CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN COMPUTACIÓN

ANÁLISIS COMPARATIVO DE ALGORITMOS DE PLANIFICACIÓN DE TIEMPO REAL EN SISTEMAS DE MULTIPROCESAMIENTO

Tesis que presenta:

OMAR ULISES PEREIRA ZAPATA

Para obtener el grado de:

MAESTRO EN CIENCIAS

En la especialidad de:

INGENIERÍA ELÉCTRICA
OPCIÓN COMPUTACIÓN

Director de Tesis:

DR. PEDRO MEJÍA ALVAREZ

México D.F. Enero del 2003

Resumen

El problema de planificar un conjunto de tareas de tiempo real sobre un sistema multiprocesador de tal forma que ninguna de ellas pierda su plazo de respuesta, y que el número de procesadores resultante de dicha planificación sea mínimo, es conocido como un problema computacionalmente intratable. En los algoritmos encargados de la asignación de un conjunto de tareas a sistemas multiprocesadores se presenta el problema de encontrar un equilibrio entre la complejidad computacional y el rendimiento. El problema de encontrar la asignación óptima de tareas a procesadores es generalmente impráctico, por lo cual se utilizan algoritmos heurísticos de bajo costo computacional que proporcionan resultados muy cercanos al óptimo.

Los sistemas de tiempo real que aquí se consideran son aquellos en los cuales las tareas son ejecutadas de manera periódica, cada tarea del conjunto es crítica (no se permite la pérdida de plazos de respuesta) y se ejecuta de manera concurrente con otras tareas. El objetivo principal de esta tesis, es presentar un análisis comparativo de distintos algoritmos heurísticos que resuelven el problema de planificación de un conjunto periódico de tareas desalojables sobre un número mínimo de procesadores posibles, de tal forma que todos los plazos de las tareas se cumplan en cada procesador, utilizando el algoritmo Rate Monotonic.

En esta tesis, se diseñó una interfaz gráfica que incluye algunos de los mejores algoritmos en línea y fuera de línea sobre múltiples procesadores. En esta interfaz se muestran los rendimientos en los peores casos y los rendimientos en los casos promedios de los algoritmos. Los rendimientos en los peores casos de los algoritmos son presentados a través de un análisis complejo en el cual se visualizan resultados cercanos al óptimo. Los rendimientos en los casos promedios son exhibidos por medio de experimentos de simulación.

Abstract

The scheduling problem involving a set of real-time tasks over a multiprocessor systems such that all tasks meet deadlines and the number of processors is minimum, is known to be computationally intractable for large tasks sets. Any practical scheduling algorithm for assigning real-time tasks to a multiprocessor system introduces a trade-off between its computational complexity and its performance. Finding the optimal scheduling solution is generally impractical, therefore in the solution of this problem we use computationally low-cost heuristic algorithms that produce results very close to the optimal one.

We consider real-time systems in which tasks are executed periodically; each task has an infinite number of requests and there are multiple tasks being executed at a giving time. Our main goal is to introduce a comparative analysis of algorithms for the scheduling problem of a set of periodic, preemptive tasks using as few processors as possible such that tasks deadlines are met on each processor by the Rate Monotonic Scheduling Algorithm.

In this thesis, a graphical interface is presented that includes the best on-line and off-line scheduling algorithms currently known. We observe the worst case performance and average case performance of each algorithm studied. The worst case performance of the algorithms is shown to have constant tight bounds through complex analysis. The average case performance is assessed by conducting simulation experiments.

,

En memoria de mi pequeñito,

José Yharím Pereira
19/Marzo/2000 - 1/Marzo/2002

Quien me enseñó hasta el último minuto,
lo que es luchar por la vida.

Agradecimientos

Al CONACYT por su apoyo económico durante todo el transcurso de la maestría.

Al CINVESTAV por su apoyo económico a través del programa de becas terminales.

Un agradecimiento especial a mi asesor el Dr. Pedro Mejía Alvarez por sus consejos y dedicación, ya que sin su ayuda no hubiera terminado este trabajo.

A todos los doctores de la sección de computación por sus excelentes clases, en especial a los doctores: Dr. Pedro Mejía Alvarez, Dr. Carlos Artemio Coello Coello, Dr. Gerardo de la Fraga, Dr. José Oscar Olmedo, Dr. Adriano de Luca, Dr. Arturo Díaz, Dr. Jorge Buenabad, Dra. Xiaoou Li, Dr. Guillermo Morales y al Dr. Sergio Chapa.

A mis padres y a mi hermano por la educación, comprensión y apoyo que a través de los años me han brindado.

A la Dra. Guillermina Gil Gaytán que por sus consejos y regaños me ha enseñado a superarme cada día.

A Sofia Reza por todo su apoyo en el manejo administrativo.

Índice General

Índice de Figuras

Índice de Tablas

Lista de Símbolos

α	**La máxima utilización permitida de una tarea $\alpha = max_i(C_i/T_i)$.**
β	**La diferencia entre dos valores S_i de tareas.**
$\sum$	**Un conjunto de tareas.**
τ_i	**La i-ésima tarea.**
$\tau_{j,i}$	**La i-ésima tarea asignada al procesador P_j.**
A	**Representa un algoritmo heurístico.**
B_j	**Representa el j-ésimo bin o depósito.**
c	**Indica la capacidad de un bin o depósito.**
C_I	**El tiempo de cómputo de la tarea τ_i.**
D_i	**El plazo de la tarea τ_i.**
K_j	**El número de tareas asignadas al procesador j.**
L	**Representa una lista de objetos.**
m	**El número de tareas en un conjunto.**
M	**El número de clases de un algoritmo.**
n	**El número de tareas en un conjunto.**
N	**El número de procesadores requerido para planificar un conjunto $\sum$ de tareas.**
N_0	**El mínimo número de procesadores requerido para planificar un conjunto $\sum$ de tareas.**
$N(A)$	**El número de procesadores requerido para planificar un conjunto $\sum$ de tareas**

	utilizando la heurística A.
P_i	**El i-ésimo procesador.**
r	**La diferencia máxima entre dos periodos de tareas cualesquiera.**
$\Re_A^\infty$	**El rendimiento en el peor caso de la heurística A.**
S_i	**El valor de la tarea τ_i. $S_i = log_2 T_i - \lfloor log_2 T_i \rfloor$.**
T_i	**El periodo de la tarea τ_i.**
U_T	**La utilización del conjunto $\sum$ de tareas. $U = \sum_{i=1}^{n} C_i/T_i = \sum_{i=1}^{n} u_i$.**
U_i	**La utilización del procesador P_i.**
u_i	**La utilización de la tarea τ_i.**
$u_{j,i}$	**La utilización de la tarea $\tau_{j,i}$.**
W_i	**El peso del objeto i.**

Capítulo 1

Introducción

El problema de planificar un conjunto de tareas de tiempo real sobre un sistema multiprocesador de tal forma que ninguna de ellas pierda su plazo de respuesta y que el número de procesadores resultante de dicha planificación sea mínimo, es conocido como un problema computacionalmente intratable (NP-Duro [6]).

Muchas aplicaciones tales como aviónica, realidad virtual, control de procesos, robótica, plantas nucleares, sistemas de defensa, etc., no serían factibles sin el uso de sistemas multiprocesadores. Por ejemplo, algunos aviones de combate (como el DD-21) requieren de 500 a 1000 procesadores [13].

El uso de estos sistemas multiprocesadores en aplicaciones de tiempo real se debe principalmente a dos razones: (1) El dominio de los sistemas de tiempo real se ha extendido considerablemente, con lo cual los requerimientos de cómputo utilizados para satisfacer las necesidades de éstos son cada vez mayores, (2) El estado del arte en la tecnología de hardware permite actualmente el soporte de sistemas con múltiples procesadores para las aplicaciones de tiempo real. Debido a estas razones, la planificación de tareas periódicas sobre multiprocesadores ha llegado a ser un problema de actualidad.

En los últimos años, el desarrollo de algoritmos de planificación que utilizan

un modelo de tareas periódicas sobre multiprocesadores ha sido objeto de estudio por un gran número de investigadores en el área de los sistemas de tiempo real [2, 4, 6, 9, 11, 12, 7], en donde muchas de las heurísticas planteadas son aplicadas a control de procesos, telecomunicaciones, etc. En los algoritmos encargados de la asignación de un conjunto de tareas a sistemas multiprocesadores se presenta el problema de encontrar un equilibrio entre la complejidad computacional y el rendimiento. El problema de encontrar la asignación óptima de tareas a procesadores es generalmente impráctico debido a su alta complejidad computacional, por lo cual se utilizan algoritmos heurísticos de bajo costo computacional que proporcionan resultados muy cercanos al óptimo. La principal motivación de este trabajo se debe a que en la actualidad, no existe un estudio que compare el rendimiento de cada uno de estos algoritmos bajo condiciones similares.

1.1 Generalidades

Los sistemas de tiempo real son aquellos en los cuales su correcto funcionamiento depende del tiempo en que los resultados son entregados. Por lo tanto, la ejecución correcta de un sistema de tiempo real no sólo depende de su funcionamiento lógico, sino también del tiempo en el cual el sistema responde. Estos sistemas son de alta complejidad y los podemos encontrar en ambientes tales como control de procesos, robótica, control de tráfico aéreo, realidad virtual, multimedia, o sistemas distribuidos. Existen principalmente dos categorías en las cuales se pueden clasificar a los sistemas de tiempo real: Sistemas de tiempo real *duros (Hard)* y sistemas de tiempo real *suaves (Soft).* En un sistema de tiempo real duro, una respuesta tardía es una respuesta incorrecta e inservible. Por el contrario, en un sistema de tiempo real suave la respuesta tardía puede ser caracterizada dándole un cierto valor de utilidad, dependiendo de qué tan

tarde se produce dicha respuesta.

Con el propósito de maximizar el número de tareas de tiempo real que pueden ser procesadas sin que se violen las restricciones de tiempos, los sistemas de tiempo real utilizan algoritmos de planificación que deciden un orden de ejecución de las tareas. Los algoritmos de planificación pueden dividirse en algoritmos con manejo *fijo* de prioridades y algoritmos con manejo *dinámico* de prioridades. En los algoritmos con manejo de prioridades fijas, la prioridad de cada tarea del sistema es asignada *a priori* (el conjunto completo de tareas es conocido antes de la ejecución) y no cambia durante la ejecución. Mientras que en un algoritmo con manejo de prioridad dinámico la prioridad de una tarea puede cambiar durante su ejecución en cualquier momento (entre una petición y otra). Liu y Layland [12] desarrollaron las condiciones para planificar un conjunto de tareas con manejo de prioridad fija en un sistema uniprocesador; ellos detallaron el algoritmo de planificación ***Rate Monotonic (RM)***, el cual asigna las prioridades más altas a las tareas con periodos más pequeños, y el algoritmo ***Earliest Deadline First (EDF)***, el cual asigna prioridades tomando en cuenta la cercanía de cada instancia de las tareas con su plazo de respuesta (la tarea con plazo más cercano recibe la más alta prioridad). Las condiciones de planificabilidad necesarias y suficientes para RM fueron presentadas por Lehoczky, et al. [9].

Se dice que un conjunto de tareas es *independiente*, si la petición de ejecución de cualquiera de las tareas no depende del inicio o fín de ejecución de alguna otra tarea en el sistema (restricción de precedencia), y además ninguna tarea comparte recursos entre sí. De la misma forma, si cualquier tarea del sistema puede interrumpirse (por alguna tarea de mayor prioridad) en cualquier punto de su ejecución (para luego continuar), se dice que es ***desalojable (preemptive)***, de lo contrario, una tarea que no es desalojable deberá ejecutarse de principio a

fín sin que otra tarea pueda interrumpirla.

La planificación puede llevarse a cabo de forma estática o de forma dinámica. En la planificación estática (*fuera de línea)*, el conjunto completo de tareas es conocido a priori desde el principio de la ejecución. En este esquema, no se permite el arribo de tareas nuevas durante la ejecución del sistema. Los algoritmos fuera de línea poseen la ventaja de ser eficientes y causar un mínimo tiempo de ejecución extra (*overhead*).

En una planificación dinámica (*en línea),* el tiempo de arribo y las características temporales de algunas tareas no son conocidas a priori, por lo que se requiere que el planificador decida en forma dinámica (en línea), mediante un mecanismo de aceptación, si dichas tareas pueden ser admitidas o no, de forma que no interfieran con el cumplimiento de plazos de respuesta de las tareas existentes en el sistema. En cualquiera de los siguientes escenarios, el conjunto de tareas deberá ser planificado de manera dinámica: (a) Cuando existen tareas que arriban al sistema en forma esporádica, (b) Cuando la falla de alguna tarea podría requerir su recuperación en línea, o (c) Cuando las características temporales de las tareas cambian en forma dinámica.

En los últimos años la disponibilidad de los sistemas de múltiples procesadores se ha incrementado, debido generalmente al descenso de costos, a su confiabilidad, a la necesidad de incrementar el poder de cómputo del sistema y a su relativa facilidad de programación. El cumplimiento de los tiempos de respuesta de un conjunto de tareas de tiempo real en un sistema multiprocesador, requiere de un algoritmo de planificación que determine para cada tarea del sistema en qué procesador debe ejecutarse (problema de asignación) y cuándo y en qué orden, con respecto a otras tareas debe comenzar su ejecución (problema de planificación). Los algoritmos de asignación proveen un conjunto de reglas que

determinan el procesador o procesadores en donde cada tarea deberá ejecutarse. Una vez que todas las tareas del sistema son asignadas, el planificador decide el orden de prioridades de ejecución.

Se dice que un algoritmo de planificación es *factible,* si la ejecución de cada tarea en el sistema puede ser completada sin violar sus plazos de respuesta. En un ambiente de múltiples procesadores, se dice que una planificación factible es *mínima*, si no existe otra planificación factible con menos procesadores. Un algoritmo de planificación es *óptimo,* si para cualquier conjunto de tareas el algoritmo siempre encuentra una mínima planificación. En general, el problema de encontrar una planificación óptima para un conjunto de tareas es ***NP-Duro*** tanto para esquemas con un manejo fijo de prioridades, como en esquemas con un manejo dinámico de prioridades [11].

La planificación de tareas de tiempo real sobre multiprocesadores puede ser desarrollada siguiendo alguno de los siguientes esquemas: un esquema *particionado* o un esquema *global* [6, 1, 14]. Cuando se utiliza un esquema particionado, el control de admisión no solamente decide cuáles tareas deben ser aceptadas, sino también debe de crear una asignación de tareas a procesadores (figura 5.1).

En un esquema particionado todas las instancias de una tarea en particular son ejecutadas sobre el mismo procesador. En constrate, en un esquema global todas las instancias son almacenadas en una sola cola global y, en cualquier instante de tiempo, los procesadores ejecutan las instancias de las tareas que posean las prioridades más altas (figura 5.2).

De entre los dos esquemas (particionado y global), el método particionado ha recibido la mayor atención, debido principalmente a que se puede fácilmente garantizar una planificación factible de un conjunto de tareas usando como prueba de admisión una condición de planificabilidad para un sistema con un único

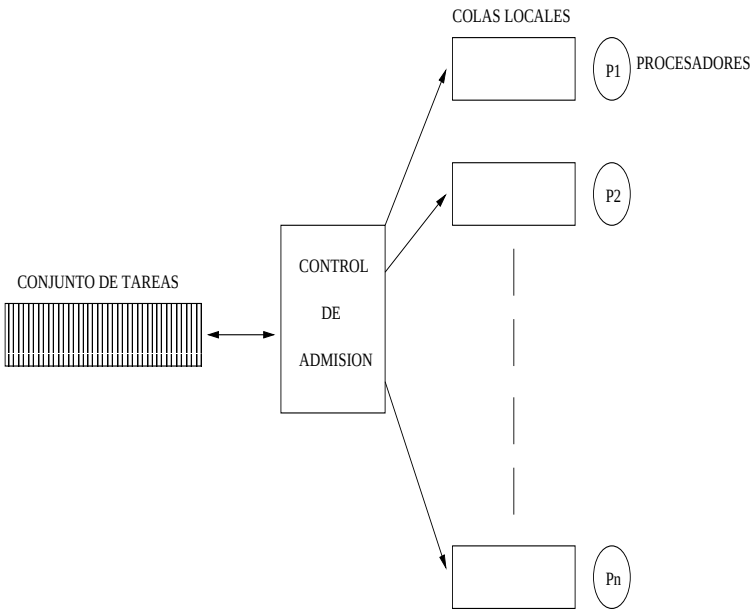

Figura 1.1: Esquema de Planificación Particionado

procesador. Podría parecer que el esquema particionado logra un menor rendimiento que el esquema global, por el hecho de imponer restricciones de lugar de ejecución a las tareas. Sin embargo, se ha demostrado [6] que una planificación global con RM puede dar una utilización del procesador arbitrariamente baja.

El esquema global ha recibido considerablemente menos antención, principalmente porque no existe una prueba de planificabilidad eficiente para él. Una prueba de planificabilidad necesaria y suficiente para el método global tiene una complejidad exponencial [10]. Otra prueba de planificabilidad suficiente con complejidad polinomial es presentada en [14]. Esta prueba llega a ser pesimista cuando el número de procesadores se incrementa.

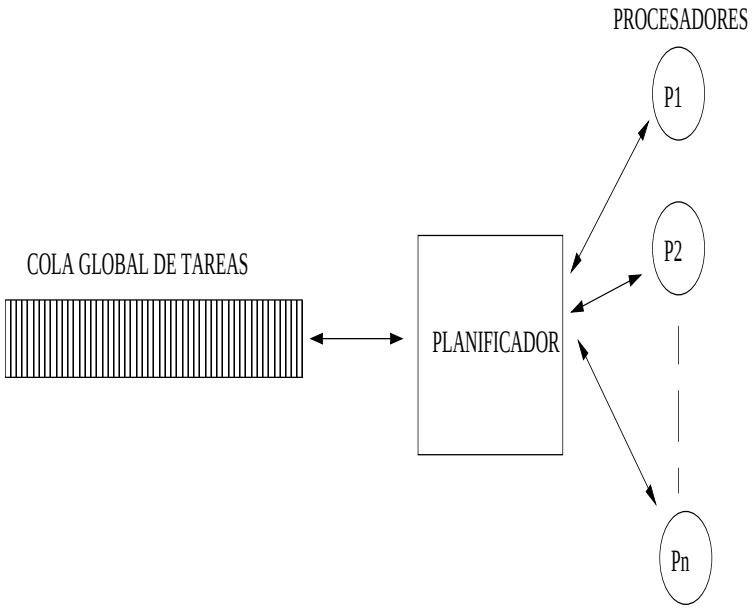

Figura 1.2: Esquema de Planificación Global

1.2 Antecedentes

Dhall y Liu [6] fueron los primeros en proponer dos algoritmos heurísticos para resolver el problema de asignación de tareas de tiempo real sobre multiprocesadores. En [6] fueron desarrollados los algoritmos: *Rate Monotonic First Fit* (RMFF) y *Rate Monotonic Next Fit* (RMNF). Estos algoritmos se basan en las técnicas heurísticas *Bin Packing First Fit y Next Fit* respectivamente. En ambos esquemas, las tareas son ordenadas de manera decreciente respecto de sus periodos antes del proceso de asignación. Las tareas son asignadas al procesador *actual* (al procesador que en ese momento el algoritmo de asignación utiliza) hasta que la condición de planificabilidad es violada. Después de esto, el procesador se marca como *utilizado* y se selecciona otro procesador ocioso (no utilizado). RMFF a diferencia de RMNF, primero trata de asignar una tarea en un procesador marcado como utilizado antes de asignarlo al procesador actual.

Davari y Dhall desarrollaron dos algoritmos: *First Fit Drecreasing Utiliza-*

tion Factor **(FFDUF) [4] y *Next Fit M* (NF-M) [5]. El algoritmo FFDUF es un algoritmo fuera de línea, el cual primero ordena las tareas de manera decreciente respecto de sus utilizaciones y después las asigna a los procesadores en ese orden. El algoritmo NF-M clasifica las tareas en M clases respecto de sus utilizaciones. Los procesadores también son clasificados en M clases, por lo que un procesador en la k-ésima clase ejecutará únicamente a las tareas correspondientes a su clase. En un sentido general, NF-M es un algoritmo en línea, pero su rendimiento depende en gran medida de una previa selección del número de clases a definir.**

Algunos de los algoritmos propuestos recientemente que resuelven el problema de asignación de tareas de tiempo real a multiprocesadores son: *Rate Monotonic Small Tasks* (RMST) [19], *Rate Monotonic General Tasks* (RMGT) [19], *Rate Monotonic General Tasks-M* (RMGT/M) [18], *Rate Monotonic First Fit Decreasing Utilization* (RM-FFDU) [21], *Rate Monotonic Best Fit* (RMBF) [22].

RMST es un algoritmo enfocado a conjuntos de tareas donde el factor de carga $u_i = C_i/T_i$ de cada tarea es pequeño comparado con la velocidad de cada procesador [19]. El algoritmo RMST considera los factores de carga de las tareas y los valores de sus periodos. La asignación de un conjunto de tareas con factores de carga pequeños usando el algoritmo RMST, da como resultado una alta utilización en los procesadores resultantes (con un alto factor de carga). Esto se debe principalmente a que en la condición de planificabilidad del algoritmo se puede incrementar la utilización del procesador siempre y cuando todos los periodos de todas las tareas del sistema posean valores cercanos unos de otros. El algoritmo RMST provee resultados satisfactorios cuando el máximo factor de carga de cualquier tarea del sistema es limitado por $u_i \leq 1/2$.

RMGT por su parte, es un algoritmo aplicable a conjuntos de tareas que no

poseen la restricción de tener factores de carga pequeños. RMGT particiona el conjunto de tareas en dos grupos, de tal forma que las utilizaciones de las tareas en el primer y segundo grupo satisfacen que $u_i \leq 1/3$ y $u_i > 1/3$ respectivamente. Las tareas correspondientes al primer grupo son asignadas usando el algoritmo RMST. Así mismo se utiliza la heurística first-fit para asignar las tareas que corresponden al segundo grupo.

RMGT/M es una versión en línea del algoritmo RMGT. En el algoritmo RMGT/M, el parámetro M denota el número de procesadores en los cuales una tarea puede ser asignada. RMGT/M posee las siguientes propiedades. Si una tarea se asigna dinámicamente a un procesador, las tareas existentes en dicho procesador permanecen sin ningún cambio. Sin embargo, si una tarea es removida de un procesador, posiblemente todas las tareas sobre ese procesador en particular sean reasignadas a otros procesadores.

Los algoritmos RMFF, RMNF, RMBF fueron extendidos para considerarlos en un esquema de asignación en línea. A estos nuevos algoritmos los nombraron: RMFF-WC [20], RMNF-WC [20], RMBF-WC [20]. En los tres algoritmos anteriores, WC significa que utilizan la condición de planificabilidad en el peor caso (*Worst Case*). Esta condición de planificabilidad (WC) [12] implica que un conjunto de tareas puede ser factiblemente planificada si el factor de utilización de todo el conjunto es menor que una cota máxima dada por $m(2^{1/m} - 1)$, donde m es el número de tareas a planificar.

Existen en la literatura varias extensiones al algoritmo RM, incluyendo manejo de tareas aperiódicas mediante el servidor esporádico [24], tolerancia a fallos [15] y manejo de sincronización de tareas [23]. El algoritmo R-BOUND-MP [16] fue propuesto como una extensión de la cota de planificabilidad del algoritmo Rate Monotonic para múltiples procesadores, en donde se incluye tolerancia a

fallos[16].

Más detalles sobre los fundamentos teóricos de los algoritmos antes discutidos serán presentados en el capítulo 4. De la misma manera, en el capítulo 5 se presentará un estudio comparativo del desempeño de estos algoritmos.

1.3 Modelo del Sistema

Generalmente un problema de planificación se define por cuatro parámetros: (1) El ambiente del sistema, (2) La caracterización de las tareas, (3) El ambiente de planificación, y (4) Los objetivos de planificación. El ambiente del sistema especifica los tipos de procesadores a utilizar. Se dice que los procesadores son homogéneos si la ejecución de una tarea en particular sobre cualquier procesador toma el mismo tiempo de cómputo (capacidad de procesamiento, acceso a memoria y acceso a dispositivos de entrada y salida), de otra manera, se considera que los procesadores son heterogéneos. La caracterización de las tareas permite especificar las restricciones de tiempos de las tareas, las relaciones y jerarquías entre ellas. El ambiente de planificación describe las restricciones impuestas sobre el algoritmo de planificación: si las tareas son desalojables o no; si la planificación se lleva a cabo en un esquema en línea o fuera de línea. Finalmente los objetivos nos definen las metas a alcanzar: reducir el número de procesadores, lograr que todas las tareas cumplan plazos, minimizar el tiempo de cómputo, etc. Considerando estos puntos, para nuestro estudio asumiremos las siguientes condiciones:

(1) Para el ambiente del sistema, se consideran procesadores homogéneos; el número de procesadores disponible se considera infinito (recordemos que es de nuestro interés usar el menor número de procesadores posible).

(2) Para la caracterización de las tareas, se consideran a éstas como indepen-

dientes. Los parámetros que definen a una tarea τ_i son, el tiempo de ejecución, C_i, el periodo, T_i, y el plazo de respuesta, D_i. Se consideran sólo tareas periódicas, es decir, que su arribo se realiza a intervalos de tiempo de duración fija, y asumiremos que el plazo de respuesta de una tarea corresponde a su siguiente tiempo de arribo (o periodo). El periodo y el tiempo de cómputo de la tarea τ_i satisface que: $T_i > 0$ y $0 \leq C_i \leq T_i = D_i, (i = 1, \ldots, n)$. Así mismo se define $u_i = C_i/T_i$ como el factor de utilización de la i-ésima tarea. El factor de utilización del conjunto de tareas U_T es la suma de los factores de utilización de las tareas en el conjunto, $U_T = \sum_{i=1}^{n} \frac{C_i}{T_i}$.

(3) Para el ambiente de planificación, se asume que las tareas pueden ser desalojadas en cualquier momento, es decir, en cierto instante el despachador determina qué tarea va a ejecutar. El costo del desalojo y arribo de una tarea se considera nulo. Las tareas no requieren un acceso exclusivo a otro recurso más que al procesador. Se utilizará un esquema de asignación *particionado* asumiendo que la ejecución de las tareas en cada procesador se lleva a cabo siguiendo un orden de prioridad decreciente. Esto es, siempre se ejecutará primero la tarea con la más alta prioridad de todas las que han arribado. Otro parámetro en el ambiente de planificación, es aquel relacionado con las características del conjunto completo de tareas. En este caso, consideraremos a la planificación fuera de línea (*off-line*) así como a la planificación en línea (*on-line*).

(4) Los objetivos de planificación son (i) Que todos los plazos de respuesta de las tareas se cumplan, (ii) Minimizar el número de procesadores requerido para planificar un conjunto de tareas de tiempo real.

1.4 Planteamiento del Problema

Considerando el modelo de tareas descrito en la sección anterior, el problema de planificación el cual estamos interesados en analizar se enfoca a tareas periódicas desalojables, en donde un esquema particionado es utilizado como estrategia de asignación de tareas a multiprocesadores. Se analizarán en particular aquellos algoritmos de planificación en línea y fuera de línea los cuales posean un manejo de prioridades fijas. La taxonomía del problema de planificación de tareas de tiempo real a multiprocesadores puede observarse en la figura 5.3 y una descripción formal del problema se describe a continuación.

Problema: Considere un sistema en el cual todas las tareas son periódicas. Para cada una de ellas, existe un número infinito de peticiones, en la cual cada una deberá ser ejecutada antes del siguiente arribo de la tarea. El plazo de respuesta de una petición es definido como el arribo de la siguiente petición. Dado un conjunto con estas características. ¿Cuál es el número mínimo de procesadores requerido para ejecutar el conjunto de tareas, de tal manera, que para cada una de ellas su tiempo de ejecución termine antes de su plazo de respuesta, si se utiliza una política de planificación Rate Monotonic?

1.5 Objetivos

1.5.1 Objetivo General

Como objetivo principal de este proyecto de tesis, se considera el desarrollo de una herramienta de simulación, que permita analizar el desempeño de distintos algoritmos de planificación de tareas de tiempo real para múltiples procesadores.

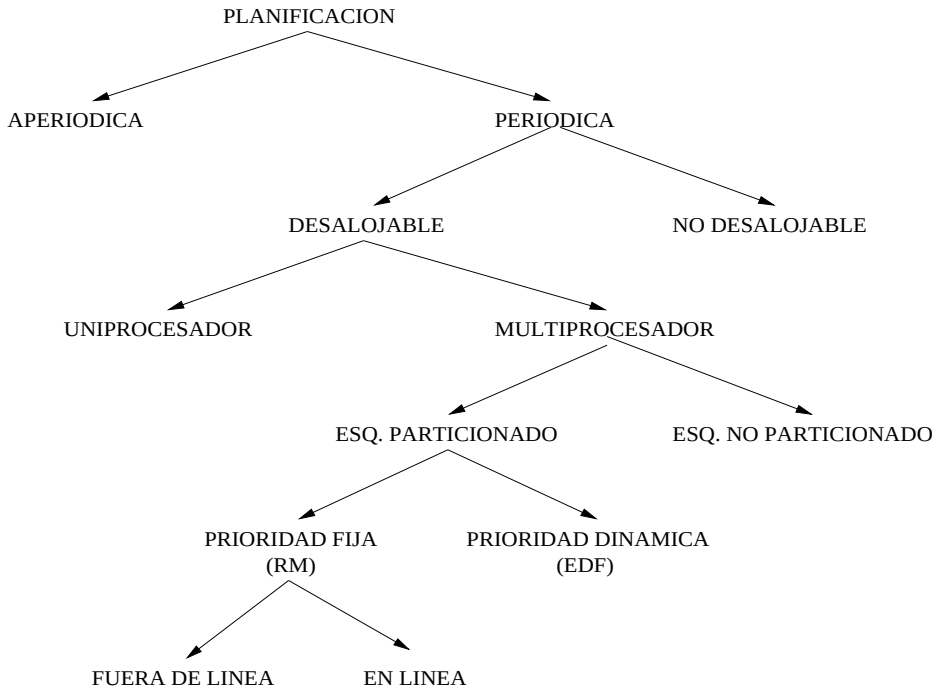

Figura 1.3: Taxonomía del Problema de Planificación

1.5.2 Objetivos Específicos

1. **Observar la importancia que presenta el algoritmo Rate Monotonic como política de planificación en los sistemas de tiempo real.**

2. **Verificar los rendimientos en los peores casos y en los casos promedio de diversos algoritmos heurísticos que resuelven el problema de planificación de tareas de tiempo real sobre sistemas multiprocesadores.**

   - **Los rendimientos en los peores casos se mostrarán a través de un análisis matemático.**
   - **Los rendimientos en los casos promedio se llevará a cabo a través de experimentos de simulación.**

3. **Redefinir algunas heurísticas de planificación, para mejorar rendimientos**

tanto en los peores casos como en los casos promedio.

4. Desarrollar una interfaz gráfica que permita visualizar el comportamiento de los algoritmos y sirva como base para analizar y comparar sus rendimientos.

En este documento de tesis se presenta un análisis comparativo de diversos algoritmos de planificación de tareas de tiempo real sobre multiprocesadores homogéneos utilizando un esquema particionado, en donde cada procesador es planificado por la política Rate Monotonic. Los algoritmos en este estudio son evaluados en condiciones similares con el propósito de realizar una comparación del rendimiento. Los algoritmos que se incluyen son: Rate Monotonic (RM), Rate Monotonic Next Fit (RMNF), Rate Monotonic First Fit (RMFF), Rate Monotonic Best Fit (RMBF), Rate Monotonic Small Tasks (RMST), Rate Monotonic General Tasks (RMGT), Rate Monotonic First Fit Decreasing Utilization (RM-FFDU), R-BOUND-MP, FFDUF y RMGT-M.

1.6 Organización del Documento

Se presenta en el capítulo 2 el algoritmo de planificación Rate Monotonic junto con diversas condiciones de planificabilidad. En el capítulo 3, se describe el problema Bin Packing presentando diversas heurísticas que resuelven este problema. El objetivo de este capítulo es presentar una solución al problema de asignación de tareas de tiempo real a sistemas multiprocesadores, utilizando heurísticas Bin Packing. Un estudio de diversos algoritmos de planificación de tareas de tiempo real sobre multiprocesadores es presentado en el Capítulo 4, mostrando los rendimientos en los peores casos de cada uno de los algoritmos. En el capítulo 5 se presentan los rendimientos de los algoritmos en los casos promedio, los cuales

son mostrados a través de experimentos de simulación. En este capítulo se describen los parámetros y condiciones de simulación. En el capítulo 6, se presenta la herramienta de simulación, describiendo los módulos que la componen, así como también la manera de uso de ésta. Finalmente las conclusiones y el trabajo futuro son descritos en el capítulo 7.

Capítulo 2

Algoritmos de Planificación en Sistemas Uniprocesadores

Entre los algoritmos de planificación con manejo de prioridades, la planificación de tareas periódicas desalojables [12] ha sido ampliamente aceptada como una teoría de planificación en los sistemas de tiempo real. En esta teoría, las tareas poseen un valor jerárquico conocido como la prioridad de la tarea, en donde una tarea con una prioridad baja puede en cualquier momento ser desalojada (se le retira el CPU) por una tarea con más alta prioridad. Por lo tanto, la importancia del diseño de estos algoritmos de planificación radica en la forma en que estas prioridades son asignadas. Dos algoritmos destacan en esta teoría: Rate Monotonic (RM) y Earliest Deadline First (EDF). Estos dos algoritmos tienen la propiedad de ser óptimos. Rate Monotonic es óptimo entre los algoritmos con manejo de prioridad de tareas fijas o estáticas. Earliest Deadline First es óptimo entre los algoritmos con manejo de prioridades dinámicas. La optimalidad de Rate Monotonic se debe a que si un conjunto de tareas es planificable con cualquier otro algoritmo con manejo de prioridad fija, entonces también es planificable bajo Rate Monotonic. En este capítulo se describirá el análisis de

planificabilidad de los algoritmos Earliest Deadline First y Rate Monotonic.

2.1 Earliest Deadline First (EDF)

EDF fue desarrollado por Liu y Layland en su trabajo "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment" [12]. En este algoritmo, las prioridades de las tareas son asignadas dependiendo de la cercanía del plazo de respuesta; la tarea con el plazo más cercano recibe la prioridad más alta. En cualquier instante, la tarea con más alta prioridad puede desalojar a la tarea que en ese momento utilice el procesador. La condición necesaria y suficiente para planificar un conjunto de tareas bajo EDF es:

$$\sum_{i=1}^{n}(C_i/T_i) \leq 1 \tag{2.1}$$

donde n es el número total de tareas.

EDF es un algoritmo óptimo en el sentido de que si un conjunto de tareas es factiblemente planificado por cualquier algoritmo de planificación de proridades dinámicas, entonces también es planificado bajo EDF. Algoritmos con manejo de prioridad dinámica tienen cierta ventaja sobre los algoritmos de prioridad fija. Una ventaja de este algoritmo, radica en el hecho de que la cota límite es (máxima) 100% para cualquier conjunto de tareas. Esto significa que el CPU puede ser utilizado totalmente. La principal desventaja de EDF se encuentra en que no existe una manera de identificar qué tareas perderán sus plazos de respuesta si un evento de sobrecarga ($\sum_{i=1}^{n}(C_i/T_i > 1)$ ocurre.

2.2 Algoritmo Rate Monotonic

Uno de los algoritmos que comúnmente se utiliza para planificar tareas desalojables de tiempo real es Rate Monotonic (RM). Esta política de planificación utiliza una regla que asigna las prioridades de las tareas de acuerdo a sus periodos: ***las tareas con periodos de activación más cortos reciben las prioridades más altas*** [12].

En [12], Liu y Layland demostraron que RM es óptimo entre todos los algoritmos con manejo de prioridades fijas. El criterio de optimalidad de RM se basa en el hecho de que ningún otro algoritmo de prioridad fija puede planificar un conjunto de tareas si éste no es planificable bajo RM.

2.2.1 Condición de Planificabilidad en el Peor Caso

Un control de admisión de tareas para RM que se basa en la utilización del procesador es descrito en [12]. En este control de admisión, se compara la utilización del conjunto de tareas con una cota límite que solamente depende del número de tareas en el sistema. Es decir, un conjunto de n tareas no perderá ningún plazo si se cumple la siguiente condición:

$$\sum_{i=1}^{n} C_i/T_i \leq n(2^{1/n} - 1) \tag{2.2}$$

Esta condición se conoce como Condición WC (Worst Case), la cual se formaliza en el teorema 1. En la tabla siguiente se describe el comportamiento de la ecuación 2.2 para distintos valores de n.

Teorema 1 ***(Condición WC) [12]: Si un conjunto*** $\sum$ ***de*** n ***tareas es planificable de acuerdo al algoritmo RM, entonces la mínima utilización del CPU es*** $n(2^{1/n} - 1)$. ***Cuando*** $n \to \infty$***, se obtiene,*** $n(2^{1/n} - 1) \to ln\ 2$.

Valor de n	$n(2^{1/n} - 1)$
1	1
2	0.8284
3	0.7788
4	0.7568
5	0.7434
..	..
..	..
$n \to \infty$	$n(2^{1/n} - 1) \to ln\ 2$

Tabla 2.1: Comportamiento de la Condición WC para diferentes valores de n

Es importante hacer notar, que si un conjunto de tareas cumple con la condición WC, entonces todas las tareas cumplirán con sus plazos de respuestas. Sin embargo, puede darse el caso en el que el conjunto de tareas no cumpla con esta condición, y aún así, el conjunto de tareas sea planificable. Por ejemplo, en la figura 2.1 se tiene el conjunto de tareas[1] $\sum = \{\tau_1(30, 10), \tau_2(40, 10), \tau_3(50, 12)\}$ **, donde claramente se observa que la utilización total de este conjunto es** $\sum_{i=1}^{3} C_i/T_i = 0.823$**, la cual es mayor que 0.778 establecida por la condición WC. La tarea** τ_3 **del conjunto pierde su plazo de respuesta en el tiempo T=50 (La tarea** τ_3 **posee la menor prioridad del conjunto utilizando el esquema de asignamiento de prioridad del algoritmo Rate Monotonic).**

Por otro lado, en la figura 2.2 se presenta un conjunto de tareas cuya utilización $\sum = \{\tau_1(16, 4), \tau_2(40, 5), \tau_3(80, 32)\}$ **no excede a la cota proporcionada por la**

[1] Generalmente en un modelo simple de tareas, una tarea de tiempo real se puede representar como un par ordenado (T,C) donde T representa el periodo de la tarea, mientras que C representa el tiempo de cómputo que ésta requeire para su ejecución. Así mismo, el plazo de respuesta de la tarea D es considerado igual al periodo, es decir, D=T.

condición WC, es decir, $\sum_{i=1}^{3} C_i/T_i = 0.775 < 0.778 =$ Condición WC. Por lo cual se garantiza que este conjunto de tareas no perderá ningún plazo de respuesta. Sin embargo, en la figura 2.3 se describe un conjunto de tareas cuya utilización del procesador es del 100% la cual obviamente no satisface la condición en el peor caso (WC), y aún así, el conjunto de tareas cumple con los plazos de respuesta. Es por esta razón, que a esta condición se le conoce como una condición de planificabilidad suficiente, pero no necesaria.

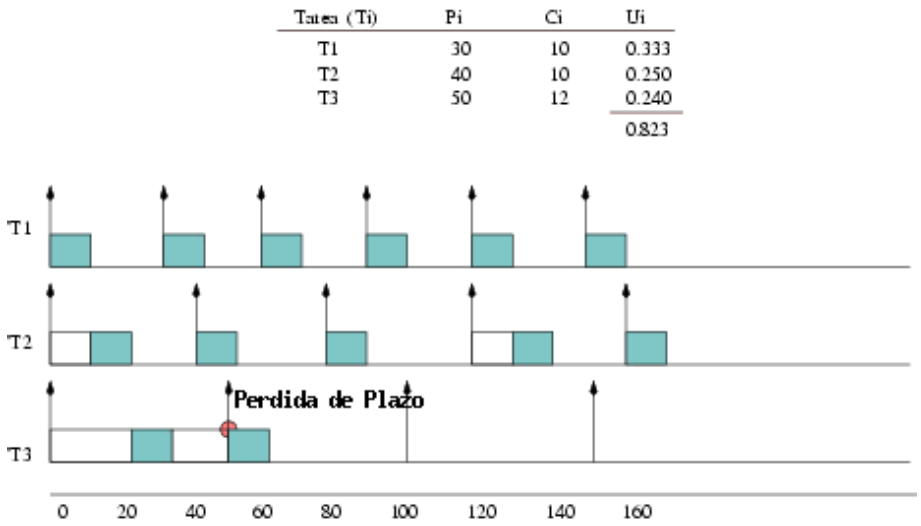

Figura 2.1: Conjunto de tareas que no satisface la condición WC y no es planificable

En la figura 2.1 se pueden apreciar representadas en una gráfica de Gantt el conjunto de tareas $\sum = \{\tau_1, \tau_2, \tau_3\}$. Las tareas se encuentran ordenadas de acuerdo a sus prioridades asignadas; la tarea τ_1 posee la mayor prioridad debido a que tiene el periodo más corto. Las flechas en la gráfica nos marcan los tiempos en donde las tareas solicitan la atención del procesador (el periodo de la tarea; es importante recalcar que en este caso, la siguiente petición de ejecución de una tarea es también su plazo de respuesta de la misma). Un rectángulo sombreado nos muestra la cantidad de tiempo de cómputo utilizado por la tarea; un

Tarea (Ti)	Pi	Ci	Ui
T1	16	4	0.250
T2	40	5	0.125
T3	80	32	0.400
			0.775

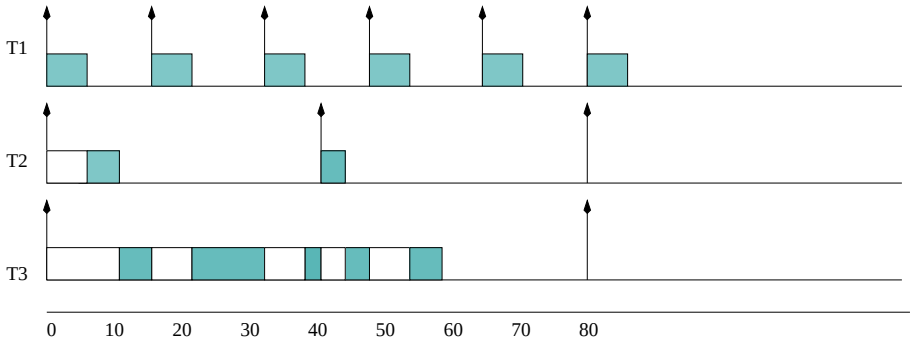

Figura 2.2: Conjunto de tareas que cumple con la condición WC y es planificable

Tarea (Ti)	Pi	Ci	Ui
T1	20	5	0.250
T2	40	10	0.250
T3	80	40	0.500
			1.000

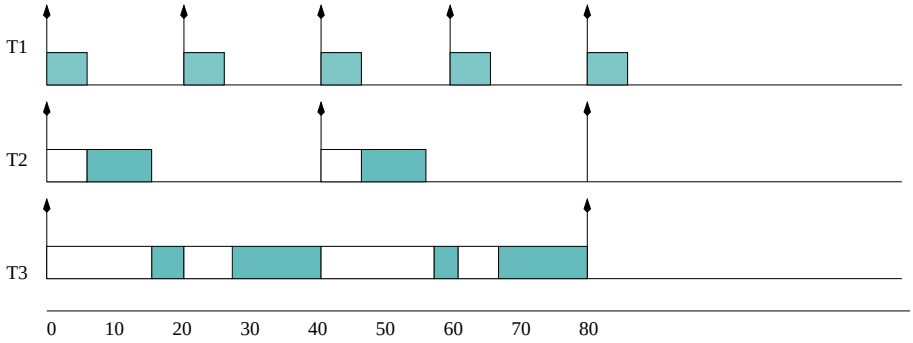

Figura 2.3: Conjunto de tareas que no cumple la condicón WC y es planificable

rectángulo en blanco nos indica la cantidad de tiempo que una tarea debe de esperar para poder utilizar el procesador.

2.2.2 Condición de Planificabilidad Suficiente y Necesaria

Lehoczky et al. [9] desarrollaron una condición de planificabilidad necesaria y suficiente para un modelo de tareas ejecutándose en un procesador. Esta condición, la cual denotaremos como Condición IFF, se define en el teorema 2:

Teorema 2 ***(Condición IFF) [9]: Sea*** $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ ***un conjunto de*** n ***tareas con*** $T_1 \leq T_2 \leq \ldots \leq T_n$***.*** τ_i ***puede encontrar una planificación factible usando RM si y sólo si,***

$$L_i = min_{\{t \in S_i\}} \left(W_i\left(t\right) / t\right) \leq 1 \tag{2.3}$$

El conjunto entero de tareas puede encontrar una planificación factible por el algoritmo RM si y sólo si:

$$L = max_{\{\, l \leq i \leq n \,\}} L_i \;\leq\; 1 \tag{2.4}$$

donde

$$S_i = \{\; k\, T_j \mid j = 1, \ldots, i;\; k = 1, \ldots, \lfloor T_i/T_j \rfloor \;\},$$
$$W_i\left(t\right) = \sum_{j=1}^{i} C_j \lceil t/T_j \rceil,$$
$$L_i\left(t\right) = W_i\left(t\right) / t,$$
$$L_i \;=\; min_{\{\, t \in S_i \,\}} L_i\left(t\right)$$

La condición IFF es exacta, en el sentido de que si el conjunto de tareas no es planificable bajo esta condición, se debe a que éstas no son planificables, y por lo tanto alguna(s) tarea(s) estaría(n) perdiendo su plazo de respuesta.

2.2.3 Condiciones de Planificabilidad Orientadas al Periodo

Entre las condiciones de planificabilidad orientadas al periodo se encuentran: Condición IP (Increasing Period), Condición RBOUND, y la Condición PO. La condición IP propuesta por Dhall y Liu [6] es utilizada para determinar el rendimiento de los algoritmos RMNF y RMFF.

Teorema 3 ***(Condición IP) [6]: Sea*** $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ ***un conjunto de*** n ***tareas con*** $T_1 \leq T_2 \leq \ldots \leq T_n$ ***y sea:***

$$u = \sum_{i=1}^{n-1} C_i \,/\, T_i \;\leq\; (n-1)(2^{1/(n-1)} \;-\; 1) \tag{2.5}$$

Si se cumple que,

$$C_n \,/\, T_n \;\leq\; 2(1 + \frac{u}{(n-1)})^{-(n-1)} \;-\; 1 \tag{2.6}$$

entonces el conjunto de tareas puede encontrar una planificación factible usando el algoritmo RM. Cuando $n \to \infty$***, la utilización mínima de*** τ_n ***se aproxima a*** $(2\, e^{-u} \;-\; 1)$**.**

En esta condición (teorema 3) se requiere que el conjunto de tareas esté ordenado de forma creciente con respecto de sus periodos, por lo cual, las características del conjunto de tareas deben ser conocidas a priori.

La Condición PO utilizada en el desarrollo de los algoritmos RMST y RMGT [19], considera que se puede incrementar la utilización del procesador si todos los periodos en el conjunto de tareas poseen valores cercanos unos a otros.

Teorema 4 ***(Condición PO) [19]: Dado un conjunto*** $\sum$ ***de tareas de tiempo real*** $\{\tau_1, \ldots, \tau_n\}$ ***, se define:***

$$S_i = log_2\ T_i - \lfloor\ log_2\ T_i\ \rfloor \qquad i = 1, \ldots, n \tag{2.7}$$

y

$$\beta = max\ S_i - min\ S_i \qquad 1\ \leq\ i\ \leq\ n \tag{2.8}$$

(a) si $\beta\ <\ 1\ -\ 1/n$ ***y la carga total satisface que:***

$$U\ \leq\ (n-1)(2^{\beta\ /\ (n\ -\ 1)}\ -\ 1)\ +\ 2^{1-\beta}\ -\ 1 \tag{2.9}$$

entonces el conjunto de tareas es planificable sobre un procesador utilizando el algoritmo RM.

(b) si $\beta\ \geq\ 1\ -\ 1/n$ ***y la carga total satisface que:***

$$U\ \leq\ n(2^{1/n}\ -\ 1) \tag{2.10}$$

entonces el conjunto de tareas es planificable sobre un procesador utilizando el algoritmo RM.

La condición establecida en el teorema 4.b). es similar a la condición dada por Liu y Layland [12] en el teorema 1. El teorema 4 mejora al teorema 1 cuando $\beta\ <\ 1\ -\ 1/n$**.**

La condición de planificabilidad dada en el teorema 4 en su versión simple se encuentra definida en el corolario 1 la cual denotaremos como Condición PO.

Corolario 1 ***(Condición PO) [19]: Dado un conjunto*** $\sum$ ***de tareas de tiempo real*** $\{\tau_1, \ldots, \tau_n\}$ ***y sea*** β ***definido como en el teorema 4. Si la carga total satisface:***

$$U \leq max\{ln\ 2, 1 - \beta\ ln\ 2\} \tag{2.11}$$

entonces el conjunto de tareas puede ser planificado en un solo procesador por el algoritmo Rate Monotonic.

RBOUND proporciona una cota de utilización para el algoritmo Rate Monotonic. Esta condición utiliza la información de los periodos de las tareas para obtener una alta utilización del procesador. RBound se describe a continuación.

Teorema 5 ***(Condición RBound) [16]: Considere un conjunto*** $\sum$ ***de*** m ***tareas ordenadas de manera creciente respecto de sus periodos, donde el mínimo*** (T_1) ***y máximo*** (T_m) ***periodo es fijo y conocido, y la razón de cualquier par de periodos es menor que 2. Si***

$$\sum_{i=1}^{m} C_i/T_i \ \leq \ (m-1)(r^{1/(m-1)} - 1) \ + \ (2/r) \ - \ 1 \tag{2.12}$$

donde $r = T_1/T_m$***, entonces el conjunto de tareas puede ser planificado en un solo procesador por el algoritmo Rate Monotonic.***

Corolario 2 ***(Condición RBound) [16]: Cuando*** $m \to \infty$ ***la utilización del procesador para la condición RBound se aproxima a*** $ln\, r + 2/r - 1$**.**

2.2.4 Condición de Planificabilidad Orientada a la Utilización

Cuando una condición de planificabilidad considera las utilizaciones de las tareas, se dice que es una condición orientada a la utilización (Condición UO). La siguiente es una condición suficiente para planificar un conjunto de tareas usando RM:

Teorema 6 ***(Condición UO) [21]: Sea*** $\sum = \{\tau_1, \tau_2, \ldots, \tau_{n-1}\}$ ***un conjunto de*** $n-1$ ***tareas con utilizaciones*** $\{u_1, u_2, \ldots, u_{n-1}\}$ ***y planificadas factiblemente utilizando Rate Monotonic. Una nueva tarea*** τ_n ***puede ser planificada factiblemente junto***

con las $n-1$ ***tareas en un procesador con el algoritmo Rate Monotonic, si satisface la condición***

$$C_n \,/\, T_n \;\leq\; 2[\prod_{i=1}^{n-1}(1+u_i)]^{\quad -1} - 1 \tag{2.13}$$

Note que la complejidad de esta condición es O(n). La condición UO supera a la condición WC en rendimiento (la condición UO siempre mantiene una mayor utilización del procesador que la condición WC). Esta condición también mejora a la condición necesaria y suficiente (IFF) en complejidad, debido a su comportamiento lineal con respecto al número de tareas. Esta condición es usada para determinar el rendimiento del algoritmo RM-FFDU la cual puede ser consultada en [21].

Capítulo 3

Algoritmos Aproximados para el Problema Bin Packing

Bin packing es conocido como uno de los problemas clásicos, con una gran variedad de aplicaciones prácticas, en donde la minimización de un cierto espacio (área u objetos) o de tiempo se convierte en un problema relevante. Por mencionar algunas de las aplicaciones que pueden ser modeladas a través del problema bin packing tenemos: (1) La planificación de tareas sobre múltiples procesadores; (2) La asignación de espacio de almacenamiento en redes de computadoras; (3) El copiado de una serie de archivos en una cantidad dada de discos de almacenamiento; (4) La asignación de comerciales de televisión dentro de un área de tiempo aire específica.

En este capítulo se presenta el problema Bin Packing y algunas de las heurísticas que lo resuelven. La finalidad de este capítulo radica en plantear el problema de asignación de tareas de tiempo real a multiprocesadores como un problema Bin Packing.

3.1 Descripción del Problema

En el problema bin packing, una lista L=$\{a_1, a_2, \ldots, a_m\}$ de objetos (*items*) deben ser asignados a un mínimo conjunto de depósitos (*bins*) $B_1, B_2, \ldots, B_m$ sujetos a la restricción de que el conjunto de objetos asignado en cada depósito no debe exceder su capacidad. Este problema puede ser descrito utilizando la terminología del problema knapsack de la siguiente manera:

Dados m objetos y m Knapsacks (o depósitos) con:

W_j = peso del objeto j.

c = capacidad de cada depósito.

Minimizar:

$$z = \sum_{i=1}^{m} Y_i$$

tal que:

$$\sum_{j=1}^{m} W_j X_{ij} \leq cY_i, \qquad i \in \mathcal{N} = \{i, \ldots, m\}.$$

$$\sum_{i=1}^{m} X_{ij} = 1, \qquad j \in \mathcal{N}.$$

$$Y_i = \textbf{0 ó 1}. \qquad i \in \mathcal{N},$$

$$X_{ij} = \textbf{0 ó 1}, \qquad i \in \mathcal{N}, j \in \mathcal{N}.$$

donde:

$$Y_i = \begin{cases} 1 & \textbf{si el depósito i es usado} \\ 0 & \textbf{de otra manera} \end{cases}$$

$$X_{ij} = \begin{cases} 1 & \textbf{si el objeto j es asignado al depósito i;} \\ 0 & \textbf{de otra manera} \end{cases}$$

Se supone que los pesos W_j son enteros positivos, y que c es un entero positivo, $W_j \leq c$ para $j \in N$.

En el problema clásico del bin packing, la capacidad c del bin es solamente un factor de escala. De esta manera podemos, sin pérdida de generalidad, asumir la normalización $c = 1$. En un contexto algorítmico, clasificamos a los depósitos o bins no vacíos como abiertos (*open*) y a los depósitos llenos como cerrados (*close*). Un bin que se encuentra abierto, es aquel que tiene la capacidad de admitir otro elemento u objeto. Un bin cerrado es un bin que no se encuentra disponible y permanece en esa condición todo el tiempo, es decir, una vez que un bin es cerrado, éste no puede ser re-abierto para incluir más objetos. Un bin (necesariamente vacío) es abierto con el único propósito de recibir su primer elemento. Este bin será posteriormente cerrado (cuando no puedan admitirse mas elementos), dependiendo del algoritmo y del tamaño del objeto. Asumiremos que todos los algoritmos abren bins en un orden creciente correspondiente al índice del bin $B_1, B_2, \ldots, B_m$. Un algoritmo que resuelve el problema bin packing se dice que es en línea, si éste asigna cada objeto a_i conociendo únicamente los tamaños de los objetos a_j, $1 \leq j \leq i$, y sin conocer de los elementos subsecuentes. Las decisiones de un algoritmo en línea son irrevocables; un elemento ya asignado no podrá en ningún momento ser reasignado a otro bin. Un algoritmo que resuelve el problema bin packing con el total conocimiento de los elementos del conjunto a asignar, es llamado un algoritmo fuera de línea.

3.2 Análisis en los Peores Casos

En el caso del problema bin packing, la métrica estándar para medir el rendimiento es la *razón asintótica en el peor caso* (*asymptotic worst-case performance ratio*). Si A describe un algoritmo arbitrario que resuelve el problema bin packing, y OPT describe el algoritmo que produce el resultado óptimo para el mismo problema: Sean A(L) y OPT(L) los números de bins resultantes de

aplicar el algoritmo A y OPT respectivamente. El rendimiento asintótico en el peor caso (R_A^∞) está definido por:

$$R_A^\infty = \lim_{OPT(L)\to\infty} A(L)/OPT(L)$$

Entre los objetivos principales en el desarrollo de los algoritmos que resuelven el problema bin packing se encuentra el de obtener valores pequeños para R_A^∞, así como el que posean complejidades bajas (un bajo tiempo de cómputo del algoritmo).

3.2.1 Algoritmos en Línea

Un algortimo simple que resuelve el problema bin packing es el Next Fit (NF). NF realiza un recorrido sobre la lista de objetos L asignando uno tras otro en un solo bin activo[1]. En el caso de que el objeto exceda la capacidad del bin activo, NF cierra el bin (nunca es usado otra vez) y asigna el objeto a un bin vacío. De esta forma, el nuevo bin abierto se convierte en el bin activo. Este algoritmo puede ser implementado con un tiempo de ejecución lineal O(n). El rendimiento asintótico en el peor caso para NF es $R_{NF}^\infty = 2$ [8].

Una desventaja del algoritmo NF es que éste nunca usa los espacios restantes de los bins cerrados. Por medio de una simple modificación al algoritmo NF resulta el algoritmo First Fit (FF). FF también realiza un recorrido en la lista de objetos L, pero nunca cierra un bin considerado como activo. Cuando FF asigna un objeto, primero trata de colocarlo dentro de un bin con el menor índice posible (Los bins se encuentran ordenados de manera creciente a un índice $B_1, B_2, \ldots, B_m$; los algoritmos abren bins en un orden creciente al índice del bin), de otra manera, si el objeto excede la capacidad restante en todos los bins y no

[1] Un bin activo es un bin no cerrado al que, en cierto momento, el algoritmo de asignación hace referencia para asignar un nuevo elemento.

puede ser asignado, FF abre un nuevo bin y deposita el objeto. La complejidad computacional del algoritmo FF es O(n log n). La complejidad de FF es mayor a la complejidad de NF, sin embargo, el rendimiento asintótico en el peor caso de FF es mejor que NF, $R_{FF}^{\infty} = 17/10$ [17].

Otros algoritmos en línea clásicos que corresponden a la clase de algoritmos llamados Any Fit (AF) [8] son: Best Fit (BF), Worst Fit (WF), y Almost Worst Fit (AWF). En los algoritmos AF se realiza un recorrido de la lista L de objetos, y nunca se asigna un objeto a un bin vacío, a menos que no pueda acomodarse en algun bin activo. BF posee un comportamiento muy similar a FF, excepto que el bin elegido para asignar el siguiente elemento u objeto será el bin que posea el menor espacio disponible. WF asigna el siguiente objeto dentro de un bin activo el cual tenga la mayor cantidad de espacio disponible. AWF por su parte, trata primero de acomodar el objeto en el bin que posea el segundo mayor espacio disponible, si el objeto no puede acomodarse en ese bin (es decir, el peso del los objetos ya asignados al bin junto con el peso del objeto excede a la capacidad del bin), AWF se comportará igual que WF. Las complejidades de los algoritmos BF, WF y AWF son O(n log n). El rendimiento asintótico en el peor caso para BF y AWF es de 17/10, es decir, $R_{BF}^{\infty} = R_{AWF}^{\infty} = 17/10$. El rendimiento de WF es el mismo que NF, es decir, $R_{WF}^{\infty} = R_{NF}^{\infty} = 2$.

3.2.2 Algoritmos Fuera de Línea

Los algoritmos FF y BF ofrecen el peor rendimiento cuando la lista de objetos L a asignar se encuentra ordenada de manera creciente respecto de sus pesos (W). De manera intuitiva, los objetos con pesos grandes al final de la lista deberían ser combinados con los objetos de pesos pequeños que se encuentran al comienzo de la lista, donde FF y BF producen una densa combinación de objetos con pesos

Algoritmo	Tipo	Complejidad	R_A^∞
BF	En línea	O(n log n)	17/10
FF	En línea	O(n log n)	17/10
WF	En línea	O(n log n)	2
AWF	En línea	O(n log n)	17/10
NF	En línea	O(n)	2
FFD	Fuera de línea	O(n log n)	11/9
BFD	Fuera de línea	O(n log n)	11/9

Tabla 3.1: Algoritmos que resuelven el problema bin packing

pequeños dejando los objetos pesados al final. Por lo tanto, es natural desarrollar algoritmos de tal forma que inviertan el ordenamiento de los objetos en forma decreciente respecto de sus pesos.

Si ordenamos los objetos de manera decreciente respecto de sus pesos, y aplicamos los algoritmos BF y FF, obtenemos los algoritmos Best Fit Decreasing (BFD) y First Fit Decreasing (FFD). Las complejidades de los algoritmos BFD y FFD son O(n log n). El rendimiento de estos algoritmos comparado contra sus similares en línea es dramático. Johnson [8] demostró que $R_{FFD}^\infty = R_{BFD}^\infty = 11/9$. Un resumen de las características de estos algoritmos es mostrado en la tabla 3.1.

Como se describió en capítulos anteriores, la planificación de tareas de tiempo real sobre sistemas multiprocesadores puede llevarse a cabo siguiendo un esquema particionado o un esquema global. Cuando se considera un esquema particionado, el control de admisión asigna un subconjunto de tareas a ejecutarse en cierto procesador. Esta asignación de tareas a procesadores puede ser modelado a través del problema bin packing. Si consideramos a las tareas como

los elementos u objetos (y a sus utilizaciones como W_i) y a los procesadores como bins o depósitos, entonces el problema de buscar un particionamiento de tareas a procesadores podría ser resuelto bajo un algoritmo bin packing.

Para resolver el problema de asignamiento de tareas a procesadores utilizando un algoritmo bin packing se podría usar la capacidad del bin (en este caso la utilización del procesador) igual a 1. Sin pérdida de generalidad se podría normalizar la utilización del procesador como el 100%. Sin embargo, esto no es necesariamente cierto para el algoritmo de planificación Rate Monotonic debido a la naturaleza de su cota de planificabilidad [12] como se explica en el capítulo 2. La cota de planificabilidad para RM garantiza que ninguna tarea perderá su plazo de respuesta si la utilización del procesador es menor que (ln 2) [12]. Sin embargo, esta condición es suficiente pero no necesaria, ya que garantiza lo que ocurre con una utilización menor a (ln 2) pero no asegura si las tareas serán planificables en el rango de utilización del procesador entre (ln 2) y 1. Si se utiliza el valor de (ln 2) como capacidad del bin en el problema de asignación de tareas a procesadores, se obtendría un resultado subóptimo, ya que no se aprovecharía el 100% del procesador para cualquier conjunto de tareas.

En esta tesis, analizaremos diversas heurísticas que resuelven el problema de asignación y planificación de tareas de tiempo real sobre múltiples procesadores, en donde las tareas son planificadas utilizando la política Rate Monotonic. El objetivo de las heurísticas consiste en producir resultados que utilicen el menor número de procesadores y que por lo tanto, los procesadores resultantes ofrezcan un mejor rendimiento.

Capítulo 4

Algoritmos de Planificación Rate Monotonic sobre Sistemas Multiprocesadores

Considerando el modelo de tareas y la descripción del problema de planificación de tareas de tiempo real sobre multiprocesadores dado en el capítulo 1, un conjunto de algoritmos que ofrecen una solución factible, puede ser formado por la combinación de las heurísticas Bin Packing, junto con las condiciones de planificabilidad para el algoritmo RM. Este conjunto de algoritmos se describe de la siguiente manera:

$$C = \{NF,FF,BF,FFD,\ldots\} \times \{WC,IFF,IP,UO,\ldots\},$$

Donde NF, FF, BF, FFD son las heurísticas Bin Packing y WC, IFF, IP, UO son las condiciones de planificabilidad para el algoritmo RM. Por ejemplo, Rate Monotonic Next Fit (RMNF) propuesto por Dhall y Liu [6] puede ser categorizado como RMNF-IP y Rate Monotonic First Fit (RMFF) como RMFF-IP, dado que la condición IP es usada.

En este capítulo se describen los mejores algoritmos de planificación de tareas de tiempo real sobre sistemas multiprocesadores. Se define también, la medida de rendimiento de un algoritmo con respecto del comportamiento en los peores casos.

4.1 Métricas de Rendimiento de los Algoritmos

Dado que existen varias heurísticas para resolver el problema de planificación de tareas sobre multiprocesadores, nos interesa encontrar soluciones las cuales nos proporcionen los mejores resultados. Por ejemplo, aquellas soluciones que nos proporcionen el menor número de procesadores utilizados. Además, si suponemos que ningún algoritmo nos garantiza el encontrar la solución óptima (la mínima planificación factible) para cualquier conjunto de tareas de entrada, entonces es de gran importancia conocer qué tan cerca se encuentra una heurística de la solución óptima.

Para nuestro problema, la medida de rendimiento de un algoritmo es el número de procesadores que éste requiere para planificar un conjunto de tareas dadas. De esto, una medida aceptable es la razón entre el número de procesadores requerido por una heurística y el número de procesadores utilizados por un algoritmo óptimo. En otras palabras, si N_A y N_0 denotan el número de procesadores requeridos por la heurística A y el número de procesadores requerido por un algoritmo óptimo respectivamente, entonces nuestra atención deberá enfocarse en el estudio de aquellos algoritmos que nos proporcionen valores pequeños para N_A/N_0.

Pronto nos daremos cuenta de que la razón N_A/N_0 no es un valor constante para diferentes entradas. Esto es debido a que una heurística en particular tiende a observar un buen comportamiento en ciertas entradas y un rendimiento

pobre ante otro tipo de entradas. Para tener una medida con mayor confiabilidad se podrían obtener diferentes valores de entrada de N_A/N_0. Otra solución es encontrar el valor máximo de N_A/N_0 para cualquier conjunto de valores de entrada dados. La primera solución nos provee un *rendimiento en el caso promedio* de las diversas heurísticas, sin ofrecer ninguna información acerca del *comportamiento en el peor caso.* Muchas veces es importante considerar un análisis en el caso promedio por la simple razón de que cierto algoritmo no siempre da una respuesta influenciada por el peor caso. La segunda solución provee toda la información de los algoritmos en el peor caso. Con el propósito de tener un análisis comparativo de diversas heurísticas de planificación de tareas de tiempo real que sea completo, se aplicarán las dos soluciones.

Para obtener el *rendimiento en el caso promedio* de las heurísticas, uno puede realizarlo a través de experimentos de simulación (En el capítulo 5 se describirán los rendimientos en el caso promedio de las diversas heurísticas a traves de experimentos de simulación). Para obtener el *rendimiento en el peor caso*, lo haremos bajo la descripción del siguiente criterio: ***Si*** N_A ***es el número de procesadores que nuestra heurística*** A ***requiere para planificar un conjunto de tareas y*** N_0 ***es el número mínimo óptimo de procesadores que se requieren para planificar el mismo conjunto de tareas, entonces el rendimiento de la heurística A en el peor caso estará representado por:*** $\Re_A^\infty = lim_{N_0 \to \infty} N_A/N_0$.

Existen varias razones para obtener el rendimiento en el peor caso: Primero, es una buena medida de comparación de diversas heurísticas. Segundo, desde el momento en que estamos tratando con sistemas de tiempo real duros, es para nosotros crucial el garantizar que todas las tareas cumplan con sus plazos de respuesta. Esto es particularmente importante cuando el algoritmo heurístico se rige bajo un esquema en línea.

4.2 Algoritmos Fuera de Línea y sus Rendimientos en los Peores Casos

4.2.1 Rate Monotonic Next Fit (RMNF)

El primer algoritmo heurístico a analizar que resuelve el problema de planificación de un conjunto de tareas de tiempo real sobre multiprocesadores es conocido como Rate Monotonic Next Fit (RMNF) [6]. En este algoritmo, las tareas son ordenadas de forma creciente con respecto de sus periodos antes del proceso de asignación. El pseudocódigo del algoritmo se presenta en la Figura 4.1.

En el algoritmo (Figura 4.1) la tarea τ_i es asignada al procesador P_j (paso 5) siempre y cuando se encuentre una planificación factible de acuerdo a la condición IP (paso 4). De lo contrario, la tarea τ_i es asignada al procesador P_{j+1}. La asignación de una tarea τ_i a un procesador P_j consiste en incrementar la utilización del procesador realizando $U_j = U_j + u_i$ donde U_j y u_i denotan la utilización del procesador j y la utilización de la tarea τ_i respectivamente. Cuando la tarea τ_i es asignada se procede a continuar con la siguiente tarea incrementando el índice i (paso 9). Los pasos 3 al 9 se repiten hasta que todas las tareas se asignen.

Cuando el algoritmo termina, el número total de procesadores requeridos para planificar el conjunto de tareas está dado por la variable j. La complejidad computacional del algoritmo es O(n log n), donde n es el número de tareas del conjunto a asignar. El límite o cota superior de desempeño del algoritmo RMNF es establecido en el Teorema 7, y el límite inferior es descrito en el Teorema 8.

Teorema 7 ***(RMNF) [6]: Para cualquier conjunto de tareas,*** $\lim_{N_0 \to \infty} N/N_0 \leq 2.67$***, donde*** N_0 ***es el número mínimo de procesadores requerido para una planificación factible del mismo conjunto de tareas, y*** N ***es el número de procesadores obtenido mediante el Algoritmo RMNF.***

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_m\}$ y un conjunto de procesadores $\{P_1, \ldots, P_m\}$.

Salida: j; número de procesadores requeridos.

```
1.    Ordenar las tareas de manera creciente respecto de sus periodos
2.    i := 1; j := 1        /* i representa la i-ésima tarea, j representa el j-ésimor */
                            /* procesador. */
3.    while (i ≤ m) do      /* Mientras no se asignen las m tareas */
4.        if ( u_i ≤ Condición IP) then
5.            U_j := U_j + u_i      /* Se asigna al procesador P_j la tarea τ_i */
6.        else
7.            U_{j+1} := U_{j+1} + u_i      /* Se asigna la tarea τ_i al procesador P_{j+1} */
8.            j := j + 1      /* Se incrementa el índice de procesadores */
9.        i := i + 1      /* Se incrementa el índice de tareas */
10.   end
```

Figura 4.1: Algoritmo Rate Monotonic Next Fit (RMNF)

Teorema 8 ***(RMNF) [22]: Sea*** N ***el número de procesadores requerido para planificar factiblemente un conjunto de tareas mediante el algoritmo RMNF, y*** N_0 ***el mínimo número de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas. Entonces,*** $\lim_{N_0 \to \infty} N/N_0 \geq 2.67$***. Junto con el Teorema 7 se concluye que*** $\Re\ (RMNF)^{N_0 \to \infty} = 2.67$***.***

Note que la condición de planificabilidad usada para el algoritmo RMNF es la condición IP, por lo que podríamos nombrar a este algoritmo como RMNF-IP. La clasificación de este algoritmo como fuera de línea se debe principalmente a

$\tau_i =$	1	2	3	4	5	6	7	8	9	10
T_i	7	20	36	45	60	65	150	230	280	400
C_i	2	3	11	14	19	16	31	70	27	113
u_i	0.2857	0.15	0.3055	0.3111	0.3167	0.2461	0.2067	0.3043	0.0964	0.2825

Tabla 4.1: Ejemplo de 10 Tareas de Tiempo Real

que la condición IP (Increasing Period) requiere que las tareas del conjunto sean ordenadas con respecto de sus periodos, antes del proceso de asignación; por lo cual se necesita del conocimiento a priori de los parámetros temporales de todo el conjunto de tareas.

Para lograr una mejor comprensión del algoritmo RMNF, se ilustrará el esquema de asignación con un ejemplo. En la tabla 4.1 se presentan los parámetros de un conjunto de 10 tareas de tiempo real. El conjunto de tareas es ordenado de manera creciente con respecto de sus periodos.

En el algoritmo 4.1 las tareas τ_1, τ_2, τ_3 son asignadas al procesador P_1. La tarea τ_4 ya no puede ser asignada al procesador P_1 ya que ésta excede la cota de planificabilidad dada por la condición IP. De esta manera, el procesador P_1 es marcado como lleno (ya no es utilizado posteriormente) y a un *nuevo procesador vacío* P_2 se le asigna la tarea τ_4. La tarea τ_5 es asignada al procesador P_2, pero no la tarea τ_6, ya que con esta tarea excede nuevamente la cota de planificabilidad de la condición IP para el procesador P_2. Un procesador ocioso es nuevamente usado para asignar la tarea τ_6. El procedimiento prosigue de manera similar hasta que todas las tareas del conjunto son asignadas. La asignación final del algoritmo RMNF puede observarse en la figura 4.2. La asignación óptima de este conjunto de tareas es presentado en la figura 4.3.

La asignación óptima del conjunto de tareas mostrado en la figura 4.3 se

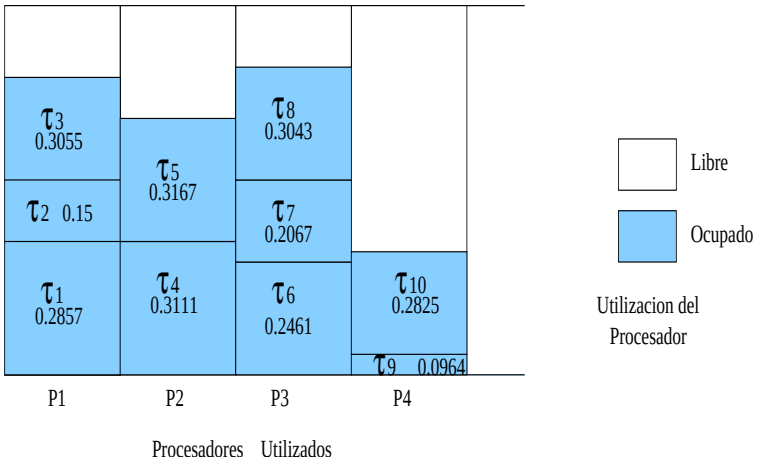

Figura 4.2: Resultado de la Asignación del Conjunto de Tareas de la Tabla 4.1 con RMNF

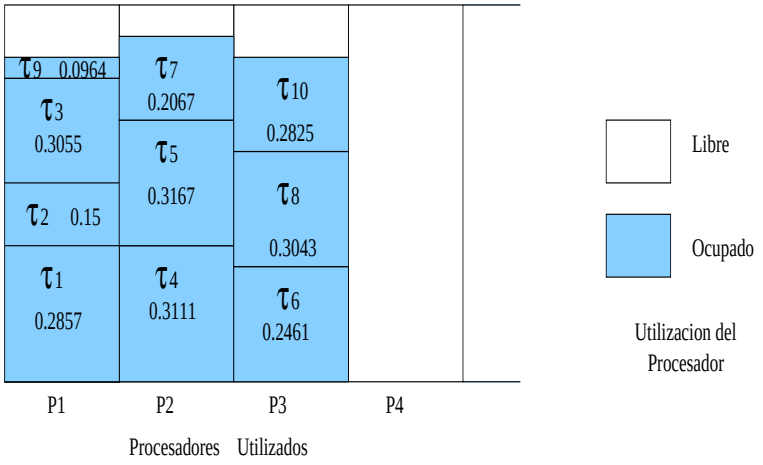

Figura 4.3: Resultado de la Asignación Óptima del Conjunto de Tareas de la Tabla 4.1

obtiene utilizando la condición de planificabilidad necesaria y suficente [9].

4.2.2 Rate Monotonic First Fit (RMFF)

En el proceso de asignación de tareas a procesadores, el algoritmo RMNF sólo verifica el procesador activo para ver si la tarea a asignar, junto con todas las tareas previamente asignadas, pueden encontrar una planificación factible. En caso de no encontrar una planificación factible, RMNF asigna la tarea a un

procesador vacío (ocioso), sin considerar que posiblemente la tarea pueda ser planificada factiblemente en algún otro procesador utilizado anteriormente. Para evitar este desperdicio de utilización en los procesadores, el algoritmo Rate Monotonic First Fit (RMFF) siempre comienza verificando la planificabilidad de la tarea en cada uno de los procesadores utilizados.

En el algoritmo Rate Monotonic First Fit (RMFF) [6], las tareas son primero ordenadas de manera creciente con respecto de sus periodos. Para planificar una tarea τ_i, se requiere encontrar el menor índice j tal que la tarea τ_i, junto con todas las tareas previamente asignadas al procesador P_j puedan encontrar una planificación factible utilizando la condición IP, para así asignar la tarea τ_i a P_j. El algoritmo se describe en la figura 4.4.

En el algoritmo (Figura 4.4) se establece que si τ_i satisface la condición IP dada en el paso 4, entonces τ_i es planificable. Esto es, se realiza la asignación de la tarea τ_i al procesador P_m (Paso 6). k_m y U_m representan el número de tareas previamente asignadas al procesador P_m, y la utilización total de las k_m tareas respectivamente. u_i representa la utilización de la tarea τ_i. Si $i > n$, es decir, si todas las tareas ya fueron asignadas (paso 3), entonces el algoritmo termina. De lo contrario, se incrementa el índice de tareas (paso 9) y se continúa con los pasos 4 al 9. El número total de procesadores usados al finalizar la asignación esta determinado por la variable j.

Se espera que el algoritmo RMFF mejore en rendimiento al algoritmo RMNF, debido a que en el algoritmo RMFF un procesador vacío no será utilizado hasta que todos los procesadores se hayan revisado para ver si una nueva tarea encuentra una asignación factible. Esta suposición es demostrada por Dhall y Liu [6], en donde presentaron los siguientes resultados obtenidos:

Lema 1 ***[6]: Si*** m ***tareas no pueden encontrar una planificación factible en*** $m-1$

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. Ordenar las tareas de manera creciente respecto de sus periodos.
2. $i := 1$; $j := 1$ /* i representa la i-ésima tarea, j representa el j-ésimo procesador */
3. **while** $(i < n)$ **do** /* Mientras no se asignen las n tareas */
4. $m := 1$; **while** $(u_i > 2(1 + U_m/k_m)^{-k_m} - 1)$ **do**
5. $m := m + 1$ /* m representa el procesador índice */
6. $U_m := U_m + u_i$; $k_m := k_m + 1$ /* Se asigna la tarea i al procesador P_m */
7. **if** $(m > j)$ **then**
8. $j := m$ /* Se actualizan el No. de procesadores usados */
9. $i := i + 1$ /* Se incrementa el índice de tareas */
10. **end**

Figura 4.4: Algoritmo Rate Monotonic First Fit (RMFF)

procesadores, de acuerdo al algoritmo RMFF, entonces el factor de utilización del conjunto de tareas es mayor que $m/(1 + 2^{1/3})$.

Lema 2 ***[6]: Si las tareas son asignadas a los procesadores, utilizando el algoritmo RMFF, de entre todos los procesadores en los cuales dos tareas fueron asignadas, existe al menos un procesador cuyo factor de utilización del conjunto de las dos tareas es menor que 1/2.***

Teorema 9 ***(RMFF) [6]: Sea*** N ***el número de procesadores requerido para planificar de manera factible un conjunto de tareas mediante el algoritmo RMFF, y*** N_0 ***el número mínimo de procesadores requeridos para encontrar una planifi-***

cación factible del mismo conjunto de tareas. Entonces, cuando N_0 ***se aproxima a infinito,*** $2 \leq N/N_0 \leq 4x2^{1/3}/(1+2^{1/3})(\approx 2.23)$

Desafortunadamente en el trabajo de Dhall y Liu [6] el Lema 1 es incorrecto y el Lema 2 da un resultado débil para el algoritmo RMFF. Estos dos errores implican un resultado incorrecto en el cálculo de la cota superior. Estas afirmaciones son demostradas en [22] en donde se presentan los siguientes resultados para el algoritmo RMFF.

Lema 3 ***[22]: Si*** m ***tareas no pueden encontrar una planificación factible en*** $m-1$ ***procesadores utilizando el algoritmo RMFF, entonces el factor de utilización del conjunto de tareas es mayor que*** $m/(1+2^{1/2}) = m(2^{1/2}-1)$.

Lema 4 ***[22]: Si las tareas son asignadas a los procesadores utilizando el algoritmo RMFF, se puede concluir que, de entre todos los procesadores en los cuales dos tareas fueron asignadas, existe al menos un procesador cuyo factor de utilización del conjunto de las dos tareas en menor o igual que*** $2(2^{1/3}-1) \approx 0.52$.

Teorema 10 ***(RMFF) [22]: Sea*** N ***el número de procesadores requerido para planificar de manera factible un conjunto de tareas mediante el algoritmo RMFF, y*** N_0 ***el número mínimo de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas. Entonces, cuando*** N_0 ***se aproxima a infinito,*** $2 \leq N/N_0 \leq 2 + (3 - 2^{3/2}) \,/\, (2(2^{1/3} - 1)) \approx 2.33$

Siguiendo el algoritmo RMFF, con el conjunto de tareas dado en la tabla 4.1, se obtiene como resultado que las tareas τ_1, τ_2 **y** τ_3 **son asignadas al procesador** P_1**. La tarea** τ_4 **no encuentra una planificación factible en el procesador** P_1 **por lo que se asigna a un nuevo procesador ocioso** P_2**. La tarea** τ_5 **es planificable (y asignada) en el procesador** P_2**. El procedimiento prosigue de manera similar con**

el resto de las tareas. La asignación final del algoritmo RMFF puede observarse en la figura 4.5. La complejidad algorítmica de este algoritmo es O(n log n) en donde n es el número de tareas.

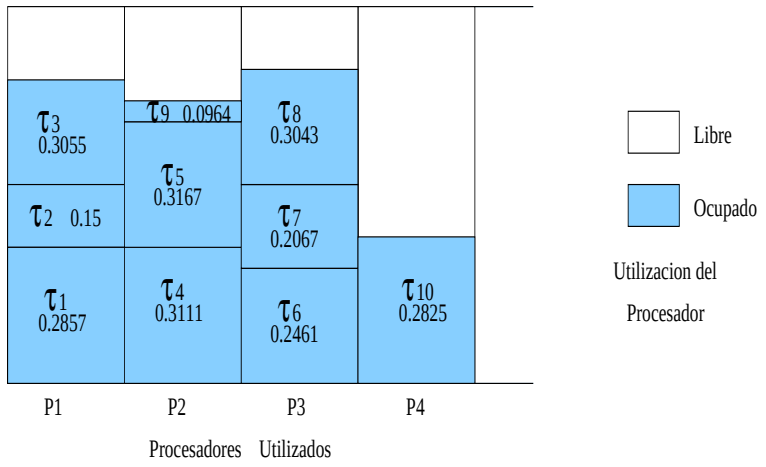

Figura 4.5: Resultado de la Asignación del Conjunto de Tareas de la Tabla 4.1 con RMFF

4.2.3 Rate Monotonic Best Fit (RMBF)

En el proceso de asignación de tareas a procesadores, el algoritmo RMFF siempre asigna la tarea al procesador con el menor índice en donde la tarea se planifique de manera factible. Sin embargo, esta estrategia podría no ser óptima para algunos casos. Por ejemplo, supóngase que el procesador con el menor índice en el cual la tarea puede ser planificada de manera factible es también el procesador que posee la mayor utilización disponible de entre todos los procesadores no vacíos. Supongamos también que la tarea a asignar puede ser planificable en otro procesador diferente no vacío al del menor índice. Si la tarea se asigna al procesador con el menor índice, entonces la utilización disponible del procesador se reducirá dependiendo de la carga de la tarea, causando que una tarea futura con una utilización lo suficientemente grande, que sólo era planificable en el

procesador que tenía la mayor utilización disponible, tenga que ocupar otro procesador vacío.

RMBF evita esta situación asignando la tarea al procesador que posea la menor utilización en donde ésta sea planificable. RMBF [22] se basa en la heurística Best Fit del problema Bin Packing. En este algoritmo, las tareas primero son ordenadas de forma creciente con respecto de sus periodos. El esquema de asignación de RMBF se presenta en la figura 4.6.

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. Ordenar ls tareas de manera creciente respecto de sus periodos.
2. $i := 1; j := 1$ /* i representa la i-ésima tarea, j representa el j-ésimo procesador */
3. **while** $(i < n)$ **do** /* Mientras existan tareas sin asignar */
4. **if** ($u_i \leq$ Condición IP) **and** $(2(1 + U_j \,/\, k_j)^{-k_j} - 1)$ sea lo menor posible
5. $U_j := U_j + u_i$ /* Se asigna la tarea al procesador P_j*/
6. $i := i + 1$ /* Se incrementa el índice de tareas */
7. **end**

Figura 4.6: Algoritmo Rate Monotonic Best Fit (RMBF)

Para planificar τ_i (paso 4) se requiere encontrar el menor índice j tal que la tarea τ_i, junto con todas las tareas que han sido asignadas al procesador P_j, puedan encontrar una planificación factible utilizando la Condición IP, y el valor de $2(1 + U_j \,/\, k_j)^{-k_j} - 1$ sea tan pequeño como sea posible. La tarea τ_i se asigna al procesador P_j en el paso 5, en donde k_j y U_j representan el número de tareas ya

asignadas al procesador P_j y la utilización total de las k_j tareas respectivamente, y u_i es la utilización de la tarea τ_i. El número total de procesadores utilizados en este algoritmo, al finalizar la asignación, está determinado por la variable j.

Nótese que con $2(1 + U_j / k_j)^{-k_j} - 1$ se obtiene el valor de utilización disponible en cada procesador. El procesador que posea la menor utilización disponible, y en el cual la tarea sea planificable de manera factible, es el procesador en el cual se asignará la tarea. De hecho, si revisamos el algoritmo, el problema de encontrar el procesador en el cual la tarea sea planificable y éste posea la menor utilización disponible, es similar a encontrar el procesador que posea la mayor utilización disponible. Aunque tenemos los dos esquemas para asignar una tarea, el rendimiento (teórico) en el peor caso del algoritmo RMBF, asignando las tareas al procesador con la mayor utilización, es un problema no resuelto hasta el momento. Por lo que para este análisis, se considera la asignación de la tarea al procesador con la menor utilización disponible.

Sorpresivamente, a pesar de las modificaciones del algoritmo RMBF en cuanto al método de asignación de tareas a procesadores, el rendimiento en el peor caso de RMBF no mejora a RMFF como se observa en el teorema 11. Además, el costo de ejecución del algoritmo RMBF es mayor al costo de ejecución de RMFF.

Teorema 11 *[22]: **Sea** N **el número de procesadores requerido para planificar de manera factible un conjunto de tareas usando el algoritmo RMBF, y** N_0 **el número mínimo de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas. Entonces,*** $\lim_{N_0 \to \infty} N/N_0 \leq 2 + (3 - 2^{3/2}) / a \approx 2.33$***, donde*** $a = (2\,(2^{1/3} - 1))$.

Al realizar la asignación del conjunto de tareas descrito en la tabla 4.1 utilizando el algoritmo RMBF (figura 4.7), encontramos que el resultado es similar

que el observado por el algoritmo RMFF dado en la figura 4.5.

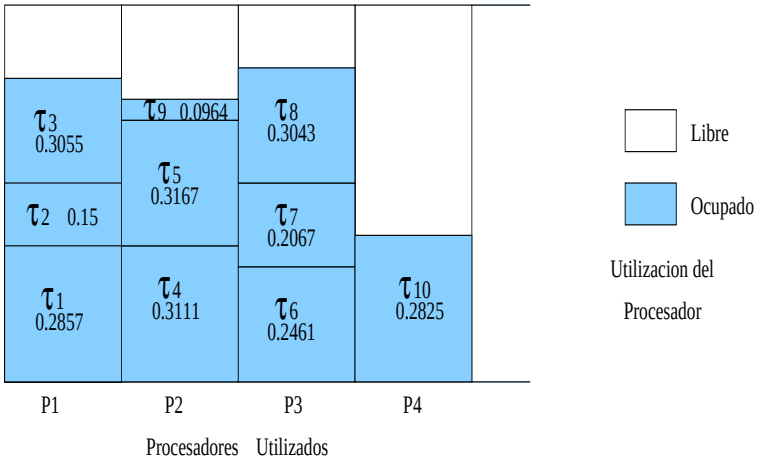

Figura 4.7: Resultado de la Asignación del Conjunto de Tareas de la Tabla 4.1 con RMBF

4.2.4 Rate Monotonic First Fit Decreasing Utilization (RM-FFDU)

En la solución al problema Bin Packing, las heurísticas Next Fit, First Fit y Best Fit, ofrecen su peor rendimiento cuando los elementos a asignar(tareas) se encuentran ordenados de manera creciente con respecto de sus pesos (utilizaciones). Por esta razón, algunas heurísticas realizan un pre-ordenamiento de las tareas antes del proceso de asignación.

La heurística First Fit es una de las más estudiadas y es considerada como una heurística en línea. Si el conjunto de tareas se ordena de manera decreciente y se aplica la heurística First Fit, se obtiene la técnica First Fit Decreasing (FFD), la cual se ejecuta fuera de línea.[1]

El algoritmo RM-FFDU [21] mostrado en la figura 4.8 ordena el conjunto de tareas de forma decreciente con respecto de sus utilizaciones y utiliza la

[1]La razón principal para considerar a FFD como una técnica fuera de línea se basa en el hecho de que se requiere conocer todo el conjunto de tareas a priori para realizar el ordenamiento.

estrategía First Fit para la asignación de tareas a procesadores. La condición de planificabilidad utilizada para RM-FFDU es la condición UO.

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. Ordenar las tareas de manera decreciente respecto de sus utilizaciones.
2. $i := 1$; $j := 1$ /* i representa la i-ésima tarea, j representa el j-ésimo procesador*/
3. **while** $(i < n)$ **do** /* Mientras no se asignen las n tareas */
4. $m := 1$; **while** $(u_i > 2/ \prod_{h=1}^{k_m} (u_{mh} + 1))$
5. $m := m + 1$
6. $U_m := U_m + u_i$; $k_m := k_m + 1$ /* Se asigna la tarea i al procesador P_m */
7. **if** $(m > j)$ **then**
8. $j := m$ /* Se actualiza el no. de procesadores usados */
9. $i := i + 1$ /* Se incrementa el índice de tareas */
10. **end**

Figura 4.8: Algoritmo Rate Monotonic First Fit Decreasing Utilization (RM-FFDU)

Cuando el algoritmo termina su ejecución, el valor de j es el número de procesadores requeridos por RM-FFDU para planificar el conjunto de tareas. Note que la diferencia entre el algoritmo RM-FFDU y la heurística Bin Packing FFD consiste en el uso de una condición diferente sobre un procesador (o bin). La complejidad del algoritmo RM-FFDU es O(n log n), la cual está influenciada en su mayor parte por el ordenamiento del conjunto de tareas.

Aunque el algoritmo RM-FFDU implementa la asignación de las tareas usan-

$\tau_i =$	5	4	3	8	1	10	6	7	2	9
T_i	60	45	36	230	7	400	65	150	30	280
C_i	19	14	11	70	2	113	16	31	3	27
u_i	0.3167	0.3111	0.3055	0.3043	0.2857	0.2825	0.2461	0.2067	0.15	0.0964

Tabla 4.2: Ejemplo de 10 Tareas de Tiempo Real

do la heurística FFD, ninguno de los resultados presentados por FFD puede ser aplicado directamente al análisis del rendimiento de RM-FFDU. Esto se debe a que en la heurística FFD el tamaño del bin permanece fijo (la capacidad del bin es igual a 1), mientras que en RM-FFDU la utilización de un procesador posee un valor variable. El rendimiento de RM-FFDU se establece en el Teorema 12.

Teorema 12 ***(RM-FFDU)[21]: Sea*** N ***el número de procesadores requerido para planificar factiblemente un conjunto de tareas mediante el algoritmo RM-FFDU, y*** N_0 ***el número mínimo de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas.***
Entonces, $\Re(RM - FFDU)^{N_0 \to \infty} \leq 5 / 3$.

Considerando los parámetros del conjunto de tareas dado en la tabla 4.1, a continuación ejemplificaremos el esquema de asignación del algoritmo RM-FFDU. Para efectos de mayor claridad en el proceso de asignación de tareas, en la tabla 4.2 se muestra el conjunto de tareas ordenado de manera decreciente con respecto de las utilizaciones.

De acuerdo al algoritmo RM-FFDU, las tareas τ_5, τ_4 son asignadas al procesador P_1 sin que se viole la condición de planificabilidad. Al asignar la tarea τ_3, se observa que ésta no es planificable dentro del procesador P_1, por lo que se asigna a un nuevo procesador ocioso. Nótese que en este algoritmo, así como en el RMFF, antes de asignar una tarea a un procesador vacío, se verifica la

planificabilidad de la misma en todos los procesadores no ociosos. El proceso de asignación de las tareas continúa de manera similar para todas las tareas. El resultado final del algoritmo RM-FFDU, es presentado en la figura 4.9.

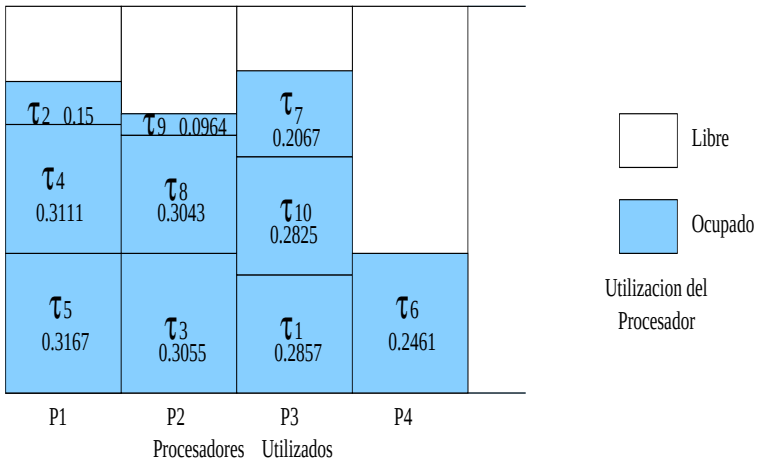

Figura 4.9: Resultado de la Asignación del Conjunto de Tareas de la Tabla 4.2 con RM-FFDU

4.2.5 Rate Monotonic Small Tasks (RMST)

Una de la propiedades que presenta la política de planificación Rate Monotonic, es que se puede incrementar la utilización en los procesadores si los periodos de las tareas poseen valores cercanos unos de otros. Rate Monotonic Small Tasks (RMST) [19] es un algoritmo que considera esta propiedad y es utilizado en conjuntos de tareas con un factor de carga[2] pequeño. La idea principal del algoritmo RMST es particionar el conjunto de tareas de tal forma que sobre cada procesador, la variable β (definido en la ecuación 2.8, sección 2.2.3) tenga un valor pequeño. Recuérdese que la carga mínima garantizada por la condición PO sobre un procesador, está dado por $1 - \beta\ ln\ 2$.

[2]El factor de carga máximo de cualquier tarea en el conjunto se define como: $\alpha = max(\ u_i); \quad (i = 1, \ldots, n)$

RMST asigna las tareas a los procesadores de la misma manera que la heurística Next Fit lo hace para el problema bin packing. RMST primero ordena el conjunto de tareas con base en sus valores S_i (definidos en la ecuación 2.7, sección 2.2.3). Es conveniente visualizar los valores de S_i para un conjunto de tareas dado, como puntos dentro de un círculo de circunferencia 1 (figura 4.11). Comenzando en cualquier punto del círculo y recorriendolo en el sentido de las manecillas del reloj se asignan las tareas a los procesadores. De esta manera, los valores de β en cada procesador son obtenidos por la longitud del arco trazado por las tareas asignadas a ese procesador. El pseudocódigo del algoritmo RMST se describe en la Figura 4.10.

La complejidad computacional del algoritmo RMST es O(n log n) la cual se encuentra determinada principalmente por el ordenamiento de las tareas (paso 1). Cuando el algoritmo termina, el número mínimo de procesadores para encontrar una planificación factible del conjunto de tareas está determinado por la variable j. El rendimiento de RMST se encuentra establecido en el Teorema 13 [19].

Teorema 13 ***(RMST) [19]: Sea*** N ***el número de procesadores requerido para planificar factiblemente un conjunto de tareas mediante el algoritmo RMST,*** ***y*** N_0 ***el número mínimo de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas. Entonces,*** $\Re(RMST)^{N_0 \to \infty} = 2$ ***y*** $\Re(RMST)^{N_0 \to \infty} \leq 1/1 - \alpha$, ***donde*** $\alpha = max_{i=1,\ldots,n} (C_i \,/\, T_i) < 1/2$.

Ejemplificando el algoritmo RMST, en la tabla 4.3, se encuentran los parámetros del conjunto de tareas descrito anteriormente (tabla 4.1) ordenados con respecto de los valores S_i de las tareas. Para este conjunto de tareas se observa que $\alpha = 0.3167$. Al aplicar el asignamiento de este conjunto usando el algoritmo RMST nos resulta que las tareas τ_6, τ_9, τ_3 y τ_7 son asignadas al procesador P_1,

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_m\}$ y un conjunto de procesadores $\{P_1, \ldots, P_m\}$.

Salida: j; número de procesadores requeridos.

1. Ordenar el conjunto de tareas tal que $0 \leq S_i \leq ... \leq S_m < 1$
2. $S_{m+1} := S_1 + 1; i := 1; j := 1$ /*i representa la i-ésima tarea, j representa el j-ésimo*/ /* procesador */
3. $S := S_i; \beta_j := 0$
4. **while** $(i \leq m)$ **do** /* Mientras no se asignen las m tareas */
5. **if** $(U_j + u_i \leq max\ \{ln\ 2,\ 1\ -\ \beta_j\ ln\ 2\})$ **then**
6. $U_j := U_j + u_i$ /* Se asigna la tarea τ_i al procesador P_j */
7. $\beta_j := S_i - S$ /* Se actualiza el valor de β_j */
8. **else**
9. $j := j + 1; S := S_i; \beta_j := 0$ /*Se incrementa el índice de procesadores */
10. $U_j := U_j + u_i$ /* Se asigna la tarea al procesador P_{j+1} */
10. $i := i + 1$ /* Se incrementa el índice de tareas */
11. $\beta_j := S_{m+1} - S$
12. **end**

Figura 4.10: Algoritmo Rate Monotonic Small Tasks (RMST)

$\tau_2, \tau_4, \tau_{10}$ son asignadas al segundo procesador P_2, y τ_1, τ_8, τ_5 se asignan al procesador P_3. Esta asignación puede observarse en la figura 4.11 en donde las áreas obscuras nos indican las asignaciones de las tareas usando RMST. Las áreas son marcadas con los valores de β para cada procesador. Nótese que cualquier asignación de este conjunto de tareas realizado por otro algoritmo da como resultado una planificación sobre cuatro procesadores.

$\tau_i =$	6	9	3	7	2	4	10	1	8	5
T_i	65	280	36	150	20	45	400	7	230	60
C_i	16	27	11	31	3	14	113	2	70	19
u_i	0.2461	0.0964	0.3055	0.2067	0.15	0.3111	0.2825	0.2857	0.3043	0.3167
S_i	0.0223	0.1292	0.1699	0.2288	0.3219	0.4918	0.6438	0.8073	0.8454	0.9068

Tabla 4.3: Ejemplo de 10 Tareas de Tiempo Real con sus Valores S_i

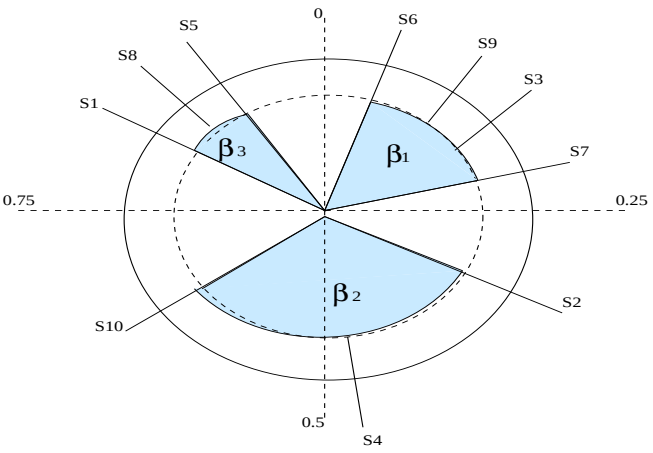

Figura 4.11: Resultado de la Asignación del Conjunto de Tareas de la Tabla 4.3 con RMST

4.2.6 Rate Monotonic General Tasks (RMGT)

Una de las principales desventajas del algoritmo RMST es que cuando α crece, el rendimiento del algoritmo decrece. Es un hecho que la mayoría de los algoritmos bajan su rendimiento cuando α aumenta, lo que ocurre es que RMST lo hace de una manera muy rápida. RMST tiende a lograr un buen rendimiento cuando el factor de carga de las tareas $\alpha < 1/2$.

Con la finalidad de obtener un mejor rendimiento para el algoritmo RMST, se propuso un nuevo esquema de asignación conocido como Rate Monotonic General Tasks (RMGT). El algoritmo RMGT [19], es un esquema de asignación que se aplica para conjuntos de tareas que no poseen la restricción de tener

factores de carga pequeños ($\alpha \leq 1$). El algoritmo RMGT particiona el conjunto de tareas en dos grupos, tal que las utilizaciones del primer y segundo grupo satisfacen que $u_i \leq 1/3$ y $u_i > 1/3$ respectivamente. La asignación de las tareas del primer grupo se realiza utilizando el algoritmo RMST [19]. Las tareas del segundo grupo se asignan utilizando la estrategia First Fit, en donde a lo más se asignarán dos[3] tareas por procesador bajo la condición de planificabilidad suficiente y necesaria (IFF) [9]. El pseudocódigo del algoritmo RMGT se describe en la figura 4.12.

Lema 5 ***[19, 9]: Dado un conjunto de tareas de tiempo real*** $\tau_1, \tau_2, \ldots, \tau_k$. ***Asumiendo que las tareas se encuentran ordenadas de manera creciente con respecto de sus periodos*** $T_1 \leq T_2 \leq \ldots \leq T_k$. ***Entonces, una tarea*** τ_k ***siempre cumplirá su plazo de respuesta*** T_k ***utilizando la política de planificación Rate Monotonic, si y sólo si existe un tiempo*** $0 \leq t \leq T_k$ ***tal que***

$$t \geq \sum_{i=1}^{k} \lceil t/T_i \rceil C_i \tag{4.1}$$

Si el conjunto consiste solamente de dos tareas entonces la ecuación 4.1 se reduce a:

$$\lfloor T_2/T_1 \rfloor (T_1 - C_1) \geq C_2 \quad o \quad T_2 \geq \lceil T_2/T1 \rceil C_1 + C_2 \tag{4.2}$$

Note que el particionamiento del conjunto de tareas en el algoritmo RMGT (figura 4.12) posee una complejidad computacional de O(n), donde n es el número de tareas en el conjunto. La asignación de las tareas correspondientes al primer grupo tiene una complejidad de O(n_1 log n_1) donde n_1 es el número de tareas correspondientes al grupo G_1. Si la heurística First Fit puede ser implementada

[3]El número de ecuaciones a evaluar utilizando la condición IFF crece considerablemente dependiendo del número de tareas a asignar. El asignar dos tareas bajo esta condición se reduce a la evaluación de dos ecuaciones, manteniendo la complejidad computacional del algoritmo baja.

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. Particionar el conjunto de tareas en dos grupos:

   $G_1 = \{ \tau_i \mid u_i \leq 1/3\}$ $\quad$ $G_2 = \{ \tau_i \mid u_i > 1/3\}$

2. Asignan las tareas del primer grupo usando RMST.
3. Las tareas del segundo grupo se asignan de la siguiente manera:
4. $i := 1$; $U_1 := u_i$; /* Se asigna τ_i al procesador P_1 */
5. $i + i + 1$ /* Se incrementa el índice de las tareas */
6. Usar la heurística First Fit para encontrar un procesador m que contenga una sola tarea, digamos τ_W que cumpla:

   $\lfloor T_w/T_i \rfloor (T_i - C_i) \geq C_w$ ó $T_w \geq \lfloor T_w/T_i \rfloor C_i + C_w$ $\quad$ sí $T_i < T_w$

   $\lfloor T_i/T_w \rfloor (T_w - C_w) \geq C_i$ ó $T_i \geq \lfloor T_i/T_w \rfloor C_w + C_i$ $\quad$ sí $T_i \geq T_w$
7. **if** el procesador existe **then**
8. $U_m = U_m + u_i$ /* Se asigna la tarea τ_i al procesador P_m */
9. **else**
10. Se asigna τ_i a un nuevo procesador ocioso.
11. **goto 5**
12. Terminar cuando todas las tareas del grupo G_2 hayan sido asignadas.

Figura 4.12: Algoritmo Rate Monotonic General Tasks (RMGT)

con una complejidad de O(n_2 log n_2) donde n_2 es el número de tareas correspondientes al segundo grupo G_2, se concluye que la complejidad computacional del algoritmo RMGT es O(n log n). El rendimiento del algoritmo RMGT se describe en el Teorema 14.

Teorema 14 ***(RMGT) [19]: Sea*** N ***el número de procesadores requerido para planificar factiblemente un conjunto de tareas mediante el algoritmo RMGT,*** ***y*** N_0 ***el número mínimo de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas. Entonces,*** $\Re(RMGT)^{N_0 \to \infty} = 7/4$.

La asignación del conjunto de tareas que se describió en la tabla 4.3 utilizando el algoritmo RMGT da como resultado una planificación similar a la conseguida por el algoritmo RMST (figura 4.11), debido principalmente a que el factor máximo de carga del conjunto es $\alpha \leq 1/3$.

4.2.7 RBound-MP

Es claro que si en el desarrollo de una condición de planificabilidad se utiliza más información acerca del conjunto de tareas, se obtendría un mejor control de admisión. Por ejemplo, la condición de planificabilidad dada por Lehoczky [9] utiliza los parámetros C_i y T_i del conjunto de tareas. Sin embargo, la complejidad de esta condición de planificabilidad puede llevar un tiempo mayor que el exponencial [11].

RBound [16] fue propuesto como una extensión a la cota de planificabilidad del algoritmo Rate Monotonic. RBound obtiene cotas de planificabilidad altas cuando es usado para servidores aperiódicos [24] o tolerancia a fallos [15]. Con el propósito de mantener la complejidad de la cota RBound baja, el algoritmo utiliza la información de todos los periodos en el conjunto, mediante un valor llamado razón de periodo (r). Dado un conjunto de tareas $\sum$, RBound primero transforma el conjunto $\sum$ en otro conjunto $\sum'$ aplicando el algoritmo ScaleTaskSet (figura 4.13). ScaleTaskSet encuentra un conjunto de tareas equivalente donde la razón entre el máximo y mínimo periodo es menor que 2. De esta forma se aplica un control de admisión sobre el conjunto $\sum'$. En el lema 6 se muestra que

si el conjunto $\sum'$ es planificable bajo RM, entonces $\sum$ también es planificable.

Entrada: Un conjunto $\sum$ de m tareas.

Salida: Un conjunto $\sum'$ de m tareas.

1. **begin**
2. Se ordena el conjunto de tareas $\sum$ de manera creciente respecto de los periodos.
3. **for** $(i = 1$ **to** $m - 1)$ **do**
4. $T_i' = T_i \, 2^{log \lfloor \frac{T_m}{T_i} \rfloor}$
5. $C_i' = C_i \, 2^{log \lfloor \frac{T_m}{T_i} \rfloor}$
6. Se ordena el conjunto de tareas $\sum'$ de manera creciente respecto de los periodos.
7. **return** $(\sum')$
6. **end**

Figura 4.13: Algoritmo Scale Task Set

Lema 6 ***[16]: Dado un conjunto de tareas $\sum$, sea $\sum'$ el conjunto de tareas resultante de la aplicación del algoritmo ScaleTaskSet. Si $\sum'$ es planificable sobre un procesador usando Rate Monotonic, entonces $\sum$ es también planificable en un procesador usando Rate Monotonic.***

Note que RBound es solamente una condición de planificabilidad necesaria. Por lo cual, si el conjunto $\sum'$ no es planificable con la condicion RBound, no podemos inferir nada acerca de la planificabilidad de $\sum$.

RBound-MP mostrado en la figura 4.14 es un algoritmo de planificacion de tareas de tiempo real sobre multiprocesadores, el cual utiliza el control de admisión RBound de la siguiente manera. Primero el algoritmo ScaleTaskSet es aplicado para transformar el conjunto original de tareas. Segundo, una vez transformado el conjunto de tareas, éstas son asignadas utilizando la heurística First

Fit. La heurística bin packing utiliza el control de admisión dado por RBound (teorema 5, sección 2.2.3) para determinar si un procesador puede o no aceptar una nueva tarea. Finalmente, cada tarea original es asignada al procesador en donde fue asignada su tarea transformada.

Entrada: Un conjunto $\sum$ de tareas.

Salida: j; número de procesadores requerido para planificar $\sum$.

1. ScaleTaskSet($\sum$) /* Se hace una llamada a ScaleTaskSet con el conjunto original*/
2. Se asigna el conjunto de tareas que retorna ScaleTaskSet ($\sum'$) usando First Fit. Se utiliza RBound (Teorema 5, sección 2.2.3) como condición de planificabilidad.
3. Cada tarea del conjunto original $\sum$ se asigna al procesador en la cual su tarea transformada del conjunto $\sum'$ fue asignada.

Figura 4.14: Algoritmo RBound-MP

Si consideramos que existe un conjunto m tareas y p procesadores para encontrar una planificación factible con el algoritmo RBound-MP, entonces la complejidad computacional de RBound-MP es O(m(p+log m)). Esto se explica de la siguiente manera: El algoritmo ScaleTaskSet requiere O(m log m), mientras que la heurística First Fit requiere O(mp) para asignar el conjunto de m tareas transformadas a los p procesadores. Finalmente, se necesita O(m) para mapear el conjunto original de tareas a los procesadores.

4.3 Algoritmos en Línea

4.3.1 RMGT/M

RMGT/M es una versión en línea del algoritmo RMST. El esquema RMGT/M se basa en las condiciones de planificabilidad utilizadas por el algoritmo RMST. La idea principal del algoritmo RMGT/M [19] consiste en dividir el conjunto de tareas y procesadores en M clases, tal que las tareas en la k-ésima clase sean asignadas a los procesadores de la k-ésima clase únicamente. La clase que corresponde a una tarea está determinada por:

$$m = \lfloor M\ (log_2\ (T)\ -\ \lfloor\ log_2\ (T)\ \rfloor)\ \rfloor\ +\ 1 \tag{4.3}$$

A cada procesador se le asignan tareas de una misma clase. De esta manera, para cada procesador el valor de β definido en el Teorema 4 (sección 2.2.3) es delimitado mediante el factor: $1\ -\ ln\ 2/M$. Para cada clase, el algoritmo mantiene un índice el cual corresponde al procesador actual. Si una nueva tarea de la clase k es incluida en el conjunto de tareas, entonces el algoritmo primero tratará de asignarla al procesador actual $P_{curr()}$ en la k-ésima clase. Si la tarea puede ser planificada sobre el procesador actual de acuerdo a la condición anterior, entonces se asigna la tarea. De lo contrario, la tarea es asignada a un procesador ocioso (obtenido por el procedimiento newproc()), el cual será el procesador actual. El algoritmo RMGT/M se describe en la Figura 4.15.

Teorema 15 ***(RMGT/M) [19]: Sea N y N_0 el número de procesadores requerido por RMGT/M y el mínimo número de procesadores requeridos para encontrar una planificación factible de un conjunto de tareas, respectivamente. Entonces, la siguiente condición se satisface***

Entrada: Un conjunto $\sum$ de tareas.

Salida: j; número de procesadores requerido para planificar $\sum$.

1. **begin**
2. $m := \lfloor M\ (log_2\ (T)\ -\ \lfloor\ log_2\ (T)\ \rfloor)\ \rfloor\ +\ 1$ /* Se determina la clase de la tarea */
3. **if** $(U_{curr(m)}\ +\ u_i\ \leq\ 1\ -\ (ln\ 2)/M)$ **then** /* si la tarea es planificable en $P_{curr(m)}$ */
4. $U_{curr(m)} := U_{curr(m)} + u_i$ /* Se asigna la tarea */
5. **else**
6. curr(m):=newproc();
7. $P_{curr(m)} := u_i$ /*se asigna la tarea τ_i al nuevo procesador oscioso */
8. **end**

Figura 4.15: Algoritmo Rate Monotonic General Tasks M (RMGT-M)

$$N\ <\ \frac{N_0}{1-(ln\ 2)/M-\alpha}\ +\ M \tag{4.4}$$

Donde, $\alpha\ =\ max_{i=1,...,n}\ (C_i/T_i)$***, si se cumple que*** $\alpha\ \leq\ (1-(ln\ 2)/M)\ /\ 2$ ***y***

$$N\ <\ \frac{2N_0}{1-(ln\ 2)\ /\ M}\ +\ M \tag{4.5}$$

Si se cumple que $\alpha\ >\ (1-(ln\ 2)\ /\ M)\ /\ 2$**.**

4.3.2 Algoritmos RMNF-WC, RMFF-WC y RMBF-WC

En las secciones 4.2.1, 4.2.2 y 4.2.3 se describieron los esquemas de asignación de los algoritmos RMNF, RMFF y RMBF respectivamente. En estos algoritmos, la condición de planificabilidad utilizada es la condición IP. Debido a la naturaleza de esta condición estos algoritmos son fuera de línea por la característica de que las tareas tienen que estar ordenadas de manera creciente con respecto a sus

periodos.

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_m\}$ y un conjunto de procesadores $\{P_1, \ldots, P_m\}$.

Salida: j; número de procesadores requeridos.

```
1.  i := 1; j := 1   /* i representa la i-ésima tarea, j representa el j-ésimo procesador */
2.  while (i ≤ m) do   /* Mientras no se asignen las m tareas */
3.      if (U_j + u_i ≤ Condición WC) then   /* U_j es la utilización del procesador j */
4.          U_j := U_j + u_i   /* Se asigna al procesador P_j la tarea τ_i */
5.      else
6.          U_{j+1} := U_{j+1} + u_i   /* Se asigna la tarea τ_i al procesador P_{j+1} */
7.          j := j + 1   /* Se incrementa el índice de procesadores */
8.      i := i + 1   /* Se incrementa el índice de tareas */
9.  end
```

Figura 4.16: Algoritmo Rate Monotonic Next Fit Worst Case (RMNF-WC)

Un esquema muy similar al de estos algoritmos es presentado en [20] con la diferencia de que los algoritmos a los que llamaremos RMFF-WC, RMNF-WC y RMBF-WC utilizan la condición WC como condición de planificabilidad. De esta forma, sin la necesidad de ordenar las tareas previamente, estos algoritmos se pueden usar en un esquema en línea.

Cabe hace notar que la única diferencia entre los algoritmos RMNF-IP y RMNF-WC es el uso de una condición diferente de planificabilidad (lo mismo se aplica para RMFF y RMBF), por lo que solamente se describirán los algoritmos y los rendimientos de éstos sin dar más detalles. En las figuras 4.16, 4.17 y 4.18

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. $i := 1$; $j := 1$ /* i representa la i-ésima tarea, j representa el j-ésimo procesador*/
2. **while** $(i < n)$ **do** /* Mientras no se asignen las n tareas */
3. $m := 1$; **while** $(U_m + u_i > (K_m + 1)(2^{1/(k_m+1)} - 1)$ **do**
4. $m := m + 1$ /* m representa el procesador índice */
5. $U_m := U_m + u_i$; $k_m := k_m + 1$ /* Se asigna la tarea i al procesador P_m */
6. **if** $(m > j)$ **then**
7. $j := m$ /* Se actualiza el no. de procesadores usados */
8. $i := i + 1$ /* Se incrementa el índice de tareas */
9. **end**

Figura 4.17: Algoritmo Rate Monotonic First Fit Worst Case (RMFF-WC)

U_m y K_m representan la utilización total del procesador m y el número de tareas asignadas al procesador m respectivamente. u_i es la utilización de la i-ésima tarea.

Teorema 16 ***(RMNF-WC) [20]: Para todos los conjuntos de tareas,*** $N \leq (2/(ln\ 2))N_0 + 1 \approx 2.88N_0 + 1$***, donde*** N_0 ***es el mínimo número de procesadores requeridos para planificar de manera factible el mismo conjunto de tareas, y*** N ***es el número de procesadores obtenido por el algoritmo RMNF-WC.***

Teorema 17 ***(RMFF-WC) [20]: Sea*** N ***el número de procesadores requerido para planificar un conjunto de tareas por el algoritmo RMFF-WC, y*** N_0 ***el***

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. $i := 1; j := 1$ /* i representa la i-ésima tarea, j representa el j-ésimo procesador */
2. **while** $(i < n)$ **do** /* Mientras existan tareas sin asignar */
3. **if** $(U_j + u_i \leq$ Condición WC) **and** $((K_m + 1)(2^{1/(K_m+1)} - 1) - (U_m + u_i))$ sea un valor lo menor posible
4. $U_j := U_j + u_i$ /* Se asigna la tarea al procesador P_j*/
5. $i := i + 1$ /* Se incrementa el índice de tareas */
6. **end**

Figura 4.18: Algoritmo Rate Monotonic Best Fit Worst Case (RMBF-WC)

mínimo número de procesadores requerido para encontrar una planificación factible del mismo conjunto de tareas. Entonces,

$$\boldsymbol{lim}_{No \to \infty}\ N/N_0 \ \leq\ 2 + (3\ -\ 2^{3/2})\ /\ (2\ (2^{1/3}\ -\ 1))\ \approx\ 2.33.$$

Teorema 18 ***(RMBF-WC) [20]: Sea*** N ***el número de procesadores requeridos para planificar de manera factible un conjunto de tareas usando el algoritmo RMBF-WC, y*** N_0 ***el número mínimo de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas. Entonces,***

$$\boldsymbol{lim}_{N_0 \to \infty}\ N/N_0 \ \leq\ 2\ +\ (3\ -\ 2^{3/2})\ /\ a\ \approx\ 2.33,$$

donde $a = 2\ (2^{1/3}\ -\ 1)$.

4.4 Algoritmos Redefinidos Fuera de Línea

4.4.1 Rate Monotonic Small Tasks Modificado (RMSTMod)

En el algoritmo RMST (figura 4.10) se utiliza el valor de

$$max\{\ ln\ 2,\ 1 -\ \beta\ ln\ 2\} \tag{4.6}$$

como condición de planificabilidad para ver si una tarea puede o no ser planificada factiblemente sobre un procesador en particular. Sin embargo, sin un costo computacional mayor al establecido por el algoritmo RMST, pueden aplicarse condiciones de planificabilidad más precisas. Estas condiciones de planificabilidad pueden derivarse directamente del teorema 4:

$$u_i + U_j \leq \begin{cases} \beta\ ln\ 2\ +\ 2^{1-\beta}\ -1 & para\ \ \beta\ \leq 1/2 \\ ln\ 2 & para\ \ \beta\ > 1/2 \end{cases} \tag{4.7}$$

donde u_i y U_j son las utilizaciones de la tarea τ_i y del procesador P_j respectivamente.

Aún más, si consideramos al número de tareas K como un parámetro dentro de la cota de planificabilidad, podríamos utilizar la siguiente condición:

$$u_i + U_j \leq \begin{cases} \beta\ ln\ 2\ +\ 2^{1-\beta}\ -1 & para\ \ \beta\ \leq 1 - 1/K \\ ln\ 2 & para\ \ \beta\ > 1 - 1/K \end{cases} \tag{4.8}$$

donde u_i y U_j son las utilizaciones de la tarea τ_i y del procesador P_j respectivamente, y K es el número de tareas en el conjunto.

La condición dada por la ecuación 4.8 mejora a la ecuación 4.6 cuando el valor de $\beta \leq 1-1/K$. Si consideramos que una función $\mathcal{F}(\mathrm{x})=x(2^{1/x}-1)$ es estrictamente decreciente con respecto a x entonces:

$$K(2^{1/K} - 1) > \lim_{K \to \infty} K(2^{1/K} - 1) = ln\ 2 \tag{4.9}$$

y

$$(K-1)(2^{\beta/(K-1)} - 1) + 2^{1-\beta} - 1 \ > \ \lim_{K \to \infty} (K-1)(2^{\beta/(K-1)} - 1) + 2^{1-\beta} - 1 = \beta\ ln\ 2 + 2^{1-\beta} - 1 \tag{4.10}$$

De la ecuación 4.10 podemos observar que:

$$\beta\ ln\ 2 + 2^{1-\beta} - 1 > 1 - \beta\ ln\ 2 \tag{4.11}$$

Utilizando la cota da planificabilidad dada en la ecuación 4.8, en donde se considera el número de tareas, se espera que el rendimiento en el caso promedio del algoritmo RMSTMod mejore a RMST. Sin embargo, el límite asintótico de N/N_{opt} permanece igual al establecido en el teorema 13. El pseudocódigo del algoritmo RMSTMod se encuentra definido en la figura 4.19.

La complejidad computacional del algoritmo RMSTMod se considera como O(n log n), la cual se encuentra determinada principalmente por el ordenamiento de las tareas (paso 1). Cuando el algoritmo termina, el número mínimo de procesadores para encontrar una planificación factible del conjunto de tareas está determinado por la variable j.

4.4.2 Rate Monotonic General Tasks Modificado (RMGTMod)

El algoritmo RMGT utiliza una estrategia conocida como divide y vencerás, con la cual el conjunto de tareas de entrada es dividido en dos grupos. Una tarea puede pertenecer a un grupo u otro dependiendo de su utilización. En el primer y segundo grupo las tareas satisfacen la condición de que $u_i \leq 1/3$ y

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_m\}$ y un conjunto de procesadores $\{P_1, \ldots, P_m\}$.

Salida: j; número de procesadores requeridos.

1. Ordenar el conjunto de tareas tal que $0 \leq S_i \leq ... \leq S_m < 1$
2. $S_{m+1} := S_1 + 1; i := 1; j := 1$ /*i representa la i-ésima tarea, j representa el j-ésimo*/ /* procesador. */
3. $S := S_i; \beta_j := 0$
4. **while** $(i \leq m)$ **do** /* Mientras no se asignen las m tareas */
5. **if** $(u_i + U_j) \leq ((\beta\, ln\ 2\ +\ 2^{1-\beta}\ -1 \quad \text{para}\ \ \beta\ \leq 1 - 1/K)$ ó $(ln\ 2 \quad \text{para}\ \ \beta\ > 1 - 1/K))$ **then**
6. $U_j := U_j + u_i$ /* Se asigna la tarea τ_i al procesador P_j */
7. $\beta_j := S_i - S$ /* Se actualiza el valor de β_j */
8. **else**
9. $j := j + 1; S := S_i; \beta_j := 0$ /*Se incrementa el índice de procesadores */
10. $U_j := U_j + u_i$ /* Se asigna la tarea al procesador P_j */
10. $i := i + 1$ /* Se incrementa el índice de tareas */
11. $\beta_j := S_{m+1} - S$
12. **end**

Figura 4.19: Algoritmo Rate Monotonic Small Tasks Modificado (RMSTMod)

$u_i > 1/3$ **respectivamente. El primer grupo es asignado utilizando el algoritmo RMST y el segundo grupo se asigna utilizando la heurística First Fit en donde se considera a la condición IFF como cota de planificabilidad para determinar si una tarea es planificable o no dentro de un procesador en particular. Sin embargo, nótese que el algoritmo RMGT nunca asigna tareas "grandes" (con** $u_i > 1/3$)

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. Particionar el conjunto de tareas en dos grupos:

   $G_1 = \{ \tau_i \mid u_i \leq 1/3\} \qquad G_2 = \{ \tau_i \mid u_i > 1/3\}$

2. Las tareas del segundo grupo se asignan de la siguiente manera:

3. $i := 1$; $U_1 := u_i$; /* Se asigna τ_i al procesador P_1 */

4. $i + i + 1$ /* Se incrementa el índice de las tareas */

5. Usar la heurística First Fit para encontrar un procesador m que contenga una sola tarea, digamos τ_W que cumpla:

   $\lfloor T_w/T_i \rfloor (T_i - C_i) \geq C_w$ ó $T_w \geq \lfloor T_w/T_i \rfloor C_i + C_w$ sí $T_i < T_w$

   $\lfloor T_i/T_w \rfloor (T_w - C_w) \geq C_i$ ó $T_i \geq \lfloor T_i/T_w \rfloor C_w + C_i$ sí $T_i \geq T_w$

6. **if** el procesador existe **then**

7. $U_m = U_m + u_i$ /* Se asigna la tarea τ_i al procesador P_m */

8. **else**

9. Se asigna τ_i a un nuevo procesador ocioso.

10. Las tareas del primer grupo se asignan de la siguiente manera:

11. Con la heurística Next Fit encontar un procesador del grupo 2 donde la tarea sea planificada utilizando la condición de planificabilidad UO.

12. **if** el procesador existe **then**

13. Se asigna la tarea τ_i al procesador perteneciente al segundo grupo.

14. **else**

15. Asignar la tarea utilizando el algoritmo RMST.

16. Terminar cuando todas las tareas del grupo G_1 hayan sido asignadas.

Figura 4.20: Algoritmo Rate Monotonic General Tasks Modificado (RMGTMod)

y tareas "pequeñas" (con $u_i \leq 1/3$) sobre el mismo procesador. Sin duda, es posible que algunos procesadores donde se planifiquen tareas "grandes" puedan acomodar un número determinado de tareas "pequeñas". Esto definitivamente incrementaría el rendimiento en el caso promedio y podría en algún momento, mejorar el límite en el peor caso dado por N/N_{opt}. El esquema de asignación del algoritmo RMGTMod se presenta en la figura 4.20. El algoritmo RMGTMod a diferencia del algoritmo RMGT, primero asignará las tareas correspondientes al segundo grupo y luego asignará las tareas correspondientes al primer grupo. Las tareas del segundo grupo se asignarán utilizando la condición de planificabilidad IFF. Cada tarea perteneciente al primer grupo tratará primero de ser planificada utilizando la heurística Next Fit en algún procesador resultante de la asignación del segundo grupo. Si no encuentra un procesador dentro del segundo grupo, entonces la tarea se planificará utilizando el algoritmo RMST. Para mantener una complejidad computacional baja en la asignación del primer grupo se utilizará la condición UO para decidir si la tarea es o no planificable dentro de un procesador correspondiente al segundo grupo.

4.5 Algoritmos Redefinidos en Línea

4.5.1 Rate Monotonic First Fit Modificado (RMFFMod)

El algoritmo RMFF tiende a presentar un buen comportamiento cuando es usado para planificar un conjunto de tareas con cargas de trabajo pequeñas, pero su comportamiento se degrada rápidamente cuando el factor de carga máximo del conjunto (α) crece. Por lo tanto, es común el tratar de encontrar un esquema diferente para la asignación de las tareas con factores de carga relativamente grandes.

RMFFMod divide a los procesadores en dos grupos ($G1$ y $G2$), de tal forma que en cada grupo existe un número de procesadores infinito. El conjunto de tareas es también dividido en dos grupos. De tal forma que las tareas correspondientes al primer grupo sólo podrán ser asignadas a los procesadores de ese grupo. De la misma manera se considera la asignación para las tareas correspondientes al segundo grupo. Una tarea τ_i pertenece al primer grupo siempre que satisfaga $u_i \leq 1/3$, de otra manera, la tarea τ_i pertenecerá al segundo grupo. Para planificar una tarea, el algoritmo RMFFMod primero identificará su grupo correspondiente y luego buscará al procesador con el menor índice donde la tarea sea planificada factiblemente. La heurística First Fit es la usada para asignar las tareas a procesadores en ambos grupos. Una descripción algorítmica del algoritmo RMFFMod se presenta en la figura 4.21.

La condición de planificabilidad para el primer grupo de tareas es la condición WC. El segundo grupo de tareas es planificado usando la condición IFF. Debido a que el algoritmo asume que solamente dos tareas puede ser planificadas sobre un procesador en el segundo grupo, la complejidad de la condición IFF se reduce a solamente dos operaciones de comparación. De esta forma la complejidad computacional de RMFFMod es O(n log n). El rendimiento en el peor caso de RMFFMod se encuentra definido en la proposición 1.

Proposición 1 ***(RMFFMod): Sea*** N ***el número de procesadores requerido para planificar factiblemente un conjunto de tareas mediante el algoritmo RMFFMod, y*** N_0 ***el número mínimo de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas.***
Entonces, $\Re(RMFFMod)^{N_0 \to \infty} \leq 7/4$.

La prueba de la proposición 1 se realizará con la ayuda de los siguientes teoremas:

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. Particionar el conjunto de tareas en dos grupos:

   $G_1 = \{ \tau_i \mid u_i \leq 1/3\} \qquad G_2 = \{ \tau_i \mid u_i > 1/3\}$

2. Las tareas pertenecientes al primer grupo (G1), se asignan a los procesadores utilizando la heurística First Fit.
3. Las tareas del segundo grupo se asignan de la siguiente manera:
4. $i := 1$; $U_1 := u_i$; /* Se asigna τ_i al procesador P_1 */
5. $i + i + 1$ /* Se incrementa el índice de las tareas */
6. Usar la heurística First Fit para encontrar un procesador m que contenga una sola tarea, digamos τ_W que cumpla:

   $\lfloor T_w/T_i \rfloor (T_i - C_i) \geq C_w$ ó $T_w \geq \lfloor T_w/T_i \rfloor C_i + C_w$ sí $T_i < T_w$

   $\lfloor T_i/T_w \rfloor (T_w - C_w) \geq C_i$ ó $T_i \geq \lfloor T_i/T_w \rfloor C_w + C_i$ sí $T_i \geq T_w$
7. **if** el procesador existe **then**
8. $U_m = U_m + u_i$ /* Se asigna la tarea τ_i al procesador P_m */
9. **else**
10. Se asigna τ_i a un nuevo procesador ocioso.
11. **goto 5**
12. Terminar cuando todas las tareas del grupo G_2 hayan sido asignadas.

Figura 4.21: Algoritmo Rate Monotonic First Fit Modificado (RMFFMod)

Teorema 19 ***[19] Si la carga total de un conjunto de K tareas de tiempo real satisface que:***

$$U \leq \frac{K}{2^{1/k}+1}$$

entonces el conjunto de tareas puede ser planificado con el algoritmo RM, en menos de K procesadores.

El corolario 3 presenta una buena aproximación del teorema 19.

Corolario 3 ***[19]: Si la carga total U de un conjunto de K tareas de tiempo real (K $\geq$2) satisface que:***

$$U \leq K/2 - ln\ 2/4$$

entonces el conjunto de tareas puede ser planificado por el algoritmo RM en menos que K procesadores.

Lema 7 ***[22]: Si el conjunto de tareas es asignado a los procesadores utilizando el algoritmo RMFF, de entre todos los procesadores en los cuales*** $n \geq 1$ ***tareas fueron asignadas, existe a lo más un procesador en donde el factor de utilización de las n tareas es menor o igual que*** $n(2^{1/n+1}-1)$

Prueba de la proposición 1: Sea $\sum = \{\tau_1, \tau_2, \ldots, \tau_m\}$ **un conjunto de** m **tareas de tiempo real. Supóngase que** $N = N_{G1} + N_{G2}$ **es el número de procesadores utilizados por la heurística RMFFMod para planificar el conjunto de tareas** $\sum$**. Donde** N_{G1} **y** N_{G2} **son el número de procesadores utilizados por el grupo** $G1$ **y** $G2$ **respectivamente. De entre los** N_{G2} **procesadores, sea** n_1 **el número de procesadores en los que se asignó solamente una tarea y** n_2 **el número de procesadores donde se asignaron dos tareas. Entonces** $N_{G2} = n_1 + n_2$**.**

Para los n_1 **procesadores en donde solamente una tarea fue asignada, tenemos por el teorema 19 que:**

$$\sum_{i=1}^{n_1} u_i > n_1/(2^{1/n_1}+1) > n_1/2 - ln\ 2/4 \tag{4.12}$$

Para los n_2 procesadores en los cuales dos tareas fueron asignados, se tiene que:

$$\sum_{i=1}^{n_2}(u_{i,1} + u_{i,2}) > 2n_2/3 \tag{4.13}$$

esto se debe a que las dos tareas asignadas al procesador poseen valores mayores a 1/3. Es decir, $u_{i,1} > 1/3$ y $u_{i,2} > 1/3$.

Si se considera que las tareas asignadas en los procesadores del grupo $G1$ poseen valores menores a 1/3. A lo menos tres tareas serán asignadas a cada procesador con excepción tal vez del último procesador. Por lo tanto considerando el lema 7, de entre todos los procesadores en los que se asignaron al menos tres tareas, existe a lo más un procesador cuya utilización no es mayor a $3(2^{1/4} - 1)$. De esto se tiene que:

$$\sum_{i=1}^{N_{G1}} u_i \geq 3(2^{1/4} - 1)(N_{G1} - 1) \tag{4.14}$$

Si observamos que la utilización total del conjunto de tareas $\sum$ está dada por $\sum_{i=1}^{m} u_i$. Y considerando las ecuaciones 4.12, 4.13 y 4.14 se obtiene que:

$$\sum_{i=1}^{m} u_i = \sum_{i=1}^{n_1} u_i + \sum_{i=1}^{n_2}(u_{i,1} + u_{i,2}) + \sum_{i=1}^{N_{G1}} u_i \tag{4.15}$$

$$\geq n_1/2 - ln\ 2/4 + 2n_2/3 + 3(2^{1/4} - 1)(N_{G1} - 1) \tag{4.16}$$

Ahora si consideramos que $N = n_1 + n_2 + N_{G1}$ y además que $N_0 \geq \sum_{i=1}^{m} u_i$, es decir, que el número óptimo de procesadores es mayor o igual a la carga total del conjunto $\sum$. Se tiene que:

$$N_0 \geq n_1/2 - ln\ 2/4 + 2n_2/3 + 3(2^{1/4} - 1)(N_{G1} - 1) \tag{4.17}$$

Substituyendo $n_2 = N - n_1 - N_{G1}$ en la ecuación 4.17

$$N_0 \geq n_1/2 - ln\ 2/4 + 2(N - n_1 - N_{G1})/3 + 3(2^{1/4} - 1)(N_{G1} - 1) \tag{4.18}$$

$$\geq n_1/2 - ln\ 2/4 + \frac{2N - 2n_1 - 2N_{G1}}{3} + 3(2^{1/4} - 1)N_{G1} - 3(2^{1/4} - 1) \quad (4.19)$$

$$\geq 2N/3 + n_1(1/2 - 2/3) - ln\ 2/4 - 3(2^{1/4} - 1) + N_{G1}(3(2^{1/4} - 1) - 2/3) \quad (4.20)$$

$$\geq 2N/3 - ln\ 2/4 - 3(2^{1/4} - 1) + N_{G1}(3(2^{1/4} - 1) - 2/3) - n_1/6 \quad (4.21)$$

Partiendo del hecho de que cualquier par de tareas que fueron asignadas a los n_1 procesadores no pueden ser planificadas sobre un solo procesador, se tiene $N_0 \geq n_1$. De esta forma:

$$N_0 \geq 2N/3 - ln\ 2/4 - 3(2^{1/4} - 1) + N_{G1}(3(2^{1/4} - 1) - 2/3) - n_1/6 \quad (4.22)$$

$$N_0 \geq 2N/3 - ln\ 2/4 - 3(2^{1/4} - 1) + N_{G1}(3(2^{1/4} - 1) - 2/3) - N_0/6 \quad (4.23)$$

Obteniendo N/N_0

$$2N/3 \leq ln\ 2/4 + 3(2^{1/4} - 1) - N_{G1}(3(2^{1/4} - 1) - 2/3) + N_0/6 + N_0 \quad (4.24)$$

Multiplicando por $3/2N_0$

$$N/N_0 \leq [ln\ 2/4 + 3(2^{1/4} - 1)]\frac{3}{2N_0} - \frac{3N_{G1}(3(2^{1/4} - 1) - 2/3)}{2N_0} + 7/4 \quad (4.25)$$

De la ecuación 4.25 se tiene que:

$$\Re(RMFFMod)^{N_0 \to \infty} \ \leq \ 7/4 \ \ Q.E.D. \quad (4.26)$$

4.5.2 Rate Monotonic Best Fit Modificado (RMBFMod)

RMBFMod tiene un comportamiento similar al algoritmo RMFFMod, excepto que utiliza la heurística Best Fit para realizar la asignación de las tareas de los dos grupos a los procesadores. Una descripción algorítmica de RMBFMod se presenta en la figura 4.22. El rendimiento en el peor caso del algoritmo RMBFMod se presenta en la proposición 2. La demostración puede verificarse de manera similar que la realizada en la proposición 1.

Entrada: Un conjunto de tareas $\sum = \{\tau_1, \tau_2, \ldots, \tau_n\}$ y un conjunto de procesadores $\{P_1, \ldots, P_n\}$.

Salida: j; número de procesadores requeridos.

1. Particionar el conjunto de tareas en dos grupos:

   $G_1 = \{ \tau_i \mid u_i \leq 1/3\}$ $\quad$ $G_2 = \{ \tau_i \mid u_i > 1/3\}$

2. Las tareas pertenecientes al primer grupo (G1), se asignan a los procesadores utilizando la heurística Best Fit.

3. Las tareas del segundo grupo se asignan de la siguiente manera:

4. $i := 1$; $U_1 := u_i$; /* Se asigna τ_i al procesador P_1 */

5. $i + i + 1$ /* Se incrementa el índice de las tareas */

6. Usar la heurística Best Fit para encontrar un procesador m que contenga una sola tarea, digamos τ_W que cumpla:

   $\lfloor T_w/T_i \rfloor (T_i - C_i) \geq C_w$ ó $T_w \geq \lfloor T_w/T_i \rfloor C_i + C_w$ $\quad$ sí $T_i < T_w$

   $\lfloor T_i/T_w \rfloor (T_w - C_w) \geq C_i$ ó $T_i \geq \lfloor T_i/T_w \rfloor C_w + C_i$ $\quad$ sí $T_i \geq T_w$

7. **if** el procesador existe **then**

8. $U_m = U_m + u_i$ /* Se asigna la tarea τ_i al procesador P_m */

9. **else**

10. Se asigna τ_i a un nuevo procesador ocioso.

11. **goto 5**

12. Terminar cuando todas las tareas del grupo G_2 hayan sido asignadas.

Figura 4.22: Algoritmo Rate Monotonic Best Fit Modificado (RMFFMod)

Proposición 2 ***(RMBFMod): Sea*** N ***el número de procesadores requerido para planificar factiblemente un conjunto de tareas mediante el algoritmo RMBF-Mod, y*** N_0 ***el número mínimo de procesadores requeridos para encontrar una planificación factible del mismo conjunto de tareas.***
Entonces, $\Re(RMBFMod)^{N_0 \to \infty} \leq 7/4$.

Para finalizar este capítulo, en la tabla 4.4 se presenta un resumen de las características de los algoritmos de planificación de tareas de tiempo real sobre multiprocesadores, en donde cada procesador es planificado bajo la política Rate Monotonic. En la tabla 4.4, α se define como el factor máximo de carga de cualquier tarea en el conjunto, $\alpha = max(\ u_i); \qquad (i = 1, \ldots, n)$. M es el número de clases en que se divide el conjunto de tareas y $N = \sum_{i=1}^{m} \lfloor T_m/T_i \rfloor$ donde T_m es el valor máximo del periodo dentro del conjunto de tareas.

Algoritmo	Asignación	Cota de Planificabilidad	Complejidad	$lim_{N_{opt}\to\infty} N/N_{opt}$
RMNF	Fuera de Linea	Condición IP	O(n log n)	2.67
RMFF	Fuera de Linea	Condición IP	O(n log n)	2.33
RMBF	Fuera de Linea	Condición IP	O(n log n)	2.33
RM-FFDU	Fuera de Linea	Condición UO	O(n log n)	5/3
FFDUF	Fuera de Linea	Condición WC	O(n log n)	2.0
RMST	Fuera de Linea	Condición PO	O(n log n)	1/1-α
RMGT	Fuera de Linea	Condición PO y IFF	O(n log n)	7/4
RBound-MP	Fuera de Linea	Condición Rbound	O(n log n)	Desconocido
EXACT-MP	Fuera de Linea	Condición IFF	O(n^3N)	Desconocido
RMNF-WC	En Linea	Condición WC	O(n)	2.88
RMFF-WC	En Linea	Condición WC	O(n log n)	2.33
RMBF-WC	En Linea	Condición WC	O(n log n)	2.33
RMGT-M	En Linea	Condición PO	O(n)	1/(1-α) +O(1/M)
Next Fit M	En Linea	Condición WC	O(n)	2.28 + O(1/M)
RMSTMod	Fuera de Línea	Condición PO	O(n log n)	1/1-α
RMGTMod	Fuera de Línea	Condición PO y IFF	O(n log n)	Desconocido
RMFFMod	En Línea	Condición WC y IFF	O(n log n)	7/4
RMBFMod	En Línea	Condición WC y IFF	O(n log n)	7/4

Tabla 4.4: Características de los Algoritmos de Planificación

Capítulo 5

Rendimientos de los Algoritmos en el Caso Promedio

En el capítulo anterior, se expusieron los rendimientos de los algoritmos considerando las condiciones de éstos en los peores casos. Aunque un análisis en el peor caso nos proporciona límites de rendimiento de los algoritmos, no nos proporciona información necesaria para determinar un rendimiento promedio de éstos.

Para obtener los rendimientos de los algoritmos en los casos promedios, se considerará una serie de experimentos de simulación. La simulación será realizada por medio de la ejecución de los algoritmos para conjuntos grandes de tareas de entrada y promediando los resultados sobre un número de ejecuciones aceptables. La simulación consiste en dos etapas: en la primera etapa, se comparará el rendimiento de los algoritmos fuera de línea, en una segunda etapa se analizarán los algoritmos considerados en un esquema en línea.

5.1 Condiciones de Simulación

Se presentan experimentos de simulación para conjuntos grandes de tareas con las siguientes características:

1. **Se definen K conjuntos de tareas donde** $100 \leq K \leq 1000$.
2. **Se varía el parámetro** $\alpha = max_{i=1,\ldots,k}\ u_i$ **entre [0,1], donde** α **es el factor de carga máximo dentro del conjunto de tareas.**
3. **Los periodos de las tareas** T_i**, son generados en forma aleatoria con valores entre** $1 \leq T_i \leq 500$ **siguiendo una función de distribución uniforme.**
4. **Los tiempos de ejecución de las tareas también son generados siguiendo una función de distribución uniforme con rango** $0 \leq C_i \leq \alpha\ T_i$.
5. **La métrica de rendimiento en todos los experimentos es el número de procesadores requeridos para planificar de manera factible el conjunto de tareas.**
6. **En las figuras, cada punto representa el promedio de 100 valores generados independientemente con parámetros idénticos.**
7. **Todos los algoritmos serán ejecutados sobre conjuntos de tareas idénticas.**

Considerando que no es práctico encontrar una planificación óptima para conjunto grandes de tareas, se utilizará la carga total del conjunto ($U = \sum_{i=1}^{K} u_i$**) como límite inferior para el número de procesadores que una planificación óptima requiere para planificar el conjunto de tareas.**

5.2 Comparación de los Algoritmos Fuera de Línea

En esta sección se presenta una comparación de los siguientes algoritmos:

- **Rate Monotonic First Fit (RMFF) [6] (Sección 4.2.2).**
- **Rate Monotonic Best Fit (RMBF) [22] (Sección 4.2.3).**
- **Rate Monotonic Next Fit (RMNF) [6] (Sección 4.2.1).**
- **Rate Monotonic Small Tasks (RMST) [19] (Sección 4.2.5).**
- **Rate Monotonic General Tasks (RMGT) [19] (Sección 4.2.6).**
- **Rate Monotonic Fist Fit Decreasing Utilization (RM-FFDU) [21] (Sección 4.2.4).**
- **First Fit Decreasing Utilization Factor (FFDUF) [4].**
- **RBound-MP [16] (Sección 4.2.7).**

Los resultados son presentados en las siguientes figuras: Figura 5.1 para $\alpha = 0.2$**, Figura 5.2 para** $\alpha = 0.5$**, Figura 5.3 para** $\alpha = 0.8$**, y Figura 5.4 para** $\alpha = 1.0$**. De los experimentos se concluye que:**

- **De entre todos los algoritmos y para cualquier factor de carga del conjunto de tareas, el peor resultado lo presenta el algoritmo RMNF.**
- **El número de procesadores usados por la heurística RMNF tiene un crecimiento proporcional al factor de carga del sistema.**
- **A pesar de las modificaciones en el esquema de asignación de tareas, el algoritmo RMBF se comporta de forma muy similar al algoritmo RMFF para los diversos factores de carga de las tareas.**
- **El algoritmo RM-FFDU tiene un comportamiento similar cuando** α **es pequeño (**$\alpha \leq 2$**) a los algoritmos RMFF y FFDUF.**

- **Aunque los algoritmos FFDUF y RM-FFDU tienden a presentar el mismo comportamiento, RM-FFDU siempre mejora a FFDUF.**
- **El rendimiento que presenta el algoritmo RMGT es bastante bueno cuando** $\alpha < 0.6$**.**
- **Los rendimientos de los algoritmos RMST y RMGT son idénticos cuando** $\alpha < 1/3$**.**
- **El algoritmo RMST es similar al algoritmo RMFF cuando** $\alpha \approx 0.8$**.**
- **RMST y RMGT proporcionan un buen rendimiento cuando** α **es pequeño.** $\alpha < 0.25$
- **En las figuras se puede apreciar un comportamiento similar del algoritmo RBOUND-MP proporcional al factor de carga.**
- **Para factores de carga de las tareas grandes** ($\alpha > 0.8$) **el mejor comportamiento lo presenta el algoritmo RM-FFDU.**
- **Si el factor de carga se incrementa, el algoritmo que degrada su rendimiento de manera muy rápida es el algoritmo RMST.**
- **El algoritmo RMGT es similar al algoritmo RM-FFDU cuando** $\alpha \approx 0.8$**.**
- **Para factores de carga grandes** ($\alpha > 0.8$) **tanto el algoritmo RMST como el algoritmo RMGT tienen rendimientos peores que RMFF.**
- **Sin importar el factor de carga de las tareas, el algoritmo RMGT siempre mejora al algoritmo RMST.**
- **El número de procesadores que el algoritmo RMFF utiliza crece de manera proporcional al factor de carga del sistema.**

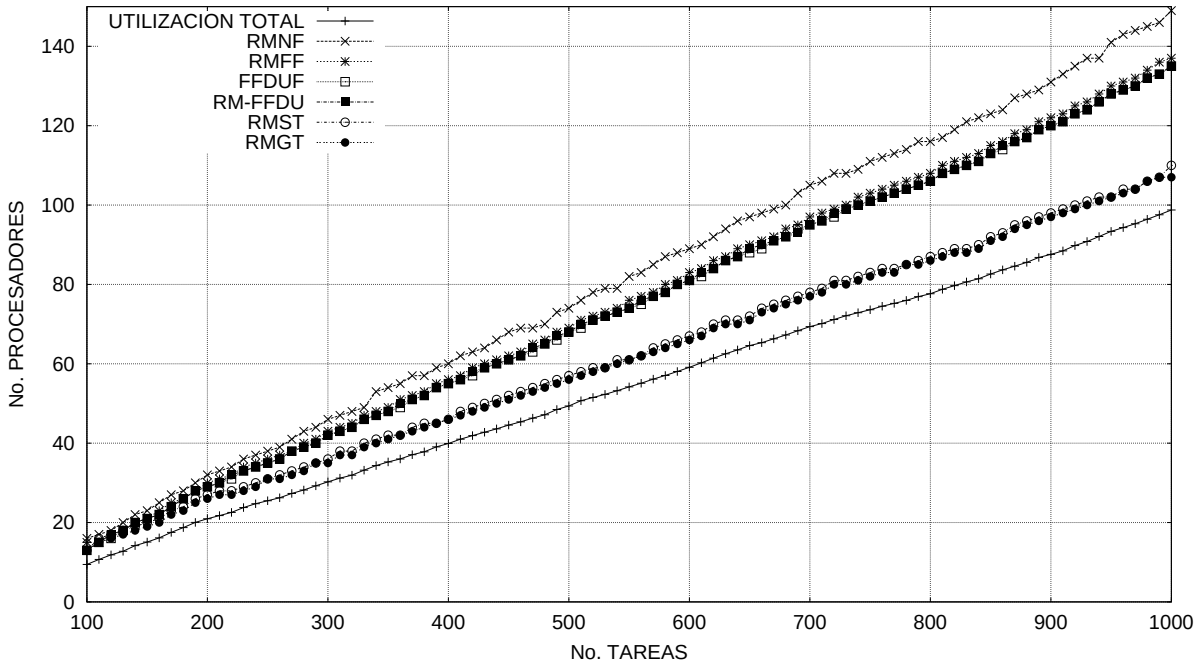

Figura 5.1: Rendimientos de Algoritmos Fuera de Línea $\alpha = 0.2$

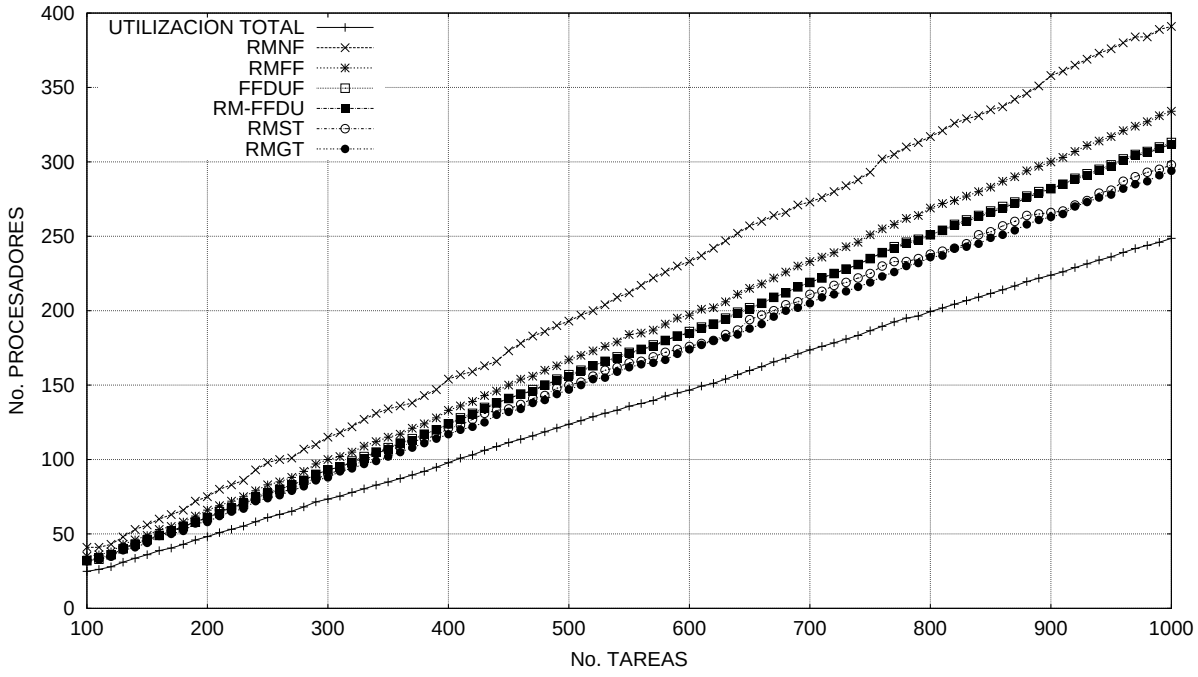

Figura 5.2: Rendimientos de Algoritmos Fuera de Línea $\alpha = 0.5$

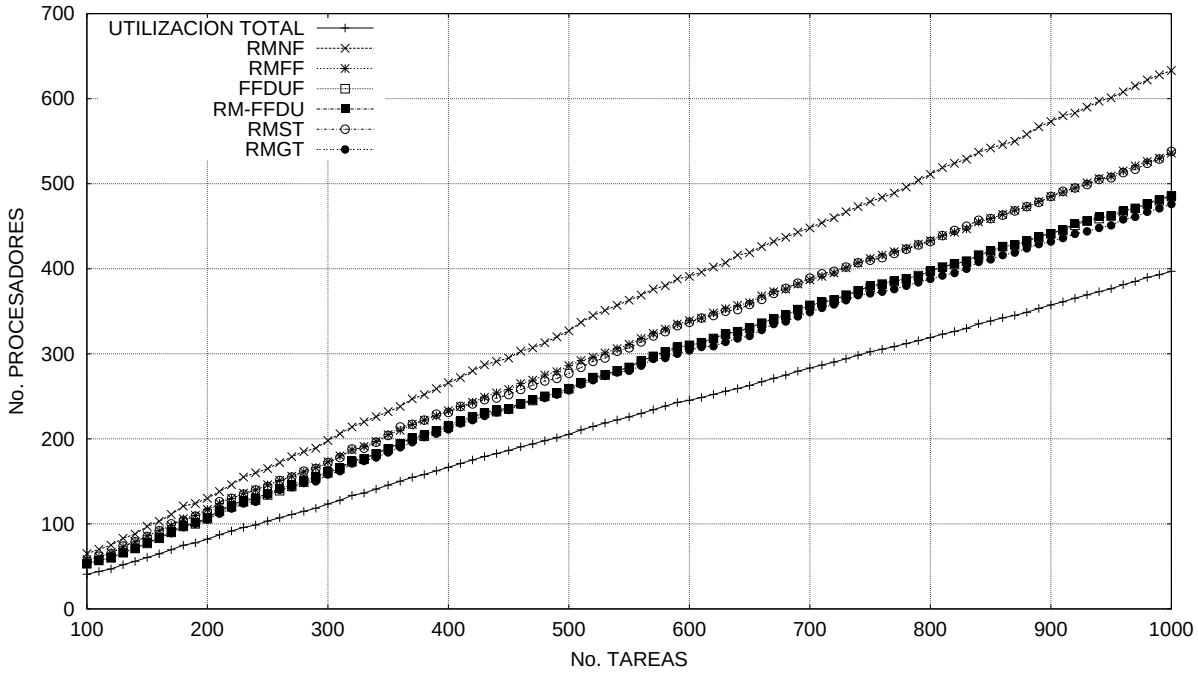

Figura 5.3: Rendimientos de Algoritmos Fuera de Línea $\alpha = 0.8$

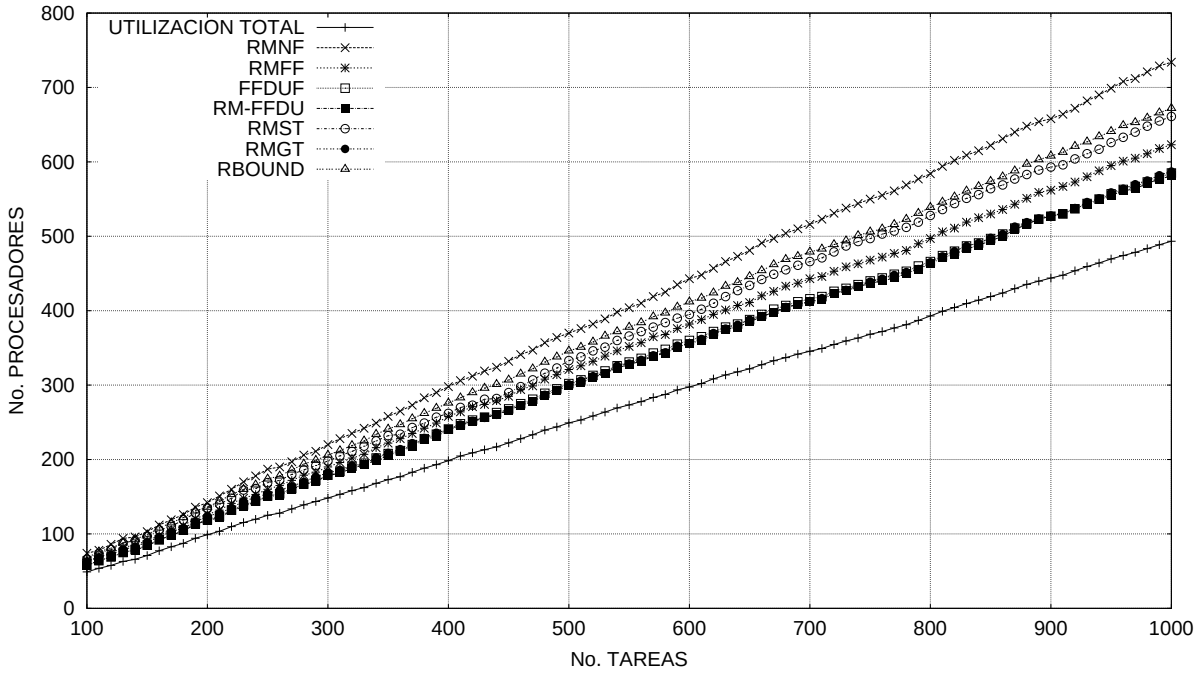

Figura 5.4: Rendimientos de Algoritmos Fuera de Línea $\alpha = 1.0$

Entre otros resultados, se presentan los porcentajes de procesadores extras que los algoritmos de planificación requieren. El porcentaje de procesadores extra se define como $(N_A - N_0)/N_0$, donde N_A es el número de procesadores que la heurística A utiliza para planificar de manera factible el conjunto de tareas y N_0 es el número de procesadores óptimo para planificar el mismo conjunto de tareas. Así, en las figuras 5.5, 5.6, 5.7 y 5.8 el eje X representa el número de procesadores óptimo N_0 y el eje Y representa los porcentajes de procesadores extras $(N_A - N_0)/N_0$ de los algoritmos de planificación. También, en las figuras 5.9, 5.10, 5.11 y 5.12 se presenta la utilización promedio por procesador de los algoritmos. De lo que se concluye que:

- **Excepto por el algoritmo RMNF todos los algoritmos, sin importar el factor de carga, utilizan menos del 60% de procesadores extras.**

- **Cuando $\alpha \leq 0.3$ y el número de tareas en el conjunto crece, los algoritmos RMGT y RMST disminuyen los porcentajes de procesadores extras.**

- **Los porcentajes de procesadores extras de los algoritmos RM-FFDU y FF-DUF tienen comportamientos similares para diferentes factores de carga.**

- **Para factores de carga $\alpha < 0.8$ el porcentaje de procesadores extra del algoritmo RMFF es $\approx 40\%$.**

- **Las utilizaciones promedio por procesador de los algoritmos RMGT y RMST son altas $\approx$ 90% cuando el factor de carga es pequeño $\alpha \leq 0.3$.**

- **Excepto por el algoritmo RMNF, todos los algoritmos presentan porcentajes de utilización por procesador altos $\approx$ entre 75% y 85% cuando el factor de carga $\alpha \approx 0.5$.**

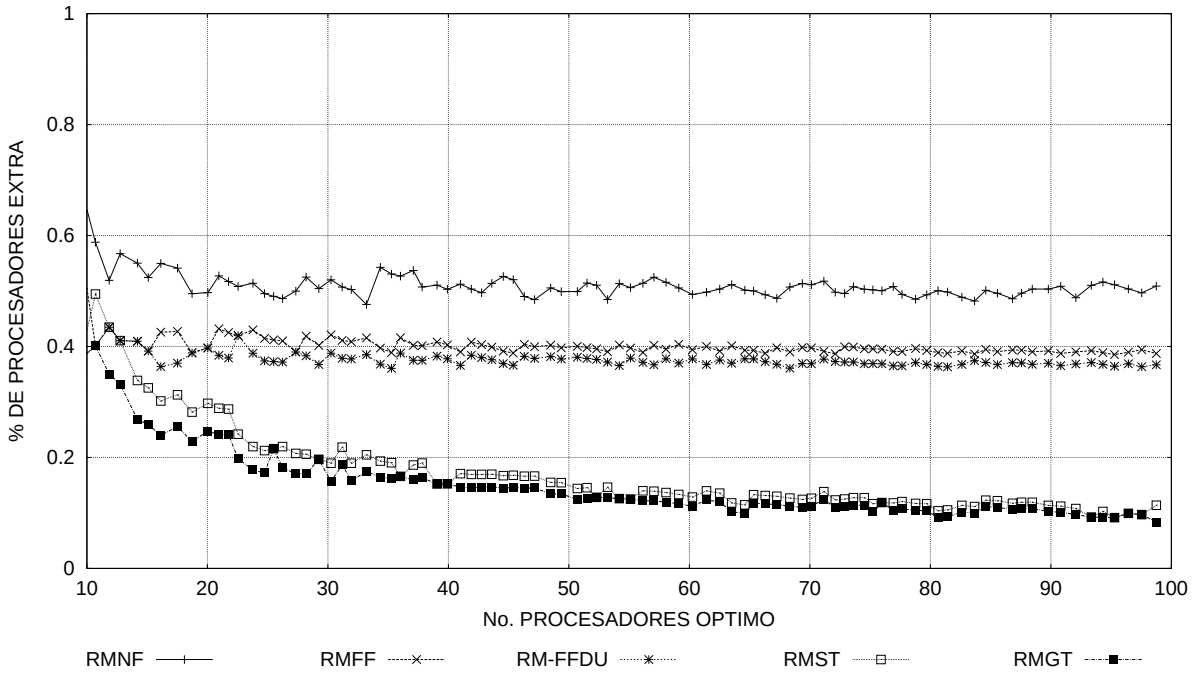

Figura 5.5: Porcentajes de procesadores extras de los Algoritmos Fuera de Línea $\alpha = 0.2$

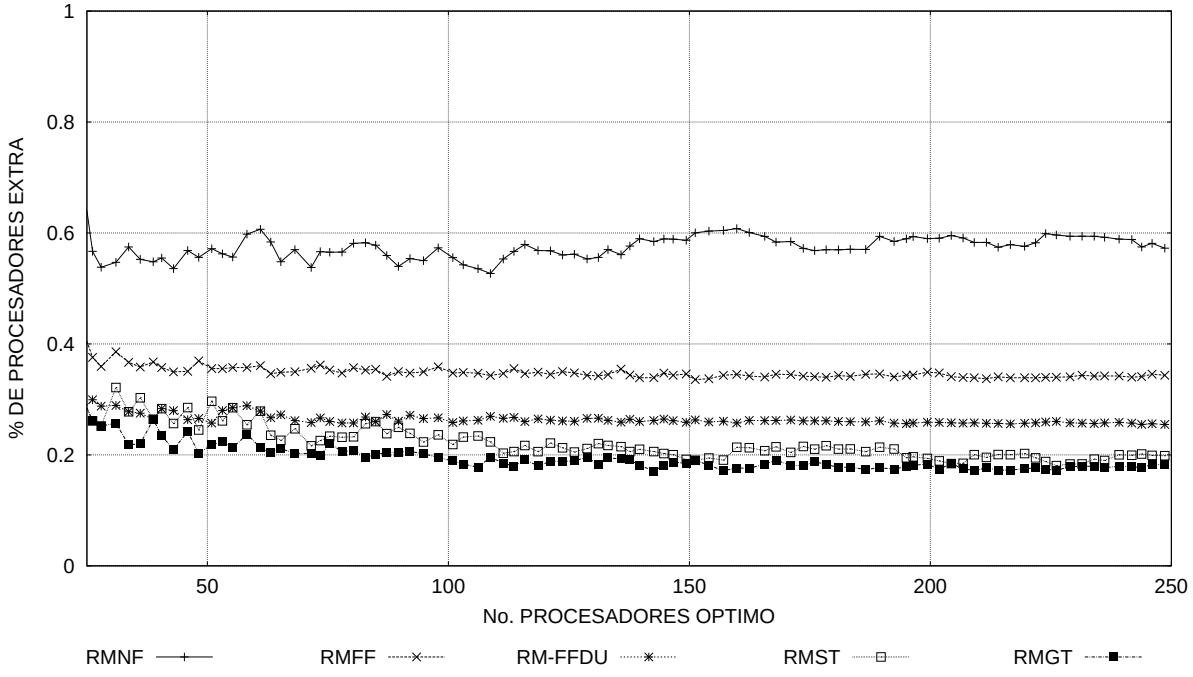

Figura 5.6: Porcentajes de procesadores extras de los Algoritmos Fuera de Línea $\alpha = 0.5$

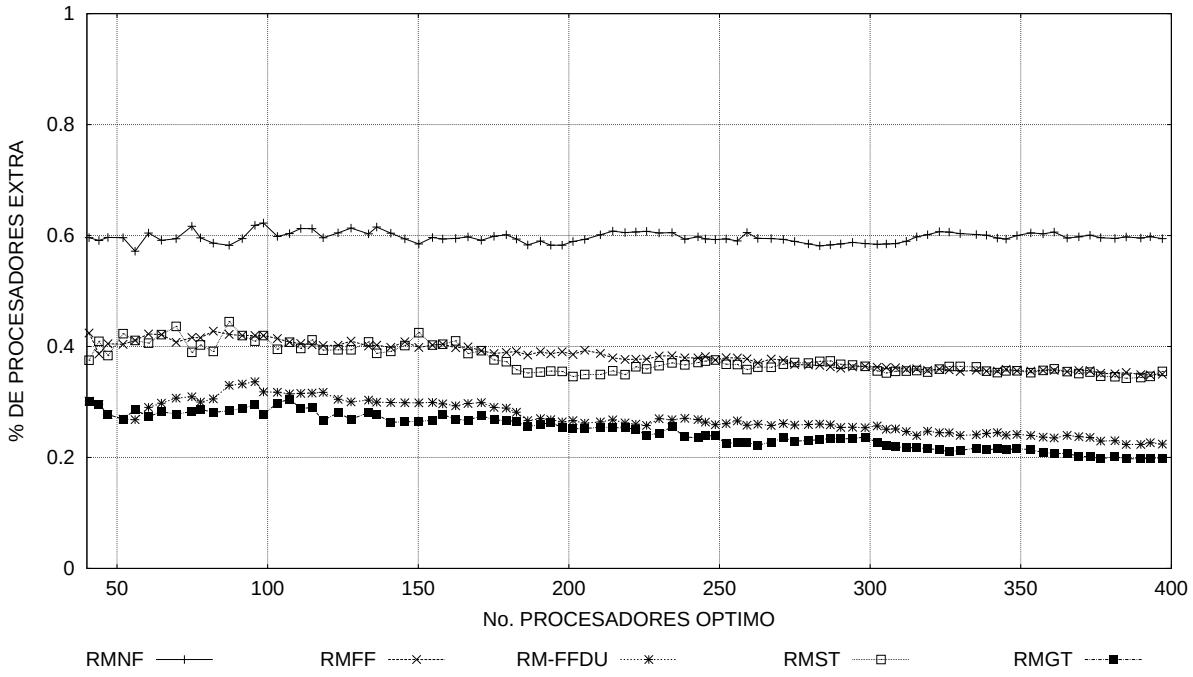

Figura 5.7: Porcentajes de procesadores extras de los Algoritmos Fuera de Línea $\alpha = 0.8$

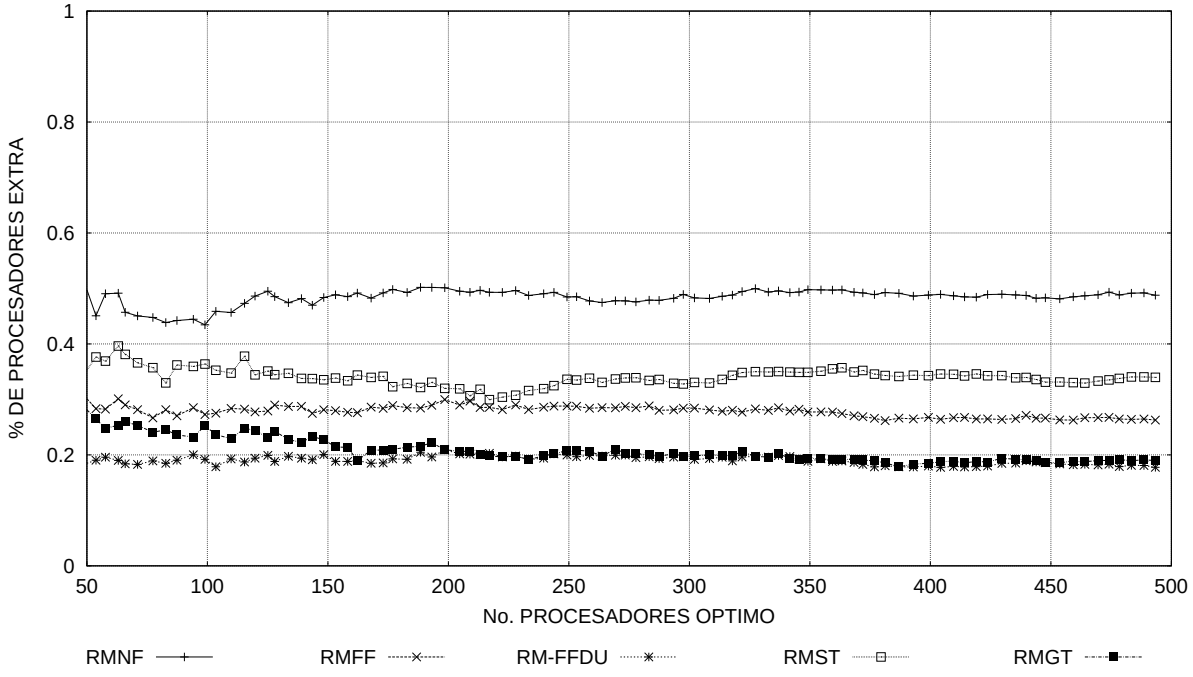

Figura 5.8: Porcentajes de procesadores extras de los Algoritmos Fuera de Línea $\alpha = 1.0$

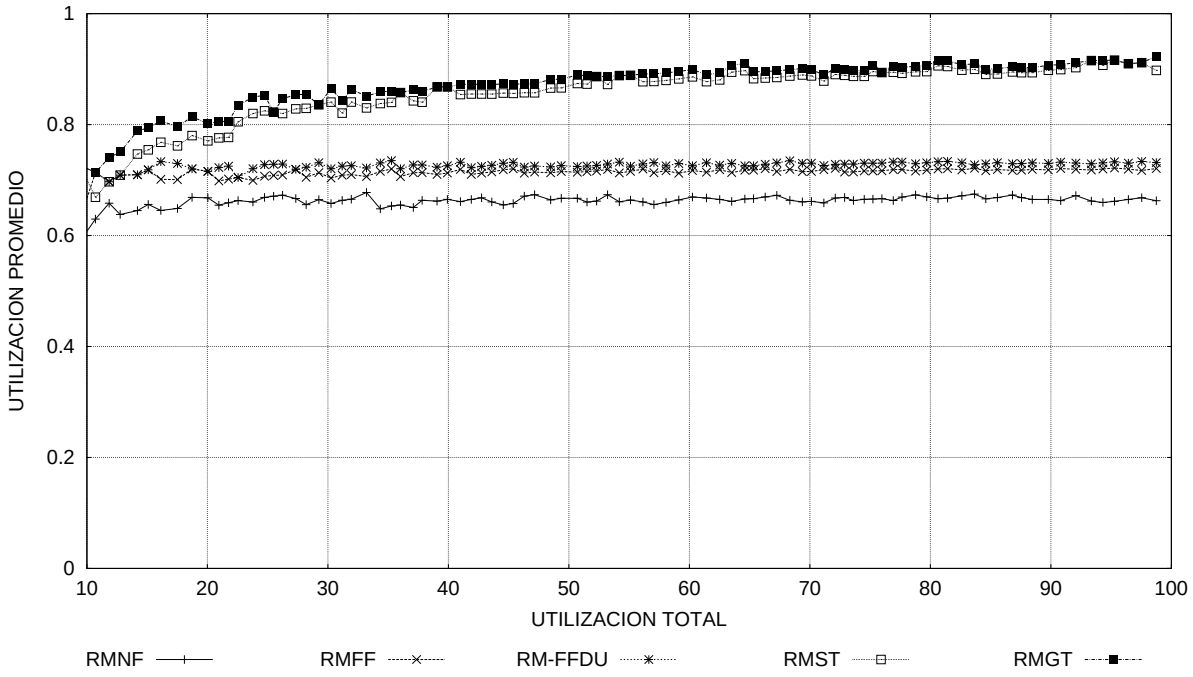

Figura 5.9: Utilización Promedio por Procesador de los Algoritmos Fuera de Línea $\alpha = 0.2$

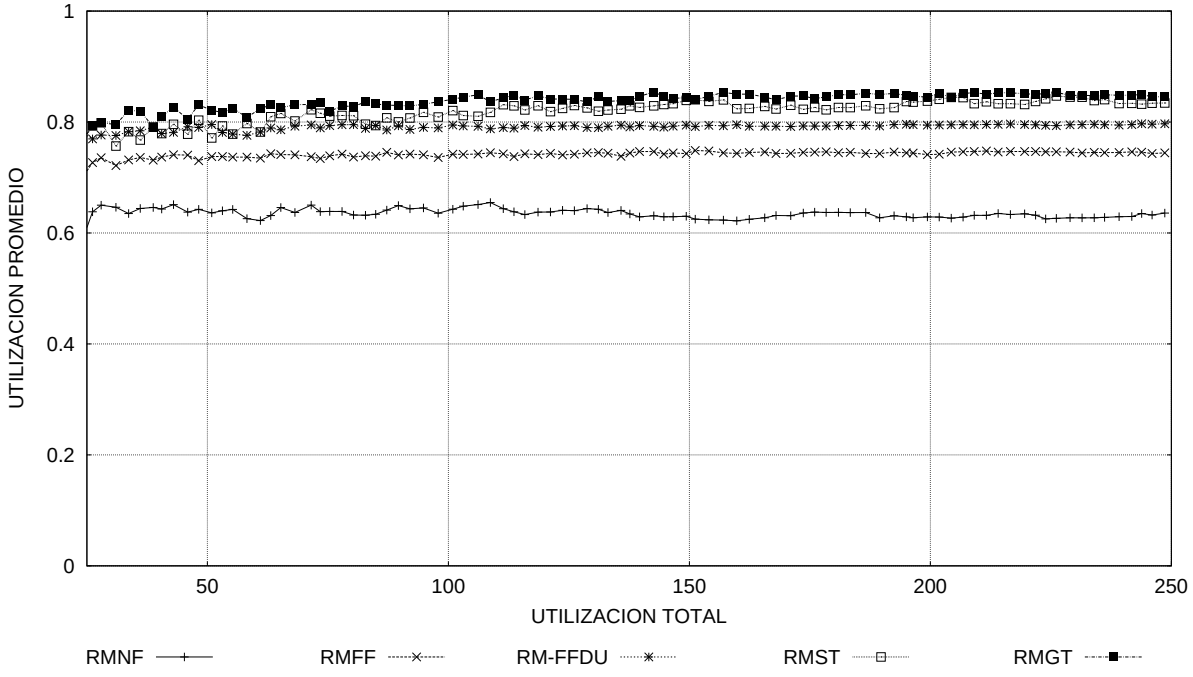

Figura 5.10: Utilización Promedio por Procesador de los Algoritmos Fuera de Línea $\alpha = 0.5$

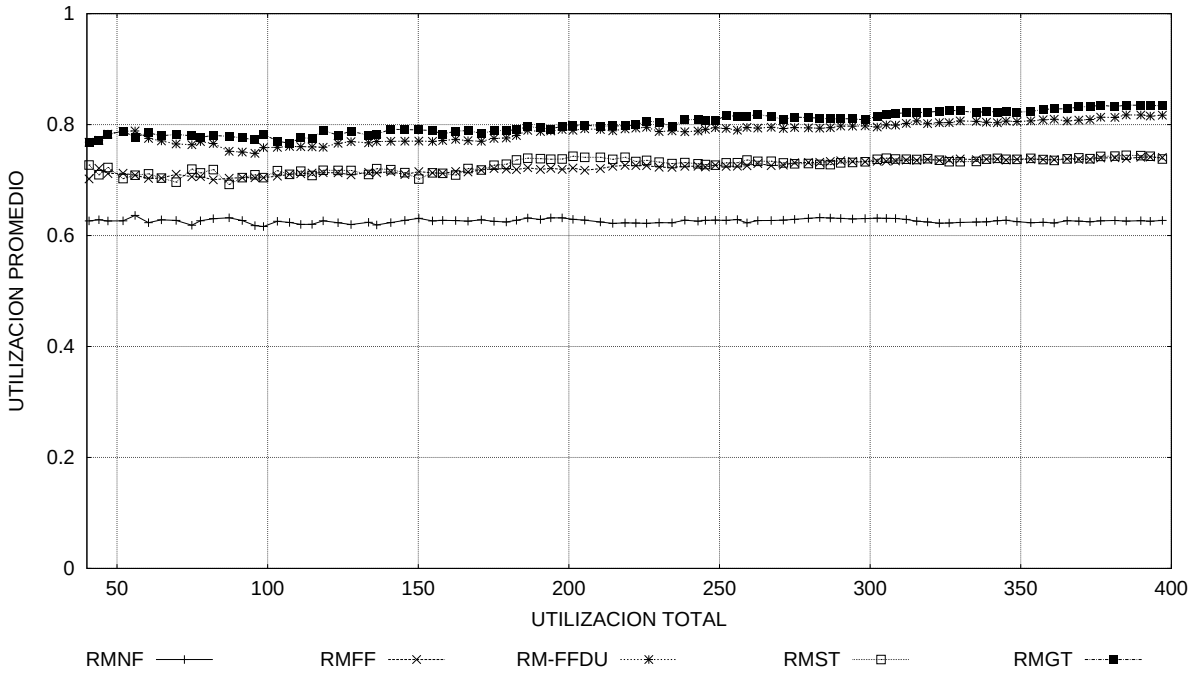

Figura 5.11: Utilización Promedio por Procesador de los Algoritmos Fuera de Línea $\alpha = 0.8$

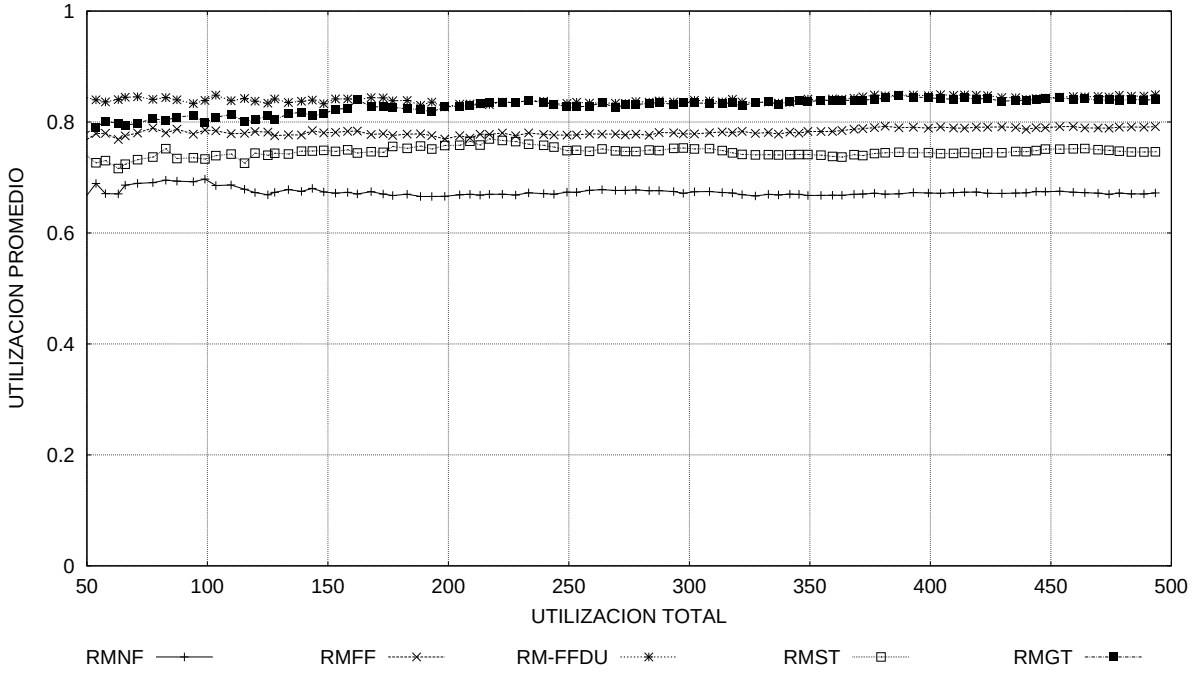

Figura 5.12: Utilización Promedio por Procesador de los Algoritmos Fuera de Línea $\alpha = 1.0$

5.3 Comparación de los Algoritmos En Línea

De la misma manera que se realizó la comparación de los algoritmos fuera de línea, se realizará ahora para los algoritmos en línea entre los cuales tenemos:

- **Rate Monotonic First Fit (RMFF-WC) [20] (Sección 4.3.2).**
- **Rate Monotonic Best Fit (RMBF-WC) [20] (Sección 4.3.2).**
- **Rate Monotonic Next Fit (RMNF-WC) [20] (Sección 4.3.2).**
- **Rate Monotonic General Tasks M (RMGT/M) [19] (Sección 4.3.1).**

Los resultados son presentados en las siguientes figuras: Figura 5.13 para $\alpha = 0.2$**, Figura 5.14 para** $\alpha = 0.5$**, Figura 5.15 para** $\alpha = 0.8$**, y Figura 5.16 para** $\alpha 1.0$**. De los experimentos se concluye que:**

- **El número de procesadores usados por la heurística RMNF tiene un crecimiento proporcional al facto de carga del sistema.**
- **A pesar de las modificaciones en el esquema de asignación de tareas, el algoritmo RMBF se comporta de forma muy similar al algoritmo RMFF para los diversos factores de carga de las tareas.**
- **Para factores de carga pequeños, el algoritmo RMGT/M tiene un comportamiento muy bueno.**
- **Cuando** $\alpha \approx 0.5$**, los algoritmos RMFF, RMBF y RMGT/M presentan rendimientos similares.**
- **El algoritmo RMGT/M presenta un rendimiento peor al algoritmo RMFF cuando** $\alpha > 0.5$**.**

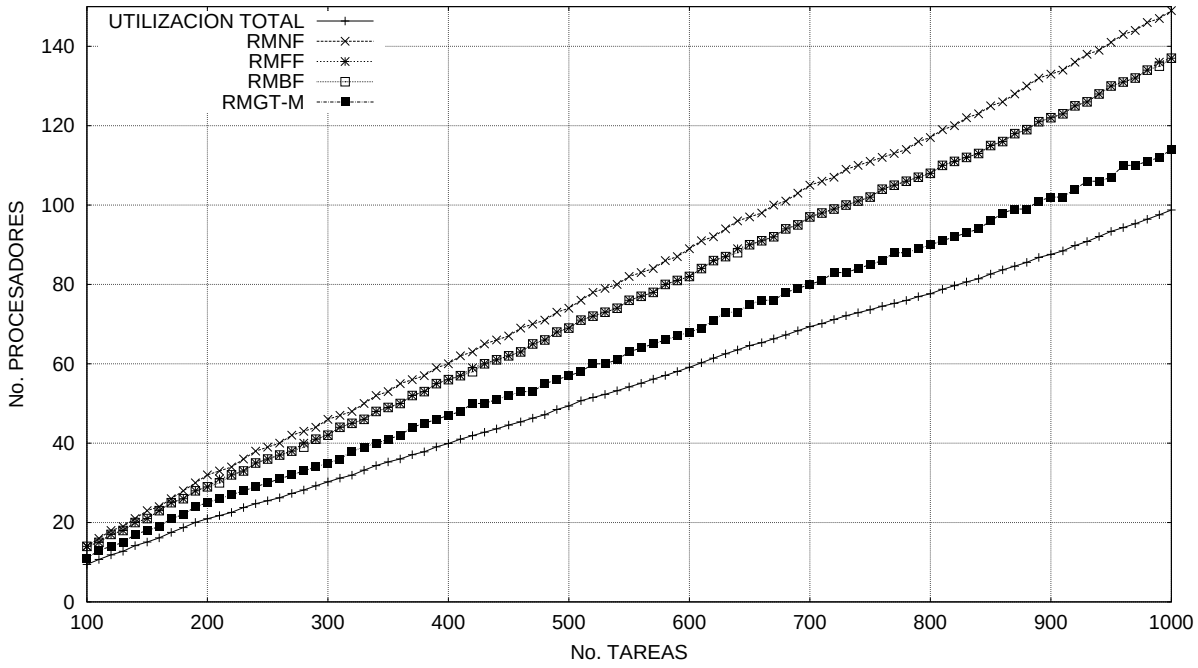

Figura 5.13: Rendimientos de Algoritmos en Línea $\alpha = 0.2$

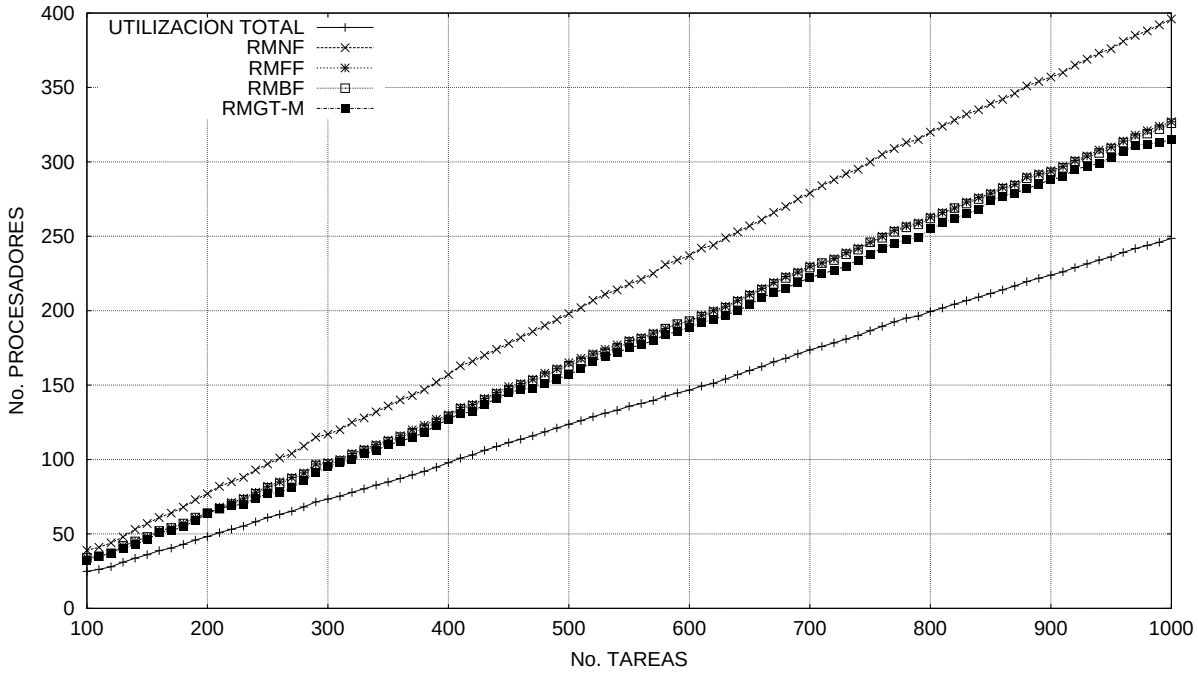

Figura 5.14: Rendimientos de Algoritmos en Línea $\alpha = 0.5$

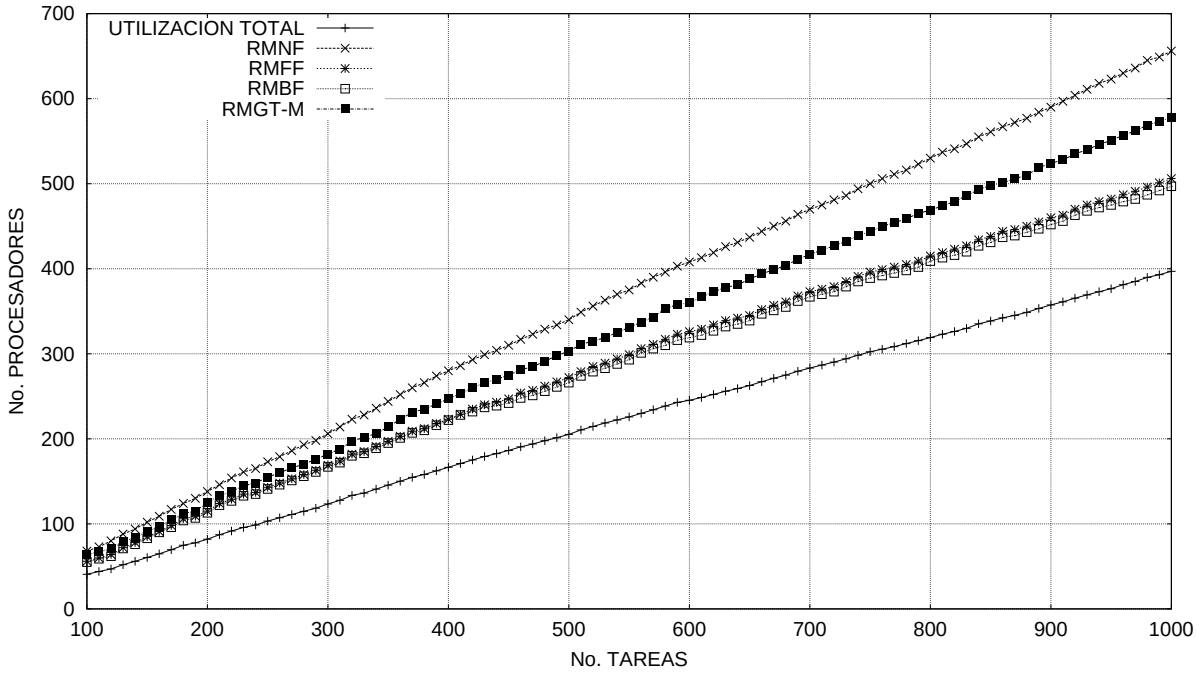

Figura 5.15: Rendimientos de Algoritmos en Línea $\alpha = 0.8$

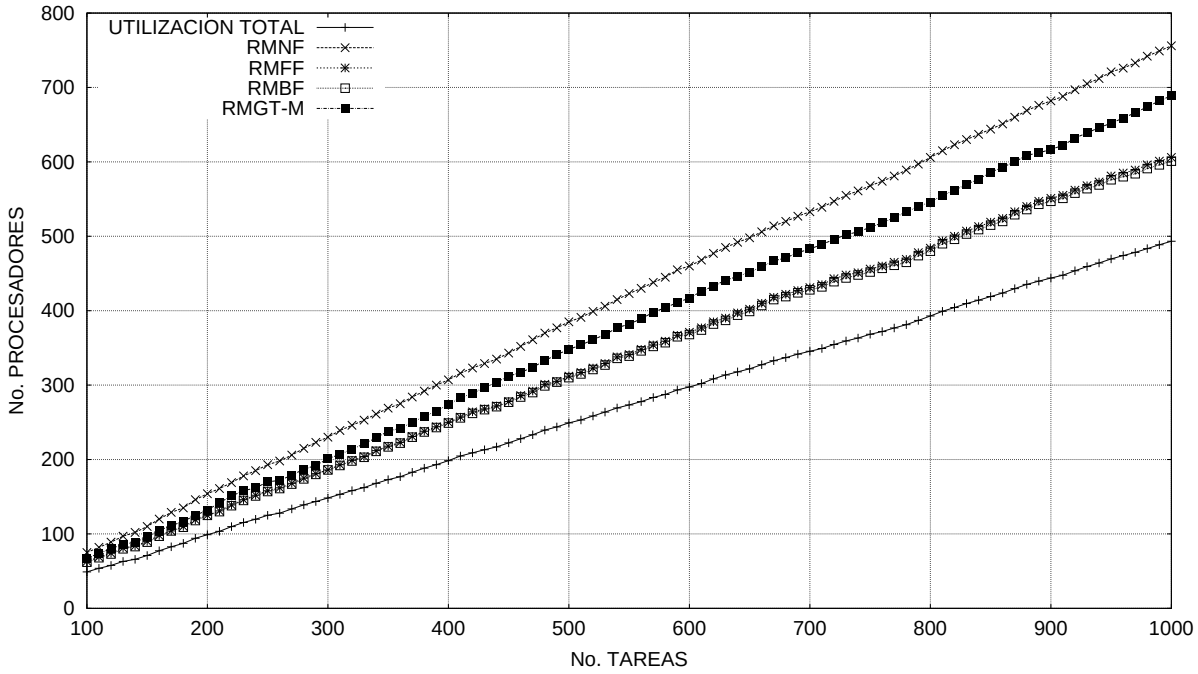

Figura 5.16: Rendimientos de Algoritmos en Línea $\alpha = 1.0$

De manera similar al realizado en la sección anterior, entre otros resultados se presentan en las figuras 5.17, 5.18, 5.19 y 5.20 los porcentajes de procesadores extras que los algoritmos requieren, así como también en las figuras 5.21, 5.22, 5.23 y 5.24 las utilizaciones promedio por procesador de los algoritmos. De lo anterior se concluye que:

- **Excepto por RMNF los porcentajes de procesadores extras de los demás algoritmos es de** $\approx 40\%$.
- **Cuando el factor de carga** $\alpha \approx 0.8$, **los algoritmos presentan los peores rendimientos en los porcentajes de procesadores extras usados.**
- **El porcentaje de procesadores extra para RMGT/M cuando** $\alpha < 0.3$ **es bastante aceptable** $\approx 20\%$.
- **Con un factor de carga de** $\alpha \leq 0.6$ **el porcentaje de procesadores extra del algoritmo RMGT/M mejora a los procentajes de los algoritmos RMBF y RMFF.**
- **Las utilizaciones promedio por procesador de los algoritmos RMFF y RMBF cuando** $\alpha > 0.5$ **son bastante aceptables** $\approx 80\%$.
- **El mayor porcentaje de utilización por procesador lo logra el algoritmo RMGT/M** $\approx 92\%$ **cuando el factor de carga del conjunto de tareas es pequeño** $\alpha < 0.3$.
- **El algoritmo RMNF mantiene una utilización promedio por procesador casi constante para cualquier factor de carga.**
- **De entre todos los algoritmos y para cualquier factor de carga del conjunto de tareas, el peor resultado lo presenta el algoritmo RMNF.**

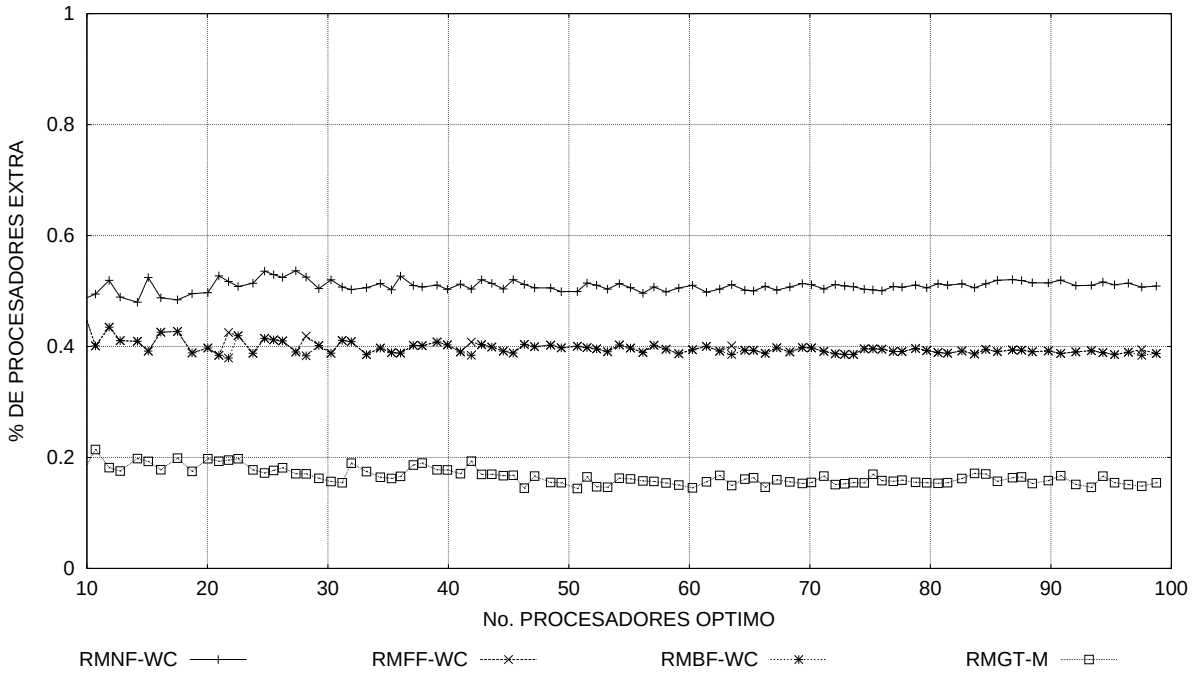

Figura 5.17: Porcentajes de procesadores extras de los Algoritmos en Línea $\alpha = 0.2$

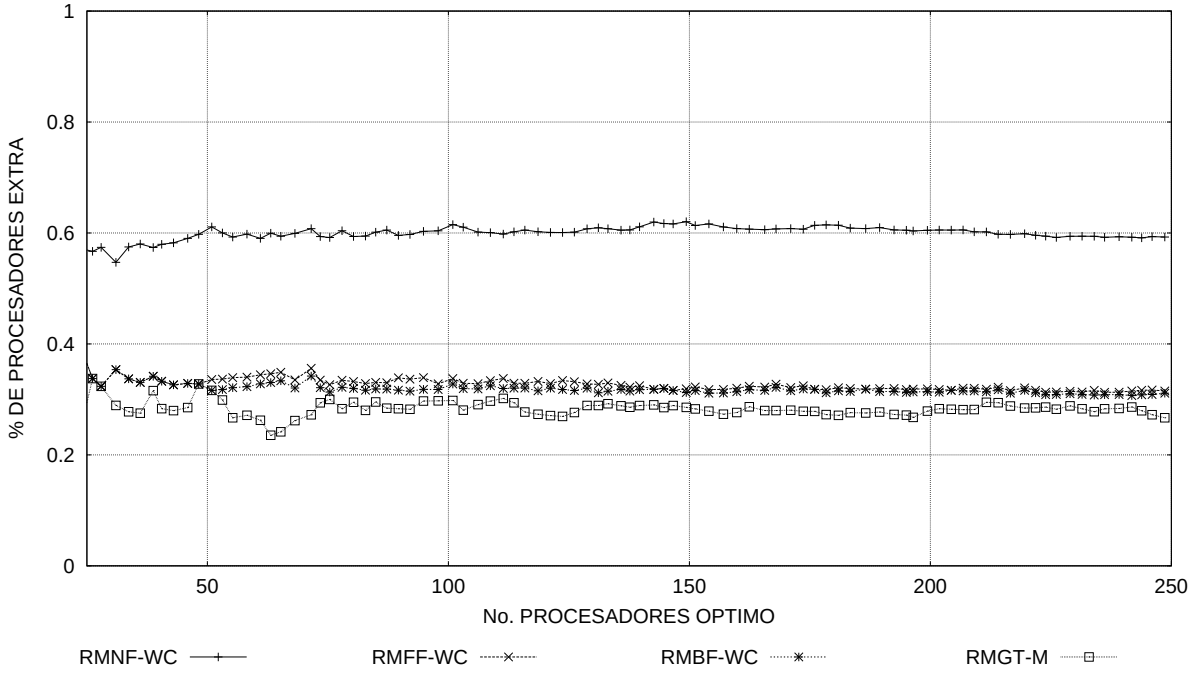

Figura 5.18: Porcentajes de procesadores extras de los Algoritmos en Línea $\alpha = 0.5$

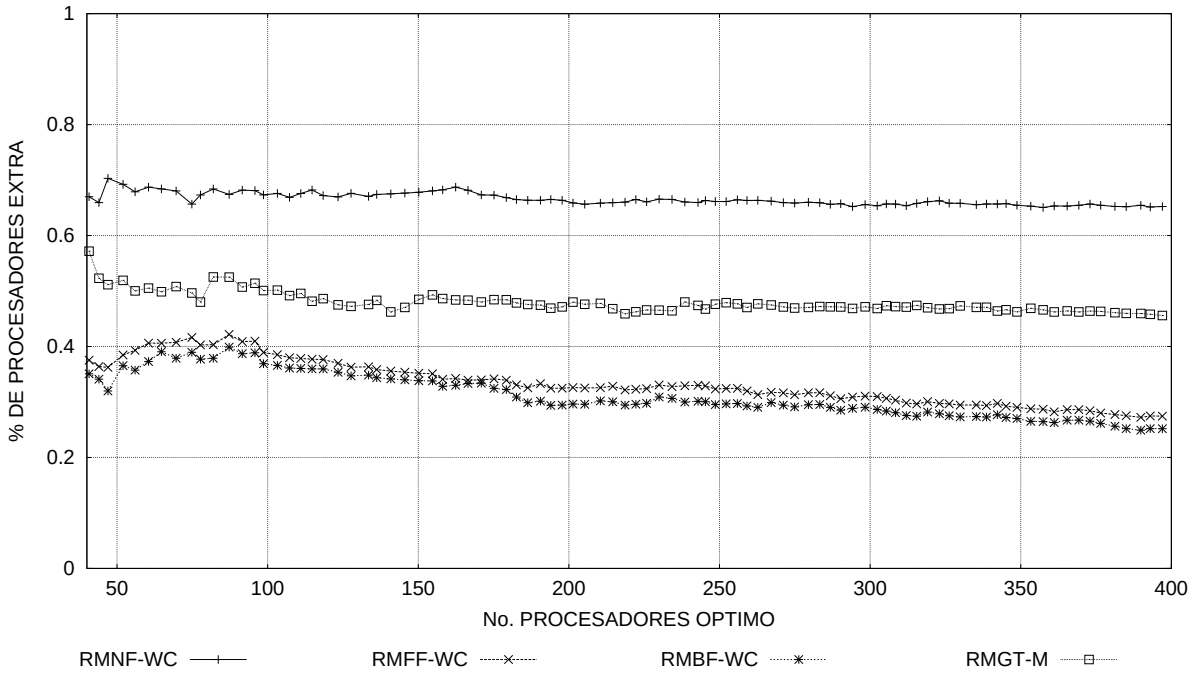

Figura 5.19: Porcentajes de procesadores extras de los Algoritmos en Línea $\alpha = 0.8$

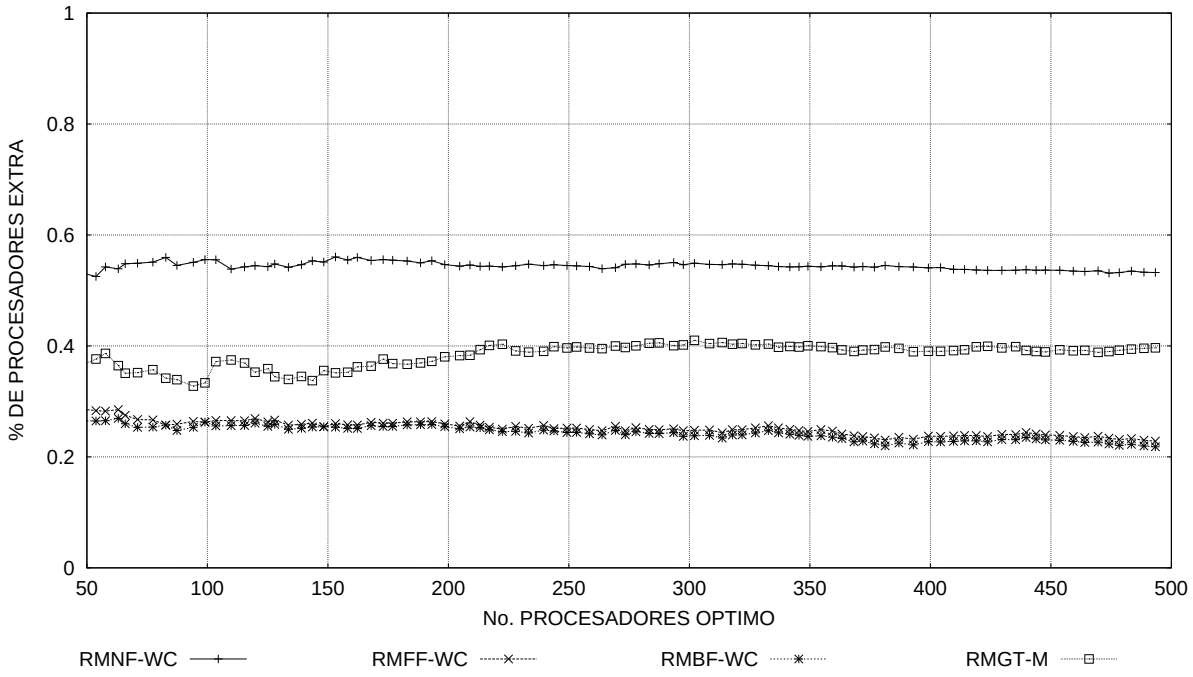

Figura 5.20: Porcentajes de procesadores extras de los Algoritmos en Línea $\alpha = 1.0$

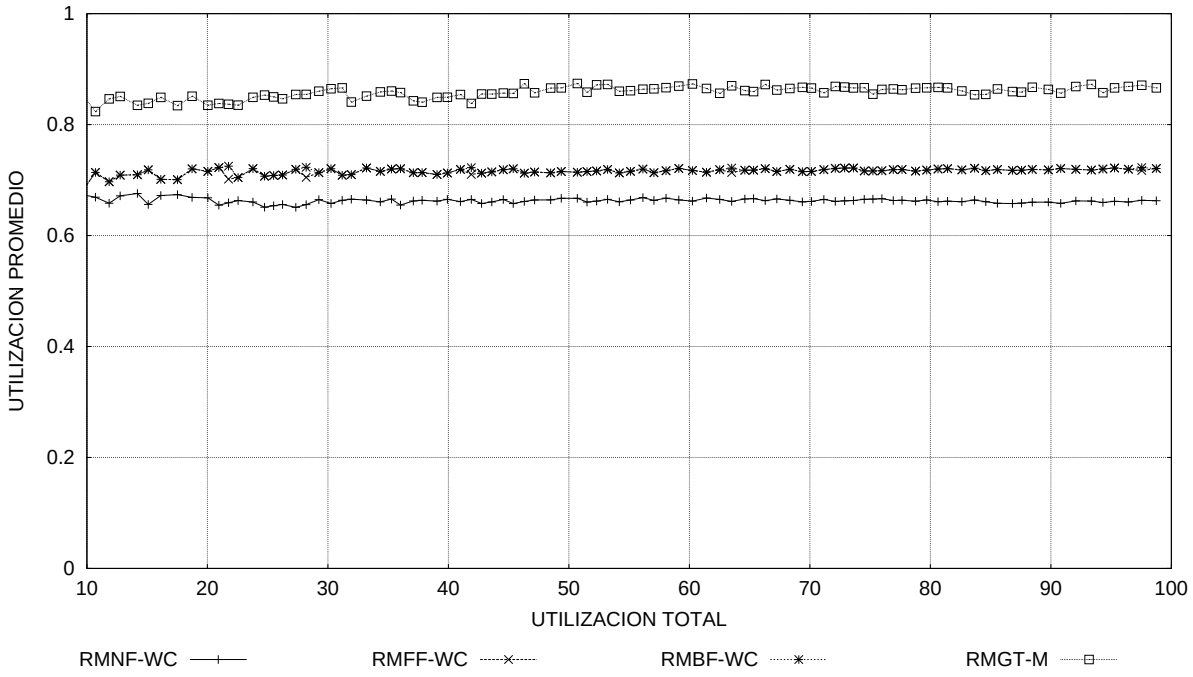

Figura 5.21: Utilización Promedio por Procesador de los Algoritmos en Línea $\alpha = 0.2$

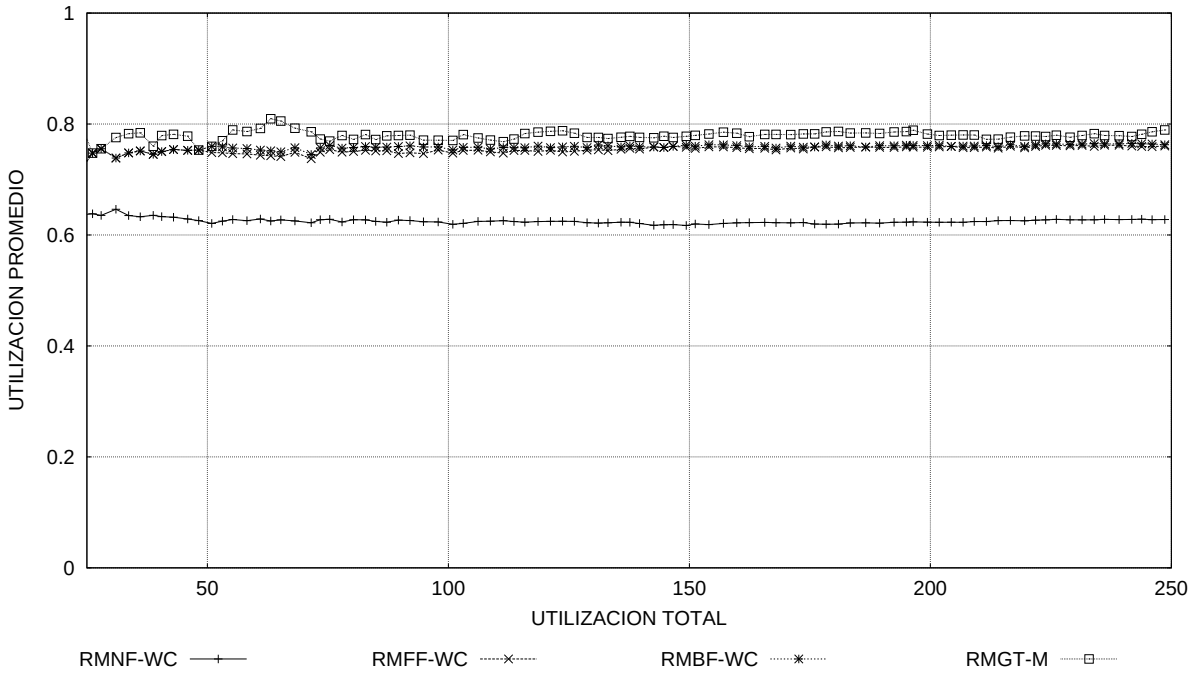

Figura 5.22: Utilización Promedio por Procesador de los Algoritmos en Línea $\alpha = 0.5$

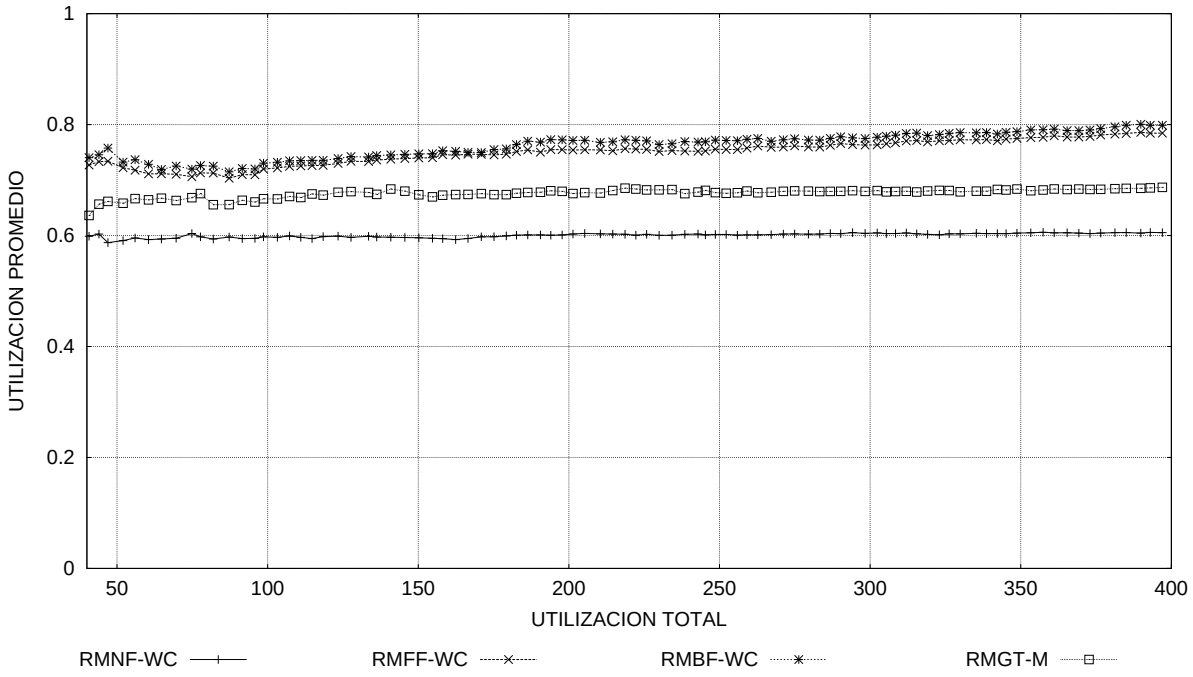

Figura 5.23: Utilización Promedio por Procesador de los Algoritmos en Línea $\alpha = 0.8$

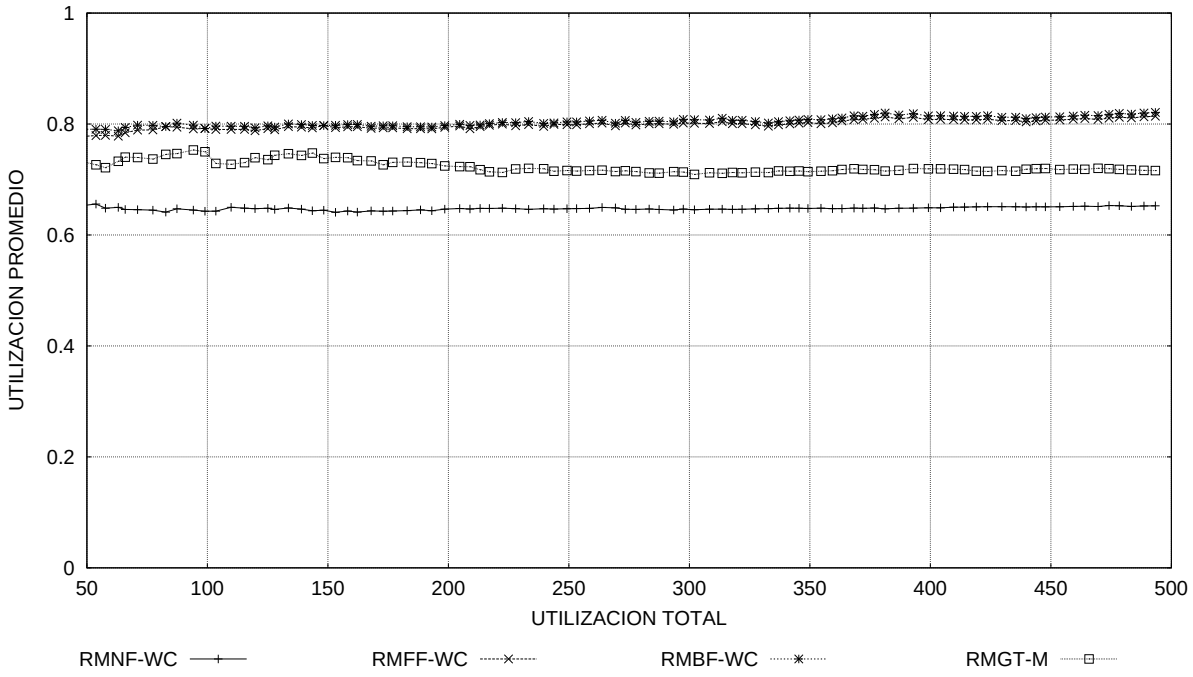

Figura 5.24: Utilización Promedio por Procesador de los Algoritmos en Línea $\alpha = 1.0$

5.4 Comparación de los Algoritmos Redefinidos

5.4.1 Algoritmos Fuera de Línea Modificados

Aplicando el mismo criterio de comparación descrito en las secciones anteriores, en esta sección se analizarán los casos promedios de los siguientes algoritmos fuera de línea:

- **RMST [19] (Sección 4.2.5).**
- **RMSTMod (Sección 4.4.1).**
- **RMGT [19] (Sección 4.2.6).**
- **RMGTMod (Sección 4.4.2).**

Los resultados son presentados en las figuras 5.25 y 5.26 para el algoritmo RMSTMod, mientras que en las figuras 5.27 y 5.28 se presentan los resultados para el algoritmo de planificación RMGTMod. De los experimentos se concluye que:

- **Para cualquier factor de carga, aunque RMSTMod mejora a RMST, los rendimientos en los casos promedios tienden a presentar comportamientos similares.**
- **Cuando** $\alpha \approx 0.3$ **los rendimientos de todos los algoritmos son similares.**
- **Cuando** $\alpha > 0.5$ **existe una mejora notable de RMGTMod sobre RMGT.**
- **Los mejores desempeños de los algoritmos modificados lo presentan cuando** $\alpha \approx 0.8$**.**
- **El algoritmo RMSTMod mejora al algoritmo de planificación RMST en** $\approx 4\%$ **cuando** $\alpha \approx 0.5$ **y en** $\approx 7\%$ **cuando** $\alpha \geq 0.8$**.**

- **El algoritmo RMGTMod mejora en rendimiento al algoritmo de planificación RMGT en $\approx$ 4% cuando $\alpha \approx 0.5$, en $\approx$ 11% cuando $\alpha \approx 0.8$, y en $\approx$ 9% cuando $\alpha \approx 1.0$.**

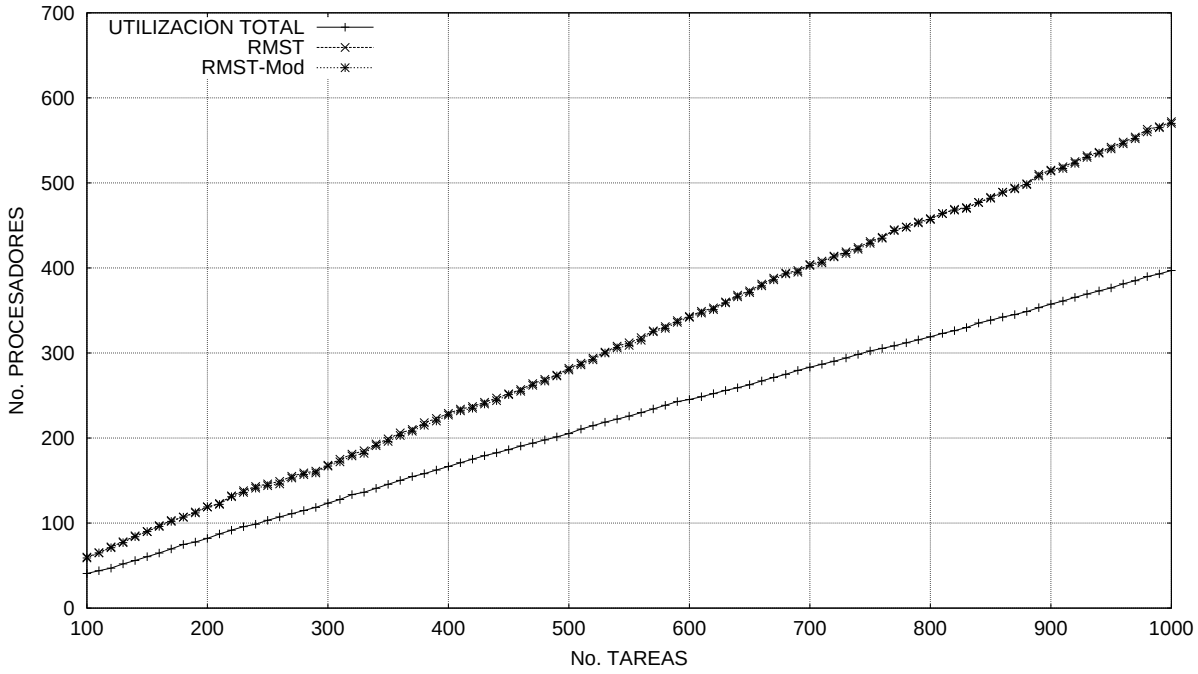

Figura 5.25: Rendimiento del Algoritmo de Planificación RMSTMod $\alpha = 0.8$

5.4.2 Algoritmos en Línea Modificados

Los algoritmos en línea modificados a comparar siguiendo un esquema similar a lo anterior son los siguientes:

- **RMFF [6] (Sección 4.2.2).**
- **RMFFMod (Sección 4.5.1).**
- **RMBF [22] (Sección 4.2.3).**
- **RMBFMod (Sección 4.5.2).**

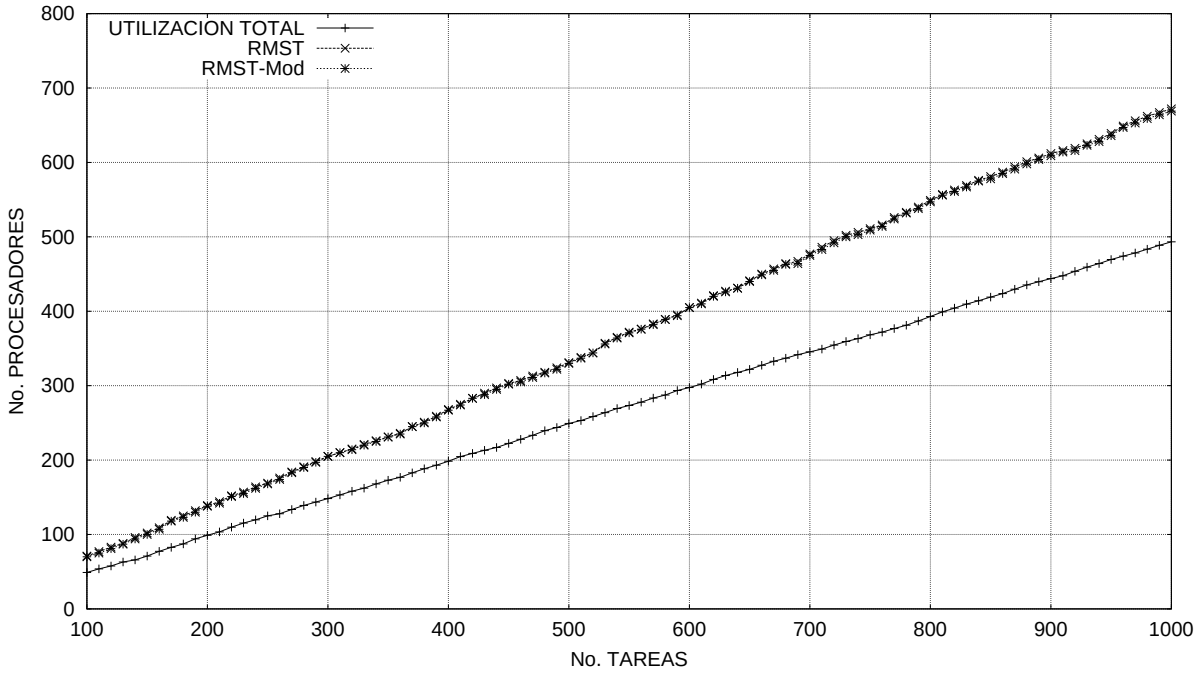

Figura 5.26: Rendimiento del Algoritmo de Planificación RMSTMod $\alpha = 1.0$

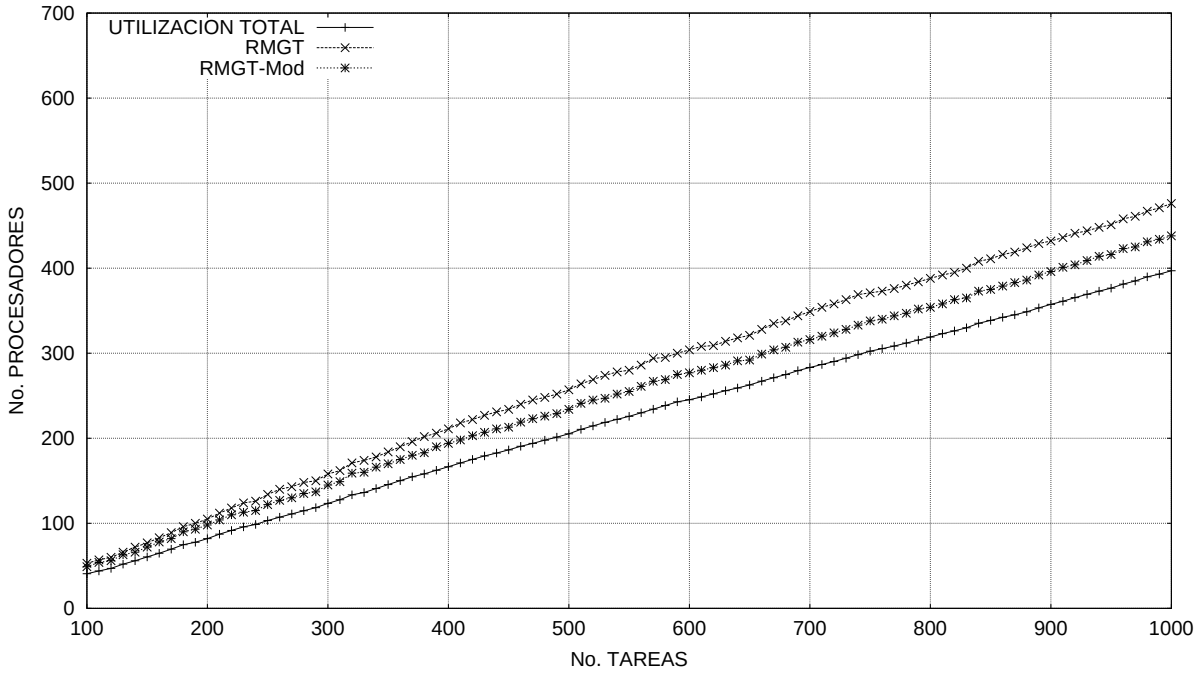

Figura 5.27: Rendimiento del Algoritmo de Planificación RMGTMod $\alpha = 0.8$

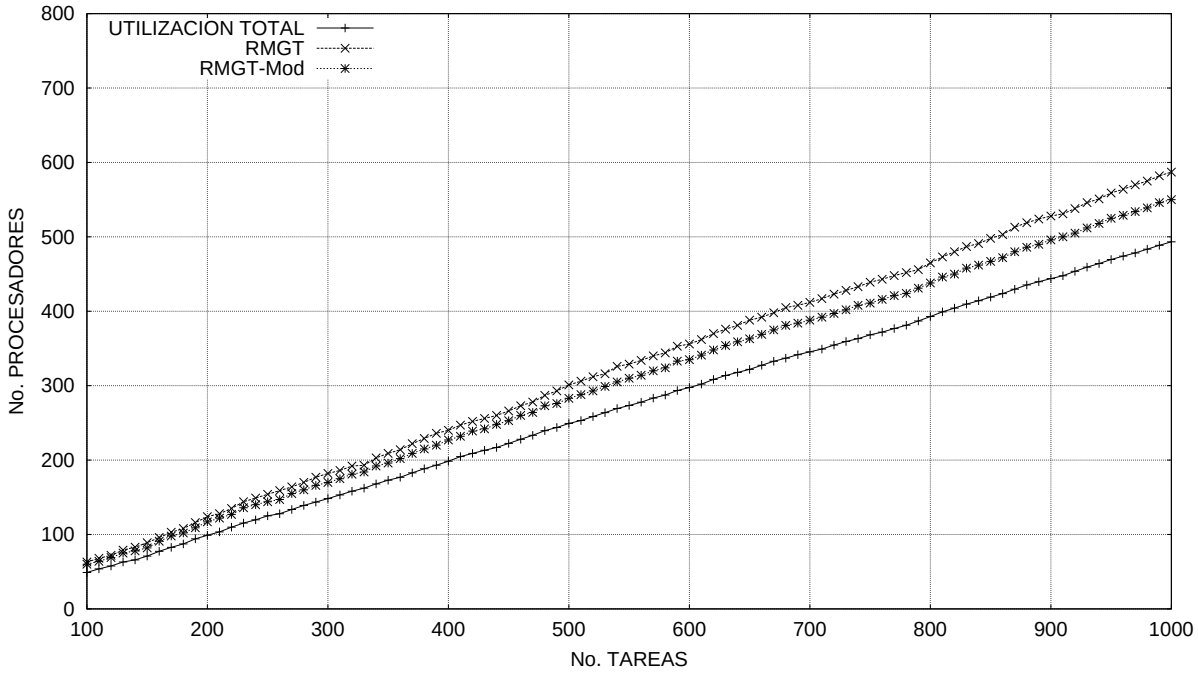

Figura 5.28: Rendimiento del Algoritmo de Planificación RMGTMod $\alpha = 1.0$

Los resultados son presentados en las figuras 5.29 y 5.30 para el algoritmo RMFFMod, y en las figuras 5.31 y 5.32 para el algoritmo RMBFMod. De los experimentos se concluye que:

- **Los algoritmos RMFF y RMBF tienen comportamientos similares. Estos comportamientos también lo presentan los algoritmos modificados RMFF-Mod y RMBFMod.**
- **Los algoritmos modificados tienen mejores comportamientos para conjuntos de tareas donde $\alpha > 1/3$.**
- **Cuando $\alpha \leq 1/3$ RMFFMod se comporta igual que RMFF. Lo mismo ocurre para RMBFMod y RMBF.**
- **Los mejores desempeños de los algoritmos modificados lo presentan cuando $\alpha \approx 0.8$.**

- **El algoritmo RMFFMod mejora en rendimiento al algoritmo de planificación RMFF en $\approx$ 10% cuando $\alpha \approx 0.5$, en $\approx$ 20% cuando $\alpha \approx 0.8$, y en $\approx$ 11% cuando $\alpha \approx 1.0$.**

- **El algoritmo de planificación RMBFMod mejora al algoritmo RMBF en $\approx$ 9% cuando $\alpha \approx 0.5$, en $\approx$ 22% cuando $\alpha \approx 0.8$, y en $\approx$ 10% cuando $\alpha \approx 1.0$.**

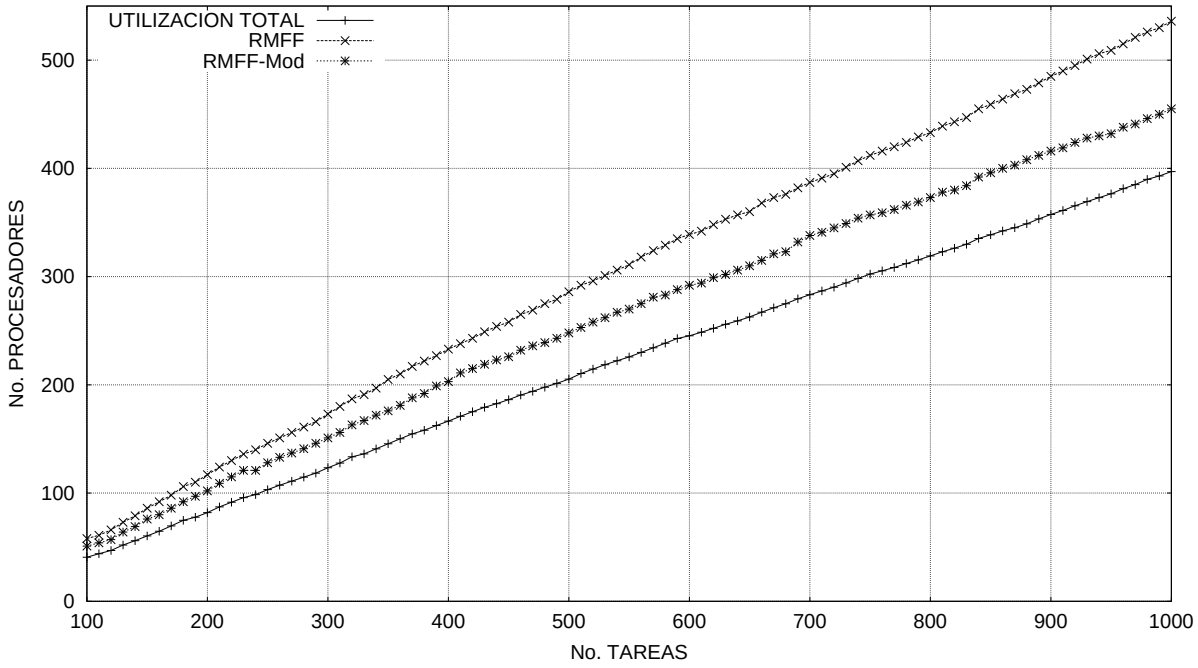

Figura 5.29: Rendimiento del Algoritmo de Planificación RMFFMod $\alpha = 0.8$

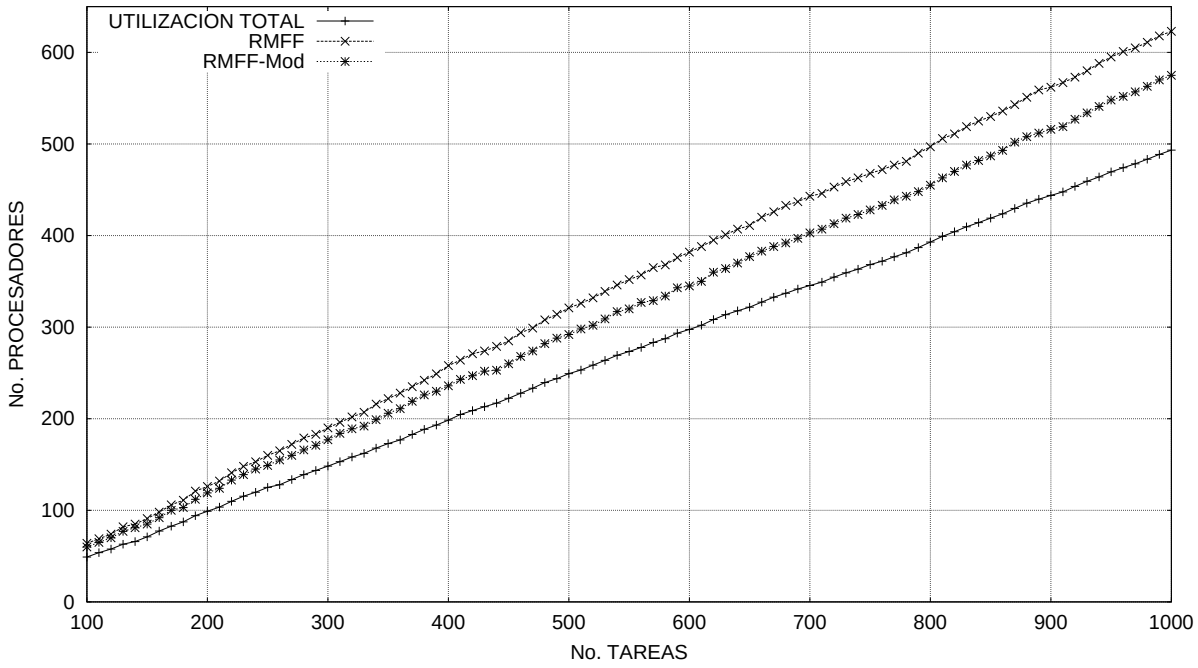

Figura 5.30: Rendimiento del Algoritmo de Planificación RMFFMod $\alpha = 1.0$

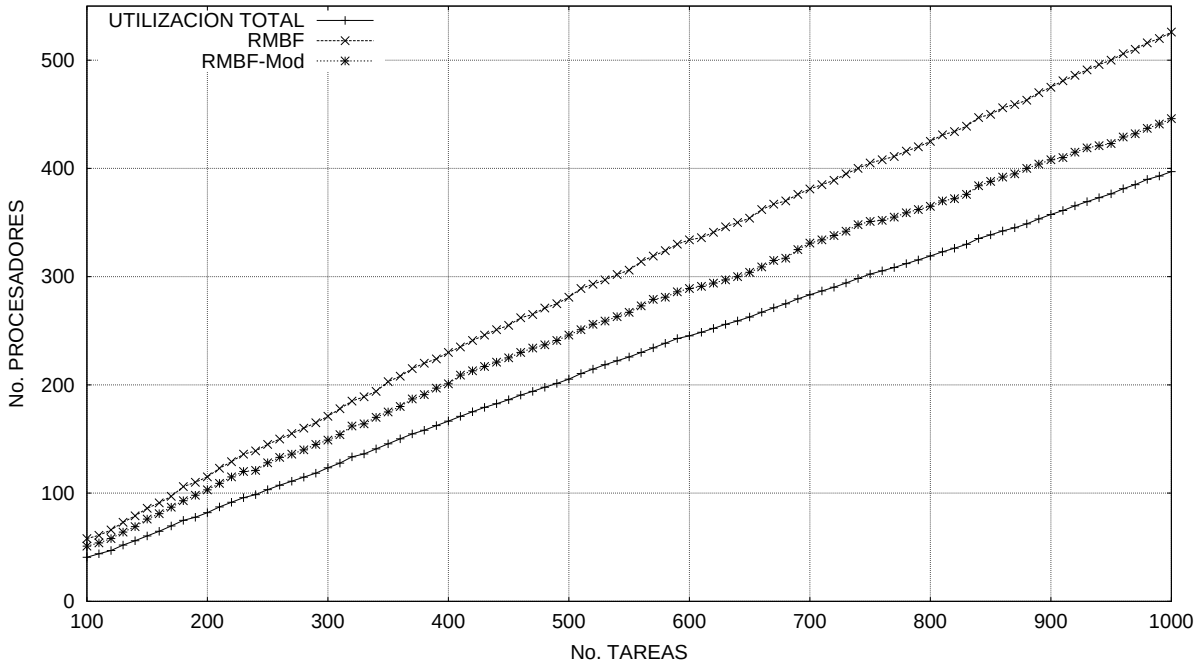

Figura 5.31: Rendimiento del Algoritmo de Planificación RMBFMod $\alpha = 0.8$

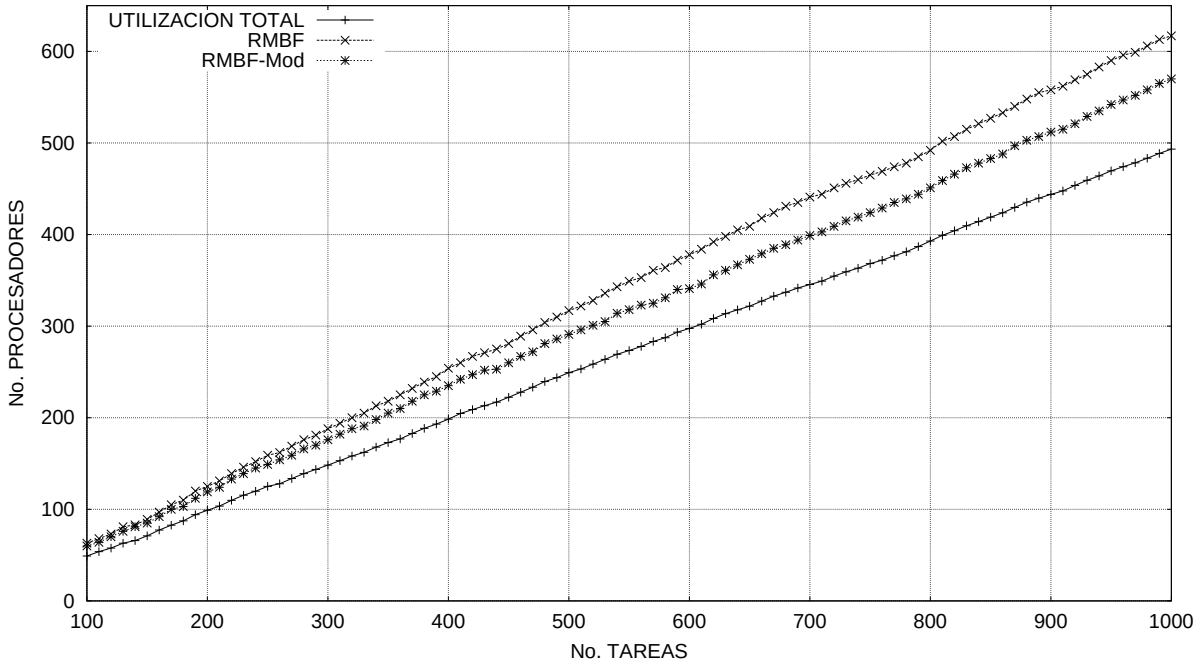

Figura 5.32: Rendimiento del Algoritmo de Planificación RMBFMod $\alpha = 1.0$

Capítulo 6

Herramienta de Simulación

En el capítulo anterior, se expusieron mediante experimentos de simulación, los rendimientos de los algoritmos de planificación considerando sus casos promedios. En este capítulo se presenta la descripción de la herramienta de simulación desarrollada para llevar a cabo los experimentos. En esta herramienta de simulación se presenta una interfaz gráfica amigable para el usuario en donde se comparan los rendimientos de los algoritmos bajo condiciones similares.

6.1 Descripción de la Herramienta de Simulación

La herramienta de simulación que se desarrolló, está compuesta básicamente de dos módulos: Un módulo encargado de la generación de las tareas y un módulo encargado de la ejecución de los algoritmos de planificación.

La interfaz gráfica de la herramienta se presenta en la figura 6.1, en la que resaltan las siguientes opciones:

1. Entrada del nombre del archivo para la generación de las tareas.
2. Selección del factor máximo de carga del conjunto de tareas a generar.
3. Comando para iniciar la generación de las tareas.

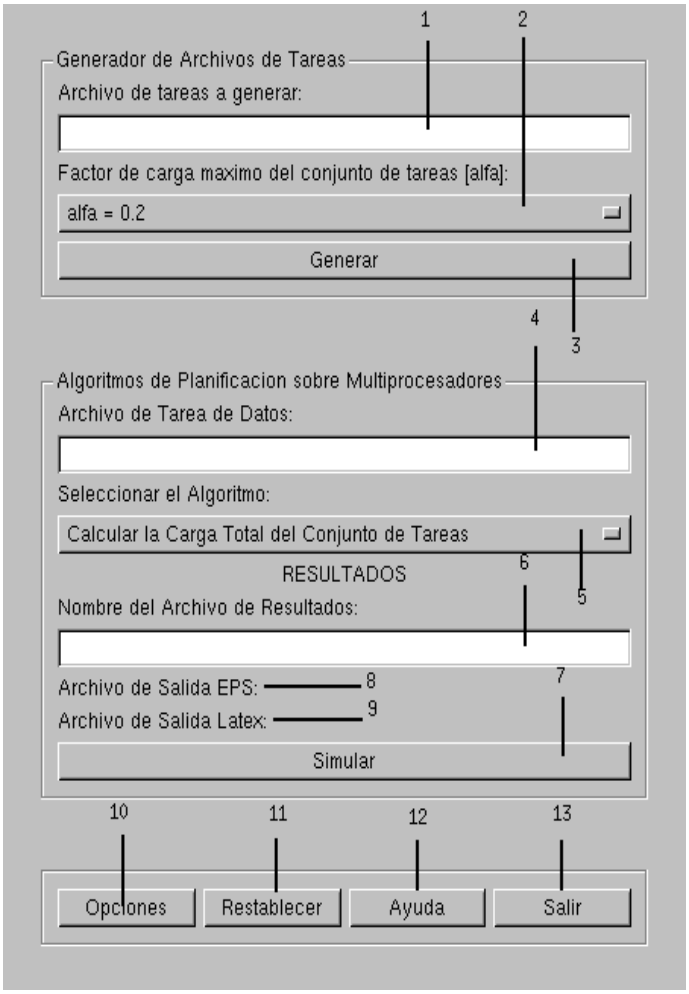

Figura 6.1: Interfaz Gráfica de la Herramienta de Simulación

4. Entrada para especificar que archivo de tareas se utilizará para ejecutar un algoritmo.

5. Selección de los algoritmos de planificación.

6. Entrada del nombre del archivo donde se almacenarán los resultados.

7. Comando para iniciar la simulación.

8. Etiqueta que especifica el nombre del archivo de resultados (gráfica) en formato EPS.

9. Etiqueta que especifica el nombre del archivo de resultado en formato LaTeX.

10. Comando que despliega las opciones de la herramienta.

11. Comando que reestablece los valores predeterminados de la herramienta de simulación.

12. Nos presenta la ayuda del sistema.

13. Comando que termina la ejecución de la herramienta de simulación.

6.1.1 Generación de los Archivos de Tareas

Para generar un conjunto de tareas, simplemente se debe especificar el archivo destino (referencia 1, figura 6.1), determinar el factor de carga máximo permitido para una tarea dentro del conjunto (referencia 2, figura 6.1), y ejecutar el comando Generar (referencia 3, figura 6.1). Es importante mencionar (como se presentará en la sección 6.1.4) que existe un número fijo que indica el número de veces que se ejecutará algún algoritmo. Este número también afecta el número de archivos de tareas que se generarán. Es decir, si se establece que se realizarán 100 experimentos por algoritmo (100 es el valor por defecto), entonces se generarán 100 archivos de datos de tareas. Por supuesto, no es necesario especificar 100 nombres de archivos destinos, los nombres de los archivos se crearán de manera interna incrementando un número índice que lleva un contador de archivos generados. Un nombre de archivo de tareas se forma de la siguiente manera:

(nombre proporcionado + [02 — 05 — 08 — 10] + No. de archivo generado)

Así, "tareas0289" nos índica que el usuario determinó como nombre de archivo de salida "tareas", que seleccionó un factor máximo de carga de 0.2 y, que es el archivo número 89 generado.

Cada archivo de tareas generado posee la siguiente estructura: Identificador

No. Tarea	Periodo	Cómputo	Utilización
0	478	89.00	0.186192
1	139	24.00	0.172662
2	425	26.00	0.061176
3	103	3.00	0.029126
4	451	17.00	0.037694

Tabla 6.1: Ejemplo de 5 Tareas Generadas

de tarea, Periodo, Cómputo y Utilización. En la tabla 6.1 se presenta un ejemplo de un subconjunto de las primeras 5 tareas generadas.

En caso de no especificar un nombre para los archivos de tareas, se hará del conocimiento al usuario a través de mensajes de error (figura 6.2).

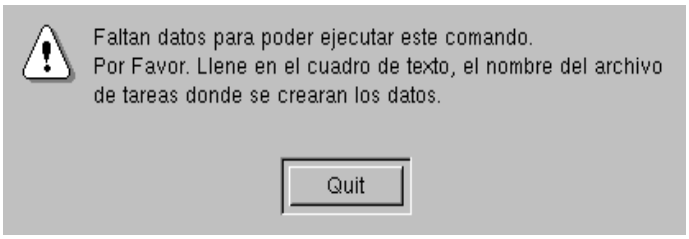

Figura 6.2: Error de Datos Incompletos

6.1.2 Simulación de los Algoritmos de Planificación

Una vez que se generaron archivos de tareas, se puede proceder a la simulación de los algoritmos de planificación. Para simular un algoritmo determinado bastará simplemente con realizar los siguientes cuatro pasos: (1) Se deberá introducir el nombre del archivo donde se encuentra el conjunto de tareas previamente generado (referencia 4, figura 6.1). (2) Se deberá seleccionar el algoritmo de planificación tal como se presenta en la figura 6.3 (referencia 5, figura 6.1). (3)

Se procede a especificar el nombre del archivo destino (referencia 6, figura 6.1) y, (4) Se deberá ejecutar el comando Simular (referencia 7, figura 6.1).

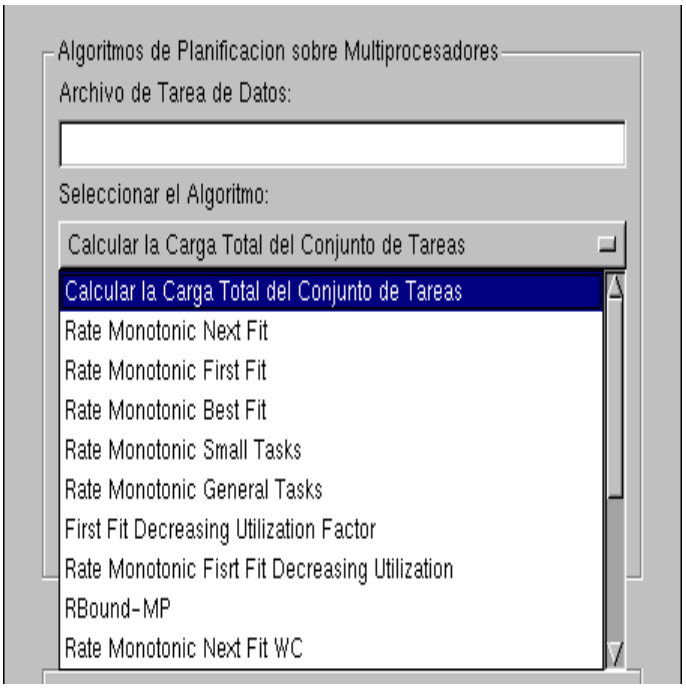

Figura 6.3: Selección de un Algoritmo de Planificación

Los algoritmos de planificación disponibles para la simulación son los siguientes: Rate Monotonic First Fit, Rate Monotonic First Fit WC, Rate Monotonic Best Fit, Rate Monotonic Best Fit WC, Rate Monotonic Next Fit, Rate Monotonic Next Fit WC, Rate Monotonic Small Tasks, Rate Monotonic General Tasks, RBoundMP, First Fit Decreasing Utilization Factor, Rate Monotonic First Fit Decreasing Utilization, Rate Monotonic General Tasks M, Rate Monotonic Small Tasks Modificado, Rate Monotonic General Tasks Modificado, Rate Monotonic First Fit Modificado y Rate Monotonic Best Fit Modificado.

Cuando el proceso de simulación de un algoritmo se completa, se generan un número de archivos de salida dado por el número de archivos de entrada. Al mismo tiempo, se genera un solo archivo de datos de salida el cual contiene el

promedio de todos los archivos producidos.

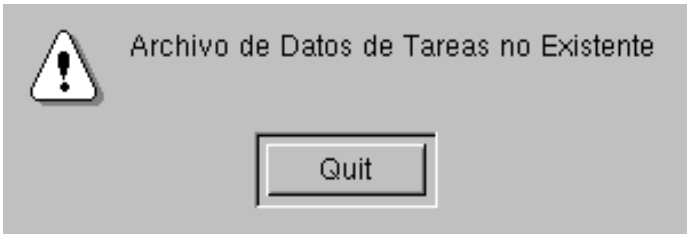

Figura 6.4: Error de Archivo Invalido

En el caso de que el usuario proporcione un nombre de archivo de tareas erróneo, se le hará saber a través de mensajes de error (figura 6.4).

6.1.3 Interpretación de Gráficas y Resultados

Al finalizar la simulación de un algoritmo en particular, se imprimirán en pantalla las gráficas correspondientes a los resultados obtenidos. Estas gráficas despliegan los datos que se encuentran en el archivo con los valores promedios de todas las ejecuciones. Las gráficas pueden ser consultadas en cualquier momento por el usuario ya que se almacenan como archivos independientes. La referencia 8 de la figura 6.1 nos indica el nombre del archivo con extensión EPS (Encapsulated Postcript) generado. La referencia 9 de la misma figura, nos presenta el nombre del archivo de texto para ser incluido en archivo que utilizen un compilador para LaTeX.

Las gráficas que se despliegan son similares a las presentadas en las figuras 6.5, 6.6 y 6.7.

La gráfica de la figura 6.5 nos presenta el número de procesadores requerido para planificar el conjunto de tareas. En el eje X nos muestra el tamaño del conjunto de tareas, mientras que en el eje Y nos presenta la cantidad de procesadores requeridos para la planificación factible de ese conjunto de tareas. Los

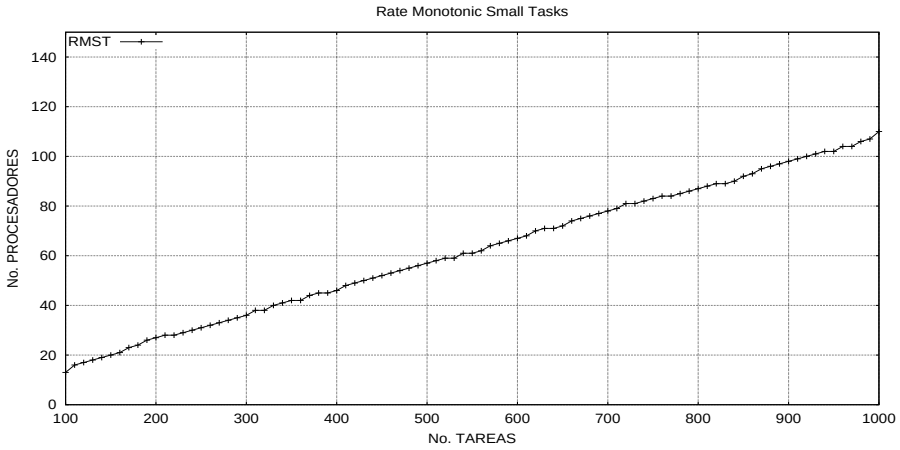

Figura 6.5: Gráfica Resultante de la Ejecución de la Herramienta de Simulación

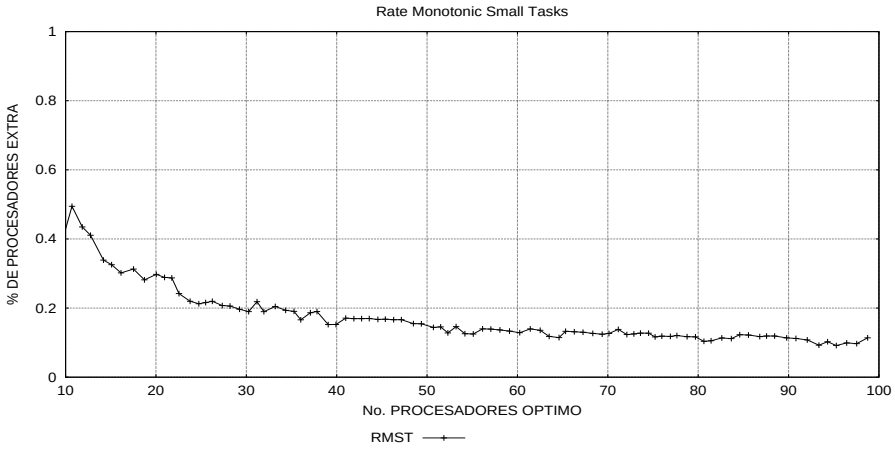

Figura 6.6: Gráfica Resultante de la Ejecución de la Herramienta de Simulación

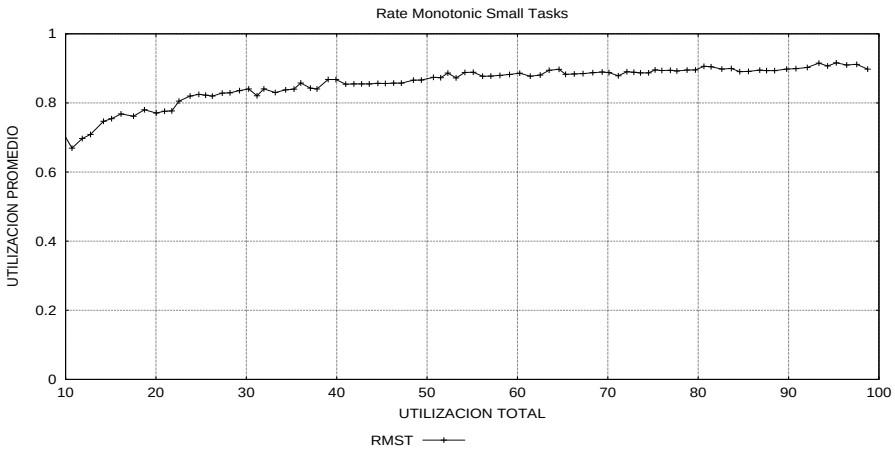

Figura 6.7: Gráfica Resultante de la Ejecución de la Herramienta de Simulación

datos mostrados en la figura 6.6 nos indican el porcentaje de procesadores extras que el algoritmo de planificación utilizó y, finalmente la figura 6.7 nos indica la utilización promedio por procesador del algoritmo de planificación.

6.1.4 Modificación de los Valores de Simulación

El usuario puede cambiar el número de repeticiones que un algoritmo puede ejecutarse, así mismo, se pueden cambiar las etiquetas que se presentan en las gráficas resultantes. Para realizar esto se tiene que ejecutar el comando Opciones presentado en la referencia 10 de la figura 6.1. De manera similar, para reestablecer los valores por defecto simplemente se deberá ejecutar el comando Reestablecer especificado en la referencia 11 de la figura 6.1.

Los comandos Ayuda y Salir (referencias 12 y 13 figura 6.1) como su nombre lo índican, presentan la ayuda del sistema así como la terminación de la herramienta de simulación.

6.2 Referencia Técnica

La herramienta de simulación presentada, fue desarrollada bajo el marco de trabajo que nos presenta la multiplataforma Qt para C++ GUI, bajo el sistema operativo Linux Red Hat 7.0. El código generado es el establecido para las versiones Qt 3.x o superior. Además para un correcto funcionamiento de la herramienta, se deberá contar con una versión de Gnuplot, así como algún visualizador de imagenes para formatos Postcript Encapsulados. Gnuplot es un programa para graficar datos, bajo un ambiente interactivo de manejo de comandos.

6.2.1 Instalación

Antes de llevar a cabo la instalación de la herramienta, primero se deberá contar con el archivo simulador.tar.gz que puede ser descargado en la siguiente dirección: http://computacion.cs.cinvestav.mx/õpereira. Para descomprimir el archivo bastará con ejecutar el comando:

tar xvfz simulador.tar.gz

Una vez descomprimido se encontrará en el mismo directorio un subdirectorio con nombre "simulador" el cual contendrá los siguientes archivos:

main.cpp
simulador.cpp
header.h
simulador.h
Makefile

El archivo main.cpp es el archivo donde se instancia un objeto de la clase sim, así como también se encuentra el código que ejecuta la ventana principal. En el archivo simulador.cpp es donde se encuentran declarados los algoritmos de planificación. En los archivos header.h y simulador.h se encuentran definidas las constantes, estructuras y los prototipos de las funciones utilizadas para la clase principal sim. El archivo Makefile contiene las banderas y la ruta de las bibliotecas necesarias para compilar los archivos fuentes. Para obtener el ejecutable simplemente invocamos el comando:

make

Con lo cual obtenemos el archivo ejecutable "simulador". Para ejecutar la aplicación bastará con ejecutar el comando "./simulador" el cual nos presentará la ventana principal.

6.2.2 Estructura de los Archivos de la Herramienta de Simulación

Debido a la cantidad de archivos que se crean tanto al generar los archivos de tareas como al ejecutar los algoritmos de planificación, es necesario que exista una organización de los archivos en directorios para una mejor administración de éstos.

Cuando se ejecuta la herramienta de simulación, se crean de manera interna un conjunto de subdirectorios. La lista y la descripción de estos subdirectorios se presentan en las tablas 6.2 y 6.3.

Subdirectorio	Descripción
carga	Se encuentran los archivos resultantes después de ejecutar el algoritmo que calcula la carga total del conjunto de tareas.
ffduf	Se encuentran los archivos resultantes después de ejecutar el algoritmo First Fit Decreasing Utilization Factor.
generador	Se encuentran los archivos de tareas generados.
graficas	Se encuentran los archivos con los resultados gráficos de los algoritmos.
latex	Se encuentran los archivos .tex
promedio	Se encuentran los archivos con los resultados promedios de le ejecución de los algoritmos de planificación.
rboundmp	Se encuentran los archivos resultantes después de ejecutar el algoritmo RBound.
rmbf	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic Best Fit.
rmbfr	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic Best Fit Modificado.
rmbfwc	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic Best Fit WC.
rmff	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic First Fit.
rmffdu	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic First Fit Decreasing Utilization.
rmffr	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic First Fit Modificado.

Tabla 6.2: Estructura de los Archivos de la Herramienta de Simulación (Parte I)

rmffwc	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic First Fit WC.
rmgt	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic General Tasks.
rmgtm	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic General Tasks M.
rmgtr	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic General Tasks Modificado.
rmnf	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic Next Fit.
rmnfwc	Se encuentran los archivos resultantes después de ejecutar el algoritmo Rate Monotonic Next Fit WC.

Tabla 6.3: Estructura de los Archivos de la Herramienta de Simulación (Parte II)

6.2.3 Modificación de la Herramienta de Simulación

En los archivos fuentes que componen a la herramienta de simulación, existen variables generales y funciones que regulan el comportamiento del sistema. El incluir nuevos algoritmos de planificación, modificar las funciones generales, así como generar distintos conjuntos de tareas, puede llevarse a cabo modificando los siguientes valores y compilando el proyecto como se definió en la sección 6.2.1.

- **Para la generación de tareas, el rango de los periodos se encuentra establecido por las variables Tmin= 1 y Tmax = 500 (declaradas en el archivo header.h).**
- **Los periodos de las tareas se obtienen usando una función de distribución uniforme con Tmin$\leq T_i \leq$Tmax.**

La función de distribución uniforme utilizada es la siguiente:

```
int uniform_func(int min, int max)
{
  int num, random, t1, t2;
  double w;
  struct timeval tv1;
  if ((max - min) == 0) random = 1;
  else{
       t1=gettimeofday(&tv1,NULL);
       t2 = tv1.tv_usec;
       srand(((int) w) + t2);
       random = rand() % (max - min);
      }
  random = random + min;
  return(random);
}
```

La cual puede ser modificada para generar un conjunto de tareas con alguna distribución distinta.

- **El factor de carga máximo (α) varía entre los valores [0.2,0.5,0.8,1.0]. De la misma manera se puede alterar la variable para valores como [0.3,0.7,etc] en el archivo header.h.**

- **Los tiempos de ejecución se encuentran definidos usando una función de distribución uniforme con $0 \leq C_i \leq \alpha T_i$. La cual puede ser modificada sencillamente.**

- **La variable NUMPROM=100 definida en header.h, es la encargada de realizar 100 experimentos por algoritmo.**

- **Nmax = 1000 (definida en header.h) es la variable encargada de generar 1000 tareas por archivo. Si sólo se requiere generar 500 tareas por archivo, bastará con modificar esta variable.**

- **Para añadir un nuevo algoritmo de planificación a la herramienta, considérese lo siguiente:**

  1. Incluir en el archivo de encabezados la declaración del nuevo algoritmo. Por ejemplo: void algoritmo(parámetros);

  2. Insertar el nombre del algoritmo en el botón de despliegue de la herramienta. Esto se realiza en el archivo simulador.cpp. Por ejemplo: InsertItem("nombre-del-nuevo-algoritmo").

  3. Declarar el código del nuevo algoritmo en el cuerpo del archivo simulador.cpp. Por ejemplo:

```
void Simulador::algoritmo([parametros])
{
Cuerpo del algoritmo.
}
```

  4. Compilar la herramienta.

  En la codificación del nuevo algoritmo, se recomienda tomar en cuenta lo siguiente: El conjunto de tareas siempre se encontrará disponible para ser utilizado por cualquier algoritmo de planificación realizando una copia a la estructura de tareas "struct qtasks tasks[Nmax]". Las función void

promedio(char *nombre) es la encargada de crear el archivo de datos finales que se imprimirá en pantalla.

Capítulo 7

Conclusiones

7.1 Contribuciones

En esta tesis nos enfocamos al estudio de algoritmos de planificación de tareas de tiempo real realizando una comparación tanto en los casos promedio así como en los peores casos. En este trabajo se diseñó una herramienta de simulación para el análisis del comportamiento de los algoritmos bajo condiciones similares.

Si bien el documento presenta una recopilación de los mejores algoritmos de planificación, también presenta algunas modificaciones a los algoritmos que sin duda mejoran el rendimiento tanto en los casos promedio como en los peores casos, como lo muestran los resultados presentados.

Aunque la mayor parte de la teoría de los Sistemas de Tiempo Real gira alrededor de sistemas uniprocesadores, en la actualidad, el desarrollo de sistemas de múltiples procesadores ha creado una tendencia hacia estos sistemas. Debido a esto, la planificación de tareas de tiempo real se ha convertido en un problema de actualidad.

En resumen, creemos que realizando este estudio, se tienen las bases para que futuros algoritmos de planificación tengan una fuente de comparación de

resultados. De igual forma, el documento puede servir como base para todas las personas interesadas en el estudio de algoritmos de planificación de sistemas de tiempo real sobre sistemas multiprocesadores.

7.2 Trabajo Futuro

Si bien existen en la actualidad un número de algoritmos para resolver los problemas de planificación sobre múltiples procesadores, éstos aún cuentan con restricciones como:

- La complejidad computacional asociada a los algoritmos existentes es muy alta. Esto indica que tales algoritmos incluyen un alto cómputo extra (overhead) en el sistema, lo cual produce un bajo desempeño. Es un hecho que pocas investigaciones en este campo han tenido como objetivo la medición del tiempo de cómputo extra que requiere la ejecución de dichos algoritmos.

- Los algoritmos existentes están desarrollados sobre modelos de tareas muy restrictivos. Pocos de ellos consideran tareas con restricciones de precedencia o con acceso a recursos compartidos. Estas restricciones hacen que su uso sea limitado.

- Pocas aplicaciones de estos algoritmos han sido investigadas. Por ejemplo, sería útil desarrollar algoritmos de planificación con múltiples procesadores que incluyan tolerancia a fallos o manejo de energía, o aplicaciones distribuidas, en donde una o varias tareas se ejecutan en cada procesador, y las tareas cuentan con restricciones de precedencia con otras tareas que se ejecutan en distintos procesadores.

A futuro sería conveniente desarrollar heurísticas que mejoren el rendimiento de los algoritmos actuales, y que demuestren las condiciones y ventajas de la

utilización de los esquemas particionados y globales.

Bibliografía

[1] B. Anderson and J. Jonson. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. *In proc. Of the IEEE Int'l Conference on Real Time Computing Systems and Applications.*, December 2000.

[2] N.C. Audsley. *Deadline monotonic scheduling.* PhD thesis, Dept. Computer Science. University of York, 1990.

[3] T.P. Baker and A. Shaw. The cyclic executive model and ada. *Proceedings of the IEEE Real-Time Systems Symposium*, December 1988.

[4] S. Davari and S. K. Dhall. An on line algorithm for real-time allocation. In *19th Ann Hawaii Int'l Conf. System Sciences*, pages 133–141, 1986.

[5] S. Davari and S.K. Dhall. On a periodic real time task allocation problem. In *Proc of 19th Annual International Conference on System Sciences*, pages 133–141, 1986.

[6] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[7] T. Carpenter. K. Driscoll. K. Hoyme. and J. Carciotini. Ar-inc659 scheduling: Problem definition. *In Proceedings of the Real Time Systems Symposium*, pages 165–169, December 1994.

[8] D. S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, MIT. Cambridge, 1973.

[9] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average behavior. *IEEE Real-Time Systems Symp.*, pages 166–171, 1989.

[10] J. Y. T. Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 4(2):209–219, 1989.

[11] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 26(2):237–250, 1982.

[12] C. L. Liu and W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[13] M. Masters. Challenges in designing future command and control systems. *Workshop on Parallel and Distributed Real-Time Systems*, 1998.

[14] S. Lauzac. R. Melhem. and D. Mosse. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. *In Proc. Of the Euromicro Workshop on a Real time System*, pages 188–195, June 1998.

[15] S. Ghosh. D. Mosse. and R. Melhem. Fault-tolerant rate monotonic scheduling. *Journal of Real time Systems*, 1998.

[16] Sylvain Lauzac. Rami Melhem. Daniel Mosse. An efficient rms admission control and its application to multiprocessor scheduling. Technical report, University of Pittsburgh. Department of Computer Science, 2000.

[17] D. S. Johnson. A. Demers. J.D. Ullman M.R.Garey and R.L. Graham. Worst case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, (3):256–278, 1974.

[18] A. Burchard. J. Liebeherr. Y. Oh. and S.H. Son. A linear online task assignment scheme for multiprocessor systems. no.

[19] A. Burchard. J. Liebeherr. Y. Oh. and S.H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, December 1995.

[20] Y. Oh and S. H. Son. Preemptive scheduling of periodic tasks on multiprocessor: Dynamic algorithms and their performance. Technical report, Univ. Of Virginia. Dept. of Computer Science., May 1993.

[21] Y. Oh and S. H. Son. Fixed priority scheduling of periodic tasks on multiprocessor systems. Technical report, Univ. Of Virginia. Dept. of Computer Science., March 1995.

[22] Y. Oh and S.H. Son. Tight performance bounds of heuristics for a real-time scheduling problem. Technical report, Univ. Of Virginia. Dept. of Computer Science., May 1993.

[23] R. Rajkumar. *Synchronization in real time systems: A priority inheritance Approach.* Kluver Academic Publishers, 1991.

[24] B. Sprunt. L. Sha. and J. Lehoczky. Aperiodic task scheduling for hard real time systems. *Journal of Real-Time Systems*, pages 27–60, 1989.