INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

No. 215 **Serie:** AZUL **Fecha:** Octubre 2005

Análisis del Tiempo de Ejecución de Algoritmos para la Asignación de Tareas de Tiempo Real en Multiprocesadores

Héctor Eduardo Silva López[1]
Sergio Suárez Guerra[2]

RESUMEN

La planificación de tareas de tiempo real en sistemas con múltiples procesadores es sabido que es computacionalmente intratable para un número grande de tareas. Los algoritmos de planificación prácticos para asignar tareas a un sistema de múltiples procesadores, presentan una disyuntiva entre su complejidad computacional y su desempeño. Por lo tanto, en este trabajo damos un panorama general donde se analiza el tiempo de ejecución de estos algoritmos, estructurando este trabajo en dos partes. En la primera parte se simula su ejecución para obtener su desempeño y el tiempo de ejecución de cada algoritmo, utilizando un número de tareas bajo. Posteriormente estos resultados se extrapolaron para obtener el tiempo de ejecución para un número de tareas mucho mayor. En la segunda parte, se escogen los que mejor desempeño tengan para ejecutarlos en un sistema de memoria compartida utilizando Pthreads y en un sistema de memoria distribuida utilizando las herramientas de la Maquina Virtual Paralela y la Interfaz de Paso de Mensajes.

Palabras clave: Asignamiento de tareas, esquema particionado, tiempo de ejecución, algoritmos heurísticos.

[1] Alumno de Doctorado en Ciencias de la Computación del Centro de Investigación en Computación, esilvab05@sagitario.cic.ipn.mx
[2] Profesor Investigador del Centro de Investigación en Computación, ssuarez@cic.ipn.mx

Instituto Politécnico Nacional
Centro de Investigación en Computación
Av. Juan de Dios Bátiz casi esq.
Miguel Othón de Mendizabal Ote.
México, 07738, D.F.
ISBN 970-36-0309-3
Impreso en México

1) INTRODUCCION

En la última década las aplicaciones que se ejecutan sobre ambientes de procesamiento paralelo han ido en aumento, originado principalmente por que ha demostrado ser una herramienta de ingeniería útil para los problemas de automatización industrial, simulación de sistemas, problemas complejos de tiempo real, etc. Esto es debido principalmente al enorme progreso en la tecnología de computadoras paralelas, herramientas de programación paralela y de los algoritmos de procesamiento paralelo.

En éste trabajo nos enfocamos en la planificación de tareas de tiempo real para procesamiento paralelo. Existen dos estrategias para la planificación de tareas de tiempo real en un sistema con multiprocesadores. Un esquema global (llamado también no-particionado) cada tarea de tiempo real puede ser ejecutada en un procesador diferente. En contraste, en un esquema particionado todas las instancias de una tarea son ejecutadas en un sólo procesador. El esquema particionado tiene algunas ventajas sobre el esquema global. En primer lugar, el esquema particionado es menos complejo por que no introduce sobrecarga al planificador de multiprocesamiento, ya que solamente son asignadas las tareas al procesador una sola vez al principio de la ejecución. En segundo lugar, un algoritmo ya conocido de planificación puede ser usado para cada procesador.

El desempeño del esquema particionado está determinado por dos factores; el algoritmo de asignación de las tareas, el cual se encarga de distribuir las tareas a los procesadores y un algoritmo de planificación, el cual determina el orden de ejecución de las tareas en cada procesador. El objetivo de un algoritmo de asignación de tareas será el tener una planificación factible (es decir, que todas las tareas conozcan sus plazos de respuesta en tiempo de ejecución) para cada procesador con el menor número de procesadores. Sin embargo, el problema de encontrar una asignación óptima para los algoritmos de prioridad fija, como el RM (Rate Monotonic), así como para la planificación con prioridades dinámicas, como el EDD (Earliest Due Date), es NP-duro. Esto lo demostraron Leung y Whitehead en [1].

Entre los dos esquemas, el particionado es en donde se ha concentrado la investigación, principalmente por que es fácil de usar para garantizar la planificabilidad en tiempo de ejecución. Muchos algoritmos heurísticos para éste método se han propuesto, por ejemplo, Dhall y Liu en [2], presentaron dos métodos de asignación, el RMNF (Rate Monotonic Next-Fit) y el RMFF (Rate Monotonic First-Fit). En ambos métodos, las tareas son ordenadas en orden decreciente a sus períodos, antes de ser asignadas. En RMNF las tareas son asignadas al procesador hasta que es violada la condición de planificabilidad, en tal caso, el procesador es marcado como lleno y es seleccionado un nuevo procesador. El RMFF primero trata de acomodar una tarea en un procesador ya marcado como lleno antes de ser asignada la tarea al procesador actual. El método FFDUF (First-Fit Decreasing-Utilization Factor) es una variación del esquema heurístico first-fit. Aquí las tareas son ordenadas en base a su factor de carga [3]. Oh y Son [4] usan el método RMBF (Rate Monotonic Best-Fit), el cual es parecido al RMFF ya que asigna tareas a los procesadores que han sido marcados como llenos. Sin embargo, los procesadores llenos son inspeccionados en un orden específico. Al igual que en [2], las tareas son ordenadas por sus períodos. Todos estos algoritmos están basados en el algoritmo heurístico bin-packing, exhibiendo en promedio un buen desempeño.

Burchard y otros, proponen otros dos algoritmos en [5], el RMST (Rate Monotonic Small Tasks), el cual particiona cada tarea en un número pequeño de sub-tareas, las cuales contienen tareas periódicas simples. Una buena aproximación para asignar la tarea, es particionar primero el conjunto de tareas periódicas en sub-conjuntos. Cada sub-conjunto de tareas periódicas es planificable todo el tiempo que la utilización total del sub-conjunto no sea mayor a uno. Si más de un sub-conjunto es asignado a un procesador, su utilización de planificación está en función del número de sub-conjuntos y no del número de tareas. El otro algoritmo es el RMGT (Rate Monotonic General Tasks) el cual primero particiona toda las tareas periódicas dentro de dos sub-conjuntos acorde a sus utilizaciones. Las tareas cuya utilización sea igual o menor a 1/3 estarán en un sub-conjunto. Esas tareas son primero asignadas a los procesadores acorde al algoritmo RMST. Después las tareas más largas cuya utilización sea más grande de 1/3, son asignadas por first-fit a los procesadores, en los cuales tiene al menos una tarea asignada por el algoritmo RMST. El método de análisis consume tiempo en checar si una tarea es grande y si puede ser asignada a un procesador que ya tenga una tarea.

El esquema no-particionado no ha recibido tanta atención, debido a las siguientes limitaciones: Primero, actualmente no existe una prueba eficiente de planificabilidad. Se conoce una prueba necesaria y suficiente de planificabilidad con una complejidad de tiempo exponencial [6]. La complejidad fue reducida a polinomial con una prueba de planificabilidad suficiente [7, 8, 9, 10] o con una complejidad de tiempo seudo-polinomial [9]. Pero en [10], la prueba llega a ser pesimista cuando el número de las tareas crece y en [7, 8, 9] llega a ser pesimista cuando el número de procesadores se incrementa. Segundo, no se ha encontrado un esquema de asignación de prioridades óptimo. La asignación de prioridades con rate monotonic (RM) [11], es óptimo en un sistema con un procesador, pero no es óptimo para multiprocesadores usando el método no-particionado [1, 2]. Un peor caso se presenta cuando un conjunto de tareas con una muy baja utilización no puede ser planificable con RM [2], conocido como el efecto Dhall.

Este trabajo está organizado de la siguiente manera: en la sección 2 se describe el modelo del Sistema, en la sección 3 se da una descripción del problema a resolver y su solución. Posteriormente en la sección 4 se simulan cuatro algoritmos representativos para obtener el tiempo de ejecución para cada Algoritmo. Una vez obtenido éste tiempo, se extrapolan los resultados para obtener el tiempo de ejecución para un número mucho mayor de tareas. En la sección 5 se paralelizaran dos algoritmos para ejecutarse en un sistema con memoria compartida y en un sistema de memoria distribuida. Los resultados obtenidos son analizados en la sección 6. Las conclusiones del trabajo se presentan en la sección 7 y finalmente termina éste trabajo con las referencias bibliograficas.

2) MODELO DEL SISTEMA

Se considera un conjunto de tareas $T=\{T_1,...,T_n\}$ de n tareas periódicas de tiempo real con desalojo, corriendo en varios procesadores. Las tareas son independientes (no comparten recursos) y no tienen restricciones de precedencia. El período de T_i es denotado por P_i, el cual es igual al plazo de respuesta de la invocación actual. Nos referimos a la invocación K_{th} de la tarea T_i como T_{ik}.

Dada una velocidad de CPU determinada por un par voltaje/frecuencia, la carga de trabajo en el peor caso es representada por el valor tradicional del tiempo de ejecución (WCET). Notar que, sin embargo, para un esqueleto de planificación con voltaje variable, donde el tiempo de ejecución actual depende de la velocidad del CPU, el número de ciclos requeridos de la carga de trabajo por el CPU en el peor caso es una medición más apropiada. Se denota el número de ciclos del procesador requeridos por T_i en el peor caso por C_i. En su trabajo inicial Liu y Layland [10] mostraron que la utilización u_i de una tarea es $u_i = C_i / T_i$, es la razón entre el tiempo de ejecución y su período. La utilización U de un conjunto de tareas es la suma de procesadores independientes que las tareas demanden para su ejecución y se expresa como:

$$U = \sum_{i=1}^{n} C_i / T_i$$

Una planificación de las tareas periódicas es factible si cada tarea T_i es asignada al menor C_i, antes de su plazo de respuesta en cada invocación.
Se considera el problema de planificar un conjunto de tareas con pequeños factores de carga es mostrado en [5], expresado como:

$$\propto := \max_{i=1,\ldots,n} u_i$$

Representa el factor de carga máximo para una tarea Ti. Para propósitos prácticos se considera que el conjunto de tareas contiene sólo pequeñas tareas sí $\alpha \leq ½$.

El sistema se comporta como sigue: Primero checa la utilización del procesador para verificar si puede aceptar la tarea, si no puede aceptarla, continua con el siguiente procesador, se le asigna una prioridad local y éstas son numeradas en orden decreciente por prioridad, esto es, la tarea τ_1 tiene la más alta prioridad.

En seguida se numeran las características que se tendrán a considerar:

1. Las tareas son independientes, arriban periódicamente y pueden ser desalojadas. En cada instante de tiempo el despachador determina cual tarea se debe ejecutar.
2. Las tareas no requieren acceso exclusivo a ningún recurso, únicamente al procesador.
3. El costo cuando una tarea arriba es cero, por que se considera que éste tiempo es parte del tiempo de ejecución.
4. El peor tiempo de ejecución de cada tarea es conocido a priori. El actual tiempo de ejecución no es conocido, ya que éste varía de una iteración a otra.
5. Únicamente se consideran los algoritmos para el esquema particionado.

3. DESCRIPCION DEL PROBLEMA

En [5] los autores mediante simulación compararon el desempeño de cuatro algoritmos de planificación para la asignación de tareas en un sistema de múltiples procesadores y utilizando diferentes factores de carga. Pero en ningún momento mencionan el tiempo de ejecución empleado en está simulación, además el número máximo de tareas que emplean es de 1000.

En éste trabajo se extiende el trabajo desarrollado en [5]. Lo primero sería reproducir la simulación y así obtener los tiempos que tardarían en ejecutarse cada algoritmo para dos factores de carga, cuando α=0.2 y α=0.5. En seguida el resultado de cada algoritmo se extrapola para obtener el tiempo que tardaría en ejecutarse cada algoritmo para un número de tareas mucho mayor (1000, 5000, 10000, 50000, 100000 y 1000000). Posteriormente, de los cuatro algoritmos se escogerían dos de ellos para ejecutarlos realmente en forma paralela, sin simular su ejecución, para observar cuanto bajaría el tiempo de ejecución de cada diferente algoritmo al programarlo en forma paralela. Lo primero será programar cada algoritmo para ejecutarse en memoria compartida utilizando Pthreads y lo segundo será utilizar la Máquina Virtual Paralela (PVM) y la Interfaz de Paso de Mensajes (MPI) para ejecutarlos en memoria distribuida.

4.- SIMULACION DEL ALGORITMO PARA LA ASIGNACION DE TAREAS

Los algoritmos a simular son el RMNF, RMFF, RMST y el RMGT. La simulación consiste en ejecutar de 100 a 1000 tareas con incrementos de 1 y en cada incremento se evalúa el algoritmo de 1 hasta el incremento. Por ejemplo, primero se evalúa de 1 a 100, enseguida se evalúa de 1 a 101, luego de 1 a 102 y así hasta llegar a 1000.

Primero el algoritmo genera las tareas de la siguiente manera:

1) El número de tareas va a ir de $100 \leq n \leq 1000$
2) El período se obtiene utilizando una función de distribución uniforme de $1 \leq T_i \leq 500$
3) El tiempo de cómputo o ejecución va de $0 < C_i \leq \alpha T_i$
4) El factor de carga máxima de cada tarea tendrá dos valores α=0.2 y α=0.5
5) Se corre 100 veces en cada incremento y se obtiene un promedio

La simulación de los cuatro algoritmos se realizó en un Pc Intel Pentium III a 650MHz con 128 Mb de RAM y corriendo en un Sistema Operativo de Linux. La función utilizada para la medición del tiempo es psched_get_time().

El simulador se encarga de generar las tareas de 100 a 1000 y en cada incremento se evalúa los cuatro algoritmos, obteniendo el tiempo de ejecución para cada algoritmo y el número de procesadores que necesita cada algoritmo para asignar las tareas. Cada incremento se ejecuta 100 veces y se obtiene un promedio. El simulador termina cuando el número de tareas sea mayor a 1000.

El factor de carga a utilizar será α=0.2 y α=0.5, para cada factor de carga se obtuvo una gráfica comparando el desempeño de los cuatro algoritmos y también se añadió la carga total del sistema. Dado que un asignamiento de tareas óptimo no puede ser calculado para un conjunto de tareas grande, se utilizó la carga total $U = \sum_{i=1}^{n} u_i$ usado en [12], para obtener la cota más baja (Lower Bound) para el número de procesadores requerido.

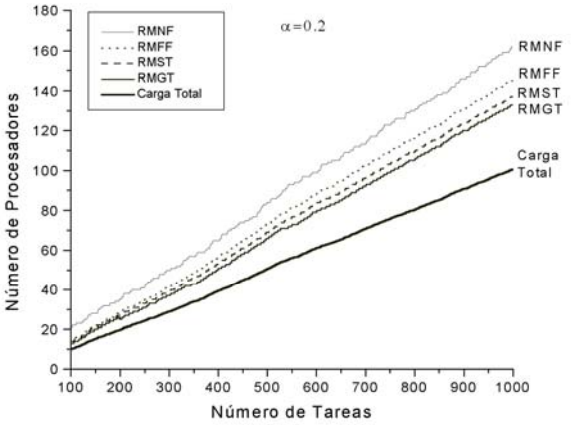

Figura 1. Simulación secuencial para α=0.2

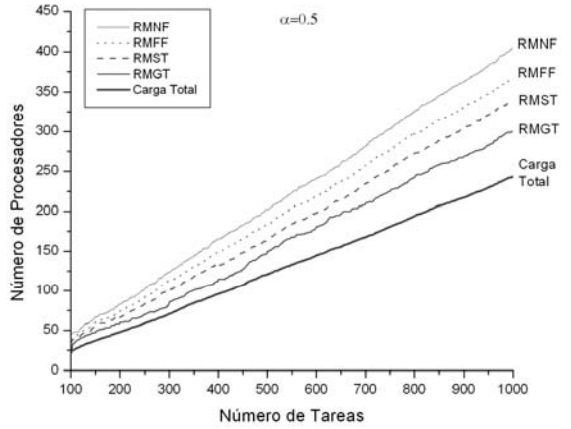

Figura 2. Simulación secuencial para α=0.5

Se puede observar en las figuras 1 y 2 con factor de carga de α=0.2 y α=0.5, que el peor algoritmo es el RMNF, seguido del RMFF, ya que necesitan un número de procesadores más grande para ejecutar la misma cantidad de tareas. Los dos algoritmos propuestos por Burchard son mejores y específicamente para el RMGT necesita un número menor de procesadores que el RMST, por ejemplo para α=0.2 y 1000 tareas, el RMST necesita 137 procesadores, mientras que para el RMGT sólo necesita 133 procesadores, he igual para α=0.5, el RMST ocupa 339 procesadores y el RMGT sólo ocupa 300 procesadores.

4.1 Extrapolación de los tiempos de ejecución para α=0.2

En la tabla 1 se muestra el tiempo de ejecución que se tardó para cada algoritmo en la simulación a un número de tareas de 100 a 1000 con un incremento de 100. El tiempo es representado en ticks de reloj.

Tabla 1. Tiempo secuencial en ticks de reloj, α=0.2.

Alg \ Tareas	100	200	300	400	500	600	700	800	900	1000
Gen_tasks	**2**	**3**	**4**	**6**	**7**	**8**	**10**	**10**	**12**	**14**
RMNF	**0**	**0**	**0**	**1**	**2**	**4**	**4**	**5**	**6**	**8**
RMFF	**1**	**9**	**33**	**74**	**151**	**261**	**410**	**632**	**934**	**1237**
RMST	**1**	**9**	**32**	**76**	**150**	**262**	**419**	**637**	**1207**	**1219**
RMGT	**1**	**9**	**31**	**76**	**151**	**262**	**419**	**634**	**895**	**1217**
Carga_total	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**2**

En seguida se obtiene un polinomio de aproximación que pueda usarse para obtener puntos adicionales fuera del dominio (número de tareas mayores) a los existentes en la tabla 1, mediante su valoración. Esto se conoce como extrapolación.
Para encontrar el polinomio de aproximación de cada algoritmo se utilizó el paquete Origin de Microcal Software, Inc. Versión 6.0.
La primera función a extrapolar es gen_tasks, que es la encargada de la generación de las tareas.

Regresión Polinomial para Gen_tasks:
$Y = A + B1 * X$

Parámetro	Valor	Error
A	0.46667	0.30781
B1	0.01297	4.96082E-4

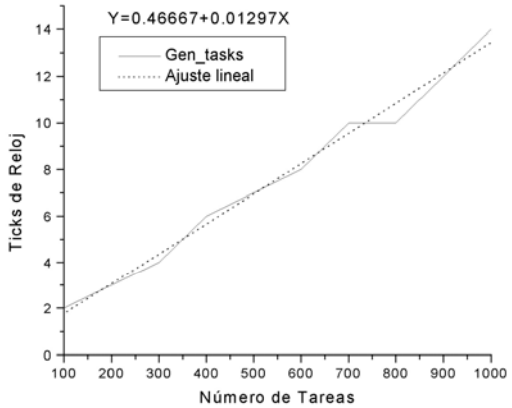

Figura 3. Ajuste para Gen_tasks, α=0.2

En seguida se realizó un ajuste polinomial de segundo grado para el algoritmo RMNF.

Regresión Polinomial para RMNF:
$Y = A + B1*X + B2*X^2$

Parámetro	Valor	Error
A	-0.66667	0.60933
B1	0.00242	0.00254
B2	6.06061E-6	2.25462E-6

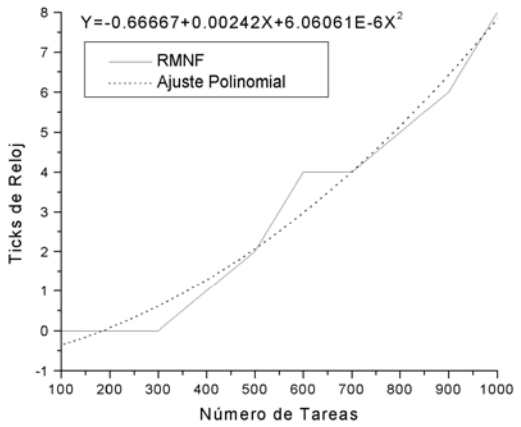

Figura 4. Ajuste para RMNF, α=0.2.

Para el algoritmo RMFF se realizó un ajuste polinomial de tercer grado.

Regresión Polinomial para RMFF:
Y = A + B1*X + B2*X^2 + B3*X^3

Parámetro	Valor	Error
A	8.53333	25.1382
B1	-0.07068	0.18842
B2	9.51632E-5	3.88654E-4
B3	1.2162E-6	2.33057E-7

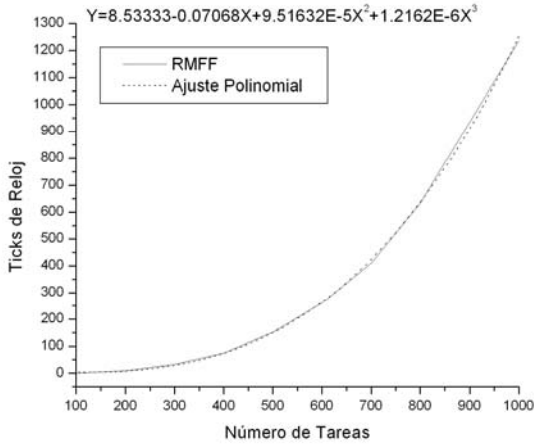

Figura 5. Ajuste para RMFF, α=0.2

Para el algoritmo RMST se utilizó un ajuste polinomial de segundo grado.

Regresión Polinomial para RMST:
Y = A + B1*X + B2*X^2

Parámetro	Valor	Error
A	113.23333	116.66324
B1	-1.05408	0.48723
B2	0.00225	4.31672E-4

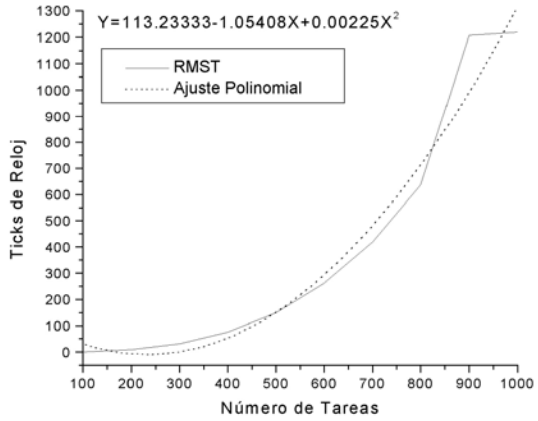

Figura 6. Ajuste para RMST, α=0.2

Para el algoritmo RMGT se realizó un ajuste polinomial de tercer grado.

Regresión Polinomial para RMGT:
Y = A + B1*X + B2*X^2 + B3*X^3

Parámetro	Valor	Error
A	9	6.34183
B1	-0.09568	0.04753
B2	2.18648E-4	9.80486E-5
B3	1.08741E-6	5.87951E-8

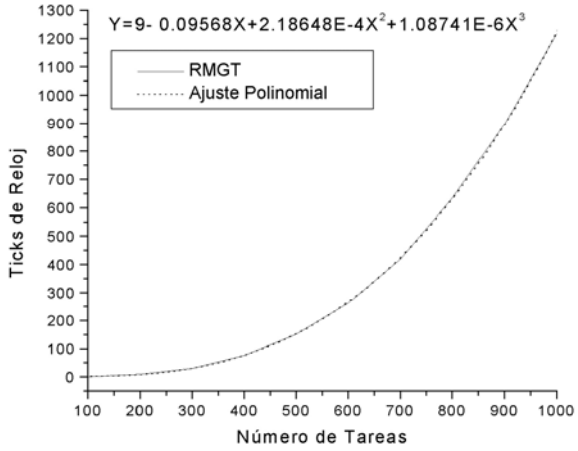

Figura 7. Ajuste para RMGT, α=0.2

Y finalmente para la carga total se utilizó un ajuste polinomial de segundo grado.

Regresión Polinomial para Carga_total
Y = A + B1*X + B2*X^2

Parámetro	Valor	Error
A	0.23333	0.29569
B1	-0.00217	0.00123
B2	3.78788E-6	1.09409E-6

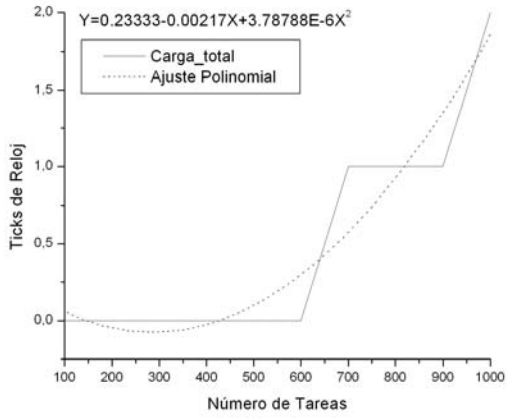

Figura 8. Ajuste para la Carga_total, α=0.2

4.2 Evaluación de los polinomios aproximados, α=0.2

Utilizando las aproximaciones polinomiales encontradas para cada algoritmo, se calculó cuanto tardaría el programa en ticks de reloj para un número de tareas más grande, ver tabla 2.

Tabla 2. Tiempo de ejecución para n tareas, α=0.2.

Alg \ Tasks	1,000	5,000	10,000	50,000	100,000	500,000	1,000,000
Gen_tasks	13.43667	65.31667	130.16667	648.96667	1297.46667	6485.46667	12970.4667
RMNF	7.81394	162.94858	629.59433	15271.8583	60847.4333	1516361.83	6063029.33
RMFF	1244.6365	153486.713	1220438.05	151686883	1212564573	1.5148E+11	1.2117E+12
RMST	1309.15333	51092.8333	214572.433	5572409.23	22394705.2	561973073	2248946033
RMGT	1219.378	140923.05	1108327	135932407	1089586921	1.3598E+11	1.0876E+12
Carga_total	1.85121	84.08033	357.32133	9361.43333	37662.0333	945885.233	3785710.23
Total	3796.26965	345814.942	2544454.57	293216981	2324646006	2.8802E+11	2.3016E+12

Para 1,000 tareas, el tiempo total de ejecución sería de

3796.26965/CLK_TCK = seg. $\Rightarrow$ 3796.26965 / 18.2 = 208.58 seg.
208.58 / 60 = minutos $\Rightarrow$ 208.58 / 60 = 3.476 minutos

Para 5,000 tareas:

345814.942/CLK_TCK = seg. $\Rightarrow$ 345814.942 / 18.2 = 19000.82 seg.
19000.82 / 3600 = horas $\Rightarrow$ 19000.82 / 3600 = 5.278 horas

Para 10,000 tareas:

2544454.57/CLK_TCK = seg. $\Rightarrow$ 2544454.57/ 18.2= 139805.19 seg.
139805.19 / 3600 = horas $\Rightarrow$ 139805.19 / 3600 = 38.83 horas

Para 50,000 tareas:

293216981/CLK_TCK = seg. $\Rightarrow$ 293216981/ 18.2= 16110823.13 seg.
16110823.13 / 3600 = horas $\Rightarrow$ 16110823.13 / 3600 = 4475.22 horas
4475.22 /24 = días $\Rightarrow$ 4475.22 /24 = 186.46 días
186.46 / 30 = meses $\Rightarrow$ 186.46 / 30 = 6.215 meses

Para 100,000 tareas:

2324646006/CLK_TCK = seg. $\Rightarrow$ 2324646006 / 18.2 = 127727802.5 seg.
127727802.5 / 3600 = horas $\Rightarrow$ 127727802.5 / 3600 = 35479.94 horas
35479.94 /24 = días $\Rightarrow$ 35479.94/24 = 1478.33 días
1478.33 / 365 = años $\Rightarrow$ 1478.33 / 365 = 4.05 años

Para 500,000 tareas:

2.8802E+11/CLK_TCK = seg. $\Rightarrow$ 2.8802E+11 / 18.2 = 1.582E10 seg.
1.582E10 / 3600 = horas $\Rightarrow$ 1.582E10 / 3600 = 4395909.646 horas
4395909.646 /24 = días $\Rightarrow$ 4395909.646 /24 = 183162.90 días
183162.90 / 365 = años $\Rightarrow$ 183162.90 / 365 = 501.81 años

Para 1,000,000 de tareas,

2.3016E+12 / CLK_TCK = seg. $\Rightarrow$ 2.3016E+12 / 18.2 = 1.2646E11 seg.
1.2646E11 / 3600 = horas $\Rightarrow$ 1.2646E11/ 3600 = 35,128, 205.13 horas
35,128, 205.13 / 24 = días $\Rightarrow$ 35,128,205.13 / 24 = 1, 463, 675.214 días

1, 463, 675.214 / 365 = años ⇒ 1, 463, 675.214 / 365 = 4010.06 años

4.3 Extrapolación de los tiempos de ejecución para α=0.5

Los tiempos de ejecución para cada algoritmo en la simulación con un α=0.5, se muestra en la tabla 3.

Tabla 3. Tiempo secuencial en ticks de reloj, α=0.5

Alg \ Tareas	100	200	300	400	500	600	700	800	900	1000
Gen_tasks	2	3	4	5	7	8	9	11	12	14
RMNF	0	0	0	1	2	3	4	5	7	9
RMFF	2	15	43	108	228	378	611	895	1363	1835
RMST	2	14	42	110	219	369	595	875	1266	1794
RMGT	1	14	37	115	212	281	455	671	958	1324
Carga_total	0	0	0	0	0	0	0	0	0	1

Al igual que en el inciso 4.1 se utilizó para encontrar el polinomio de aproximación de cada algoritmo el paquete Origin de Microcal Software, Inc. Versión 6.0.

Se realizó un ajuste polinomial para cada uno de los algoritmos, para la Gen_tasks se realizó un ajuste lineal obteniéndose los siguientes resultados.

Regresión Polinomial para Gen_tasks:
Y = A + B1 * X

Parámetro	Valor	Error
A	0.2	0.26054
B1	0.01327	4.19891E-4

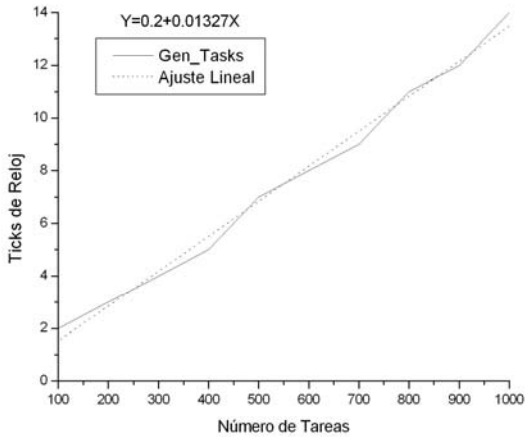

Figura 9. Ajuste para Gen_tasks, α=0.5

En seguida se realizó un ajuste polinomial de segundo grado para el algoritmo RMNF.

Regresión Polinomial para RMNF:
Y = A + B1*X + B2*X^2

Parámetro	Valor	Error
A	-0.06667	0.3017
B1	-0.00167	0.00126
B2	1.06061E-5	1.11635E-6

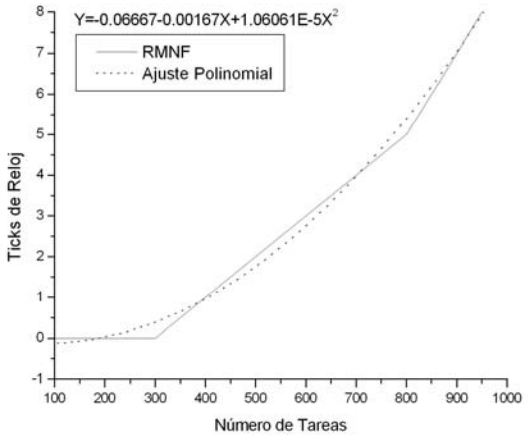

Figura 10. Ajuste para RMNF, α=0.5.

Para el algoritmo RMFF se realizó un ajuste polinomial de tercer grado.

Regresión Polinomial para RMFF:
Y = A + B1*X + B2*X^2 + B3*X^3

Parámetro	Valor	Error
A	-7.36667	36.37345
B1	0.10061	0.27264
B2	-3.65501E-4	5.62356E-4
B3	2.11752E-6	3.37218E-7

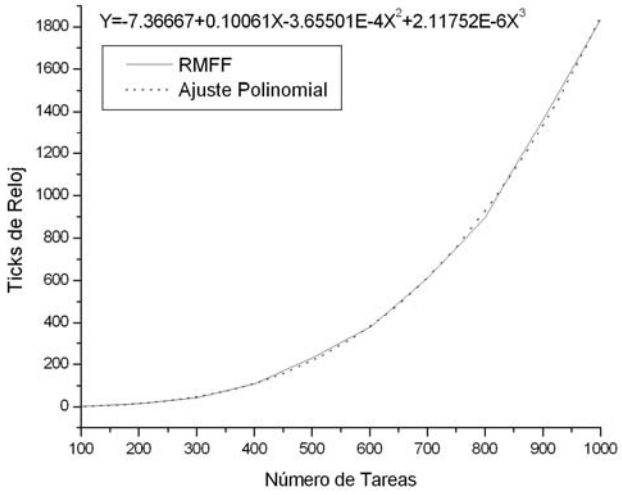

Figura 11. Ajuste para RMFF, α=0.5

Para el algoritmo RMST se utilizó un ajuste polinomial de tercer grado.

Regresión Polinomial para RMST:
Y = A + B1*X + B2*X^2 + B3*X^3

Parámetro	Valor	Error
A	-26.56667	21.7856
B1	0.28132	0.16329
B2	-7.52622E-4	3.36819E-4
B3	2.28166E-6	2.01974E-7

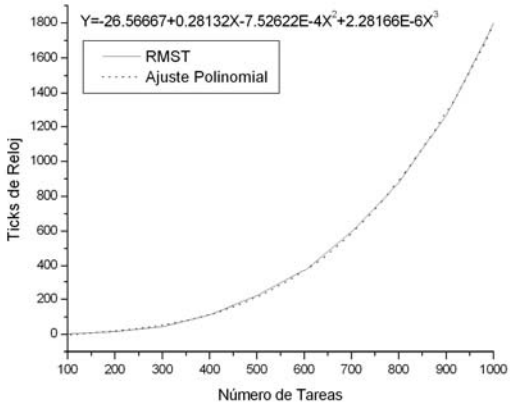

Figura 12. Ajuste para RMST, α=0.5

Para el algoritmo RMGT se realizó un ajuste polinomial de tercer grado.

Regresión Polinomial para RMGT:
Y = A + B1*X + B2*X^2 + B3*X^3

Parámetro	Valor	Error
A	-40.2	32.16673
B1	0.38658	0.2411
B2	-7.331E-4	4.97318E-4
B3	1.70785E-6	2.98218E-7

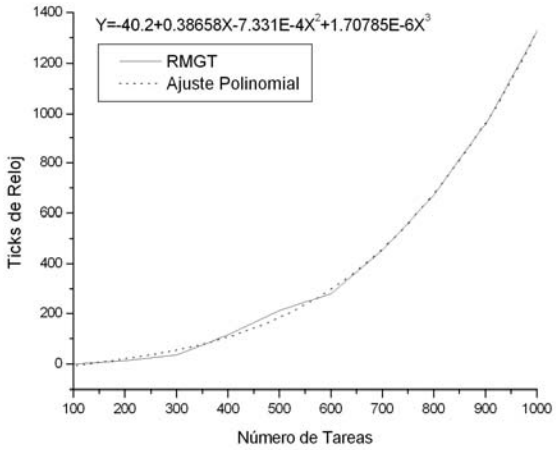

Figura 13. Ajuste para RMGT, α=0.5

Y finalmente para la carga total se utilizó un ajuste polinomial de séptimo grado.

Regresión Polinomial para Carga_total:
Y = A + B1*X + B2*X^2 + B3*X^3 + B4*X^4 + B5*X^5 + B6*X^6 + B7* X^7

Parámetro	Valor	Error
A	-1.6	0.84809
B1	0.03811	0.01865
B2	-3.40605E-4	1.51336E-4
B3	1.51963E-6	6.08123E-7
B4	-3.72474E-9	1.33744E-9
B5	5.08824E-12	1.63696E-12
B6	-3.62745E-15	1.04531E-15
B7	1.05042E-18	2.71217E-19

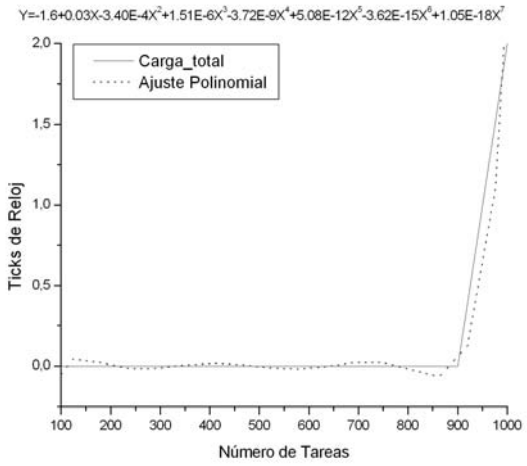

Figura 14. Ajuste para la Carga_total, α=0.5

4.4 Evaluación de los polinomios aproximados, α=0.5

Utilizando las aproximaciones polinomiales encontradas para cada algoritmo, se calculó cuanto tardaría el programa en ticks de reloj para un número mayor de tareas. Ver tabla 4.

Tabla 4. Tiempo de ejecución para n tareas, α=0.5

Alg \ Tasks	1,000	5,000	10,000	50,000	100,000	500,000	1,000,000
Gen_tasks	13.47	66.55	132.9	663.7	1327.2	6635.2	13270.2
RMNF	8.26943	256.13583	1043.24333	26431.0833	105893.333	2650689.33	10604429.3
RMFF	1845.26233	256048.158	2081968.63	263781271	2113875044	2.646E+11	2.1172E+12
RMST	1836.90367	267825.117	2209237.57	285183961	2274161939	2.8502E+11	2.2809E+12
RMGT	1321.13	197046.45	1638365.6	211667789	1700557618	2.133E+11	1.7071E+12
Carga_total	2.005	39139571.3	7349812549	7.6553E+14	1.0147E+17	8.1499E+21	1.0468E+24
Total	5027.04043	39860813.7	7355743297	7.6553E+14	1.0147E+17	8.1499E+21	1.0468E+24

Para 1,000 tareas, el tiempo total de ejecución sería de

5027.04043/CLK_TCK = seg. $\Rightarrow$ 5027.04043/ 18.2 = 276.21 seg.
276.21 / 60 = minutos $\Rightarrow$ 276.21 / 60 = 4.60 minutos

Para 5,000 tareas:

39860813.7/CLK_TCK = seg. $\Rightarrow$ 39860813.7/ 18.2 = 2190154.59 seg.
2190154.59 / 3600 = horas $\Rightarrow$ 2190154.59 / 3600 = 608.37 horas
608.37 / 24 = días $\Rightarrow$ 608.37 / 24 = 25.35 días

Para 10,000 tareas:

7355743297/CLK_TCK = seg. $\Rightarrow$ 7355743297/ 18.2 = 404161719.6 seg.
404161719.6 / 3600 = horas $\Rightarrow$ 404161719.6 / 3600 = 112267.14 horas
112267.14 / 24 = días $\Rightarrow$ 112267.14 / 24 = 4677.8 días
4677.8 / 365 = años $\Rightarrow$ 4677.8 / 365 = 12.82 años

No se continuo con el tiempo de ejecución por resultar realmente impractico seguir el análisis para un número mayor de tareas.

5. PROGRAMACION PARALELA

Se escogió el RMST y el RMGT por ser los que mejor desempeño obtuvieron de los cuatro. Antes de paralelizarlos, primero se explica el funcionamiento de cada uno de ellos.

El RMGT fue propuesto por Burchard [5], el cual primero particionan todas las tareas periódicas dentro de dos sub-conjuntos acorde a sus utilizaciones. Las tareas cuya utilización es igual o menor a 1/3 están en un sub-conjunto. Estas tareas son primero asignadas a los procesadores acorde al algoritmo RMST. Después las tareas más largas cuya utilización es más grande de 1/3 son asignadas por first-fit a los procesadores los cuales tienen al menos una tarea asignada por el algoritmo RMST. El algoritmo RMST primero ordena las tareas periódicas en orden creciente de acuerdo a sus parámetros Xi´s, los cuales son calculados de los períodos *pi* de las tareas de acuerdo a la siguiente ecuación:

$$Xi = \log_2 pi - \lfloor \log_2 pi \rfloor$$

Posteriormente, asigna las tareas a los procesadores utilizando la forma First-Fit. La condición de planificabilidad es la siguiente:

$$u_i + U_i \leq \max(\ln 2, 1 - \xi_k \ln 2)$$

En donde:

$$\xi = \max_l X_l - \min_l X_l$$

Y la utilización planificable para m ≥ 2, (m es el número de procesadores) es:

$$U_{RMST} = (m-2)(1-u_{\max}) + 1 - \ln 2$$

5.1 Programación paralela para memoria compartida

Un programa paralelo consiste de uno o más threads de control. Un *thread*, también llamado proceso liviano (lightweight process LWP), es una unidad básica de utilización de CPU y consiste de una secuencia de instrucciones ejecutadas en un programa, un conjunto de registros y un espacio de *stack*. Un *thread* comparte con otros *threads* de un mismo proceso su sección de código, sección de datos y recursos del sistema de operación, tales como archivos abiertos y señales. Todos los threads corren independientemente uno de otro en el mismo espacio de direcciones y no son visibles fuera del proceso. El hecho de tales compartimientos de recursos hace que el cambio de contexto entre *threads* de un mismo proceso y la creación de *threads* no sean costosos comparado con el cambio de contexto entre procesos pesados (*heavyweigth process*, un proceso tradicional con un sólo *thread*). Esto principalmente porque no se requiere

realizar trabajo de administración de memoria. La librería de Pthreads cumple con los estándares POSIX y nos permite trabajar con distintos hilos de ejecución (threads) al mismo tiempo.

Para la ejecución del programa paralelo para memoria compartida se realizó en una Computadora de Multiprocesamiento, integrada por 4 Procesadores (Pentium III a 750 MHz) en cascada y un Sistema Operativo Linux NET4.0 for Linux 2.4

El método de particionamiento de los algoritmos RMST y RMGT consistió en asignar a cada esclavo el número correspondiente de cada esclavo menos 1, más 100 con un incremento de acuerdo al número de esclavos requeridos, así se continúa hasta 1000, es decir: con dos esclavos, el primero le tocaría los números pares y el segundo los números impares hasta 1000.

Tabla 5. Ejemplo con 2 esclavos

# de esclavos / Tareas	1	2
	100	101
	102	103
	104	105

Con tres esclavos, el primero le tocaría 100 con incrementos de 3 y los demás en ese orden hasta 1000.

Tabla 6. Ejemplo con 3 esclavos.

# de esclavos / Tareas	1	2	3
	100	101	102
	103	104	105
	106	107	108

Para la implementación con Pthreads, se obtuvieron los tiempos en segundos tanto para los valores de α=0.2 y α=0.5. Se comparan los resultados obtenidos para el algoritmo RMST con el algoritmo RMGT.

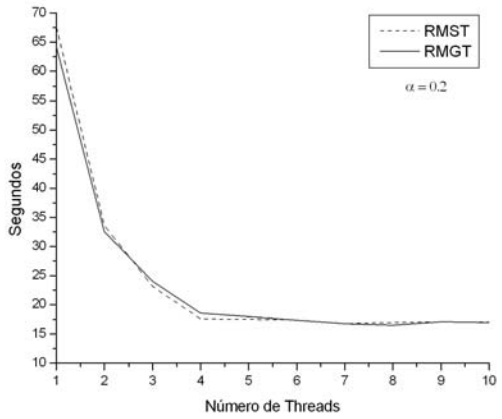

Figura 15. Threads con α=0.2

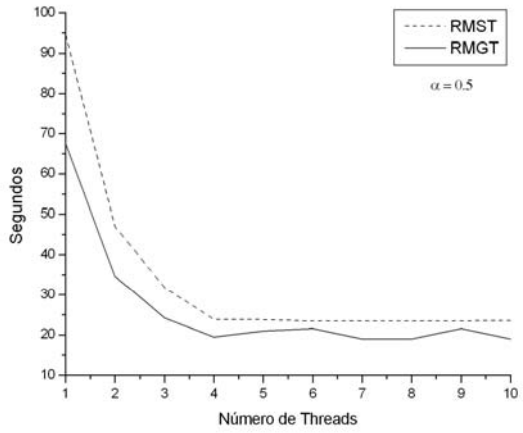

Figura 16. Threads con α=0.5

En la figura 15 se observa que el tiempo de ejecución para el algoritmo RMST y RMGT es el mismo para un número máximo de threads de 10, mientras que en la figura 16 el algoritmo RMST tiene un tiempo de ejecución mayor comparado con el algoritmo RMGT. De hecho el RMGT después de cuatro threads tiene un comportamiento un poco irregular. En los dos casos se obtiene que al ir aumentando el número de threads, el tiempo de ejecución baja, pero a partir de 4 threads el tiempo ya no cambia, esto es debido a que el número de procesadores donde se simuló está parte del trabajo sólo se contaban con 4 procesadores.

5.2 Programación paralela para memoria distribuida

Un sistema de memoria distribuida tiene su propia memoria local. Los Procesadores pueden compartir información solamente enviando mensajes, es decir, si un procesador requiere los datos contenidos en la memoria de otro procesador, deberá enviar un mensaje solicitándolos. Esta comunicación se le conoce como Paso de Mensajes.
En éste trabajo se utilizaron dos herramientas para la programación paralela. La primera es la Máquina Virtual Paralela (PVM) y la segunda es la Interfaz de Paso de Mensajes (MPI).

Para la implementación de éstas dos herramientas se utilizó un Cluster de 9 computadoras (un servidor y 8 clientes) Pentium II a 450 MHz y 256 Mb de RAM, con un Sistema Operativo LINUX Red Hat. Para la comunicación de cada nodo se utilizó un Switch Fast Ethernet (Switch intel Express 510T 10/100 Fast Ethernet) y un cableado tipo FTP.

5.2.1 Modelo de Rendimiento

El objetivo del Modelo de Rendimiento es desarrollar expresiones matemáticas que especifiquen el tiempo de ejecución. El tiempo de ejecución de un programa paralelo es el tiempo que transcurre desde que el primer procesador inicia su ejecución hasta que el último procesador termina su ejecución y se expresa como sigue:

$$T_p = T_{com} + T_{esp}$$

En donde:

T_p.- Es el tiempo paralelo o de cómputo, es el tiempo invertido en procesamiento (no en comunicaciones, ni en espera) y depende normalmente del tamaño del problema.
T_{com} .- El tiempo de comunicaciones, es el tiempo invertido en comunicar información de un procesador a otro.
T_{esp}.- El tiempo de espera, es el tiempo perdido en espera de la ocurrencia de eventos, los tiempos de sincronización y los tiempos de trabajo adicional.

En éste trabajo se considera que el tiempo de espera está implícito en el peor tiempo de ejecución de cada tarea. El tiempo de comunicaciones se obtiene de la siguiente ecuación:

$$T_{com} = \lambda + \beta n$$

Donde:

λ = Latencia de comunicación
β = El tiempo para transmitir un dato (inverso del ancho de banda)
n = Tráfico total en la red

El Modelo de Rendimiento para nuestra aplicación queda de la siguiente manera:

$$T_{paralelo} = \left[\frac{\sum_{n=100}^{1000} n \log n}{S} * C \right] + \left[(n-100) S * \left(\lambda + \beta \frac{(n-100)}{S} \right) \right]$$

En donde:

n son los conjuntos de tareas a analizar (1000).
S número de esclavos (1 a 8).
C es el tiempo que toma hacer las operaciones básicas (17×10^{-6} tiempo experimental).
λ Latencia de las comunicaciones (0.000196).
β tiempo requerido para enviar un símbolo (0.09453×10^{-6}).

(n – 100) - rango de ejecución del algoritmo
(n – 100) / S – rango de ejecución para cada esclavo
n log n – complejidad del algoritmo RMST.

Para obtener λ y β se llevó a cabo de la siguiente manera. Para λ se envió un mensaje de tamaño 1, sacando un promedio de 100 repeticiones. Y para la β se enviaron varios mensajes de diferentes tamaños y a sus valores se realizó un ajuste lineal en donde el valor de la pendiente B es el valor de β. El ajuste lineal correspondiente se ve en la siguiente gráfica.

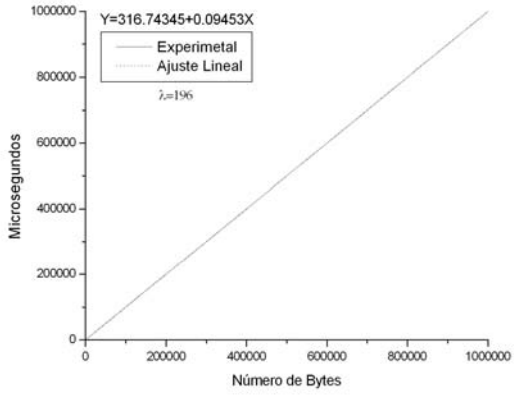

Figura 17. Modelo de rendimiento

5.2.2 Máquina Virtual Paralela

La Máquina Virtual Paralela (Parallel Virtual Machine, PVM) es un conjunto de rutinas de paralelización creados en el *Oak Ridge National Laboratory* en 1989. Es un conjunto integrado de herramientas y librerías de software, que emulan un marco de trabajo de computación concurrente de propósito general, flexible y heterogéneo. Todo esto sobre un conjunto de computadoras de distintas arquitecturas interconectadas entre sí. El principal objetivo del sistema PVM, es permitir que tal conjunto de máquinas, sean usadas en forma cooperativa para hacer computación concurrente o paralela. El sistema PVM está compuesto por dos partes. La primera es un demonio, llamado pvmd3 que reside en todas las máquinas que conforman la máquina virtual. La segunda parte del sistema es una librería de interfaces de rutinas de PVM, que contiene un repertorio de primitivas necesarias para la cooperación entre las tareas de una aplicación. Tales rutinas pueden ser llamadas por el usuario y permiten realizar paso de mensajes, expansión de procesos, coordinación de tareas y modificación de la máquina virtual. Las interfaces están provistas para los lenguajes Fortran y C, siendo implementadas como subrutinas en el primer caso y como funciones en el último. En base al tiempo de comunicaciones, al modelo y a los resultados experimentales se obtuvieron los siguientes datos correspondientes para α=0.2, estos se pueden observar en la tabla 7, el tiempo está en segundos.

Tabla 7. Resultados experimentales con PVM, α=0.2.

Esclavos	Tcomunicaciones	Tmodelo	Texp-RMST	Texp-RMGT
1	**0.253**	**78.41**	**97.50**	**125.27**
2	**0.429**	**39.51**	**52.09**	**66.82**
3	**0.606**	**26.66**	**34.17**	**42.40**
4	**0.782**	**20.32**	**27.48**	**35.41**
5	**0.958**	**16.59**	**20.64**	**25.87**
6	**1.134**	**14.14**	**19.24**	**24.24**
7	**1.311**	**11.86**	**14.99**	**20.54**
8	**1.486**	**9.24**	**13.95**	**18.78**

La figura 18 corresponde a resultados experimentales y al modelo de rendimiento. Se aprecia que el algoritmo RMST tiene un comportamiento más parecido al modelo de rendimiento, mientras que para el algoritmo RMGT tiene un tiempo de ejecución más grande que para el modelo y el RMST.

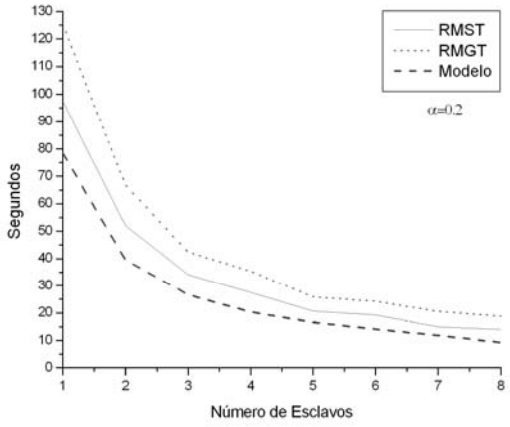

Figura 18. PVM, α=0.2

En base al tiempo de las comunicaciones, al modelo y a los resultados experimentales, se obtuvieron los siguientes datos correspondientes para α=0.5, estos se muestran la tabla 8, el tiempo está en segundos.

Tabla 8. Resultados experimentales con PVM, α=0.5.

Esclavos	Tcomunicaciones	Tmodelo	Texp-RMST	Texp-RMGT
1	**0.253**	**78.41**	**99.07**	**123.62**
2	**0.429**	**39.51**	**59.48**	**73.60**
3	**0.606**	**26.66**	**34.24**	**58.30**
4	**0.782**	**20.32**	**27.30**	**48.43**
5	**0.958**	**16.59**	**20.42**	**35.98**
6	**1.134**	**14.14**	**18.26**	**30.85**
7	**1.311**	**11.86**	**15.23**	**27.89**
8	**1.486**	**9.24**	**13.76**	**26.60**

En la figura 19 que corresponde cuando el factor de carga es igual a 0.5, el RMST tiene un comportamiento parecido al presentado en la figura 18 comparado contra el modelo. Pero el algoritmo RMGT se aleja más de su comportamiento presentado en la figura 18. Esto es debido principalmente a que como el factor de carga es mayor (0.5) el número de tareas a ser asignadas a los procesadores por el algoritmo RMFF es mayor comparado cuando el factor de carga es 0.2. Y el RMFF como ya se demostró presenta un desempeño malo.

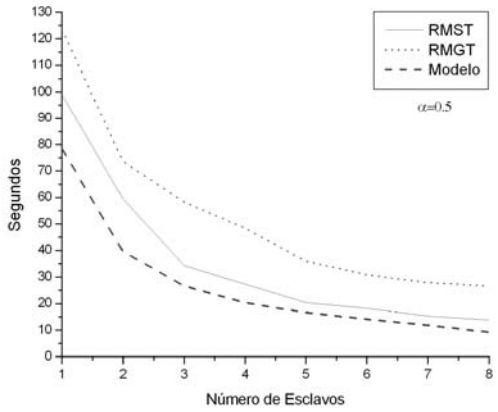

Figura 19. PVM, α=0.5

5.2.3 Interfaz de Paso de Mensajes

Interfaz para el Paso de Mensajes (Message Passing Interface, MPI) es una interfaz estandarizada para la realización de aplicaciones paralelas basadas en paso de mensajes. Su ventaja más inmediata es su implementación y portabilidad a una gran variedad de sistemas: desde máquinas con memoria compartida hasta una red de estaciones de trabajo.

Ejecutando un programa en MPI, cada procesador tiene una copia el programa. Todos los procesadores comienzan a ejecutar el mismo listado del programa, pero cada proceso se ejecutará distintas sentencias del programa, por bifurcaciones introducidas basadas en el rango del proceso (el rango identifica a cada proceso). De está forma, cada procesador ejecuta el mismo programa, pero hará diferentes cosas dependiendo del procesador donde se ejecute (p/e. si el proceso 1 envía un mensaje al proceso 2, éste último deberá tener en el listado una condición de si es el proceso 2, recibirá un mensaje del proceso 1).

En base al tiempo en las comunicaciones, al modelo y a los resultados experimentales para α=0.2 que se obtuvieron al programar los algoritmos RMST y RMGT para ser ejecutados en MPI, estos se pueden observar en la tabla 9, el tiempo está en segundos.

Tabla 9. Resultados experimentales con MPI, α=0.2.

Esclavos	Tcomunicaciones	Tmodelo	Texp-RMST	Texp-RMGT
1	**0.253**	78.41	**122.35**	**125.42**
2	**0.429**	**39.51**	**62.84**	**78.78**
3	**0.606**	**26.66**	**41.95**	**50.07**
4	**0.782**	**20.32**	**31.08**	**32.37**
5	**0.958**	**16.59**	**25.60**	**26.28**
6	**1.134**	**14.14**	**21.11**	**21.36**
7	**1.311**	**11.86**	**18.14**	**18.34**
8	**1.486**	**9.24**	**16.11**	**16.33**

En la figura 20 se observa un comportamiento parecido no igual al de la figura 18, pero en MPI para el algoritmo RMST tiene un comportamiento casi igual al algoritmo RMGT cuando el número de esclavos es igual a 1 y va de 4 a 8.

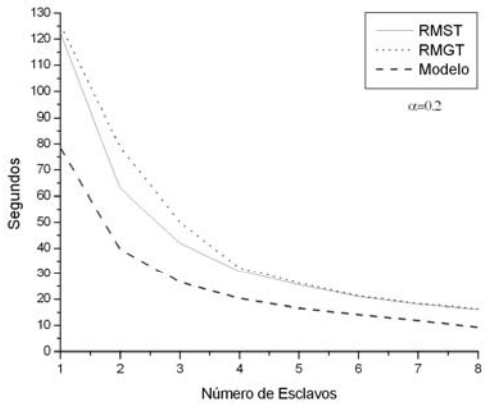

Figura 20. MPI, α=0.2.

En base al tiempo de comunicaciones, al modelo y a los resultados experimentales correspondientes para α=0.5, se pueden ver en la tabla 10, el tiempo está en segundos.

Tabla 10. Resultados experimentales con mpi, α=0.5.

Esclavos	Tcomunicaciones	Tmodelo	Texp-RMST	Texp-RMGT
1	0.253	78.41	173.95	124.21
2	0.429	39.51	85.06	65.86
3	0.606	26.66	55.34	50.26
4	0.782	20.32	42.82	40.64
5	0.958	16.59	33.21	32.22
6	1.134	14.14	28.15	22.86
7	1.311	11.86	24.28	21.85
8	1.486	9.24	21.29	20.25

En la figura 21 el comportamiento de los algoritmos es diferente comparado con las figuras 18, 19 y 20, ya que el algoritmo que consume más tiempo de ejecución es el RMST. Y tiene un comportamiento casi igual al RMGT cuando el número de esclavos va de 4 a 5 y de 7 a 8.

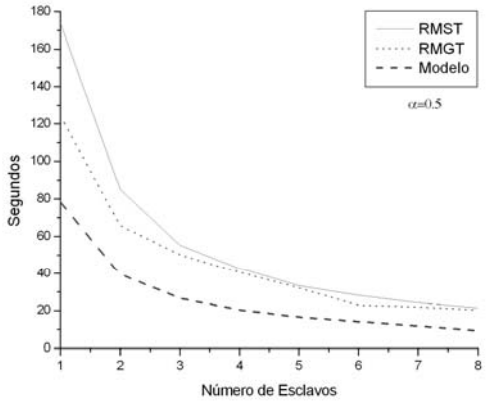

Figura 21. MPI, α=0.5

6.- COMPARACION DE RESULTADOS

En está sección se presentan los tiempos de ejecución que tarda en ejecutarse el algoritmo RMGT tanto en el tiempo secuencial como en el tiempo paralelo. Este algoritmo se escogió por tener un mejor desempeño que el RMST y como los tiempos de ejecución son muy parecidos, no tiene mucho interés en presentar los dos algoritmos.
En forma secuencial el algoritmo RMGT que presentó un tiempo de ejecución menor para un factor de carga de 0.2 y 0.5 y los tiempos obtenidos en forma experimental y teórica para 1,000 tareas se pueden ver en la tabla 11, el tiempo está en segundos. En está tabla los tiempos de ejecución encontrados en forma teórica y experimental son muy parecidos por lo que se concluye que los resultados teóricos presentados en la tabla 2 son muy cercanos a la ejecución real para una cantidad mucho mayor de tareas.

Tabla 11. Tiempo secuencial para el RMGT

Factor de Carga	Experimental	Teórica
0.2	66.87	70.00
0.5	72.75	72.59

Los tiempos paralelos para Threads, PVM y MPI se pueden observar en la tabla 12, tanto para un factor de carga de 0.2 y 0.5. El tiempo está en segundos.

Tabla 12. Tiempo Paralelo para el RMGT

Factor de Carga	Threads	PVM	MPI
0.2	17.18	22.35	20.57
0.5	20.56	30.33	24.29

En la tabla 12, el tiempo presentado para los Threads es el resultado de sacar un promedio de la suma del tiempo obtenido entre 4 y 10 threads, para los dos factores de carga. Para PVM y MPI se obtuvo el promedio de la suma de los tiempos obtenidos entre 5 y 8 esclavos.
Los tiempos presentados concuerdan con lo esperado, para PVM y MPI tienen un tiempo de ejecución más grande que el tiempo de ejecución de los threads, esto es debido al tiempo de comunicaciones que está implícito en la ejecución de un sistema distribuido.

7.- CONCLUSIONES

Se simuló el desempeño de cuatro algoritmos para la asignación de tareas de tiempo real en un ambiente de multiprocesadores, en donde se obtuvieron sus tiempos de ejecución. Estos algoritmos son el RMNF, RMFF, RMST y el RMGT. Los tiempos de ejecución obtenidos se extrapolaron para obtener polinomios aproximados que nos den una idea del tiempo que tardaría cada algoritmo en ejecutar un número de tareas mucho mayor, resultando en un tiempo prácticamente imposible de llevar a cabo. Por ejemplo, para un factor de carga igual a 0.2 resultaría imposible su ejecución a partir de 50,000 tareas ya que el programa en ejecutarse tardaría 6.215 meses y para un factor de carga de 0.5 resulta imposible a partir de 10,000 tareas, ya que tomaría 12.82 años para ejecutarse.

En la segunda parte de éste trabajo, se escogieron dos algoritmos cuyo desempeño resultaron mucho mejor que los otros dos para paralelizarlos, estos son el RMST y el RMGT para memoria compartida se utilizaron Threads y para memoria distribuida se utilizaron en PVM y MPI.
La comparación final de resultados nos muestra que al obtener los polinomios aproximados dan resultados muy cercanos al obtenido en forma experimental, por lo que los resultados que se presentaron para un número mayor de tareas no están muy alejados de la realidad. Por otra parte, al paralelizar dos algoritmos se obtuvieron los resultados que se esperaban, esto es, para los Threads llega el momento a partir de 4 Threads el tiempo de ejecución baja muy poco. Y aún así se compararon con los obtenidos en PVM y MPI y utilizando Threads presentan un mejor tiempo de ejecución, ya que en PVM y MPI al tiempo de ejecución se le tiene que sumar el tiempo de comunicaciones

REFERENCIAS

[1] J. Y. T. Leung and J. Whitehead. "On the complexity of fixed-priority scheduling of periodic, real time tasks". Performance Evaluation, 2(4):237-250, December 1982.

[2] S.K. Dhall and C. L. Liu. "On a real time scheduling problem". Operations Research, 26(1):127-140, January/February 1978.

[3] S. Davari and S. K. Dhall, "An on line algorithm for real time allocation", 19th Ann. Hawaii Int´l Conf. System Sciences, pp 133-141, 1986.

[4] Y. Oh and S. H. Son. "Tight performance bounds of heuristics for a real time scheduling problem", Technical Report CS-93-24, Univ. of Virginia, Dept. of Computer Science, May 1993.

[5] 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.

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

[7] B. Andersson. "Adaption of time-sensitive tasks on shared memory multiprocessor: A framework suggestion. Master's thesis", Department of Computer Engineering, Chalmers University of Technology, January 1999.

[8] S. Lauzac, R. Melhem, and D. Mossé. "Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor". In 10th Euromicro Workshop on Real Time Systems, pages 188-195, Berlin, Germany, June 17-19, 1998.

[9] L. Lundberg. "Multiprocessor scheduling of age constraint processes. In 5th International Conference on Real Time Computing Systems and Applications", Hiroshima, Japan, October 27-29, 1998.

[10] C. L. Liu. "Scheduling algorithms for multiprocessor in a hard real-time environment". In JPL Space Programs Summary 37-60, volume II, pages 28-31, 1969.

[11] 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.

[12] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "A Linear Time Online Task Assignment scheme for multiprocessor systems", Proc. 11th IEEE Workshop Real-Time Operating Systems and Software, pp. 28-31, May 1994.