignar las posiciones en vector_posición a la solución, donde el subíndice i es el valor del
        renglón y el valor contenido en la celda representa el valor de la columna j.
1.  Fin
```

Linear_global_swap: de manera similar a linear_local_swap, éste selecciona los elementos de la diagonal como una solución inicial y aplica el operador swap, si la nueva solución minimiza la longitud total del calendario de transmisión, la complejidad en tiempo es también $O(m + k)$. Los *swaps* se realizan utilizando un vector de posiciones de tamaño m. El pseudocódigo se muestra a continuación.

Heurística Linear_global_swap

1. **Inicio**
2. tomar los elementos de la diagonal como solución inicial y asignarlos al vector de posición
   (esto es *i* = *j*)
3. Asignar un valor escogido k
4. **para** i = 1 hasta k **hacer**
5. generar dos números aleatoriamente R1 y R2
6. **si** (costo de los todos los elementos en el vector posición encontrados por el
   intercambio entre vector_posición[R1] y vector_posición[R2] es menor)
   entonces
7. Intercambiar los valores contenidos en vector_posición[R1] por
   vector_posición[R2]
8. **fin si**
9. **fin para**
10. asignar las posiciones en vector_posición a la solución, donde el subíndice i es el valor del
    renglón y el valor contenido en la celda representa el valor de la columna j.
1. **Fin**

V.4 Resultados y análisis experimental: arreglo lineal de procesadores

La figura 25 muestra la relación entre el número de operaciones *swap* y la calidad de la solución. Esta presenta el porcentaje promedio (sobre 30 experimentos) de la reducción del costo versus el porcentaje promedio en el incremento del numero de swaps y su diferencia absoluta. El número máximo de swaps probados es m^2 (100%). Los elementos de la diagonal se escogen como una solución inicial y su costo es considerado como 100%. Si se aplica linear_global_swap con un incremento de 10% se consigue un 90% la reducción del costo. Se puede encontrar un equilibrio entre el número de swaps y la calidad de la solución.

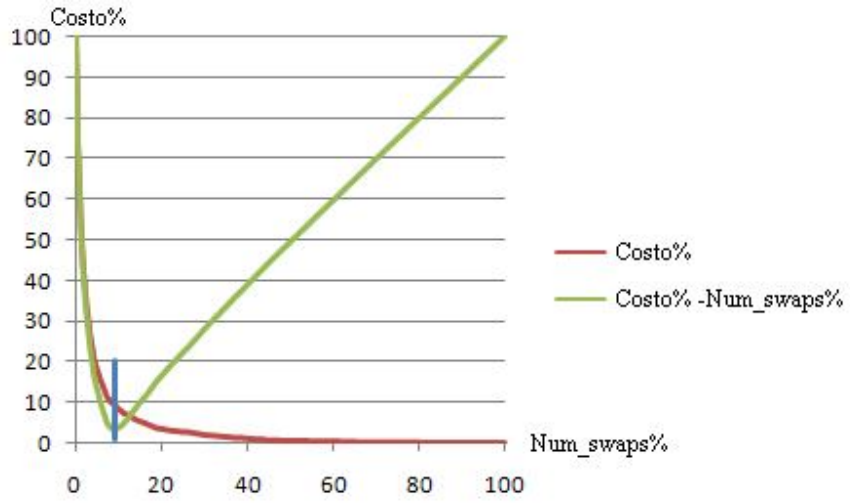

Figura 25. Intercambio entre el número de swaps y la calidad de la solución para arquitectura de arreglo lineal de procesadores.

El objetivo de las heurísticas propuestas es minimizar la longitud del calendario de transmisión de mover las tareas de $\mathcal{T}^{cur}$ a su destino $\mathcal{T}^{new}$. Los subconjuntos de $\mathcal{T}_i^{cur}$ y $\mathcal{T}_i^{new}$ se generaron con una distribución uniforme para m = 100 y n = 1000. Se aplicaron las heurísticas linear_diag, linear_local_swap y linear_global_swap. Para linear_local_swap y linear_global_swap se tomo la diagonal de la matriz C como solución inicial y se ejecutaron swaps con $k = m$, hasta $k = m^2$. Se generaron 30 diferentes experimentos. Se toman en cuenta procesadores idénticos. Los resultados se presentan a continuación.

La figura 26 muestra la longitud promedio absoluto del calendario de transmisión para las heurísticas propuestas. Linear_diag representa la solución (factible) inicial para todas las heurísticas. La heurística linear_local_swap con $k = m$ y $k = m^2$, no encuentran una solución buena con respecto a la solución inicial. Es mas, es interesante observar que cuanto mas grande es el número de swaps, es posible aumentar la longitud del calendario, dado que el intercambio (menor) entre dos elementos de la matriz de costos no garantiza la disminución de la longitud del calendario total. La heurística linear_global_swap con $k = m$, $k = m^2$ presentan

soluciones mejores en la solución respecto de la solución inicial. Si el número de swaps es mayor, mejor es la calidad de solución encontrada.

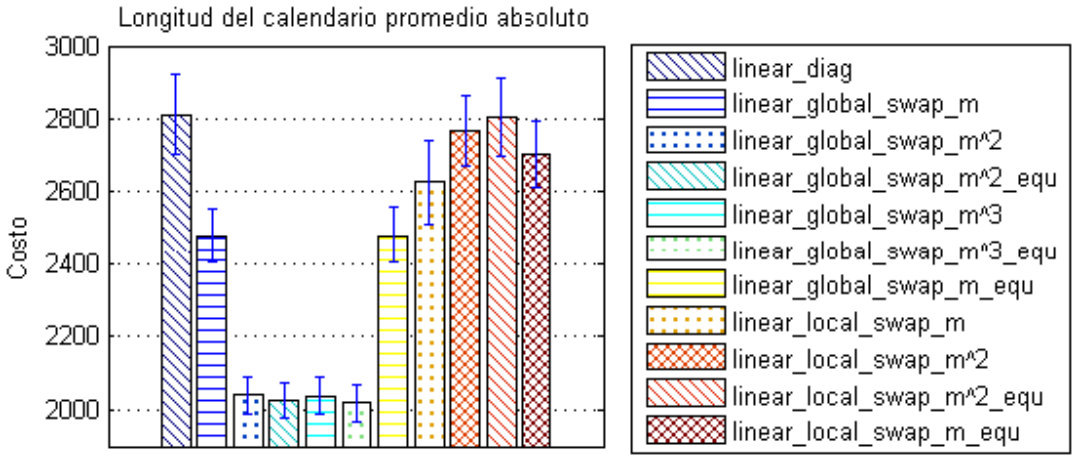

Figura 26. Costo promedio absoluto para soluciones menores y menores iguales con m=100 y topología de arreglo lineal de procesadores.

La figura 27 muestra la longitud promedio porcentual del calendario de transmisión para las heurísticas propuestas. De manera similar a lo observado en la figura 21 linear_diag representa la solución (factible) inicial para todas las heurísticas y la heurística linear_global_swap con $k = m$, $k = m^2$ presentan las mejores soluciones solución respecto de la solución inicial. Las figuras 21 y 22 presentan una variación del algoritmo descendente. Para propósitos de comparación se incluyen las heurísticas con terminación "*equ*", que representa la condición de actualización de la solución si $F(\bar{x}) \leq F(x_n)$. Para las heurísticas sin esta terminación la condición para la actualización de la solución es $F(\bar{x}) < F(x_n)$. En ambas figuras se puede observar que la condición de actualización para $F(\bar{x}) < F(x_n)$ obtiene soluciones mejores.

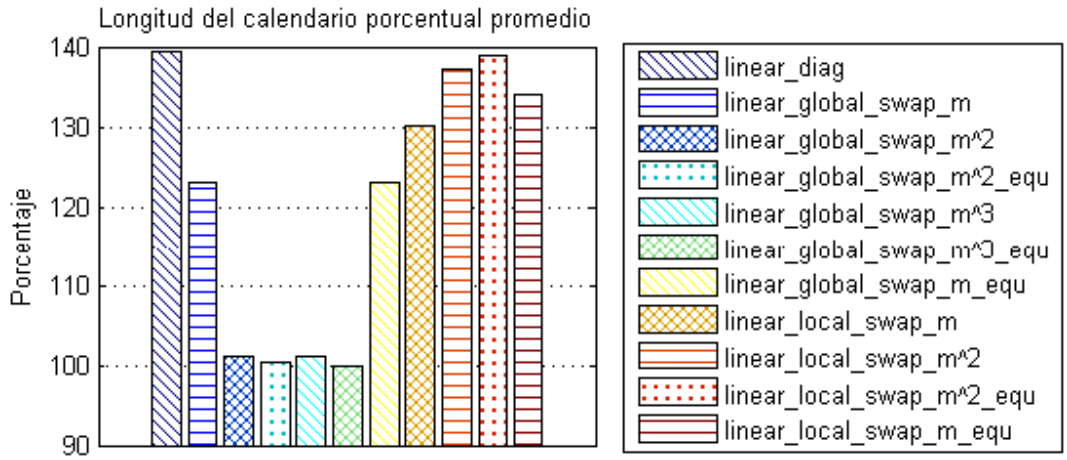

Figura 27. Costo promedio porcentual para soluciones menores y menores iguales con m=100 y topología de arreglo lineal de procesadores.

En las figuras 28 y 29 se omiten las heurísticas linear_local_swap con terminación "*equ*". La figura 28 muestra la longitud promedio absoluto del calendario de transmisión para las heurísticas linear_diag, linear_local_swap con $k = m$ y k=m^2 y linear_global_swap con $k = m$, $k = m^2$ y $k = m^3$. Se puede observar que las mejores soluciones encontradas están dadas por las heurísticas linear_global_swap para cualquier número k de swaps. Es importante notar que para las heurísticas linear_global_swap con $k = m^2$ y $k = m^3$, la mejora en la calidad de la solución es pequeña comparado con su complejidad en tiempo.

La figura 29 muestra la longitud promedio porcentual del calendario de transmisión para las heurísticas linear_diag, linear_local_swap con $k = m$ y k=m^2 y linear_global_swap con $k = m$, $k = m^2$ y $k = m^3$. De manera similar a la figura 23, se observa que las mejores soluciones encontradas están dadas por las heurísticas linear_global_swap para cualquier número k de *swaps*.

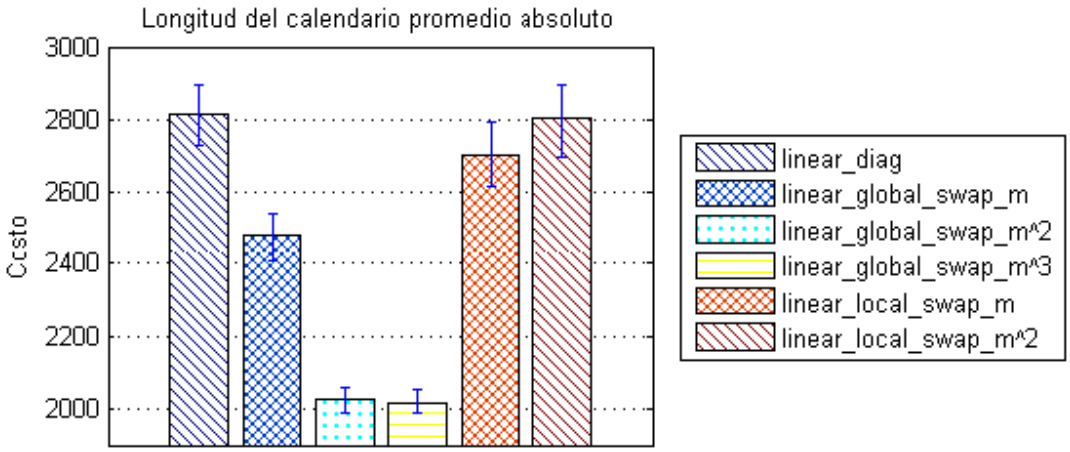

Figura 28. Costo promedio absoluto con m=100 y topología de arreglo lineal de procesadores.

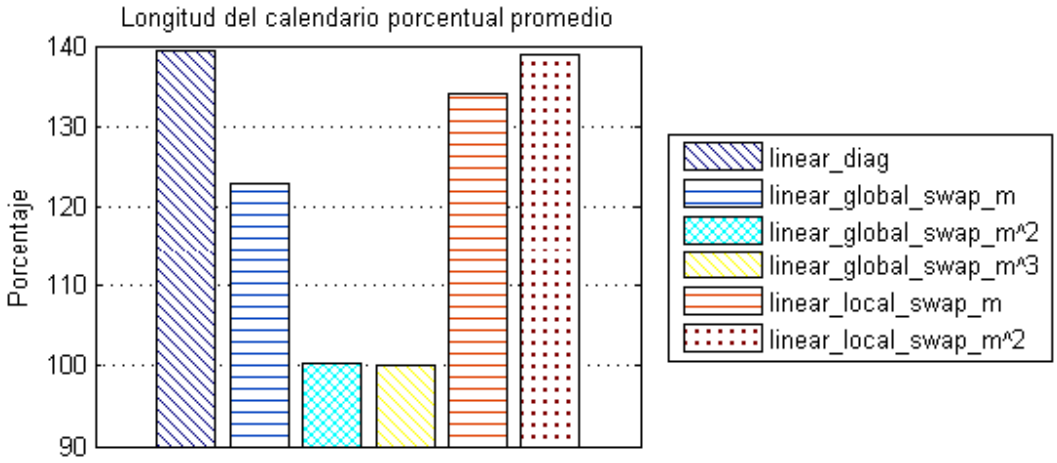

Figura 29. Costo promedio porcentual con m=100 y topología de arreglo lineal de procesadores.

IV.6 Conclusiones: arreglo lineal de procesadores

En este capítulo se muestran varias heurísticas para la minimización del calendario de transmisión para la topología de arreglo lineal de procesadores con restricciones de sistemas de tiempo real. Para el caso de procesadores idénticos, las heurísticas propuestas muestran un rendimiento satisfactorio en la relación tiempo versus calidad de la solución. Además todas estas heurísticas consiguen

obtener una solución factible para el problema de reasignación de tareas con procesadores idénticos y con topología de arreglo lineal de procesadores.

Las heurísticas linear_local_swap con $k = m$ y k=m^2 no representan mejoría respecto de la solución inicial dada, es mas incluso pueden aumentar la longitud promedio del calendario de transmisión. Por otro lado las heurísticas linear_global_swap con $k = m$, $k = m^2$ y $k = m^3$, son heurísticas efectivas para encontrar una solución factible y con una mejora de hasta 40% respecto de una solución inicial. Además se debe notar que cuanto más grande es el número k de *swaps* aplicados la solución tiende a mejorar pero a un costo mayor en tiempo respecto de la calidad de la solución.

Capítulo VI

Reasignación Mediante Técnicas de *Bin Packing*

En la Sección VI.1 se describe la de reasignación mediante técnicas de *bin packing* y su definición formal. En la sección VI.2 se describe las técnicas más populares de *bin packing*. En la Sección VI.3 se describe el problema de suma múltiple de subconjuntos. Estas dos secciones se toman en cuenta para el desarrollo de las heurísticas de reasignación mediante técnicas de *bin packing* presentadas en la Sección VI.4. Los resultados de la experimentación se presentan en la Sección IV.5 y finalmente en la Sección IV.6 se observan las conclusiones relacionadas con este problema.

VI.1 Reasignación mediante técnicas de *bin packing*

Como se describe en la Sección III.1, el cambio de las exigencias de control de los sistemas de tiempo real conduce a nuevas condiciones de cronometraje para sus cómputos. Como consecuencia, los tiempos de cómputo de los procesos de control y parámetros de cronometraje como períodos y fechas límites pueden cambiar. Las condiciones de funcionamiento pueden cambiarse rápidamente, y el sistema de tiempo real tiene que adaptarse continuamente a modos de cambio. Como consecuencia, los modos de cambio de operación pueden conducir a la necesidad de migrar de un procesador a otro y así las acciones de control se distribuyen entre los procesadores.

Este problema sigue las características descritas en la Sección III.2, con ligeras variantes y se define como sigue. Se supone la existencia de un conjunto de

tareas $\mathcal{T} = \{T_1, \dots, T_n\}$; las cuales se ejecutan repetidamente dentro de un periodo dado $\pi_j := \pi(T_j)$. La tarea está caracterizada por una dupla de requerimientos de procesamiento (p_j, π_j) . Para la dupla se tiene una utilización del procesador para cada tarea T_j de $u_j = \frac{p_j}{\pi_j}$. También se supone que las tareas en $\mathcal{T}$ son independientes entre si. Para el procesamiento de las tareas se usa un conjunto de procesadores $\mathcal{H} = \{H_1, \dots, H_m\}$. Para cada procesador H_i se asocia la utilización total del procesador $U = \sum_{j=1}^{n} \left(\frac{p_j}{\pi_j}\right)$, y además debe cumplirse que $U \leq 1$ para cualquier conjunto de tareas $\mathcal{T}_i$ asignadas al procesador H_i. También se tiene que la utilización máxima del procesador $U^{max} = 1$ y la utilización mínima $U^{min} = 0$.

Inicialmente, se supone una asignación inicial $\alpha^{init}(\mathcal{T}^{init})$. Así que podemos decir que en el tiempo actual se puede realizar una asignación actual $alloc^{cur} = \alpha^{cur}(\mathcal{T}^{cur})$. También, se supone que dados algunos cambios en los periodos de las tareas, y por tanto nuevos requerimientos de procesamiento, se debe instalar una nueva asignación $\mathcal{T}^{new} = \{\mathcal{T}_1^{new}, \dots, \mathcal{T}_m^{new}\}$ para responder a los nuevos requerimientos del sistema de tiempo real. Se toma en cuenta que para una asignación $alloc^{cur} = \alpha^{cur}(\mathcal{T}^{cur})$ en el cual se exceda la utilización $U^{max} = 1$ se deben migrar algunas tareas buscando que la condición de calendarizabilidad $U \leq 1$ se mantenga. También $F_i \in \{0, 1\}$ es una variable binaria, que denota si una asignación $\mathcal{T}^{new} = \{\mathcal{T}_1^{new}, \dots, \mathcal{T}_m^{new}\}$ no puede ser encontrada. Esto es:

$$F_i = \begin{cases} 0, & si\ no\ falla \\ 1, & si\ falla \end{cases}. \tag{12}$$

El objetivo es determinar el mapeo de $\alpha^{new}: \mathcal{T}^{new} \rightarrow \mathcal{H}$ tal que no ocurra una falla F_i. Por lo tanto, el problema de reasignación utilizando técnicas de *bin packing* se puede formular como:

$$Dada\ una\ asignacion\ alloc^{cur}\ mantener\ que\ U_j \leq 1\ para\ todo\ H_i$$
$$tal\ que, F_i\ se\ mantenga\ \neq\ 1. \quad (13)$$

VI.1.1 Ejemplo de reasignación mediante técnicas de *bin packing*

Para mostrar el comportamiento de reasignación presentado en este capítulo se presenta el siguiente ejemplo. Suponer m = 2 procesadores, n = 4 tareas, una asignación actual $\mathcal{T}_1^{cur}$ = $\{T_1, T_2, T_3\}$ en H_1 y $\mathcal{T}_2^{cur}$ = $\{T_4\}$ en H_2. Los tiempos de procesamiento p_j, sus periodos π_j y su utilización u_j para los subconjuntos de tareas $\mathcal{T}^{cur}$ son los siguientes:

Tabla IV. Tiempos de procesamiento p_j, periodos π_j y utilización u_j para subconjuntos $\mathcal{T}^{cur}$ para el ejemplo de reasignación mediante técnicas de bin packing.

	T_1	T_2	T_3	T_4
p_i	2	1	2	3
π_i	8	4	10	10
u_i	0.25	0.25	0.2	0.3

Por tanto la U_1 = 0.25 + 0.25 + 0.2 = 0.7 y U_2 = 0.3. Suponer también dado un cambio en los parámetros extrínsecos, los periodos se reducen por la mitad o lo que es lo mismo se reducen un 50%. Los tiempos de procesamiento p_j, sus periodos π_j y su utilización u_j para los subconjuntos $\mathcal{T}^{new}$ serían como sigue:

Tabla V. Tiempos de procesamiento p_j, periodos π_j y utilización u_j para subconjuntos $\mathcal{T}^{new}$ para el ejemplo de reasignación mediante técnicas de bin packing.

	T_1	T_2	T_3	T_4
p_i	2	1	2	3
π_i	4	2	5	5
u_i	0.5	0.5	0.4	0.6

Entonces la nueva utilización total del procesador U_1 = 0.5 + 0.5 + 0.4 = 1.4 y U_2 = 0.6. Es claro que dada la condición necesaria para la calendarización $U \leq 1$ para H_1 es violada. El objetivo es determinar el mapeo de $\alpha^{new}: \mathcal{T}^{new} \rightarrow \mathcal{H}$ tal que no ocurra una falla F_i. Si $\mathcal{T}_1^{cur} = \{T_1, T_2\}$ se asigna a H_1 y $\mathcal{T}_2^{cur} = \{T_3, T_4\}$ a H_2 se tiene que U_1 = 0.5 + 0.5 = 1 y U_2 = 0.4 + 0.6=1 mantienen la condición necesaria de calendarización.

Por tanto, se debe hacer uso de heurísticas que permitan encontrar una asignación que mantenga la condición de calendarización $U \leq 1$, buscando así que $F_i \neq 1$. En la Sección VI.4 se proponen heurísticas efectivas para encontrar una reasignación factible. Las heurísticas están basadas en técnicas de *bin packing*.

VI.2 Algoritmos de aproximación unidimensional *bin packing*

En el problema clásico de *bin packing* uni-dimensional, se tiene una secuencia $L = (a_1, a_2, \ldots, a_n)$ de elementos, de tamaño $s(a_i) \in (0,1]$ y donde se busca empacar los elementos de la lista dentro del número mínimo de recipientes (*bins*) de capacidad unitaria, por ejemplo, particionando los elementos dentro del número mínimo m de subconjuntos $B_1, B_2, \ldots, B_m$ tal que $\sum_{a_i \in B_j} s(a_i) \leq 1, 1 \leq j \leq m$ (Coffman *et.al*., 1996).

Este problema se caracteriza por ser $NP - dificil$, y tiene muchas aplicaciones en el mundo real, desde la carga de camiones sujetos a limitaciones de peso o espacio hasta problemas de corte de cintas, donde los recipientes corresponden a longitudes estándar de algún material, como cable o papel el cual debe ser cortado. Debido a que este es un problema $NP - dificil$ los algoritmos más eficientes conocidos usan heurísticas para hacer aproximaciones a la solución, en la mayoría de los casos se encuentran soluciones bastante buenas pero no es posible garantizar la optimalidad de las mismas.

Para el caso de los algoritmos de *bin packing* la métrica estándar para el rendimiento del peor caso es *el cociente de rendimiento asintótico del peor caso* (Coffman *et.al.*, 1996). Esto es, dada una lista L y un algoritmo A, sea $A(L)$ el número de recipientes usados por el algoritmo A cuando se aplica a la lista L, sea $OPT(L)$ el número de recipientes óptimo para empacar la lista L y $R_A(L) \equiv A(L)/OPT(L)$, e*l cociente de rendimiento absoluto del peor caso* R_A para un algoritmo A se define como:

$$R_A \equiv \inf\{\, r \geq 1 : R_A(L) \leq r \; para \; toda \; L \,\}. \tag{14}$$

El cociente de rendimiento asintótico del peor caso R_A^∞ se define como:

$$R_A^\infty \equiv \inf\{\, r \geq 1 : para \; alguna \; N > 0, R_A(L) \leq r \; para \; toda \; L \; con \; OPT(L) \geq N \}. \tag{15}$$

Las estrategias más conocidas y ampliamente usadas son:

- Siguiente acomodo (NF por las siglas en inglés de *Next Fit*): Este algoritmo empaca los elementos de la lista con el índice B_m más pequeño (más a la izquierda) que tenga espacio. Si un nuevo elemento de la lista no puede empacarse en el recipiente B_m, se cierra y se abre uno nuevo para continuar con el empacado. Tiene una complejidad en tiempo de $O(n)$.

Además, mantiene una cota superior para cualquier lista L, $NF(L) \leq 2 \cdot OPT(L) - 1$ (Coffman *et.al*., 1996).

- Primer acomodo (FF por las siglas en inglés de *First Fit*): Este algoritmo empaca los objetos de la lista en el recipiente con el índice B_m más pequeño (más a la izquierda) que tenga espacio. Si el recipiente B_m esta completamente lleno se cierra y el siguiente recipiente B_{m+1} será el nuevo primer recipiente. Si los recipientes existentes no pueden empacar un elemento de la lista, un nuevo recipiente se abre para después empacarlo. Tiene una complejidad en tiempo $O(n \log n)$. Además, mantiene una cota superior para cualquier lista L, $FF(L) \leq \lceil (17/10) \cdot OPT(L) \rceil$ (Coffman *et.al.*, 1996).

- Mejor acomodo (BF por las siglas en inglés de *Best Fit*): En esta estrategia de empacado la cual es similar a la estrategia de FF, donde los elementos de la lista se empacan poniendo los objetos de la lista en el recipiente más lleno que tenga espacio, tomando en cuenta el recipiente con el índice B_m más pequeño. Si los recipientes existentes no pueden empacar un elemento de la lista un nuevo recipiente se abre para después empacarlo. Tiene una complejidad en tiempo de $O(n \log n)$. Además, de manera similar a FF mantiene una cota superior para cualquier lista L, $BF(L) \leq \lceil (17/10) \cdot OPT(L) \rceil$ (Coffman et.al., 1996).

- Último acomodo (LF por las siglas en inglés de *Last Fit*): En este algoritmo de empacado, empaca los objetos de la lista en el recipiente con el índice B_m más grande (más a la derecha) que tenga espacio. Tiene una complejidad en tiempo de $O(n \log n)$. Además se ha demostrado que

$para\ toda\ \propto$, $0 < \propto \leq 1$, $R_{FF}^{\infty}(\propto) \leq R_{LF}^{\infty}(\propto) \leq R_{NF}^{\infty}(\propto)$ (Coffman et.al., 1996).

- Peor acomodo (WF por las siglas en inglés de *Worse Fit*): Este algoritmo los elementos de la lista se empacan poniendo los objetos de la lista en el recipiente más vacío que tenga espacio, tomando en cuenta el recipiente con el índice B_m más pequeño. Si los recipientes existentes no pueden empacar un elemento de la lista un nuevo recipiente se abre para después empacarlo. Tiene una complejidad en tiempo de $O(n \log n)$. Además se ha demostrado que: $para\ toda\ \propto$, $0 < \propto \leq 1$, $R_{FF}^{\infty}(\propto) \leq R_{WF}^{\infty}(\propto) \leq R_{NF}^{\infty}(\propto)$ (Coffman et.al., 1996).

Otra mejora considerable para los algoritmos de *bin packing*, es ordenar los elementos de la lista en orden decreciente para su empaque. Las estrategias de mejor acomodo decreciente (BFD por las siglas en inglés de *Best Fit Decreasing*) y primer acomodo decreciente (FFD por las siglas en inglés de *First Fit Decreasing*) se encuentran entre los algoritmos heurísticos mejor conocidos para solucionar el problema de *bin packing*. Estos tienen una cota superior de $(11/9) \cdot OPT(L) + 1$ esto se muestra en (Yue, 1991). El más simple de estos es, la estrategia de FFD, la cual funciona ordenando primeramente los elementos a ser asignados en orden decreciente de acuerdo a su tamaño, e insertando cada elemento aplicando la regla de empaque de FF. Recientemente se a probado que el límite más ajustado para FFD es $(11/9) \cdot OPT(L) + (6/9)$ (Dósa, 2007).

VI.3 El problema de la suma múltiple de subconjuntos

El problema de la suma múltiple de subconjuntos (MSSP por las siglas en inglés de *Multiple Subset Sum Problem*) es una variante del problema de *bin packing*, en

el cual el número de recipientes es fijo y se trata de maximizar el peso promedio empacado en todos los recipientes (Caprara *et. al.*, 1998). El problema es también un caso especial del problema de la mochila múltiple en el cual todas las mochilas tienen la misma capacidad y el beneficio de los elementos y sus pesos coinciden (Mortello y Toth, 1990).

Formalmente, se tiene un conjunto de $N \coloneqq \{1, \dots, n\}$ de *elementos*, cada elemento i tiene un peso entero positivo w_i, y un conjunto $M \coloneqq \{1, \dots, m\}$ de *recipientes* idénticos con una *capacidad* entera positiva c. El objetivo es seleccionar un subconjunto de elementos con un peso total máximo que pueda ser empacado en los recipientes. El problema se puede formular como el siguiente problema de programación lineal entera:

$$maximizar \sum_{j \in M} \sum_{i \in N} w_i x_{ij}$$

$$sujeto\ a \sum_{i \in N} w_i x_{ij} \ \leq c, \ \ j \in M$$

$$\sum_{j \in M} x_{ij} \leq 1, \qquad i \in N$$

donde $x_{ij} \in \{0, 1\}, \ i \in N$.

Si se supone que $w_i \leq c$ para toda $i = 1, \dots, n$. Además, se supone que $n \geq m$, de otro modo este problema queda trivialmente resuelto. La carga de un recipiente j se puede denotar por ℓ_j y representa el peso promedio de los elementos empacados en los recipientes.

MSSP es un problema fuertemente $\mathcal{NP} - dificil$ y encuentra aplicación en logística, corte y empaque. Por ejemplo, se tiene el siguiente problema del mundo real, donde una compañía produce objetos de mármol, el cual es precisamente MSSP. Cada semana la compañía recibe un cargamento de m lozas de mármol de

una cantera. Estas lozas tienen el mismo tamaño y son mucho más largas que anchas. La compañía produce diferentes productos las cuales tienen que ser cortadas de las lozas de mármol y entonces son procesadas. Dependiendo de la disponibilidad, la compañía prepara una lista de productos que están interesados en producir. Para preparar dicha lista, algunos productos deberían ser seleccionados y cortados de las lozas y así el total de mármol gastado es el mínimo (ejemplo tomado de Caprara *et. al*., 1998).

VI.4 Heurísticas para la minimización del número de fallas de reasignación con periodos dinámicos

Las heurísticas propuestas, se basan en los principios de asignación de tareas con técnicas de *bin packing* y tomando en cuenta el problema de suma múltiple. Estos esquemas no pueden usarse de manera directa para el problema de reasignación presentado en este capítulo. Para el caso de asignación con algoritmos de *bin packing*, el objetivo es particionar los elementos dentro del número mínimo m de recipientes, para el problema de reasignación de tareas el número de recipientes (procesadores) es fijo. Por otro lado el problema de suma múltiple de subconjuntos parece ser más acertado. Pero este tiene el inconveniente que supone recipientes completamente vacios al inicio de la asignación, para el problema de reasignación de tareas siempre debe existir una asignación factible dentro de los recipientes (procesadores). Por lo tanto los recipientes posiblemente estén ocupados. Los pasos que se proponen para la reasignación de tareas mediante técnicas de bin packing son:

1. Cuando un procesador H_i tiene una $U > 1$, realiza una búsqueda para encontrar los elementos tales que $u_j = p_j/\pi_j$ sean los menores hasta que $U \leq 1$ y los asigna a una cola de tareas q. Esto para todos los procesadores H_i.

2. Ordenan los elementos en q, aplican las reglas asignación de *First Fit* (FF), *Best Fit* (BF), *Worse Fit* (WF) y *Last Fit* (LF) que se presentan en la Sección VI.2. Dado el ordenamiento sobre q, se puede ver que se trata de los algoritmos de asignación *First Fit Decresing* (FFD), *Best Fit Decresing* (BFD), *Worse Fit Decresing* (WFD) y *Last Fit Decresing* (LFD).

A continuación se muestra el pseudocódigo de las heurísticas de reasignación. La complejidad en tiempo de todas las heurísticas propuestas es de $O(n \log n)$.

Algoritmo de reasignación mediante técnicas de *bin packing*

1. **Inicio**
2. **mientras** (exista un procesador H_i que tenga una $U > 1$) **hacer**
3. realizar una búsqueda en todo H_i para encontrar los elementos tales que $u_j = p_j / \pi_j$ sean los menores hasta que $U \leq 1$
4. asignarlos a una cola de tareas q. (Esto para todos los procesadores H_i.)
5. **fin mientras**
6. Ordenan los elementos en q
7. Aplicar las reglas de asignación de: *First Fit Decresing* (FFD), *Best Fit Decresing* (BFD), *Worse Fit Decresing* (WFD) y *Last Fit Decresing* (LFD)
8. **Fin**

VI.5 Resultados y análisis experimental: reasignación mediante técnicas de *bin packing*

El objetivo de las heurísticas propuestas es encontrar una asignación que mantenga la condición de calendarización $U \leq 1$ para todos los procesadores, y que $F_i \neq 1$. Se generaron conjuntos de tareas buscando que se mantenga el límite $U^{max} = 1$ con distribuciones normal y uniforme respecto de número de tareas asignadas a los procesadores en $\mathcal{H}$ y por lo tanto a la utilización u_j de cada una de ellas. Se generaron 30 diferentes experimentos con $m = 100$ y $n = 1000$. A

partir de una asignación arbitraria con aproximadamente $U \cong \frac{1}{2}$ que representa el 0%, se decrementa el periodo en intervalos de 1% hasta un decremento máximo de 100% buscando así que los procesadores mantengan una utilización $U \cong 1$, con el objetivo de observar hasta qué porcentaje de decremento del periodo la asignación se mantiene factible, esto es mantener la condición $U \leq 1$. Se aplicaron las heurísticas para la asignación de las tareas en q bajo las reglas de FFD, BFD, WFD y LFD. Se toman en cuenta procesadores idénticos. Los resultados se presentan a continuación.

La figura 30 muestra el número de fallas promedio para heurísticas propuestas. Se puede observar que para el caso de una distribución normal el decremento del periodo, a partir de una asignación inicial dada, encuentra una asignación sin falla hasta el 97% de decremento. Todas las heurísticas presentan un rendimiento similar, donde las heurísticas obtienen una asignación sin falla hasta un 97% del decremento del periodo. Las fallas en el decremento del periodo están en el orden de las 14 fallas en 98% de decremento, 29 fallas para un decremento de 99% y 30 fallas para el decremento del 100% del periodo.

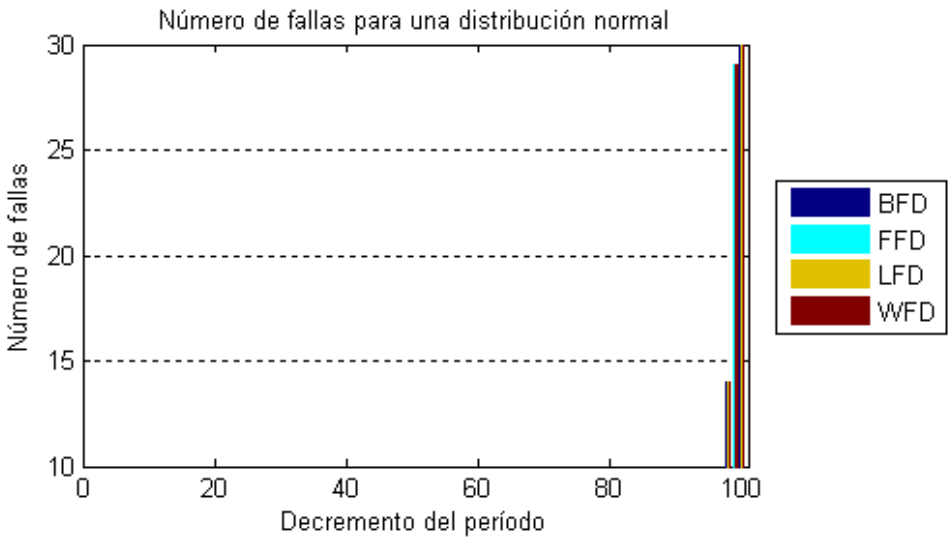

Figura 30. Número de fallas promedio con m=100 y para la asignación de tareas con técnicas de bin packing para una distribución normal.

La figura 31 muestra el promedio porcentual del número de fallas para el caso de distribución normal (sobre 30 experimentos). Se toma la mejor heurística como el 100%. La heurística FFD consigue obtener reasignaciones ligeramente mejores, para 98% de decremento del periodo presenta 11 fallas, para un decremento de 99% y 100%, 29 fallas. Por esta razón se observa que FFD tiene un valor del 100% y el resto de las heurísticas tienen un porcentaje del orden del 106%.

La figura 32 muestra el número de fallas promedio para el caso de una distribución uniforme. Se puede observar que para el caso de la distribución uniforme de las tareas las fallas en la asignación ocurren a partir del 5% del decremento del periodo, respecto de la asignación inicial dada. Se debe notar que el número de fallas es pequeño comparado con el número de experimentos (se realizan 30 experimentos), esto indica que en algunos casos las heurísticas encuentran una asignación correcta aunque el decremento del periodo este cerca del 100%. Todas las heurísticas presentan el mismo rendimiento. Por ejemplo las fallas en el decremento del periodo del 30% están en el orden de las 6 fallas, en el 60% de decremento, 8 fallas y para un decremento del 100% presentan16 fallas.

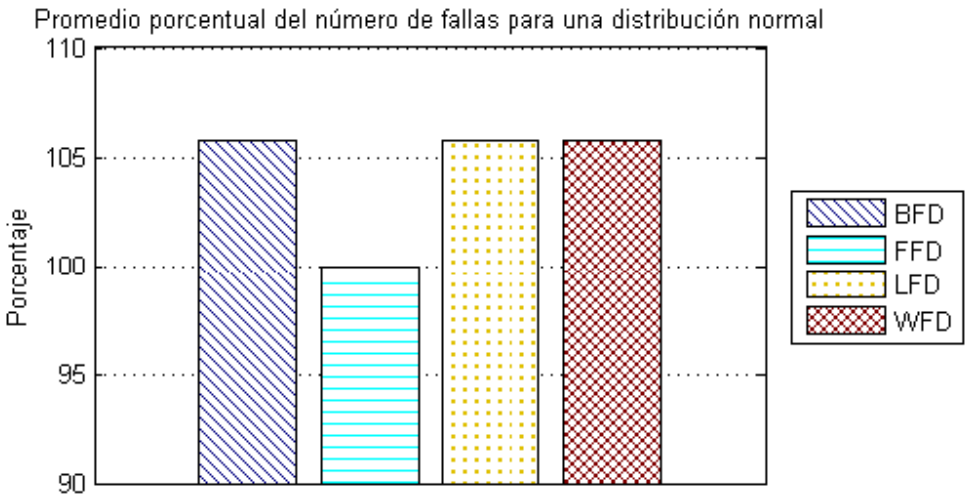

Figura 31.Promedio porcentual del número de fallas promedio con m=100 y para la asignación de tareas con técnicas de bin packing para una distribución normal.

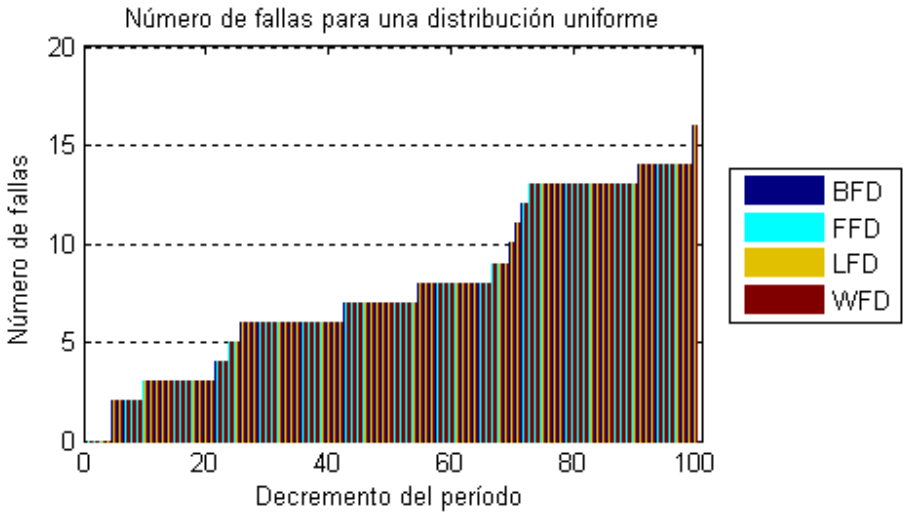

Figura 32. Número de fallas promedio con m=100 y para la asignación de tareas con técnicas de bin packing para una distribución uniforme.

La figura 33 muestra el promedio porcentual del número de fallas para el caso de distribución uniforme (sobre 30 experimentos). Dado que todas las heurísticas se comportan exactamente igual al buscar una reasignación sin fallas, todas consiguen un rendimiento del 100% en la comparación entre estas.

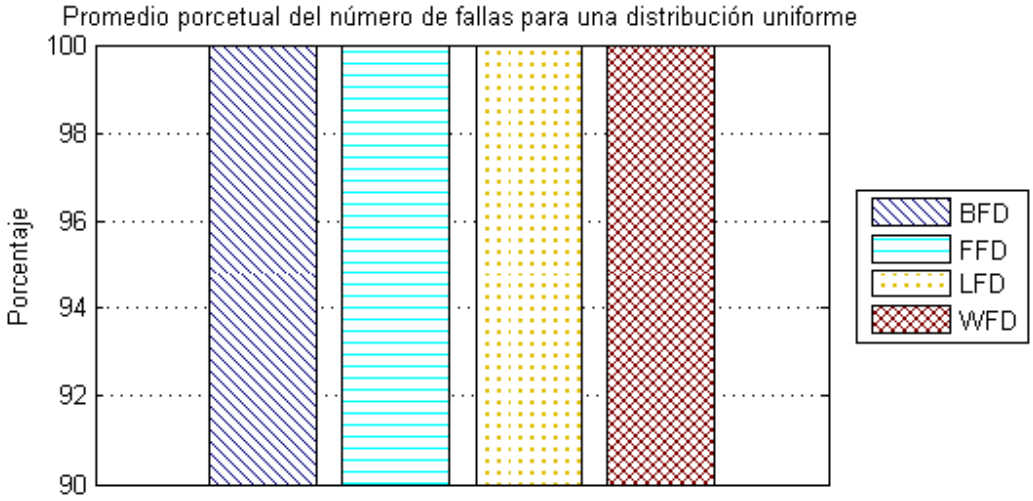

Figura 33. Promedio porcentual del número de fallas promedio con m=100 y para la asignación de tareas con técnicas de bin packing para una distribución normal.

VI.6 Conclusiones: reasignación mediante técnicas de *bin packing*

En este capítulo se muestran varias heurísticas para la asignación de tareas mediante técnicas de *bin packing*.

Para el caso de las tareas con distribución normal, las heurísticas propuestas muestran un una alta confiabilidad en la reasignación de tareas con periodos cambiantes. Las heurísticas obtienen reasignaciones sin fallas hasta el 97% del decremento del periodo de las tareas. Además, todas las heurísticas presentan un comportamiento similar, a excepción de FFD que obtiene reasignaciones ligeramente mejores comparada con el resto de las heurísticas.

Por otro lado, para el caso de las tareas con distribución uniforme, las heurísticas propuestas muestran buena confiabilidad en la reasignación de tareas con periodos cambiantes. Las heurísticas presentan que existe una relación entre el número de fallas y el decremento del periodo de las tareas. También, es importante notar que el número de fallas es pequeño comparado con el número de experimentos, esto indica que en algunos casos las heurísticas encuentran una asignación correcta aunque el decremento del periodo este cerca del 100%. Además, en promedio todas las heurísticas presentan el mismo rendimiento en la búsqueda de reasignaciones sin fallas para el caso de tareas con distribución uniforme.

Capítulo VII

Conclusiones

VII.1 Resumen

En este trabajo se cumplieron con los objetivos presentados en la Sección I.3 a través de la metodología propuesta en I.4. Se presentó el marco teórico de los temas relacionados con sistemas de tiempo real, de calendarización de tareas de tiempo real y de calendarización en sistemas distribuidos de tiempo real, esto se discutió en el Capítulo II. En los capítulos III, IV, V y VI se trató con el modelado del problema de reasignación de tareas con restricciones temporales y con la optimización de la reasignación de tareas a procesadores basados en diferentes modelos: arquitecturas de bus común, arquitectura de arreglo lineal de procesadores, y un problema que permite la aplicación de algunas técnicas conocidas de *bin packing.*

Se realizó el análisis experimental a través de simulación por computadora. Se evaluó el desempeño de las heurísticas propuestas para los diferentes modelos de reasignación propuestos a través de las métricas diseñadas. Se presentaron las conclusiones para los modelos propuestos a través de comparaciones del rendimiento promedio y desviaciones estándar para cada uno de los modelos.

VII.2 Conclusiones Finales

En este trabajo de tesis se trata con la optimización del costo de reasignación para diferentes modelos distribuidos dinámicos de tiempo real. Para cada modelo particular el se propusieron heurísticas eficientes. De acuerdo a los resultados experimentales se presentan las siguientes conclusiones para este trabajo de tesis:

1. Para el modelo I: arquitectura de bus común, el algoritmo húngaro es un algoritmo óptimo que soluciona el problema de reasignación de tareas en sistemas distribuidos de tiempo real, bajo la topología de bus común. Pero su complejidad $O(m^3)$, no es necesariamente satisfactorio, para los requerimientos de los sistemas de tiempo real. Por lo tanto, se presentan heurísticas que obtienen resultados de manera más rápida pero disminuyendo la calidad de la solución. La complejidad de la heurística Eval_swap con, $k = m^2$, es $O(m^2)$. Comparado con el algoritmo húngaro su complejidad en tiempo es m veces menor y presenta la mejor calidad de la solución en comparación con las otras heurísticas, con un 4% peor respecto del algoritmo óptimo. Además, consume sólo el 25% del tiempo de ejecución comparado con el algoritmo óptimo

2. Para el modelo II: arquitectura de arreglo lineal de procesadores, se presentan varias heurísticas para la minimización del calendario de transmisión para la topología de arreglo lineal de procesadores con restricciones de sistemas de tiempo real. Las heurísticas propuestas muestran un rendimiento satisfactorio en la relación tiempo versus calidad de la solución. Además todas estas heurísticas consiguen obtener una solución factible para el problema de reasignación de tareas y presentan mejoras de hasta el 40% respecto de una solución inicial.

3. Para el modelo III: reasignación mediante técnicas de *bin packing*, se presentan varias heurísticas para la asignación de tareas a procesadores. Para el caso de las tareas con distribución normal, las heurísticas propuestas muestran un una alta confiabilidad en la reasignación de tareas con periodos cambiantes. Las heurísticas obtienen reasignaciones sin fallas hasta el 97% del decremento del periodo a partir de una asignación inicial dada tomada como 0%. Para el caso de las tareas con distribución uniforme, las heurísticas propuestas muestran buena confiabilidad en la reasignación de tareas con periodos cambiantes. Las heurísticas presentan el número de fallas es pequeño comparado con el número de experimentos, esto indica que en algunos casos las heurísticas encuentran una asignación correcta aunque el decremento del periodo este cerca del 100%.

VII.3 Trabajo Futuro

Como trabajo futuro se planea el estudio de nuevas topologías de trasmisión para ambientes distribuidos con restricciones de sistemas de tiempo real. Esto es importante, ya que la transmisión de las tareas, y por tanto el costo de reasignación de las mismas, depende directamente de la topología que se utilice para la transmisión de las tareas.

También, se planea presentar nuevos algoritmos de optimización. Basados en el estudio y tratamiento estadístico de los datos, para diferentes casos de ejecución de los algoritmos, como pueden ser: tiempo de ejecución del mejor caso, tiempo de ejecución promedio, y tiempo de ejecución del peor caso.

También, se planea realizar las pruebas analíticas, que permitan encontrar cotas asintóticas para las heurísticas propuestas en cada uno de los modelos de reasignación presentados en este trabajo. Esto es encontrar un factor de

aproximación de las soluciones obtenidas para los modelos particulares estudiados.

Referencias

Andersson B., Baruah S., Jonsson J. 2001. *Static-Priority Scheduling on Multiprocessors*. Proceedings of the 22nd IEEE Real-Time Systems Symposium, London UK, 193-202p.

Antsaklis P., Kohn W., Nerode A., Sastry S. 1993. *Hybrid Systems II*, LNCS, Springer 999: 569pp.

Arora A., Gouda M. 1993. *Closure and convergence: A foundation for fault-tolerant computing*, IEEE Transactions on Software Engineering, vol. 19: 1015-1027p.

Bate I., Burns A. 2003. *An Integrated Approach to Scheduling in Safety-Critical Embedded Control Systems*. Real-Time Systems 25: 5-37p.

Blazewics J., Ecker K. H., Pesch E., Schmidt G., Weglarz J. 2001. *Scheduling Computer and Manufacturing Systems*. Springer, 485pp.

Blazewics J., Ecker K. H., Pesch E., Schmidt G., Weglarz J., 2007. *Handbook on Scheduling: From Theory to Applications*. Springer, 646pp.

Caprara A., Kellerer H., Pferschy U. 1998. *The Multiple Subset Sum Problem*, SIAM Journal of Optimization. 308-319p.

Coffman E. G., Garey M. R., Johnson D. S. 1996. *Bin Packing Approximation Algorithms: A Survey in Approximation Algorithms for NP-Hard Problems*. En: D. Hochbaum (ed.), PWS Publishing Co., Boston, MA,USA, 46-93p.

Dhall S. K., Liu C. L., 1978. *On a Real-Time Scheduling Problem*. Operations Research 26: 127-140p.

Dósa G., 2007. *The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is FFD(I)≤(11/9)OPT(I)+6/9*. ESCAPE: *4614*: 1-11p.

Ecker K. H., Juedes D., Welch L., Chelberg D., Bruggeman C., Drews F., Fleeman D., Parrot D., Pfarr B., 2003. *An Optimization Framework for Dynamic Distributed Real-Time Systems*. IPDPS 2003, Workshop on Parallel and Distributed Real-Time Systems, IEEE Computer Society, Washington, DC, USA, 111pp.

Ecker K. H., Tchernykh A., Drews F., Schomann S., 2004. *Continuous Mode Changes in Mechatronic Systems*. Proceedings of the Mexican International Conference on Computer Science (Encuentro internacional de ciencias de la computación), Colima, Mexico, 407-414p.

Gerber R., 1995. *Guaranteeing end-to-end timing processes*. Proceedings. IEEE Real-Time System, 192-203pp.

Grossman R. L., Nerode A., Ravn A. P., Rischel H., 1993. *Hybrid Systems*. Spinger LNCS, 73:593p.

Huang J., Du D. Z., 1994. *Resource Management for Continuous Multimedia Database Applications,* Proceedings of the IEEE Real-Time Systems Symposium, Minneapolis, MN USA, 46-54p.

Huo C. J., Shin K. G., 1994. *Load Sharing with Consideration of Future Task Arrivals in Heterogeneous Distributed Real-Time Systems*. IEEE Transactions on Computers, 43(9): 1076-1090p.

Kalogeraki V., Melliar-Smith P. M., Moser L. E., 2000. *Dynamic Scheduling for Soft Real-Time Distributed Object Systems,* Proceedings of 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Washington, DC, USA, 114-122p.

Kalogeraki V., Melliar-Smith P. M., Moser L. E., 1999. *Using Multiple Feedback Loops for Object Profiling, Scheduling and Migration in Soft Real-Time Distributed Object Systems*. Proceedings of the IEE 2nd International Symposium on Object-Oriented Real-Time Distributed Computing, Saint Malo, France, 291-300p.

Kopetz H., 1997. *Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 537pp.

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

Koymans R., Kuiper R., Zijlstra E., 1988. *Paradigms for real-time systems*. En: M. Joseph (ed.), Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer LNCS 331:159-174p.

Kuhn H., 1956. *Variants of the Hungarian Method for the Assignment Problems*, Naval Research, Logist Quart. 3: 253-258p.

Kurose J. F., Towley D., Krishna C. M., 1991. *Design an Analysis of Processor Scheduling Policies for Real-Time Systems*, En:A. v. Tilborg (ed.), Foundations of Real-Time Computing: Scheduling an Resource Management, 63-89, Kluwer Academic Publishers.

Lopez J. M., Diaz J. L., Garcia D. F., 2004. *Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems*, Real-Time Systems Journal 28, 39-68p.

Liu J. W. S., 2000. *Real-Time Systems*, Prentice Hall.

Liu C., Layland J., 1973. *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*, Journal of ACM 20, 46-71p.

Peltola M., 2008 *"Example 1: Hungarian method"*, http://www.ee.oulu.fi/~mpa/matreng/eem1_2-1.htm. (Octubre de 2008)

Mortello S., Toth P., 1990. *Knapsack Problems: Algorithms and Computer Implementations*, Wiley-Interscience series in discrete mathematics and optimization.

Peng D. T., Shin K. G., Abdelzaher T. F., 1997. *Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems*, IEEE Transactions on Software Engineering, 23(12), 745-758pp.

Pinedo M., 2002. *Scheduling. Theory, Algorithms, and Systems*, Prentice Hall.

Pirlot M., 1996. *General local search methods,* Elsevier, European Journal of Operational Research 92 (1996) 493-511pp.

Qiao Y., Wang H., Dai G., 2001. *A new Dynamic Scheduling Algorithm for Real-Time Multiprocessor Systems*, Proceedings of the International Workshop on Distributed an Parallel Systems Embedded Systems (DIPES 2000), Kluwer Academic Publishers, 173-182pp.

Rajkumar R., 1991. *Synchronization in Real-Time Systems : A Priority Inheritance Approach,* Kluwer *Academic Publishers, 200pp.*

Schomann S., 2006. *Adaptive Resource Management in Distributed Real-Time Systems with continuous Mode Changes*, PhD Thesis, Clausthal University of Technology, Germany, 130pp.

Sha L., Klein M. H., Goodenough J. B., 1991. *Rate Monotonic Analysis for Real-Time Systems*, A. v. Tilborg (ed.), Foundations of Real-Time Computing: Scheduling and Resource Management, Kluwer Academic Publishers, 129-155p.

Yue M., 1991. *A simple proof of the inequality FFD(L) ≤ (11/9)OPT(L) + 1, for all L, for the FFD bin-packing algorithm*, Acta Mathematicae Applicatae Sinica 7, 321-331p.

Apéndice A: Tabla de símbolos

La tabla de símbolos que se utilizan en el presente trabajo son las siguientes:

Tabla VI. Tabla de símbolos.

$\mathcal{T}$	Conjunto de tareas
$\mathcal{H}$	Conjunto de procesadores
R	Conjunto de recursos
p_j	Tiempo de procesamiento del peor caso
r_j	Tiempo de llegada
d_j	Fecha límite
π_j	Período de la tarea
l_j	Holgura de la tarea
$\tilde{S}$	Calendario
$u_j = \frac{p_j}{\pi_j}$	Utilización del procesador
$U = \sum_{j=1}^{n} (\frac{p_j}{\pi_j})$	Utilización total del procesador
U^{max}	Utilización máxima del procesador
U^{min}	Utilización mínima del procesador
$(\mathcal{T}_1, \dots, \mathcal{T}_m)$	Partición de tareas
α^{cur}	Asignación actual
α^{new}	Asignación nueva
$\mathcal{T}^{cur}$	Subconjunto actual de tareas
$\mathcal{T}^{cur}$	Subconjunto nuevo de tareas

Apéndice B: Ejemplo de ejecución del algoritmo húngaro

Para mostrar el comportamiento del algoritmo húngaro se presenta el siguiente ejemplo. Los costos de transmisión para la matriz C_{ij} para de los elementos de $\mathcal{T}_j^{new}$ en H_i para j = 1,2,3 e i = 1,2,3 se muestran en la figura 34:

	$\mathcal{T}_1^{new}$	$\mathcal{T}_2^{new}$	$\mathcal{T}_3^{new}$	$\mathcal{T}_4^{new}$
H_1	90	75	75	80
H_2	35	85	55	65
H_3	125	95	90	105
H_3	45	110	95	115

Figura 34. Matriz de costos C_{ij} para el ejemplo de ejecución del algoritmo húngaro.

Los pasos que realiza el algoritmo húngaro sobre la matriz de costos C_{ij} se describen a continuación:

	$\mathcal{T}_1^{new}$	$\mathcal{T}_2^{new}$	$\mathcal{T}_3^{new}$	$\mathcal{T}_4^{new}$
H_1	15	0	0	5
H_2	0	50	20	30
H_3	35	5	0	15
H_4	0	65	50	70

Figura 35. Paso 1: algoritmo húngaro.

1. Restar las entradas de cada fila para i = 1,. . ,m, por el mínimo de la fila.
    a. Cada fila tiene al menos un cero.
    b. Todas las entradas son positivas o cero.

	$\mathcal{T}_1^{new}$	$\mathcal{T}_2^{new}$	$\mathcal{T}_3^{new}$	$\mathcal{T}_4^{new}$
H_1	15	0	0	0
H_2	0	50	20	25
H_3	35	5	0	10
H_4	0	65	50	65

Figura 36. Paso 2: algoritmo húngaro.

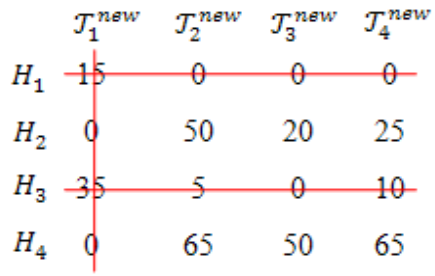

Figura 37. Paso 3 y 4: algoritmo húngaro primera iteración.

	$\mathcal{T}_1^{new}$	$\mathcal{T}_2^{new}$	$\mathcal{T}_3^{new}$	$\mathcal{T}_4^{new}$
H_1	35	0	0	0
H_2	0	30	0	5
H_3	55	5	0	10
H_4	0	45	30	45

Figura 38. Paso 5: algoritmo húngaro primera iteración.

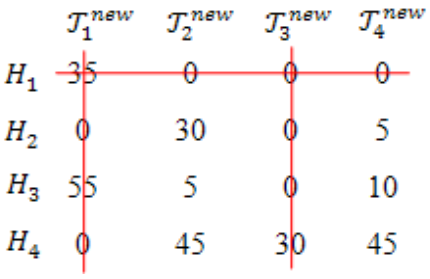

Figura 39. Paso 3 y 4: algoritmo húngaro segunda iteración.

2. Restar las entradas de cada columna para $i = 1, . . , m$ por el mínimo de la columna.
    a. Cada fila y cada columna tienen al menos un cero.

3. Seleccionar las filas y columnas a través de las cuales se dibujen líneas, de tal modo que todos los ceros sean cubiertos y que no mas líneas necesarias hayan sido dibujadas.

4. Una prueba de optimización.
    a. Si el número de las líneas es *m,* escoger una combinación modificada para el costo de la matriz de tal manera que la suma sea cero.
    b. Si el número de las líneas es < *m* ir al paso 5.

	T_1^{new}	T_2^{new}	T_3^{new}	T_4^{new}
H_1	40	0	5	0
H_2	0	25	0	0
H_3	55	0	0	5
H_4	0	40	30	40

Figura 40. Paso 5: algoritmo húngaro segunda iteración

	T_1^{new}	T_2^{new}	T_3^{new}	T_4^{new}
H_1	40	0	5	0
H_2	0	25	0	0
H_3	55	0	0	5
H_4	0	40	30	40

Figura 41. Fin algoritmo húngaro.

	T_1^{new}	T_2^{new}	T_3^{new}	T_4^{new}
H_1	40	0	5	**0**
H_2	0	25	**0**	0
H_3	55	**0**	0	5
H_4	**0**	40	30	40

Figura 42. Solución ejemplo algoritmo húngaro.

5. Encontrar el elemento más pequeño el cual no es cubierto por ninguna de las líneas, restárselo a cada entrada que no ha sido cubierta por las líneas y añadirlo a cada entrada la cual está cubierta por una línea vertical y horizontal. Volver al paso 3.

La figura 43 muestra las posiciones que representan la solución para la matriz de costos C_{ij}. Las posiciones escogidas son C_{14}, C_{23}, C_{32} y C_{41}. Con un costo total de 80 + 55 + 95 + 45 = 275.

	$\mathcal{T}_1^{new}$	$\mathcal{T}_2^{new}$	$\mathcal{T}_3^{new}$	$\mathcal{T}_4^{new}$
H_1	90	75	75	**80**
H_2	35	85	**55**	65
H_3	125	**95**	90	105
H_3	**45**	110	95	115

Figura 43. Solución de la matriz de costos C_{ij}, y las posiciones escogidas para el ejemplo de ejecución del algoritmo húngaro.