Evaluación Experimental de Estrategias de Calendarización en Grid Computacional Utilizando un Esquema de Admisibilidad

[1] José Luis González García, [1]Andrei Tchernykh, [2] Ramin Yahyapour
[1] *Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada, B.C.*
{jlgonzal, chernykh}@cicese.mx,
[2] *IT and Media Center, Dortmund University of Technology, 44221 Dortmund, Germany*
ramin.yahyapour@udo.edu

Resumen

El paradigma de computación en Grid introduce nuevos y difíciles problemas en calendarización y manejo de recursos. Los métodos de calendarización tradicional no satisfacen las necesidades actuales para la administración de recursos en un Grid. En este artículo analizamos estrategias de calendarización en Grid computacional de dos niveles utilizando un esquema de admisibilidad. En el primer nivel, el metacalendarizador selecciona una máquina del conjunto de máquinas admisibles para cada tarea según algún criterio. En el segundo nivel, el calendarizador local crea el calendario de ejecución para las tareas asignadas. Una razón de la ineficiencia de la asignación de trabajos en este modelo puede ser la ocupación de las máquinas grandes por trabajos con un grado de paralelismo pequeño, causando que los trabajos altamente paralelos esperen por su ejecución. El esquema de admisibilidad es un concepto simple y fácil de aplicar, basta con evitar la asignación de las tareas pequeñas a máquinas grandes. Mostramos resultados experimentales comparando el desempeño del esquema de admisibilidad contra algoritmos conocidos. Demostramos que un ajuste del rango de admisibilidad, independiente para cada estrategia, ofrece mejores resultados.

1. Introducción

Los problemas de calendarización se pueden ver, en general, como problemas de asignación de recursos sobre el tiempo para ejecutar un conjunto de tareas como parte de algún proceso. Las tareas tienen sus propias características como tamaño (número de procesadores requeridos), tiempo de llegada, tiempo de procesamiento estimado, tiempo de procesamiento real, etc. De esta manera se generan calendarios que indican, para cada tarea, cuándo y qué conjunto de procesadores la ejecuta. Además existen diferentes criterios de optimización al generar los calendarios, como lo son la minimización de la longitud del calendario en la que se termina la ejecución de todas las tereas, la minimización del tiempo de espera de cada tarea, etc.

El problema de calendarización no es nuevo. Se han hecho estudios para sistemas distribuidos y paralelos tales como máquinas con múltiples procesadores simétricos (SMP), máquinas con procesadores paralelos masivos (MPP) y *cluster* de estaciones de trabajo (COW). Los algoritmos de calendarización han evolucionado a la par de las arquitecturas de sistemas de cómputo.

El paradigma de computación en Grid [1] introduce nuevos y difíciles problemas en calendarización y manejo de recursos [2]. Un Grid se puede ver como un acuerdo para compartir recursos entre un número de organizaciones independientes (como laboratorios o universidades), con poco o ningún control administrativo central, forzándolos a interactuar [3]. Una organización es una entidad administrativa que agrupa usuarios y recursos computacionales. Las organizaciones son libres de unirse o dejar el sistema, si la experiencia adquirida es menor que el costo de la participación. Sin embargo, para sostener el Grid, el sistema de administración de recursos debe lograr un desempeño aceptable no sólo al nivel de la comunidad de usuarios (como en la calendarización clásica de un solo criterio), sino también al nivel de las organizaciones.

Sin embargo, al intentar compartir una amplia variedad de recursos computacionales distribuidos geográficamente (tal como supercomputadoras, *clusters*, sistemas de almacenamiento, fuentes de datos, instrumentos) y presentarlos como un solo recurso unificado, surgen algunos problemas de investigación interesantes como: seguridad (ya sea de la información o del propio sistema), descubrimiento y agregación dinámicos (la cantidad de recursos en el sistema es

dinámica, en cualquier momento puede unirse o salir un recurso), y calidad de servicio entre otros [4].

Los modelos de calendarización tradicional generalmente producen calendarios pobres en ambientes Grid. Esto se debe a las suposiciones utilizadas en la calendarización tradicional [5] como homogeneidad de los recursos, arquitectura de calendarización centralizada, interconexiones de red de alta velocidad, criterios únicos de optimización, etc. Estas suposiciones desafortunadamente no se pueden aplicar a un ambiente Grid por sus características, muchas de ellas únicas. En consecuencia, el diseño de algoritmos de calendarización para Grid computacional es más complejo [6].

Se han propuesto e implementado varios sistemas de calendarización, entre ellos se encuentran Condor [7], LSF [8], Gridbus [9], Globus [10], Legion [11], entre otros.; sin embargo, estos sistemas se basan en modelos de calendarización tradicional. Estos proyectos desarrollan un software llamado *middleware* que administra los recursos del Grid computacional, permite a los usuarios no preocuparse de la infraestructura del Grid y solamente someter la tarea al sistema.

Recientemente se propuso el esquema de admisibilidad [12] que tiene como objetivo mejorar el desempeño del Grid computacional al tratar un problema en específico: el retraso en la ejecución de tareas grandes debido a la ocupación de las máquinas por tareas pequeñas. Los autores presentan un análisis teórico del esquema de admisibilidad y, muy importante, demuestran que en algunos casos provee una cota de competitividad constante para algunos algoritmos. Sin embargo, no proporcionan un análisis experimental del mismo. Si bien este esquema es muy eficiente según el análisis teórico, se requiere de resultados experimentales para determinar el caso promedio y verificar su comportamiento al aplicarlo en un sistema real.

2. Definición del problema

En esta investigación se trata el problema de calendarización en línea con el objetivo de minimizar la longitud del calendario: n tareas paralelas $(J_1, J_2, \dots, J_n)$ se deben calendarizar en m máquinas paralelas $(M_1, M_2, \dots, M_m)$; se denota al número de procesadores idénticos de la máquina M_i como m_i. Sin pérdida de generalidad, se supone que las máquinas paralelas se encuentran ordenadas en forma no decreciente por su tamaño ($m_1 \leq m_2 \leq \cdots \leq m_m$).

La tarea J_j está representada por una tripleta $(r_j, size_j, p_j)$: el tiempo de su llegada $r_j \geq 0$, su tamaño $1 \leq size_j \leq m_m$ que se refiere a su grado de paralelismo, y su tiempo de ejecución p_j. El tiempo de llegada de cada tarea no se conoce hasta que es sometida al sistema (calendarización en línea), y su tiempo de ejecución es desconocido hasta que termina su procesamiento (calendarización no clarividente).

El presente trabajo supone que la tarea J_j sólo se puede ejecutar en la máquina M_i si $size_j \leq m_i$. También se supone una calendarización en modo de espacio compartido[1], aplicado en muchas computadoras paralelas. Además, una tarea paralela J_j se ejecuta en exactamente $size_j$ procesadores disjuntos sin interrupción.

Finalmente, $g(J_j) = M_i$ denota que la tarea J_j se asigna a la máquina M_i. Sea n_i el número de tareas asignadas a la máquina M_i.

Todos los algoritmos se analizan de acuerdo al factor de competitividad (cuántas veces se aleja del óptimo) para la longitud del calendario. Sea C^*_{max} y $C_{max}(A)$ las longitudes de los calendarios óptimo y el determinado por el algoritmo A, respectivamente. El factor de competitividad del algoritmo A se define como $\rho_A = max\left(\frac{C_{max}(A)}{C^*_{max}}\right)$ para todas las instancias del problema.

La notación GP_m describe el modelo de Grid utilizado. En la notación corta de tres campos [13] se describe como $GP_m|r_j, size_j|C_{max}$ [14]. Para referirse a este problema se utiliza el término MPS, y PS para la calendarización de tareas paralelas en una sola máquina paralela ($P_m|r_j, size_j|C_{max}$) [15].

3. Admisibilidad

Para el esquema de admisibilidad [16] se define $f(j) = first(j)$ como el índice i más pequeño tal que $m_i \geq size_j$ para la tarea J_j. También se define $l(j) = last(j)$ como el índice i más grande tal que $m_i \geq size_j$ para la tarea J_j. Debido a la restricción $size_j \leq m_m \; \forall J_j$, se tiene que $l(j) \leq m$. El conjunto de máquinas $M_{available}(j)$ son las que están disponibles para ejecutar la tarea J_j y corresponde al conjunto de índices de las máquinas $s(f(j), m) = \{f(j), f(j)+1, f(j)+2, \dots, m\}$. El conjunto completo de máquinas

[1] *Espacio compartido*: Las tareas son asignadas a un subconjunto de procesadores de una máquina paralela de manera exclusiva, pero la máquina (su "espacio") es compartida. Esto es, diferentes tareas paralelas pueden potencialmente ejecutarse al mismo tiempo si los requerimientos de las tareas permiten su asignación a diferentes subconjuntos de procesadores de esa máquina. El "espacio" se calcula al multiplicar el número de procesadores por el tiempo.

M_{total} se representa por el conjunto de enteros $s(1,m) = 1,\dots,m$. El número total de procesadores que se encuentran en las máquinas m_f a m_l es $m(f,l) = \sum_{i=f}^{l} m_i$.

El conjunto $M_{admissible}(j)$ de máquinas admisibles para la tarea J_j es el conjunto de máquinas con índices en el conjunto $s\big(f(j),r(j)\big)$, donde $r(j)$ es el número mínimo que cumple $m\big(f(j),r(j)\big) \geq \frac{1}{2}m(f(j),m)$ [12]. La definición se generaliza introduciendo un parámetro nuevo $0 \leq a \leq 1$ que configura el grado de admisibilidad usado para la asignación de tareas [15]. Por lo tanto, $s\big(f(j),r(j)\big)$ es el conjunto de índices de máquinas admisibles si $r(j)$ es el índice mínimo tal que $m\big(f(j),r(j)\big) \geq a \cdot m(f(j),m)$. Cuando $a = \frac{1}{2}$ (o 50% en el grado de admisibilidad) se genera la definición original [12]. Cuando $a = 1$ (o 100% en el grado de admisibilidad) equivale al algoritmo original sin utilizar el esquema de admisibilidad.

El análisis teórico de este esquema de admisibilidad utiliza *Min_LB* como estrategia MPS_{Alloc} (nivel de asignación de recursos) y define *Best_PS* como el mejor algoritmo conocido de calendarización local con un factor de competitividad de $2 - \frac{1}{m}$.

Teorema 1. [16] Supóngase un conjunto de máquinas con procesadores idénticos, un conjunto de tareas rígidas (aquellas tareas que solicitan un número de procesadores que no puede cambiar bajo ninguna circunstancia) y un rango de asignación admisible $0 \leq a \leq 1$. Entonces el algoritmo *Min_LB* + *Best_PS* tiene un factor de competitividad

$$\rho \leq \begin{cases} 1 + \dfrac{4}{a^2} & para\ a \leq \dfrac{m(f(d),m)}{m(f(b),m)} \\ 1 + \dfrac{4}{a(1-a)} & para\ a > \dfrac{m(f(d),m)}{m(f(b),m)} \end{cases} \quad (1)$$

con $1 \leq f(b) \leq f(d) \leq m$ siendo parámetros que dependen de la máquina y la carga de trabajo (ver Figura 1). Donde J_d es la última tarea asignada a la máquina que define la longitud del calendario del Grid y J_b la tarea más pequeña ejecutada en el conjunto $M_{available}(d)$.

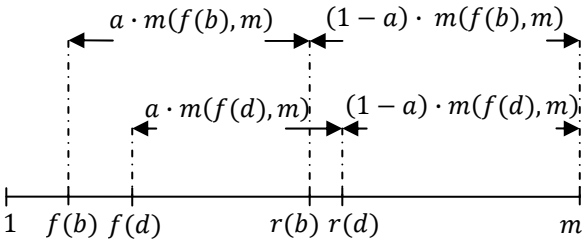

Figura 1. Asignación admisible con factor *a*.

La Figura 2 muestra el límite del factor de competitividad del algoritmo *Min_LB* + *Best_PS* para el valor de admisibilidad $a \leq \frac{m(f(d),m)}{m(f(b),m)}$ y la Figura 3 para $a > \frac{m(f(d),m)}{m(f(b),m)}$. Se puede observar que si $a \leq \frac{m(f(d),m)}{m(f(b),m)}$ el límite para el peor caso cambia de ∞ a 5 como función del valor de admisibilidad $0 < a \leq 1$. Si $a > \frac{m(f(d),m)}{m(f(b),m)}$ los límites para el peor caso cambian de ∞ a ∞ con un mínimo para $a = 0.5$.

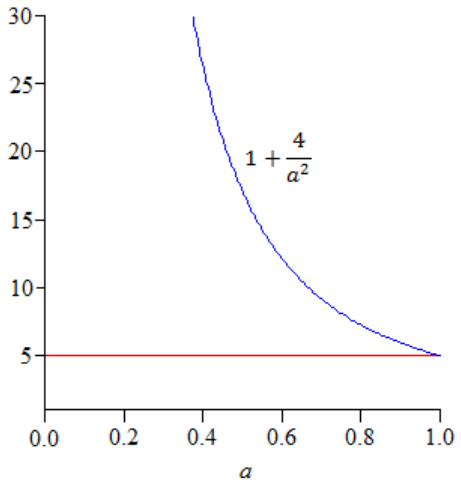

Figura 2. Factor de competitividad del algoritmo *Min_LB* + *Best_PS* para el valor de admisibilidad $a \leq \frac{m(f(d),m)}{m(f(b),m)}$.

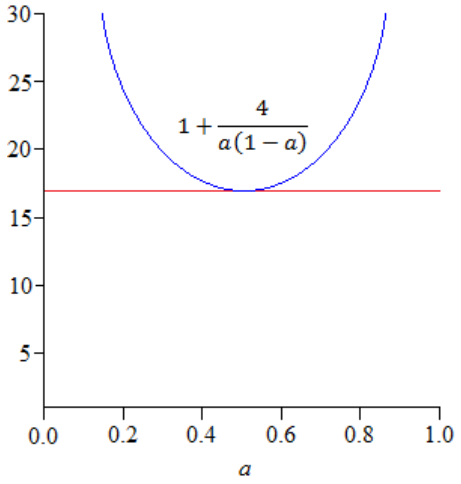

Figura 3. Factor de competitividad del algoritmo *Min_LB* + *Best_PS* para el valor de admisibilidad $a > \frac{m(f(d),m)}{m(f(b),m)}$.

La Figura 4 muestra el límite resultante de los límites del peor caso presentados en la Figura 2 y la Figura 3, como función del valor de admisibilidad

$0 < a \leq 1$. Observe que los límites producen $\rho = 17$ para $a = 0.5$.

4. Calendarización jerárquica de dos niveles

El modelo jerárquico considera dos niveles de calendarización. En el primer nivel se lleva a cabo una asignación de tareas a máquinas (selección de recursos) utilizando un esquema de admisibilidad, en el segundo nivel se considera un calendarizador local por cada una de las máquinas que integran el Grid como se muestra en la Figura 5.

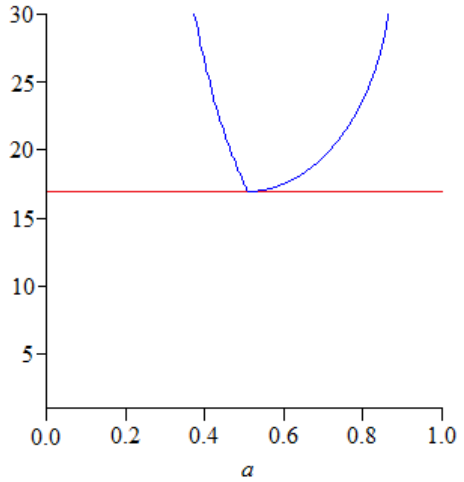

Figura 4. Factor de competitividad del algoritmo *Min_LB + Best_PS*.

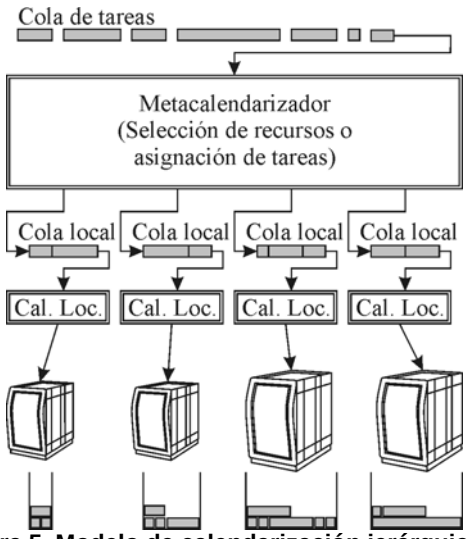

Figura 5. Modelo de calendarización jerárquica de dos niveles.

La asignación de tareas a máquinas consiste en aplicar alguna estrategia que selecciona la máquina más adecuada dentro de un conjunto de máquinas disponibles para ejecutar dicha tarea. Este conjunto de máquinas disponibles se reduce de acuerdo al esquema de admisibilidad utilizado. Esta reducción intenta evitar la asignación de tareas pequeñas a máquinas grandes, provocando que se retrasen las tareas grandes.

Durante la calendarización local, las tareas enviadas a cada máquina se asignan a los procesadores que habrán de ejecutarlas. En esta etapa es donde se crean los calendarios definitivos de ejecución.

Se utilizan catorce estrategias de asignación de tareas [17]:

1. *Random*. Selecciona la máquina de manera aleatoria.

$$f(j) \leq random(\quad) \leq r(j) \tag{2}$$

2. *Min_Lp*. Selecciona la máquina con la mínima carga por procesador.

$$min_{f(j)\leq i\leq r(j)}\left(\frac{n_i}{m_i}\right) \tag{3}$$

3. *Min_PL*. Selecciona la máquina con la mínima carga paralela por procesador.

$$min_{f(j)\leq i\leq r(j)}\left(\sum_{g(J_k)=M_i}\frac{size_k}{m_i}\right) \tag{4}$$

4. *Min_LBal*. [18] Selecciona la máquina que genera la menor desviación estándar en la distribución de la carga de cada máquina del Grid.

$$min_{f(j)\leq i\leq r(j)}\sqrt[2]{\frac{\sum_{f(j)\leq i\leq r(j)}(PL_i-\overline{PL})^2}{r(j)-f(j)+1}} \tag{5}$$

donde $PL_i = \sum_{g(J_k)=M_i}\frac{size_k}{m_i}$ es la carga paralela de la máquina M_i y $\overline{PL} = \sum_{f(j)\leq i\leq r(j)}\frac{PL_i}{r(j)-f(j)+1}$ es la media de las cargas paralelas del conjunto de máquinas admisibles.

5. *Min_LB*. Selecciona la máquina con la menor cota inferior de tiempo de terminación de las tareas. En lugar del tiempo real de ejecución de una tarea, que no está disponible en calendarización no clarividente, se usa el valor proporcionado por el usuario al someter la tarea al sistema (tiempo solicitado).

$$min_{f(j)\leq i\leq r(j)}\left(\sum_{g(J_k)=M_i}\frac{size_k\cdot p_k}{m_i}\right) \tag{6}$$

6. *Min_CT*. Se elige la máquina con el menor tiempo de terminación estimado de todas sus tareas asignadas.

$$min_{f(j)\le i\le r(j)}(C^i_{max}) \quad (7)$$

donde $C^i_{max} = max_{g(J_k)=M_i}(C^i_k)$ es el tiempo máximo de terminación de las tareas en la máquina M_i y C^i_k es el tiempo de terminación de la tarea J_k en la máquina M_i.

7. *Min_SWCT*. Se elige la máquina con la menor suma del producto del tamaño con el tiempo de terminación estimado de cada una de las tareas asignadas.

$$min_{f(j)\le i\le r(j)}\left(\sum_{g(J_k)=M_i} size_k \cdot C^i_k\right) \quad (8)$$

8. *Min_WT*. Estrategia que selecciona la máquina con el promedio menor de tiempo de espera de las tareas asignadas.

$$min_{f(j)\le i\le r(j)}\left(\sum_{g(J_k)=M_i} \frac{C^i_k - r_k - p_k}{n_i}\right) \quad (9)$$

9. *Min_WWT*. Se elige la máquina que tiene el menor promedio del producto del tamaño con el tiempo de espera de cada una de las tareas asignadas a la máquina.

$$min_{f(j)\le i\le r(j)}\left(\sum_{g(J_k)=M_i} \frac{(C^i_k - r_k - p_k)\cdot size_k}{n_i}\right) (10)$$

10. *Min_U*. Se elige la máquina con la menor utilización de sus procesadores.

$$min_{f(j)\le i\le r(j)}\left(\frac{W^i_{total}}{C^i_{max}\cdot m_i}\right) \quad (11)$$

donde $W^i_{total} = \sum_{g(J_k)=M_i} size_k \cdot p_k$ es el trabajo realizado por la máquina M_i.

11. *Min_ST*. Se selecciona la máquina que ofrece el menor tiempo de inicio de ejecución para la tarea a asignar.

$$min_{f(j)\le i\le r(j)}(C^i_j - r_j) \quad (12)$$

12. *Min_TA*. Se selecciona la máquina que tiene el promedio menor de tiempo de permanencia en el sistema de las tareas asignadas.

$$min_{f(j)\le i\le r(j)}\left(\sum_{g(J_k)=M_i} \frac{C^i_k - r_k}{n_i}\right) \quad (13)$$

13. *Min_WTA*. [19] Se selecciona la máquina que tiene el promedio menor de tiempo de permanencia en el sistema de las tareas asignadas. Este tiempo lo pondera el tamaño de la tarea.

$$min_{f(j)\le i\le r(j)}\left(\sum_{g(J_k)=M_i} \frac{(C^i_k - r_k)\cdot size_k}{n_i}\right) (14)$$

14. *Min_WWoTA*. [19] Se selecciona la máquina que tiene el promedio menor de tiempo de permanencia en el sistema de las tareas asignadas. Este tiempo lo pondera el trabajo de la tarea.

$$min_{f(j)\le i\le r(j)}\left(\sum_{g(J_k)=M_i} \frac{(C^i_k - r_k)\cdot (p_k \cdot size_k)}{n_i}\right) (15)$$

Las estrategias de asignación de tareas se dividen en 5 niveles de conocimiento dependiendo de la información que utilizan relacionada a la tarea y de las máquinas del Grid computacional. La información utilizada en cada nivel incluye la información disponible en el nivel anterior.

- Nivel 0: Se conoce el número de máquinas y el tamaño de cada una de ellas. El grado de paralelismo de cada tarea se utiliza únicamente para conocer el conjunto de máquinas admisibles. La estrategia *Random* es un ejemplo de este nivel.
- Nivel 1: Adicionalmente a la información del nivel 0, se encuentra disponible la carga de tareas de cada máquina. La estrategia *Min_Lp* corresponde a este nivel de conocimiento.
- Nivel 2: Adicionalmente a la información del nivel 1, el grado de paralelismo de cada tarea se utiliza durante los cálculos de la estrategia. Las estrategias *Min_PL* y *Min_LBal* pertenecen a este nivel.
- Nivel 3. Además de la información disponible en el nivel 2, se considera calendarización clarividente, esto es, el tiempo de ejecución de cada tarea está disponible desde el momento en que ésta llega al sistema. La estrategia *Min_LB* pertenece a este nivel.
- Nivel 4: Se tiene acceso a toda la información del nivel 3 y también a todos los calendarios locales. Las estrategias *Min_CT*, *Min_SWCT*, *Min_WT*, *Min_WWT*, *Min_U*, *Min_ST*, *Min_TA*, *Min_WTA*, *Min_WWoTA* se encuentran en este nivel de conocimiento.

Los niveles 0 a 2 describen una calendarización no clarividente. Las estrategias de niveles 0 a 3 están basadas solamente en los parámetros de las tareas, y no necesitan información alguna de los calendarios locales.

Según resultados previos, los algoritmos para calendarización local más utilizados actualmente en sistemas paralelos reales son *FCFS* y *EASY* (Extensible Argonne Scheduling sYstem) [20]. Este

último es una implementación de *FCFS* con *Backfilling* [21].

En el presente trabajo se utiliza únicamente *FCFS* como estrategia de calendarización local. Es una de las estrategias más simples. El calendarizador toma las tareas en el orden en que van llegando al sistema. Si no hay suficientes recursos disponibles, la tarea espera hasta que se liberen recursos y pueda iniciarse mientras las demás tareas permanecen detenidas en la cola. Al utilizar *Backfilling*, cuando no hay suficientes procesadores disponibles para ejecutar la tarea que se encuentra al inicio de la cola de espera, el calendarizador selecciona otras tareas de la cola que se adapten a los procesadores disponibles sin que se afecte el tiempo de inicio de ejecución de la tarea al inicio de la cola.

5. Definición del experimento

La configuración del Grid computacional utilizado para las simulaciones es el resultado de un análisis extenso de la carga de trabajo, ver Tabla I. Esta carga de trabajo contiene tareas que requieren hasta 32 procesadores. Es la razón por la cual el tamaño máximo de las máquinas del Grid es 32. Generalmente las máquinas actuales tienen una cantidad de procesadores igual a una potencia de dos, por esta razón se seleccionaron de esta forma. Aun cuando el tamaño del *cluster* máximo del Grid es relativamente pequeño, se cree que es una imagen adecuada de un Grid computacional ya que las tareas se encuentran en la misma proporción.

La carga de trabajo contiene información suficiente para realizar 30 experimentos de 36,770 tareas cada uno. Esto equivale a 30 experimentos de las tareas correspondientes a una semana; durante la primera semana se sometieron 36,770 tareas, se supone la misma cantidad en las semanas subsecuentes. Un total de 1,103,100 tareas en la carga de trabajo, correspondientes a 467 días. Los detalles completos del experimento se muestran en la Tabla II.

Tabla I. Configuración del Grid utilizado para los experimentos.

Configuración del Grid	
Tamaño máximo de tarea permitido	32
Número de máquinas	11
Tamaño de las máquinas	4, 4, 4, 4, 8, 8, 8, 16, 16, 32, 32

Tabla II. Parámetros de los experimentos.

Parámetros de los experimentos	
Número de experimentos	30
Número de tareas por experimento	36,770
Niveles de calendarización	2
Tipo de calendarización	En línea
Selección de tareas de la cola de espera del Grid	FIFO
Selección de la máquina a la cual se asignará la tarea	*Random*, *Min_Lp*, *Min_PL*, *Min_LBal*, *Min_LB*, *Min_CT*, *Min_SWCT*, *Min_WT*, *Min_WWT*, *Min_U*, *Min_ST*, *Min_TA*, *Min_WTA*, *Min_WWoTA*
Calendarizador para el nivel de asignación de tareas (usado para las estrategias que requieren calendario tentativo)	FCFS
Grado de admisibilidad	0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%
Tiempo de ejecución estimado	Tiempo solicitado por usuario
Calendarización local	FCFS

Tabla III. Resultados experimentales al aplicar el esquema de admisibilidad a las diferentes estrategias de calendarización. En blanco se muestra la métrica del algoritmo, en gris claro las métricas del sistema y en gris oscuro las métricas del usuario.

Estrategia		*Random*	*Min_Lp*	*Min_PL*	*Min_LBal*	*Min_LB*	*Min_CT*	*Min_SWCT*	*Min_WT*	*Min_WWT*	*Min_U*	*Min_ST*	*Min_TA*	*Min_WTA*	*Min_WWoTA*
Nivel de conocimiento		0	1	2	2	3	4	4	4	4	4	4	4	4	4
Porcentaje de mejora de la métrica y el grado de admisibilidad en el que se obtuvo	Factor de competitividad	22.68 0.5	24.79 0.5	1.54 0.5	0.37 0.6	2.51 0.5	1.25 0.5	0.0 -	3.06 0.5	2.54 0.6	8.59 0.5	0.78 0.6	4.03 0.5	0.0 -	3.42 0.7
	Longitud del calendario	23.67 0.5	22.68 0.5	2.58 0.5	0.37 0.5	1.50 0.5	1.87 0.5	0.0 -	3.56 0.5	2.26 0.6	2.76 0.5	0.27 0.6	4.58 0.5	0.0 -	3.56 0.6
	Utilización del sistema	29.43 0.5	31.25 0.5	1.64 0.5	0.32 0.6	2.55 0.5	1.12 0.5	0.0 -	3.01 0.5	2.44 0.6	4.88 0.5	0.67 0.6	4.56 0.5	0.0 -	2.19 0.7
	Recursos no utilizados	37.30 0.5	35.95 0.5	6.18 0.5	0.86 0.5	3.38 0.5	5.35 0.5	0.0 -	7.60 0.5	4.20 0.6	4.23 0.5	0.80 0.6	9.67 0.5	0.0 -	5.65 0.6
	Rendimiento de procesamiento	25.62 0.5	34.68 0.5	0.0 -	0.00 0.6	3.14 0.5	0.51 0.5	0.0 -	1.19 0.5	3.35 0.5	12.31 0.5	1.57 0.6	3.11 0.6	0.0 -	0.73 0.7
	Tiempo promedio de permanencia	10.22 0.5	57.83 0.5	4.79 0.5	2.75 0.5	1.11 0.5	1.06 0.5	1.28 0.2	8.28 0.2	0.67 0.2	0.20 0.5	1.80 0.5	8.73 0.2	0.72 0.5	2.47 0.7
	Tiempo promedio de permanencia ponderado	29.81 0.5	36.06 0.5	4.78 0.5	2.46 0.5	4.13 0.5	1.41 0.5	0.49 0.6	7.09 0.5	0.81 0.5	2.28 0.5	0.67 0.5	6.61 0.5	0.41 0.6	3.99 0.5
	Tiempo promedio de permanencia ponderado por trabajo	30.58 0.5	29.87 0.5	4.31 0.5	1.91 0.5	4.22 0.5	1.83 0.5	0.74 0.6	7.13 0.5	0.53 0.6	5.07 0.5	0.76 0.5	6.70 0.5	0.39 0.6	6.35 0.6
	Tiempo promedio de espera	10.24 0.5	57.88 0.5	4.80 0.5	2.75 0.5	1.11 0.5	1.06 0.5	1.28 0.2	8.29 0.2	0.67 0.2	0.20 0.5	1.80 0.5	8.74 0.2	0.72 0.5	2.47 0.7
	Tiempo promedio de espera ponderado	29.84 0.5	36.09 0.5	4.79 0.5	2.46 0.5	4.13 0.5	1.41 0.5	0.49 0.6	7.10 0.5	0.81 0.5	2.28 0.5	0.67 0.5	6.62 0.5	0.41 0.6	4.00 0.5
	Tiempo de respuesta promedio	10.22 0.5	57.83 0.5	4.79 0.5	2.75 0.5	1.11 0.5	1.06 0.5	1.28 0.2	8.28 0.2	0.67 0.2	0.20 0.5	1.80 0.5	8.73 0.2	0.72 0.5	2.47 0.7
	Cociente de respuesta promedio	5.44 0.5	65.75 0.5	5.68 0.5	4.08 0.5	0.37 0.6	1.06 0.5	7.32 0.2	12.80 0.2	6.18 0.2	8.02 0.5	1.78 0.5	12.96 0.2	0.50 0.5	1.69 0.7
	Cociente de respuesta promedio acotado	7.76 0.5	62.69 0.5	5.20 0.5	3.12 0.5	0.80 0.2	0.96 0.5	5.08 0.2	11.64 0.2	4.49 0.2	5.65 0.5	1.78 0.5	12.03 0.2	0.89 0.5	1.89 0.7
	Cociente de respuesta del sistema	9.07 0.5	59.34 0.5	6.85 0.5	4.62 0.5	2.33 0.5	1.29 0.5	2.40 0.2	10.05 0.2	2.20 0.2	9.54 0.5	1.77 0.5	9.95 0.2	0.81 0.6	2.80 0.7

6. Análisis experimental

La métrica utilizada para el análisis de las estrategias es la más citada en la literatura, *factor de competitividad*. Es la métrica del algoritmo más importante. Está definida como el cociente de la longitud del calendario generado por la estrategia de calendarización y la longitud del calendario óptimo. Sin embargo, dado la dificultad para obtener el valor de la longitud del calendario óptimo, se utiliza una cota mínima. De esta manera el factor de competitividad del algoritmo de calendarización se define como el

cociente de la longitud del calendario generado por la estrategia de calendarización y la cota mínima de finalización del calendario.

$$\frac{C_{max}}{C^*_{max}} \tag{16}$$

El factor de competitividad es la principal métrica tratada en este trabajo de investigación. El objetivo del esquema de admisibilidad es reducir la longitud del calendario, por consiguiente, reducir el factor de competitividad del algoritmo, incrementar la utilización y reducir la cantidad de recursos no utilizados. Los resultados experimentales muestran el comportamiento de las estrategias al utilizar el esquema de admisibilidad, así como los efectos en las diferentes métricas de análisis.

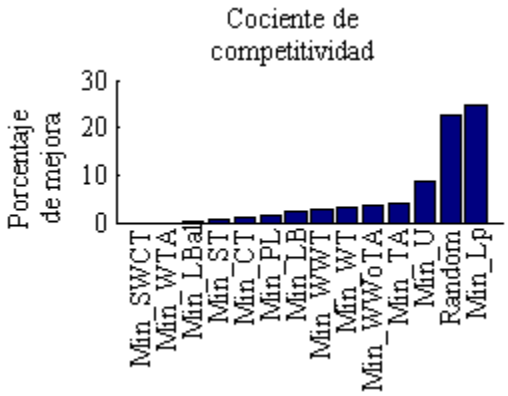

Figura 6. Porcentaje de mejora de los algoritmos, en el factor de competitividad, al utilizar el esquema de admisibilidad.

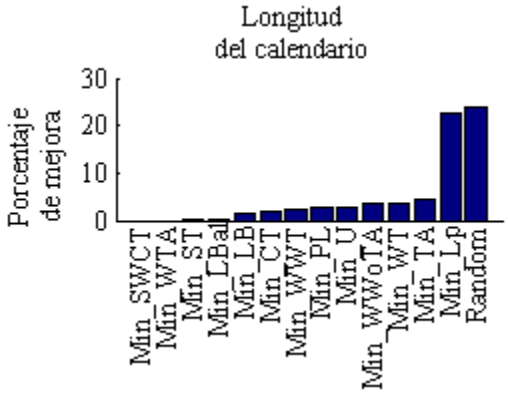

Figura 7. Porcentaje de mejora de los algoritmos, en la longitud del calendario, al utilizar el esquema de admisibilidad.

La Tabla III muestra un resumen de los resultados obtenidos durante la experimentación. Los porcentajes de mejora mostrados para cada métrica son los mayores obtenidos con los diferentes grados de admisibilidad utilizados. Algunas estrategias no presentan mejoría en ciertas métricas y en algunos casos una misma estrategia tiene mejoras en las diferentes métricas con diferentes grados de admisibilidad.

Al aplicar el esquema de admisibilidad se reducen los factores de competitividad de casi todas las estrategias analizadas. Las más afectadas son *Min_Lp* y *Random* con más de 20% de reducción. Esto significa que las longitudes de los calendarios se reducen en una proporción similar (ver Figura 7). Las dos estrategias que se mantienen invariantes son *Min_SWCT* y *Min_WTA*, (ver Figura 6).

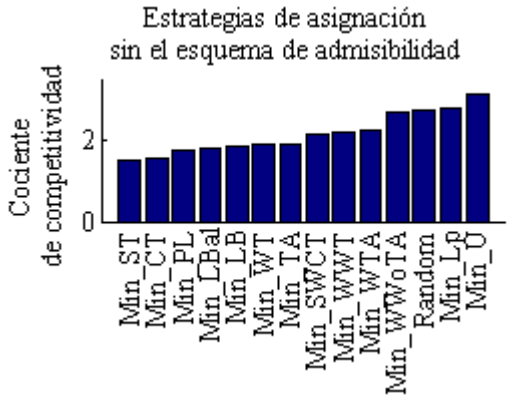

Figura 8. Factor de competitividad de las estrategias de asignación sin el esquema de admisibilidad.

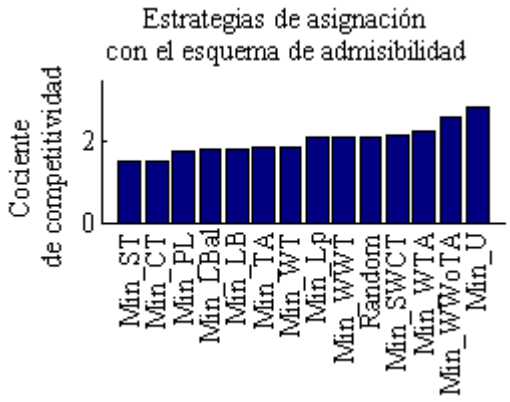

Figura 9. Factor de competitividad de las estrategias de asignación con el esquema de admisibilidad.

Las estrategias que presentan los mejores factores de competitividad sin el esquema de admisibilidad se mantienen estables después de aplicarlo; *Min_ST*, *Min_CT*, *Min_PL*, *Min_LBal* y *Min_LB*. Algunas estrategias con factores de competitividad mayores muestran un caso diferente; *Min_Lp* pasa de ser la penúltima peor estrategia a la octava, baja 5 posiciones; *Random* baja 2 posiciones. *Min_U*, la peor estrategia, sigue sin cambios. La Figura 8 muestra los

factores de competitividad de las estrategias antes de aplicar el esquema de admisibilidad y la Figura 9 los muestra después de haberlo aplicado.

Las dos estrategias que se benefician más con el esquema de admisibilidad son *Min_Lp* y *Random*, en algunas métricas con mejoras de hasta 60%. Por el contrario, *Min_WTA* es la estrategia que menos efectos tiene.

7. Conclusiones y trabajo futuro

El esquema de admisibilidad es un concepto muy simple y fácil de aplicar, basta con evitar la asignación de las tareas pequeñas a máquinas grandes. La consideración original es de restringir el 50% de los procesadores disponibles para una tarea. Se muestra que este porcentaje es el que más se acerca al óptimo si se generaliza para todas las estrategias de asignación de recursos. Sin embargo, algunas estrategias trabajan mejor con otro porcentaje. Un mejor ajuste, independiente para cada estrategia y métrica, ofrece mejores resultados como se aprecia en la Tabla III.

Se redujo la longitud de los calendarios generados; mejoras de hasta un 25% al aplicar el esquema de admisibilidad durante la etapa de selección de recursos. Este porcentaje varía dependiendo de la estrategia seleccionada, siendo *Min_Lp* la que obtiene el mayor beneficio. Las mejores estrategias sin el esquema de admisibilidad se mantienen estables al aplicarlo, aunque sí presentan mejoría. La hipótesis inicial del comportamiento del esquema de admisibilidad era que mejoría los calendarios de todas las estrategias. Este no es el caso, ya que dos estrategias, *Min_SWCT* y *Min_WTA*, no presentan mejora alguna. Se observa que todas las estrategias que utilizan ponderación generan resultados más pobres comparadas con sus versiones no ponderadas.

Las dos estrategias que no utilizan el tamaño de cada tarea para asignarlas (nivel 0 y 1 de conocimiento) son las que obtienen mejores beneficios del esquema de admisibilidad. La razón de esta situación es la introducción de la selección de máquinas dependiendo del tamaño de la tarea. Al asignar las tareas a máquinas de manera indiscriminada se presenta más frecuentemente la situación en la que tareas grandes esperan por su ejecución; situación en la que se enfoca el esquema de admisibilidad. Se presenta un caso diferente con las estrategias correspondientes a los niveles 2 al 5 cuando se aplica el esquema de admisibilidad. Estas estrategias utilizan el tamaño de la tarea para seleccionar la máquina que la ejecutará y la nueva selección del esquema de admisibilidad no tiene el mismo efecto. El tamaño de la tarea juega un papel importante en la calendarización, además la única información relacionada a las tareas que utiliza el esquema de admisibilidad.

En algunos casos, el utilizar el esquema de admisibilidad provee un factor de competitividad constante para algunas estrategias como *Min_LB* y *Min_CT*. Esto es muy importante porque asegura que la longitud de los calendarios no se alejará del óptimo en más de cierto rango. Aunque la estrategia genere calendarios buenos en promedio, siempre existe la posibilidad de la generación de un calendario extremadamente malo, y algunas veces sin ningún límite conocido. Esto no sucede con un factor de competitividad constante.

Finalmente, se muestra que el esquema de admisibilidad es un concepto simple y de fácil implementación. Puede ayudar a mejorar el desempeño del Grid computacional si se configura adecuadamente el grado de admisibilidad. Además, para los sistemas que tienen como objetivo minimizar criterios del usuario, es una buena opción ya que reduce los tiempos de espera de las tareas. Sin embargo, al aplicarlo incorrectamente puede dar resultados no deseados; un grado de admisibilidad muy pequeño generalmente incrementa la longitud de los calendarios.

Como trabajo futuro se pretende continuar con el estudio del esquema de admisibilidad al utilizar otras configuraciones de Grid. Se cree que este esquema es sensible a la configuración del sistema y de la carga de trabajo. De esta manera se desea conocer el grado de impacto que tiene el Grid computacional y la carga de trabajo en el esquema de admisibilidad.

Otro aspecto interesante es el comportamiento de otras estrategias de calendarización al utilizar el esquema de admisibilidad. Se desea introducirlas al simulador para estudiarlas de manera experimental.

Este trabajo muestra resultados experimentales con un grado de admisibilidad constante durante la calendarización. Pero no necesariamente es la mejor opción, si la carga de trabajo es principalmente secuencial, entonces un grado de admisibilidad alto es mejor. En cambio, para una carga de trabajo principalmente paralela un grado de admisibilidad menor es el adecuado. El adaptar este grado durante la calendarización podría generar mejores resultados. Esta variación depende de la carga de trabajo y del estado del sistema al momento de calcular el mejor valor. Se planea introducir este esquema de admisibilidad adaptable al simulador para realizar un análisis experimental y se esperan mejores resultados en la calendarización.

Referencias

[1] I. Foster and C. Kesselman, "Concepts and Architecture," in *The Grid 2. Blueprint for a New Computing Infrastructure*, Segunda ed. San Francisco: Morgan Kaufmann, 2004, pp. 37-64.

[2] F. Pascual, K. Rzadca, and D. Trystram, "Cooperation in multi-organization scheduling," in *Euro-Par 2007 Parallel Processing In conjunction with the First CoreGRID*, Rennes, France, 2007, 28-31 agosto.

[3] I. Foster. (2002, julio) Ian Foster Homepage. [Online]. http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf

[4] I. Foster and C. Kesselman, "The Grid in a Nutshell," in *GRID RESOURCE MANAGEMENT State of the Art and Future Trends*. Norwell, Massachusetts: Kluwer Academic Publisher, 2003, pp. 3-14.

[5] F. Berman, "High-performance schedulers," in *The GRID. Blueprint for a New Computing Infrastructure*. San Francisco: Morgan Kaufmann Publishers, 1998, pp. 279-310.

[6] Y. Zhu, "A Survey on Grid Scheduling Systems," Hong Kong University of Science and Technology, Kowloon, Hong Kong, Reporte técnico 2004.

[7] The University of Wisconsin Madison. (1988, enero) The Condor Project. [Online] http://www.cs.wisc.edu/condor/

[8] Platform Computing Inc. (2008, junio) Platform Computing Corporation. [Online] http://www.platform.com/

[9] The University of Melbourne. (2004, junio) The Gridbus project. [Online]. http://www.gridbus.org/

[10] The University of Chicago. (1997, enero) The Globus Alliance. [Online]. http://www.globus.org/

[11] The University of Virginia. (1993, agosto) The Legion Project. [Online]. http://legion.virginia.edu/

[12] A. Tchernykh et al., "Two Level Job-Scheduling Strategies for a Computational Grid," *Lecture Notes in Computer Science*, vol. 3911, no. 2006, pp. 774-781, 2006.

[13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Annals of Discrete Mathematics*, vol. 5, no. 1979, pp. 287-326, 1979.

[14] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour, "Online Scheduling in Grids," in *IPDPS 2008 Conference*, Miami, Florida, USA, 2008, 14-18 de abril.

[15] A. Tchernykh, U. Schwiegelshohn, R. Yahyapour, and N. Kuzjurin, "Online hierarchical job scheduling on Grids," in *From Grids To Service and Pervasive Computing*. Berlin: Springer, 2008, pp. 77-91.

[16] A. Tchernykh, U. Schwiegelshohn, R. Yahyapour, and N. Kuzjurin, "Online Hierarchical Job Scheduling on Grids with Admissible Allocation," *Journal of Scheduling*, Springer-Verlag, Netherlands, 2010, DOI: 10.1007/s10951-010-0169-x.

[17] D. Madrigal et al., "Estrategias de Calendarización para un Grid Computacional," in *CiComp'06, Primer Congreso Internacional de Ciencias Computacionales*, Ensenada, Baja California, México, 2006, 6 - 8 de noviembre.

[18] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Węglarz, "A multicriteria approach to two-level hierarchy scheduling in grids," *Journal of Scheduling*, vol. 11, no. 5, pp. 371-379, 2008.

[19] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz, *Handbook on Scheduling: From Theory to Applications*. Berlin: Springer-Verlag, 2007.

[20] Y. Etsion and D. Tsafrir, "A short survey of commercial cluster batch schedulers," School of Computer Science and Engineering, the Hebrew University, Jerusalem, Israel, Reporte técnico 2005-13, 2005.

[21] David A. Lifka, "The ANL/IBM SP scheduling system," in *Workshop on Job Scheduling Strategies for Parallel Processing in conjunction with IPPS '95*, Santa Barbara, CA, USA, 1995, 25 de abril.

[22] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Primera edición ed. San Francisco: Morgan Kaufmann, 1998.

[23] EC-cofunded GridTalk project. (2008, mayo) Grid Café. [Online] http://gridcafe.web.cern.ch/gridcafe/index.html