TESIS DEFENDIDA POR

Ariel Arturo Quezada Pina

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Andrey Chernykh

Director del Comité

Dr. Carlos Brizuela Rodríguez

Miembro del Comité

Dr. Vitaly Kober

Miembro del Comité

Dr. Ramin Yahyapour

Miembro del Comité

Dr. José Antonio García Macías

Coordinador del programa de posgrado
en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales

Director de Estudios de Posgrado

20 de septiembre de 2012

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

Estrategias de calendarización en línea para modelos jerárquicos y distribuidos de Grid computacional

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

DOCTOR EN CIENCIAS

Presenta:

Ariel Arturo Quezada Pina

Ensenada, Baja California, México,

Resumen de la tesis de Ariel Arturo Quezada Pina, presentada como requisito parcial para la obtención del grado de DOCTOR EN CIENCIAS en Ciencias de la Computación. Ensenada, Baja California.

Estrategias de calendarización en línea para modelos jerárquico y descentralizado de grid computacional

Resumen aprobado por:

_____

Dr. Andrey Chernykh
Director de Tesis

En esta tesis se proponen soluciones para el problema de calendarización en línea en un Grid computacional. El problema se aborda utilizando arquitecturas jerárquica de dos niveles y descentralizada de un nivel. Para el modelo de dos niveles se utiliza el esquema denominado de admisibilidad. Se presenta un análisis teórico y experimental de este esquema de admisibilidad. Para el modelo de un nivel se utilizaron estrategias basadas en migración de tareas Se implementó el algoritmo de robo de tareas el cual tiene el mejor desempeño teórico para el modelo utilizado y sólo había sido analizado de forma teórica anteriormente. Se implementaron distintas versiones del algoritmo y se comparó experimentalmente su desempeño con otros algoritmos de calendarización. Se realizaron simulaciones utilizando tres escenarios de Grid para el modelo de dos niveles y dos escenarios para el modelo de un solo nivel. Se elaboraron cargas de tareas de Grid a partir de cargas de trabajo reales de sitios de súpercómputo distribuidos en el mundo para simular cargas de trabajo de Grid para cada uno de los escenarios. Se utilizó una metodología de evaluación de desempeño en la cual se consideraron tres parámetros de desempeño no correlacionados. Considera que las métricas tienen la misma importancia y se elabora una jerarquía con las distintas estrategias para cada uno de los modelos, considerando los resultados de los distintos escenarios donde se probaron las estrategias. Para el modelo jerárquico de dos niveles, se provee de un análisis teórico estableciendo cotas de aproximación para el esquema de admisibilidad con las estrategias MLBa+PS y MCTa+PS, para el caso en línea y fuera de línea. Las cotas son para dos tipos de carga de trabajo: donde las tareas tienen a lo más el tamaño de la máquina más pequeña del Grid y donde las tareas tienen hasta el tamaño de la máquina más grande del Grid. Los resultados experimentales demostraron que nuestras estrategias se benefician al incorporar el esquema de admisibilidad. También, que estrategias más simples de calendarización, mostraron mejor desempeño. Para el modelo de un solo nivel, se provee de un análisis experimental del algoritmo de robo de tareas y de su desempeño en comparación con otras estrategias. Los resultados indican que estrategias que utilizan una sola cola de tareas tienen mejor desempeño para los escenarios propuestos.

Palabras Clave: **Grid computacional**, **calendarización en línea**, **administración de recursos**

Abstract of the thesis presented by Ariel Arturo Quezada Pina as a partial requirement to obtain the DOCTOR OF SCIENCE degree in Computer Science. Ensenada, Baja California, México.

Online job scheduling strategies for hierarchical and decentralized computational grid models

Summary approved by:

_____

Dr. Andrey Chernykh

Thesis Director

In this thesis, solutions to the problem of online Grid computing scheduling are proposed. Two-level hierarchical and one-level decentralized architectures are used. The model is defined with jobs executed online, in a non-clairvoyant fashion; multisite execution, preemption and coallocation are not allowed. An admissibility scheme is proposed for the hierarchical two-levels model. A theoretical and experimental analysis is provided for the solutions proposed for this model. For the decentralized one level model, strategies that allow job migration between sites are analyzed. This model considers point-to-point and all-to-all instant communications between sites. The stealing algorithm with the best theoretical performance for this model is implemented to carry out the first experimental analysis known up to date. Different versions of the same algorithm are implemented and compared with other one level scheduling strategies.

Experiments include three Grid scenarios for the two level model and two scenarios for the one level model. Grid workloads are built from real supercomputing traces around the world for each scenario.

A performance evaluation methodology considers three uncorrelated metrics. Each metric has the same importance in the analysis, and strategies are ranked for each model based on these metrics.

For the two-level hierarchical model, a theoretical analysis is provided for strategies MLBa+PS and MCTa+PS which use admissible scheme, for online and offline cases. The obtained bounds consider two types of workload: jobs with sizes up to the smallest machine in the Grid, and jobs with sizes up to the biggest machine of the Grid. Results indicate the strategies benefit from admissible scheme.

For the one level model, results indicate that stealing performs efficient load balancing. However, strategies that use a single queue per machine perform better in our scenarios.

Keywords: **Computational Grid, online scheduling, resource management**

Dedicatorias

A mi madre, América Pina Palacios.

Agradecimientos

A Dios y mis padres, América Pina Palacios y Fernando Quezada Tavarez. A mis hermanos Fernando, Dora y Lilián. A mis "viejitas" María Elizabeth Palacios e Isabel Quezada Terrazas. A mi mejor compañera, consejera y amiga, con todo mi cariño, Ena del Carmen Gámez Balmaceda, TE Quiero mucho. A mi "Brother" Rogelio Cano Cetina.

A mi comité de tesis. Al Dr. Andrey por su paciencia y enseñanzas. Al Dr. Vitaly por sus consejos. Dr. Carlos Brizuela por todo su apoyo, consejos, sugerencias y su siempre buena disposición para hacer de este un mejor trabajo. Al Dr. Ramin por su tiempo invertido para la terminación de este proyecto.

A los Dres. Alberto Fernández, Gilberto López, Hugo Hidalgo, por sus enseñanzas dentro y fuera del aula; porque contribuyeron de forma especial a mi formación.

A CONACYT por su apoyo para la realización de estos estudios.

A CICESE, todo su personal quienes siempre tienen la mejor disposición y actitud de servicio. Empezando por Carito, Lydia y Jorge Soria, del departamento de computación. A Cecy y Lupita de biblioteca. A Ivonne "the" Best, incansable con su buen humor. Con mucho cariño a Normita, a Citlali por toda su ayuda y ni se diga de la "mediadora por la paz", Dolores. Muchas gracias a Ustedes que hacen que las cosas funcionen y uno lleve a buen puerto éstos proyectos.

A mis compañeros de CICESE. De plano son muchos, a todos Ustedes que también fueron escuela para mi.

A los Scouts, Grupo Capoeira Brasil especialmente Guaraná y Bruxinha, "forito" de electrónicos, legión extranjera, cruzrojeros, oceanólogos y ensenadenses por nacimiento o adopción, muchas gracias a todos.

Tabla de contenido

LISTA DE FIGURAS

LISTA DE TABLAS

Capítulo 1

Introducción

1.1 Computación en Grids

El término “Grid” fue introducido a mediados de los 90’s para denotar una infraestructura de cómputo distribuida para ingeniería y estudios científicos avanzados (Foster y Kesselman, 1999). En un Grid se pueden compartir redes de telecomunicaciones, capacidad de almacenamiento digital, sensores, licencias de software y poder de cómputo a través de dominios que se encuentran geográficamente distribuidos.

Schopf (2003) describe tres fases en el proceso de calendarización en un Grid: descubrimiento de recursos, selección de máquinas y ejecución de tareas. Este trabajo se enfoca en las últimas dos fases. Siguiendo una estrategia definida, una máquina es seleccionada para la ejecución de una tarea dada. La ejecución de la tarea es un proceso que tiene implícito la reservación de recursos en la máquina y la ejecución misma de la tarea hasta su terminación. La calendarización de la tarea en esta última fase se lleva a cabo por el calendarizador local. Aún cuando el calendarizador de Grid no tiene injerencia en las decisiones del calendarizador local, su desempeño se ve impactado por la estrategia local de calendarización.

Graham et al. (1979) introdujeron la notación de tres campos $(\alpha|\beta|\gamma)$ para los problemas teóricos de calendarización. Nuestro problema de calendarización puede representarse como $GP_m|r_j, size_j, p_j, p_j^{'}|C_{\max}$, con $C_{\max}$ como criterio de optimización. El campo α se refiere al modelo de máquina. Para nuestro problema, *P* significa que los procesadores son idénticos, distribuidos en *m* máquinas y forman el Grid *G*. El campo β se refiere a las características de las tareas que son calendarizadas, las cuales ya fueron especificadas anteriormente. El campo γ se refiere al conjunto de métricas que se desean optimizar. Para nuestro problema, el objetivo es disminuir *Cmax*.

El modelo de Grid utilizado en este trabajo fue propuesto por Schwiegelshohn et al. (2008). Este modelo es una extensión del muy conocido modelo propuesto por Graham (1969) el cual describe una máquina con procesadores idénticos que ejecutan tareas independientes, secuenciales y que son sometidas concurrentemente. Garey y Graham (1975) extendieron este modelo con tareas paralelas, independientes y rígidas con tiempos de procesamiento desconocidos. Naroska y Schwiegelshohn (2002) extendieron aún más este modelo al considerar que las tareas son sometidas a través del tiempo. Al final, el modelo presentado en Schwiegelshohn et al. (2008) supone que los procesadores disponibles se encuentran no en una sola máquina, pero en un conjunto de ellas y que las tareas no pueden ejecutarse en varias máquinas a la vez.

En esta tesis se realizó un análisis teórico de dos algoritmos llamados Cota Mínima con Asignación de Tarea Admisible (MLBa) y Tiempo de Terminación Mímimo con Asignación de Tarea Admisible (MCTa) bajo restricciones específicas y optimización del tiempo de terminación del calendario en el Grid; esto es, se considera la minimización del mayor tiempo de finalización de cualquier tarea en el sistema. Específicamente, se muestra que el tiempo de finalización del calendario obtenido por los algoritmos propuestos no excede el tiempo de finalización óptimo para cualquier caso del problema por más de un factor definido. A este factor se le llamará factor de competitividad para el caso en línea, y factor de aproximación para el caso fuera de línea.

Se lograron un factor de aproximación de 9 y un factor de competitividad de 11, mejorando y extendiendo los resultados de Tchernykh (2006). También se derivó un factor de aproximación de 3 y un factor de competitividad de 5, para el caso donde todas las tareas pueden ser ejecutadas por la máquina más pequeña del Grid, como en el problema discutido por Bougeret et al. (2010).

Existen pocos resultados teóricos para el problema de calendarización en Grid. Tchernykh et al. (2005) estudiaron el desempeño de varios algoritmos en un modelo jerárquico de dos niveles con el objetivo minimizar el tiempo de terminación de tareas. Ellos presentaron algoritmos fuera de línea con factor de aproximación de 10. Schewiegelshohn et al. (2008) presentaron el algoritmo con mejor desempeño para un problema de calendarización en

línea no clarividente, es decir, el tiempo de ejecución de cada tarea se desconoce hasta su terminación. Su algoritmo de robo de tareas tiene un factor de competitividad de 5 y factor de aproximación 3 para el problema $GPm|r_j,size_j,p_j|Cmax$ con un modelo de calendarización descentralizado. Tchernykh et al. (2010) presentaron su selección de recursos con admisibilidad adaptable en un modelo jerárquico de dos niveles. Esta estrategia puede ser implementada muy eficientemente y puede ser combinada con otra estrategia de calendarización.

Para demostrar la practicidad y competencia de los algoritmos propuestos, se llevó un estudio completo de su desempeño y derivaciones usando simulación. Se tomaron en cuenta diferentes características del problema que son críticos para la adopción práctica de los algoritmos de calendarización. Se utilizaron cargas de trabajo de Grid basadas en historiales reales de ejecución, se consideraron tres escenarios de Grid basados en sistemas heteregéneos de alto desempeño y se consideraron problemas de calendarización de tareas con requisitos de tiempos de ejecución no especificados, ya que solo estimaciones de tiempo de ejecución dadas por el usuario están disponibles en un ambiente de ejecución real.

1.1.2 Funciones objetivo

Las funciones objetivo son utilizadas para evaluar el desempeño de un algoritmo de calendarización respecto a un parámetro o conjunto de ellos. La mayoría de las funciones objetivo aplicadas a ambientes de Grid fueron heredadas de aquéllas utilizadas para evaluar sistemas paralelos distribuidos (Dong y Akl 2006). En este trabajo, se utilizaron tres tipos de funciones objetivo: centradas en el usuario, centradas en los recursos y de evaluación algorítmica.

Las funciones objetivo que son centradas en el usuario miden el desempeño del Grid enfocándose en la optimización de la ejecución de cada tarea en forma individual, que es sometida al Grid (Krauter et al., 2002). Una estrategia basada en este tipo de funciones asigna una tarea sin tomar en cuenta el resto de las tareas pendientes por asignar. Un

ejemplo es la estrategia de tiempo de espera mínimo (Minimun Start Time, *MST*), la cual elige una máquina para la ejecución de una tarea donde ésta tendrá el tiempo de inicio de ejecución menor.

Algoritmos de calendarización que utilizan funciones objetivo centradas en los recursos se enfocan en la optimización de los recursos disponibles para la ejecución de tareas (Krauter et al., 2002) y están relacionadas con la utilización de los mismos. Ejemplos de este tipo de funciones objetivo son utilización y capacidad de carga. Utilización se refiere a la proporción que existen entre el tiempo en el que los procesadores de un Grid se ocupan en ejecutar tareas y el tiempo total del calendario. Capacidad de carga se refiere a la cantidad de trabajos que son ejecutados por unidad de tiempo.

Las funciones objetivo de evaluación algorítmica están centradas en parámetros que permiten la evaluación teórica de algoritmos basados en ellos. Un ejemplo de este tipo de funciones objetivo es el coeficiente de desempeño. Este consiste en la proporción que existe entre el tiempo de finalización de calendario dado por un algoritmo y el tiempo de finalización óptimo del mismo problema. Esto se explica de forma detallada en la Sección 2.2

1.1.3. Análisis de competitividad

En su forma más simple, un problema de calendarización en línea tendrá al menos restricciones en los tiempos de sometimiento de tareas (Schmoys et al., 1991). Aun más, parte de la información de entrada de un algoritmo que resuelve este problema permanecerá desconocida. Por esta razón, no se puede encontrar un algoritmo que siempre produzca un calendario óptimo para todas las instancias del problema. Sin embargo, el desempeño de los algoritmos debe ser evaluado de alguna forma. Este proceso puede llevarse a cabo estableciendo una relación entre el resultado dado por el algoritmo del peor caso posible y el mejor resultado existente para el problema en el cual el algoritmo está siendo evaluado. Este es una técnica común aplicada para evaluar el desempeño de algoritmos de aproximación que se utilizan para resolver problemas NP-difíciles. En el

contexto de calendarización en línea, a este proceso se le conoce como análisis de competitividad (Schmoys et al., 1991).

Formalmente, sea *f(A,I)* la función objetivo de un calendario producido por el algoritmo *A* y el caso *I*. Aquí, *A* denota un algoritmo en línea y *f* denota una función objetivo que se trata de minimizar, por ejemplo tiempo de terminación de un calendario. Decimos que el algoritmo *A* es *c*-competitivo si $f(A,I) \leq c\ f(OPT,I)+b$ para cualquier entrada *I* con una constante *b*. Aquí, *OPT* denota la mejor solución para cada caso del problema *I* y *c* una constante que nos determina el factor de aproximación del algoritmo *A*.

Aún cuando uno de los objetivos de esta tesis es analizar algoritmos en línea, el análisis de competitividad de algoritmos fuera de línea es importante. Esto se debe a que Shmoys et al. (1991) probaron que un algoritmo fuera de línea, con factor de aproximación de *a*, tiene factor de competitividad a lo más de *2a* en la versión en línea del mismo algoritmo. Por lo tanto, es simple establecer una cota superior para un algoritmo en línea, si se ha determinado su factor de aproximación en la versión fuera de línea.

1.1.4 Arquitecturas de calendarización en Grid

De forma general, se pueden distinguir tres tipos de arquitecturas o modelos de calendarización en Grid: centralizadas, jerárquicas y descentralizadas.

En un modelo centralizado, solamente existe un calendarizador de las tareas que se ejecutan en el Grid. Es responsable de todas las decisiones en cuanto a asignación de tareas se refiere, como se muestra en la Figura 1. Este modelo tiene varias ventajas: es de fácil administración, son de implementación sencilla y de relativa facilidad de co-asignación de recursos. Entre las desventajas de este modelo se encuentran: la escasa o nula escalabilidad, ausencia de tolerancia a fallos de tipo monolítico* y la dificultad de establecer diferentes estrategias de calendarización en un solo sistema. También, el calendarizador se convierte en un cuello de botella pues todas las solicitudes de ejecución de tareas deben cruzar este punto. Este modelo es utilizado en Grids de pequeña escala y

algunos ejemplos de sus implementaciones son Condor y Globus MDS (Abraham et al., 2000).

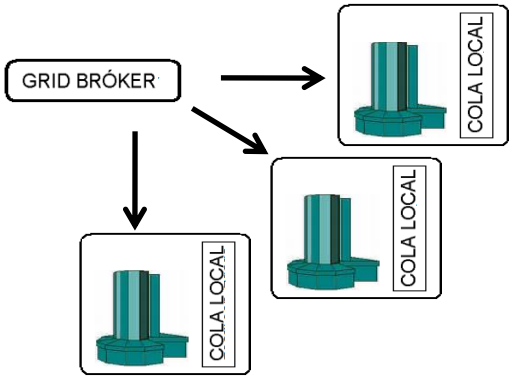


Figura 1.Arquitectura de calendarización centralizada

En un modelo jerárquico, los calendarizadores están organizados en niveles con relación de dependencia entre las mismas. En la Figura 2 se muestra un modelo jerárquico de tres niveles. Como característica general, componentes ubicados en niveles superiores tienen a su disposición mayor cantidad de recursos que los componentes de niveles inferiores y componentes de un mismo nivel, no tienen comunicación entre sí. Comparado con el modelo centralizado, este tipo de modelo aborda los problemas de escalabilidad y de tolerancia a fallas presentes en el modelo centralizado. También retiene algunas de las ventajas como son la co-asignación de tareas. Algunos calendarizadores conocidos que utilizan este esquema son 2K, Darwin y Legion (Abraham et al., 2000).

Las principales desventajas que presenta este tipo de modelo es que aún no provee autonomía a cada sitio que lo compone y múltiples políticas de calendarización. Estas desventajas son importantes para los Grid porque los distintos recursos que participan en un Grid pueden desear conservar el control sobre el uso de los mismos en distinto grado. Algunos recursos del Grid pueden no dedicarse de lleno a la ejecución exclusiva de aplicaciones del Grid. Aún más, esquemas de calendarización jerárquico deben tener la capacidad de tratar con políticas de uso dinámico de recursos.

*: se pierde la funcionalidad completa de un sistema por la falla en uno de sus componentes

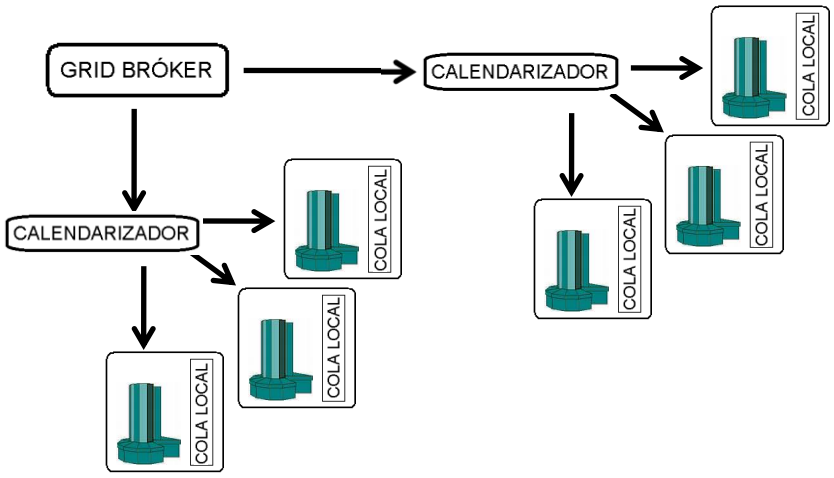


Figura 2. Arquitectura de calendarización jerárquica

Una tercera alternativa es el modelo descentralizado. De forma natural abordan cuestiones relevantes para el Grid como son la tolerancia a fallos, escalabilidad, autonomía de sitios y políticas múltiples de calendarización. Sin embargo, los modelos descentralizados introducen distintos problemas propios. Estos son referentes a la administración, utilización de recursos, rastreo y coasignación. Este tipo de esquema permite el crecimiento de grandes redes de recursos pero es necesario para los calendarizadores poder coordinarlos vía alguna forma de descubrimiento de recursos y protocolos de contrato de los mismos. Sin embargo, esto daría como resultado a un número muy grande de solicitudes causando retrasos pero no existe un punto común de fallo en el sistema.Un ejemplo de este tipo de modelo puede apreciarse en la Figura 3.

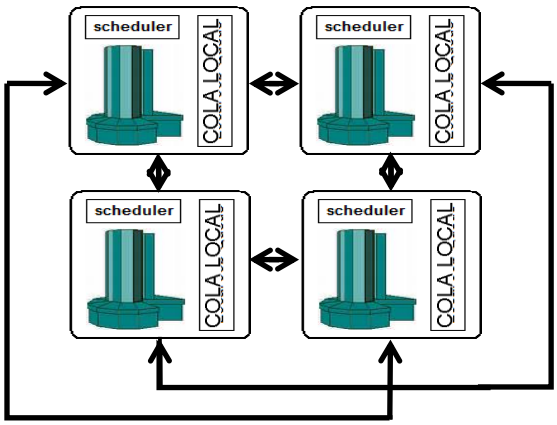


Figura 3. Arquitectura de calendarización descentralizada

1.2 Planteamiento del problema

Se han propuesto diferentes soluciones al problema de calendarización en línea para Grid. La mayoría de ellas provienen de estrategias heredadas de calendarización en máquinas paralelas. Sin embargo, las cotas obtenidas para dichas estrategias no se mantienen para el problema de Grid. La principal diferencia es que en un Grid hay más de una sola máquina paralela lo cual cambia por completo el problema de calendarización, como se mostrará más adelante.

Una solución al problema de calendarización en línea para Grid fue propuesto por Tchernykh et al. (2008). Ellos describen una estrategia de calendarización en un modelo jerárquico de dos niveles como *MPS = MPS_alloc + PS*. En el primer nivel, la estrategia *MPS_alloc* asigna la tarea a una máquina en base a algún criterio. En un segundo nivel, cada máquina aplica un algoritmo de calendarización local *PS* a las tareas asignadas a su respectiva máquina en la etapa previa.

El factor de competitividad del algoritmo *MPS* está acotado por el factor de competitividad del mejor algoritmo *PS*. Naroska y Schwiegelshohn (2002) demostraron que el mejor algoritmo en línea no clarividente tiene un factor de competitividad de *2-1/s*, con *s* denotando el número de procesadores en la máquina paralela. Hurink y Paulus (2007) demostraron que la calendarización de tareas en línea para 2 máquinas paralelas tiene una cota inferior de 2. La relevancia de este trabajo radica en que el problema que ellos

abordaron, es un caso particular de la calendarización en línea de Grid. Y al establecer esta cota inferior para el problema de calendarización en línea tareas paralelas para dos máquinas paralelas, establecen una cota inferior para el problema de calendarización en línea para Grid.

Considérese el siguiente ejemplo, el problema $P2|r_j=0,size_j,p_j|Cmax$, el cual es un caso especial de $GPm|r_j,size_j,p_j|Cmax$. Suponga que existe dos máquinas en el Grid, m=2, $\sum_j m_j = 2s_1$ con $s_1=s_2$, el número de procesadores de cada máquina. Aún más, hay n tareas con y $p_j=1$ para todos los trabajos. Para los casos del problema dado, cualquier calendario sin atrasos producirá $Cmax$=1 o $Cmax$=2. En la Figura 4 el caso óptimo está mostrado con $Cmax$=1. En la Figura 5 se muestra un calendario con el caso en el que Cmax=2.

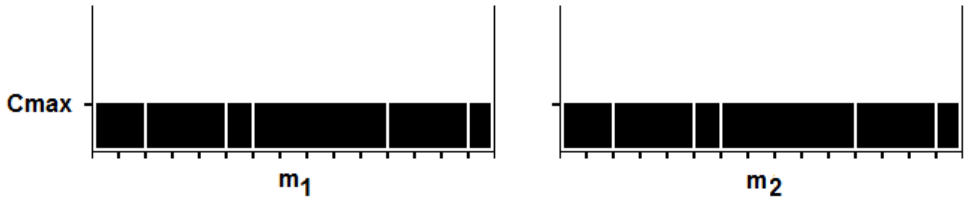


Figura 4. Calendario óptimo para el ejemplo

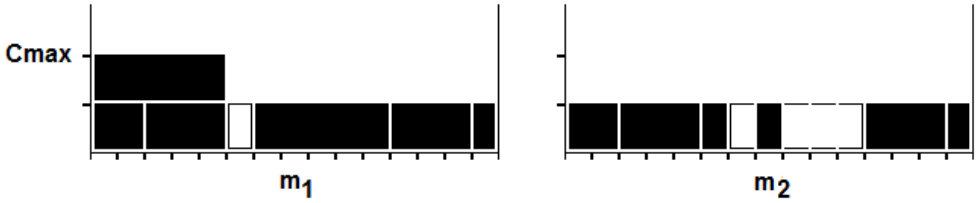


Figura 5. Calendario subóptimo para el ejemplo

De esta forma, cualquier algoritmo que garantice $\frac{C_{\max}}{C^*_{\max}} < 2$ debe de producir una solución óptima para los datos de entrada descritos para este ejemplo. Sin embargo, esto significa que se le diera solución al problema de partición el cual es NP Difícil en el caso ordinario. Por lo tanto, no existe un algoritmo de tiempo polinomial que sea menor a 2-competitivo a menos que $P=NP$ (Schwiegelshohn, Tchernykh, 2008).

Tchernykh et al. (2008) clasificaron las estrategias de asignación de acuerdo al nivel de información conocida de las máquinas para asignación de tareas a recursos.

En todos los niveles de información presentados, se conoce el número de máquinas y su tamaño. También, el número de tareas que se han calendarizado en las máquinas a un tiempo dado.

En el nivel 1 de información, se conoce el número de trabajos esperando a ser ejecutados en cada máquina. Basados en esta información, dos estrategias pueden ser implementadas. La primera es Carga Mínima (*minimum load*, *ML*). Usando esta estrategia, una tarea recién sometida al sistema es asignada a la máquina con menor número de tareas. La segunda estrategia es Carga Mínima por Procesador (*minimum load per processor*, *MLp*) Esta estrategia toma en cuenta el número de procesadores de cada máquina y selecciona una de ellas con la menor número de tareas por procesador $\left(\arg\left\{ \min_{1 \le i \le m} \left\{ \frac{n_i}{s_i} \right\} \right\} \right)$ al llegar una nueva tarea, donde n_i es el número de tareas de la máquina i y s_i el número de procesadores de la misma máquina.

Tchernykh et al. (2005) presentan un ejemplo con un Grid formado por dos máquinas paralelas con el mismo número de procesadores. Las tareas llegan en línea, con un patrón de dos tareas secuenciales y dos tareas paralelas que ocupan todos los procesadores de una de las máquinas. La Figura 6 muestra el calendario obtenido en caso de utilizar estrategias como ML y MLp. Al observar este ejemplo, se llega a la conclusión que las estrategias ML y MLp dan por resultado un calendario muy alejado del óptimo, mostrado en la Figura 7. Aún más, dichas estrategias no pueden garantizar un factor constante de aproximación para todos los problemas de calendarización en Grid (Tchernykh et al. 2005). Existen otros tres niveles de información en los cuales se pueden establecer estrategias distintas de calendarización en línea para Grid.

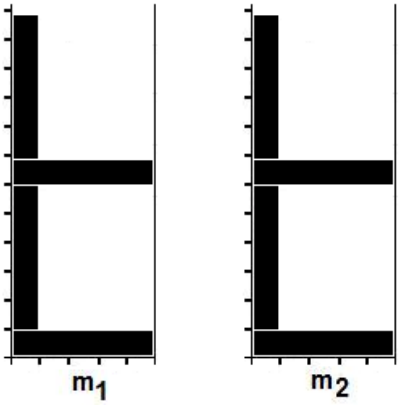


Figura 6. Calendario obtenido para el ejemplo

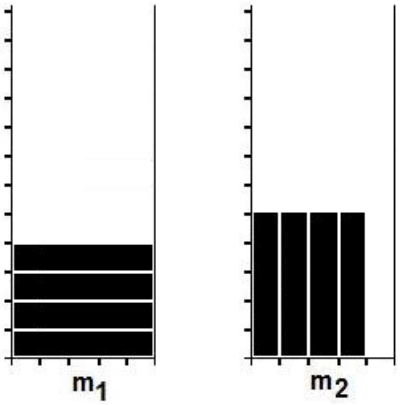


Figura 7. Calendario óptimo para el ejemplo

En el nivel 2, además de la información del nivel 1, se conoce el grado de paralelismo $size_j$ de cada tarea. La estrategia carga paralela mínima por procesador (*minimum parallel load per processor, MPL*) selecciona la máquina con la carga paralela menor por procesador calculada como $\left(\arg\left\{\min_{1\le i\le m}\left\{\sum_{i=a(j)}\frac{size_j}{s_i}\right\}\right\}\right)$. En el nivel 3, además de la información disponible en el nivel 2, se considerada el caso clarividente. Es decir, se conoce también el tiempo de ejecución de cada tarea. La estrategia cota inferior mínima (*minimum lower bound, MLB*) asigna una tarea a la máquina con la menor carga de trabajo sin ejecutar, tomando en cuenta todas las tareas asignadas a dicha máquina. Esto es $\arg\left\{\min_{1\le i\le m}\left\{\sum_{i=a(j)}\frac{size_j p_j}{s_i}\right\}\right\}$. En el nivel 4, se tiene disponible la información de los niveles

anteriores y además se conocen los calendarios locales. La estrategia de asignación (*minimum completion time, MCT*), asigna una tarea *j* a una máquina con el menor tiempo de finalización, tomando en cuenta el calendario actual de cada máquina y las tareas en sus colas locales.

En resumen, las estrategias mencionadas anteriormente, harían un calendario como el mostrado en la Figura 6 y por lo tanto, sin importar la cantidad de información que utilicen, no pueden garantizar un factor constante de aproximación (Tchernykh et al. 2005).

El algoritmo de *List* fue propuesto por Graham en 1965 y es muy conocido en teoría de calendarización en sistemas multiprocesadores. Considera que las tareas forman una lista de la cual se van tomando para su ejecución, una a una, mientras haya suficientes recursos para ejecutarlas. El algoritmo de List tiene un factor de competitividad de *2-1/s* para el problema $Pm|size_j|Cmax$. Sin embargo, Schwiegelshohn et al. (2008) demostraron que este algoritmo no puede tener un factor de aproximación constante para el problema de *GPm|size|Cmax.* Considérense los siguientes ejemplos. En el primer ejemplo, suponga que hay una lista de trabajos ordenados de forma no decreciente de acuerdo a su grado de paralelismo y se calendarizan en un Grid. Las máquinas están ordenadas de forma no decreciente de acuerdo al número de procesadores. Un calendario generado por List sería como el mostrado en la Figura 8.

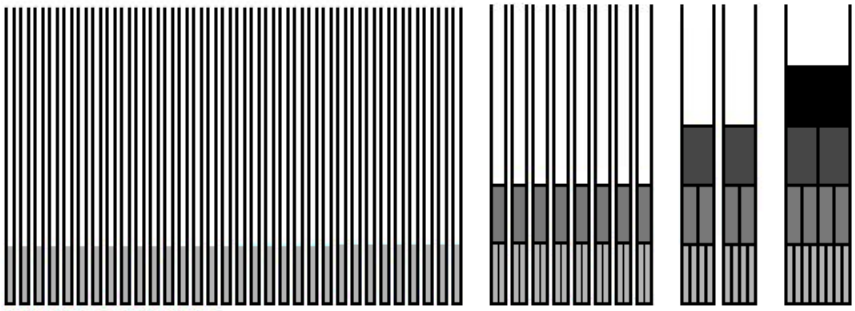


Figura 8. Calendario dado por algoritmo List

No es difícil ver que un calendario óptimo sería como el mostrado en la Figura 9, donde las tareas son asignadas a máquinas con el mismo tamaño.

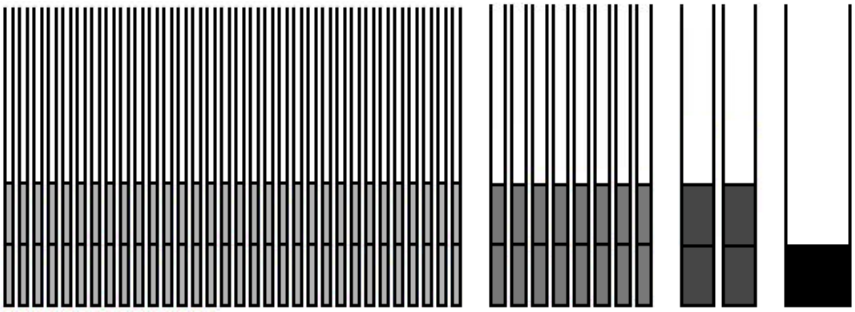


Figura 9. Calendario óptimo

El *Cmax* del calendario se incrementaría al utilizar *List* si aumenta el número de máquinas y el número de tareas. Entonces, no puede garantizarse una constante de aproximación (Schwiegelshohn et al. 2008).

Por otro lado, si las tareas están ordenadas de forma no creciente, el problema no se resuelve. Considere el caso mostrado en la Figura 10. En ella se muestra a una sola máquina que puede pertenecer a un Grid. Las tareas en color claro son tareas secuenciales. Las tareas en color gris, son tareas paralelas de distintos grados de paralelismo.

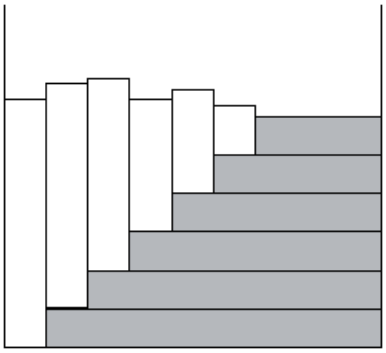


Figura 10. Calendario dado por el algoritmo List

Al calendarizar tareas de forma no creciente con *List*, tareas altamente paralelas pero de tiempo de ejecución corto son asignadas primero y los recursos disponibles se van ocupando con tareas secuenciales de tiempo de ejecución más largo. Entonces, se llega al punto en que máquinas grandes están ocupadas por tareas secuenciales. Y, tareas de mayor tamaño deben esperar a la ejecución de las tareas secuenciales, teniendo un efecto de retraso en la ejecución de tareas no deseado en todo el Grid.

En este trabajo se ataca el problema de calendarización en Grid con función objetivo de tiempo de terminación de calendario o *Cmax*. El Grid está formado por *m* máquinas. Cada máquina *i* tiene s_i procesadores. Sin pérdida de generalidad, las máquinas son indexadas de forma ascendente de acuerdo al tamaño de las mismas $s_1 \leq s_2 \leq s_3 \leq \ldots \leq s_{m.,}$ y $s_0=0$ es introducido. Nuestro modelo de máquinas para Grid queda descrito como GP_m.

Cada tarea *j* está caracterizada por la tupla (r_j,$size_j$,p_j,p'_j): su tiempo de sometimiento $r_j \geq 0,$su tamaño $1 \leq size_j \leq s_m$ y llamado grado de paralelismo, su tiempo de ejecución p_j y su tiempo estimado de ejecución p'_j. La notación $i=a(j)$ es introducida para indicar que la tarea *j* será ejecutada en la máquina *i*. El tiempo de terminación de la tarea *j* en el calendario *S* es denotado por $C_j(S)$, o simplemente C_j. Un calendario es factible si $r_j + p_j \leq C_j$ se mantiene para todos las tareas *j* y si para todo tiempo *t*,

$$s_i \geq \sum_{j \mid C_j - p_j \leq t < C_j \wedge i = a(j)} m_j \qquad \text{Ec.(1)}$$

No se permite la ejecución multisitio de tareas (*coallocation*). Tampoco la interrupción temporal de las mismas (*preemption*). Cada tarea tiene que ser ejecutada en m_j procesadores, en una sola máquina sin interrupciones.

1.3 Justificación

Se han propuesto varias soluciones para el problema de calendarización en línea de Grid. El mejor algoritmo conocido a la fecha es el algoritmo de robo de tareas. Tiene un factor de competitividad de 5 y usa un modelo de calendarización descentralizado. Hay varias suposiciones que el modelo utilizado por el algoritmo dificulta utilizarlo en implementaciones reales de Grid:

- La comunicación es punto a punto, todos a todos

- Los retrasos en las comunicaciones se desprecian

- La información respecto al estado de cada una de las máquinas es conocida por todas las máquinas de forma instantánea

El algoritmo de robo de tareas lleva a cabo la selección de máquina para cada tarea y establece procedimientos para el equilibrio de la carga. El modelo es utilizado para estudiar el problema de calendarización en línea de Grid con un número limitado de restricciones. Si algunas características son agregadas al modelo para reflejar la realidad de implementaciones de Grid, como retrasos en la comunicación o topología del sistema, produce factores de competitividad mayores. Es decir, el análisis con el modelo propuesto representa el mejor de los casos. Aún cuando se trata de un modelo con condiciones ideales, el análisis aporta información sobre la cota mínima en el factor de competitividad del algoritmo. A pesar de que el algoritmo ha sido estudiado de forma teórica, no existe un análisis experimental que establezca su desempeño con cargas de trabajo reales.

El modelo de calendarización jerárquico de dos niveles es más realista pero tiene sus propias limitaciones. Un bróker lleva a cabo la búsqueda de recursos y la asignación de tareas. Pero una vez que esto se ha realizado, no hay oportunidad de que el bróker realice equilibrio de cargas en tiempo de ejecución. Sin embargo, es común encontrar el modelo jerárquico en implementaciones reales de Grid. Por esta razón, es una tarea relevante establecer de forma experimental el desempeño de estrategias que utilizan este modelo a pesar de sus restricciones.

En un Grid real, es común que una misma máquina tenga varias particiones. Para un calendarizador de Grid, cada partición es vista como una máquina independiente, a pesar de que pueden pertenecer a una misma máquina. El modelo utilizado en el algoritmo de robo de tareas puede representar de forma más exacta si es utilizado en particiones de una misma máquina. Esto puede ser aceptable ya que las restricciones respecto a las comunicaciones en el modelo original pueden ser despreciables debido a las redes de comunicaciones de alta velocidad que se encuentran de forma común en máquinas paralelas de alto desempeño.

1.4 Objetivos

1.4.1 Objetivo general

Diseño, implementación, estudio teórico y análisis experimental de estrategias de calendarización en línea para Grid computacional con modelos de calendarización descentralizado y jerárquico.

1.4.2 Objetivos específicos

- Definir formalmente el problema de calendarización en línea en Grid
- Definir las propiedades de modelos descentralizados y jerárquicos de dos niveles que serán utilizados para el estudio de estrategias de calendarización
- Definir la relajación en los modelos para aproximarlos a implementaciones reales de Grid
- Describir las consecuencias de tales relajaciones en el desempeño de los algoritmos utilizados
- Diseñar adaptaciones dinámicas para estrategias de calendarización en línea para Grid
-Implementar un simulador de eventos discretos para llevar a cabo experimentos con modelos propuestos y estrategias seleccionadas. Al participar en el proyecto del simulador de calendarización en Grid, GSimula, los módulos requeridos para el desarrollo de esta tesis son: calendarización de Grid en modelo de un nivel con distintas estrategias, calendarización con lote de tareas por intervalo de tiempo definido, calendarización en modelo de dos niveles con admisibilidad.
- Llevar a cabo un estudio de cargas de trabajo reales para la fase de experimentación
- Definir métricas orientadas al usuario, al sistema y a algoritmos para desempeñar un análisis estadístico comparativo entre las distintas estrategias, existentes y propuestas
- Determinar el desempeño de estrategias de calendarización de Grid computacional con bitácoras de tareas reales por medio de un análisis experimental

1.4.3 Preguntas de investigación

El algoritmo de robo de tareas tiene el mejor desempeño conocido y probado teóricamente para el problema $GP_m|fix,r_j|Cmax$ con un factor de competitividad de 5.

-¿El algoritmo de robo de tareas puede superar en desempeño a otras estrategias de calendarización usando cargas reales?

- ¿Cómo se puede mejorar la calidad de la solución de un problema de calendarización en Grid usando estrategias de asignación?

-¿Cómo afectan las características de un Grid Computacional la definición de problemas en la teoría de calendarización?

-¿Qué cambio tienen que hacerse en los modelos de Grid para aproximarlos adecuadamente a una implementación de Grid real?

-¿Cuál es el impacto de tales cambios en el desempeño de algoritmos de calendarización?

-¿Puede una estrategia ser dinámicamente ajustada en respuesta a cambios de la configuración del Grid y de la carga de trabajo en un escenario de Grid real?

Cargas de trabajo de centros de súpercómputo han sido estudiadas por años y la información estadística de estas cargas está disponible.

-¿Cómo se puede utilizar esta información para la simulación de cargas de trabajo de Grid ya que éstas no están disponibles?

1.5 Organización de la tesis

En el Capítulo 1 se proporciona una introducción a la calendarización en Grid, se describe el problema que se aborda en este trabajo y los objetivos con los cuales se proponen soluciones al mismo. En el Capítulo II se define formalmente el modelo matemático utilizado. En el Capítulo III se definen las estrategias de calendarización, tanto para calendarizadores locales como para calendarizadores de Grid. En el Capítulo IV se describen los experimentos que se llevaron a cabo, se muestran resultados al utilizar las distintas estrategias de calendarización para diferentes casos de Grid y un análisis detallado

de las cargas de trabajo utilizadas y los resultados obtenidos. Las conclusiones se muestran en el Capítulo V.

Capítulo 2

Calendarización en Grid

2.1 Definición de modelo

El modelo utilizado para abordar el problema de calendarización en línea de Grid se formula como sigue: *n* tareas paralelas J_1, J_2, …, J_n tienen que ser calendarizadas en m máquinas paralelas (sitios) N_1, N_2, …, N_m. Sea m_i el número de procesadores idénticos en la máquina N_i, también llamado el tamaño de la máquina N_i. Suponga sin pérdida de generalidad, que las máquinas paralelas están ordenadas de forma no decreciente de acuerdo a su tamaño $m_1 \le m_2 \le ... \le m_m$.

Cada tarea J_j está descrito por la tupla (r_j, $size_j$, p_j, p'_j): su tiempo de sometimiento $r_j \ge 0$, su tamaño $1 \le size_j \le m_m$, también conocido como grado de paralelismo, tiempo de ejecución p_j, y el tiempo de ejecución estimado por el usuario p'_j. El trabajo de la tarea J_j se define como $w_j = p_j \cdot size_j$, también llamado el área en el calendario o su consumo de recursos. El problema que se aborda es el caso no clarividente. Por esta razón, las propiedades de cada tarea se conocen hasta el tiempo de sometimiento, excepto p_j, el cual permanece desconocido hasta terminar la ejecución de dicha tarea. Las tareas son sometidas a través del tiempo. Cada tarea es ejecutada exclusivamente en una máquina. La ejecución multisitio y coasignación (asignación de procesadores de distintas máquinas) no están permitidas. Una tarea J_j puede ejecutarse en una máquina N_i solo si $size_j \le m_i$ se mantiene. Sin embargo, la migración de tareas está permitida. Una tarea sometida a un sitio en particular puede ser ejecutada en un sitio diferente. No es requisito ejecutar la tarea inmediatamente una vez que ha sido sometida. El inicio de su ejecución puede ser atrasado en una cola local. La tarea es ejecutada utilizando el modelo de espacio compartido al asignar exactamente $size_j$ procesadores por un período ininterrumpido de tiempo p_j.

El tiempo de finalización del calendario *S*, o *makespan*, y un caso del problema *I*, se define como: $C_{\max}(S,I) = \max_{J_j}\{c_j(S,I)\}$. El makespan óptimo del caso *I* se denota como $C^*_{\max}(I)$. En futuras referencias se omitirán *I* y *S* siempre y cuando no cause ambigüedad.

2.2 Medidas de desempeño

Las medidas de desempeño son utilizadas para evaluar el desempeño de estrategias de calendarización. Existen diferentes tipos, dependiendo de la perspectiva de quién las evalúa. Por lo tanto, se tienen métricas centradas en el usuario y métricas centradas en el sistema. Aún cuando se conocen con distintos nombres y están definidas de forma diferente, existen métricas que están muy relacionadas y pueden omitirse, sin perder información al momento de evaluar un conjunto de estrategias de calendarización.

Desaceleración acotada promedio (*mean bounded slowdown*).

Se define como:

$$SD_b = \frac{1}{n}\sum_{j=1}^{n}\frac{c_j - r_j}{\max\{10, p_j\}} \qquad \text{Ec. (2)}$$

Esta métrica proporciona información respecto a qué proporción del tiempo que dura la tarea en el sistema corresponde a la ejecución de la misma. Ya que se usa el tiempo de ejecución en el denominador, se acota esta métrica a tareas que duren al menos 10 unidades de tiempo.

Tiempo promedio de espera

Se define como:

$$t_w = \frac{1}{n}\sum_{j=1}^{n}\left(c_j - p_j - r_j\right) \qquad \text{Ec. (3)}$$

Esta métrica nos indica cuánto tiempo, en promedio, una tarea tiene que esperar para ser ejecutada en el Grid.

En este trabajo se usa t_w sobre tiempo de respuesta del sistema (*response time* o *turnaround*) $TA = \frac{1}{n}\sum_{j=1}^{n}\left(c_j - r_j\right)$, tiempo de respuesta del sistema ponderado (*weighted*

turnaround time) $WTA = \frac{1}{n}\sum_{j=1}^{n} weight_j \cdot (c_j - r_j)$, tiempo de espera promedio ponderado en tamaño (*size weighted mean waiting time*) $WWT_s = \frac{1}{n}\sum_{j=1}^{n} size_j \cdot (c_j - p_j - r_j)$ y suma de tiempos de espera promedio (*sum of job waiting times*) $SWT = \sum_{j=1}^{n}(c_j - p_j - r_j)$. Las diferencias entre estas métricas son constantes independientemente de la carga de trabajo, estrategia de calendarización o Grid utilizado: $\frac{1}{n}\sum_{j=1}^{n} p_j$, $\frac{1}{n}\sum_{j=1}^{n} weight_j \cdot p_j$, $\frac{1}{n}\sum_{j=1}^{n} size_j$ y $1/n$, respectivamente. No se utilizaron tampoco métricas centradas en el sistema muy comunes, como utilización $U = \sum_{j=1}^{n} \frac{p_j \cdot size_j}{C_{\max} \cdot s_{1,m}}$ y capacidad de procesamiento (*throughput*) $Th = \frac{n}{C_{\max}}$. Hay una relación muy estrecha entre ellas ya que $C^*_{\max}$, $\sum_{j=1}^{n} \frac{p_j \cdot size_j}{s_{1,m}}$, y n son constantes para un experimento dado; una disminución de x porciento en la reducción de $C_{\max}$ corresponde a $\frac{x}{100\% - x}$ porciento de incremento en utilización y capacidad de procesamiento (Schwiegleshohn, 2009).

Cabe notar que para los experimentos, al no poder determinar de manera exacta la longitud del calendario óptimo, se utiliza una cota inferior: $\hat{C}^*_{\max} = \max\left\{\max_j (r_j + p_j), \frac{\sum_{j=1}^{n} w_j}{s_{1,m}}\right\}$ (Schwiegelshohn et al., 2008).

2.3 Trabajos previos

El problema de calendarización en sistemas multiprocesadores ha sido estudiado por décadas. De esa teoría es de donde se han heredado la mayor parte de modelos, estrategias y métricas aplicadas al problema de calendarización en Grid.

Graham presentó su algoritmo de *List* en 1969. Dos contribuciones principalmente son retomadas de este trabajo. En primera instancia, determinaró que para el modelo de máquina compuesta por procesadores idénticos donde se calendarizan tareas secuenciales, existe un factor de competitividad de *2-1/s*. Este, considerando que se calendarizan las

tareas fuera de línea. Sin embargo, Tchernykh et al. (2006) demostró que no es posible garantizar un factor de competitividad constante para todos las instancias de calendarización en Grid fuera de línea usando el algoritmo List. Hurink y Paulus (2007) demostraron que el problema de calendarización en Grid es mucho más complejo y que no existe un algoritmo de tiempo polinomial que garantice un factor de competitividad menor a 2, a menos que $P=NP$.

La segunda contribución es en sí el modelo. Propuso un modelo con procesadores idénticos. Este modelo fue extendido para el problema en el cual las tareas no son secuenciales, si no paralelas, en el trabajo que Garey y Graham presentaron en 1975. Naroska y Schwiegelshohn (2002) retomaron el modelo pero lo extendieron para el caso en el cual las tareas son sometidas en línea. Por último, Schwiegleshohn et al. (2008) aplican dicho modelo pero en lugar de considerar una sola máquina, lo hacen para un conjunto de máquinas paralelas formando un Grid.

Existen diferentes trabajos teóricos en los cuales se establecen cotas de competitividad para problemas muy relacionados con los presentados en este trabajo. Estos trabajos teóricos abordan ambos casos, en línea y fuera de línea, con variantes en el modelo de tareas, clarividentes y no clarividentes. La relevancia de las cotas establecidas para los casos fuera de línea, radica en que Schmoys et al. (1995) demostraron que un algoritmo de calendarización en línea es a lo más 2 veces peor en factor de competitividad que su versión fuera de línea.

El problema de calendarización en Grid puede verse como el problema de *multiple strip packing* (MSP). Este problema es una generalización del problema de *strip packing*, el cual consiste en un conjunto de finito de objetos de diferente área rectangular los cuales deben de acomodarse sin traslape en una tira, minimizando la longitud de la tira necesaria para hacerlo. Los rectángulos a ser colocados en las tiras, son las tareas a ser calendarizadas. Y las tiras donde se colocarán los rectángulos son las máquinas paralelas. La única restricción de MSP que no se cumple para el problema de calendarización en Grid, es que en MSP los procesadores donde se ejecutarán las tareas, deben de ser consecutivos. Bougeret et al. (2010) proporcionaron soluciones con algoritmos de aproximación. Definen formalmente

el problema como un conjunto de rectángulos $r_1,r_2,\ldots,r_n$ con alturas y anchos ≤ 1, con el objetivo de encontrar agrupamientos ortogonales no traslapados y sin rotaciones en k tiras de $[0,1] \times [0,\infty)$, minimizando el máximo de las alturas. Clasifican el problema en continuo y discontinuo. El problema continuo es aquel donde los rectángulos se consideran indivisibles y deben ocupar un espacio continuo en la tira. El problema discontinuo considera que el rectángulo puede ser divisible en una de sus dimensiones (ancho de la tira) y éste es el que se asemeja más al problema de calendarización en Grid. Una diferencia importante entre el modelo utilizado en el trabajo de Bougeret et al. (2010) y el modelo de este trabajo, es que ellos consideran conocer las dos dimensiones de los rectángulos. En calendarización en Grid, esto implicaría que se conoce no solo el tamaño de la tarea sino además el tiempo de ejecución. En nuestro trabajo, la calendarización se realiza de forma no clarividente, por lo que se desconoce el tiempo de ejecución de las tareas.

Pascual et al. (2008) propusieron soluciones al problema de calendarización en Grid en un modelo descentralizado. Proponen un algoritmo con coeficientes de competitividad de 4 y de 3 sobre el tiempo total de finalización del calendario, para distintos casos del problema. Sin embargo, existen varias restricciones en su modelo con el nuestro. Ellos consideran la calendarización fuera de línea. Su Grid está compuesto por N clústeres de m procesadores. Es decir, proponen un modelo más abstracto donde las máquinas son exactamente del mismo tamaño. Y el mejor resultado lo obtuvieron para el caso específico en el que la última tarea a ser calendarizada requiere a lo más $m/2$ procesadores.

Bougeret et al. (2010) propusieron una solución con coeficiente de competitividad de 5/2 para un modelo de calendarización de Grid jerárquico. En este caso, las máquinas tienen diferente tamaño, las tareas se consideran independientes y rígidas, y la función objetivo es el tiempo de finalización del calendario. Las diferencias principales con nuestro trabajo son que la calendarización de las tareas es clarividente y que el tamaño máximo de las tareas es el tamaño de la máquina más pequeña, es decir, cualquier trabajo puede ejecutarse en cualquier máquina. La solución propuesta se aplica también al problema de MSP continuo.

Tchernyk et al. (2006) consideró un problema de calendarización clarividente en un modelo de calendarización jerárquico. Presentaron dos algoritmos fuera de línea

MLBa+LSF y *MCTa+LSF*, con factor de competitividad de 10. En 2008, extendieron estos resultados introduciendo un factor de admisibilidad que parametriza la disponibilidad de recursos para la asignación de tareas. El algoritmo propuesto, *MLBa+PS*, presenta un factor de competitividad entre 5 y el infinito, para el caso en línea, el cual fue derivado para cargas de trabajo con características muy específicas. En 2010, Tchernykh et al. utilizaron un modelo más general. El factor de competitividad del mismo algoritmo varía entre 17 y el infinito, con cambios en el factor de admisibilidad.

En general, los trabajos previos mostrados presentan restricciones en al menos una de las siguientes características del modelo: tamaño de las tareas, tamaño de las máquinas y clarividencia. En este trabajo, las tareas puede ser hasta del tamaño de la máquina más grande del Grid, las máquinas pueden tener diferentes tamaños y las características de las tareas se conocen hasta su tiempo de sometimiento y, para las estrategias en el modelo descentralizado de calendarización, su tiempo de ejecución se conoce una vez que se ha completado la tarea.

Este trabajo tiene como punto de partida el trabajo presentado por Schwiegelshohn et al. (2008). El modelo propuesto por ellos es de máquinas paralelas de diferentes tamaños en una arquitectura descentralizada donde se calendarizan tareas en línea. Las tareas se consideran rígidas, la calendarización se ejecuta de forma no clarividente y las tareas son tan grandes como la máquina más grande del Grid. Ellos obtuvieron factor de competitividad de 3, para el caso fuera de línea, y de 5 para el caso en línea, con el tiempo de finalización de calendario como función objetivo. Propusieron un algoritmo descentralizado, carente de una cola global, donde el balanceo de la carga se lleva a cabo por medio de robo de tareas. Sin embargo, no proporcionaron evidencia experimental de la ejecución del algoritmo.

Grimme et al. (2008) propusieron un modelo de calendarización descentralizado, donde el balanceo de la carga se realiza por medio de migración de tareas a través de una cola global. Consideran un Grid de máquinas con distintos tamaños y con tareas tan grandes como la máquina más grande del Grid. Implementaron distintos algoritmos proporcionando únicamente resultados experimentales. Sin embargo, el modelo de tareas considera que se

conoce el tiempo estimado por el usuario al hacer la calendarización. A diferencia con el modelo que adoptamos para este trabajo, con un modelo de tareas no clarividente.

Existen trabajos donde la idea general de la solución, o algoritmo, que proponen es esencialmente la misma que el de Schwiegelshohn et al. (2008) pero el análisis que hacen es completamente diferente. Blumofe y Leiserson (1999) propusieron la idea de robo de trabajo dentro de una máquina paralela. Establecieron tiempos de finalización de calendario en base a tiempos de ejecución de las tareas, por ejemplo, la tarea con el tiempo secuencial más corto y el tiempo de ejecución del calendario con un número infinito de procesadores. Sin embargo consideran su modelo de máquina multihilo, de forma tal que no es una tarea en sí lo que un procesador ocioso roba si no un hilo al cual está asociado un conjunto de tareas. La idea de uso de listas de tareas descentralizadas fue retomada por Tchiboukdjian et al. (2011). Su trabajo se centra en establecer el costo de tener listas de forma descentralizada, en lugar de una sola lista central, para la asignación de tareas. Establecieron los tiempos de ejecución para tres casos de tareas (tareas unitarias, con peso y con precedencia) y su metodología puede ser utilizada para el análisis de algoritmos donde se consideren robo de tareas entre máquinas. Por otro lado, para realizar el balanceo de cargas por medio de migración de tareas Grimme et al. (2010) utilizaron un modelo descentralizado puerto a puerto (*peer-to-peer*, P2P). Realizaron un estudio experimental donde muestran el beneficio de utilizar la migración de tareas para lograr un mejor balanceo de carga entre las máquinas que componen el Grid. Sin embargo, no proporcionaron ningún estudio teórico sobre el desempeño de las estrategias estudiadas.

En Hirales_Carbajal et al. (2012) llevaron a cabo un estudio experimental con distintas estrategias de calendarización de Grid. Sin embargo, utilizan un modelo de tareas como flujos de trabajo, las cuales difieren con esta tesis en donde las tareas se consideran paralelas e independientes.

En Quezada-Pina et al. (2012) se derivaron un factor de aproximación de 9 y un factor de competitividad de 11 para los algoritmos MLBa+PS y MCTa+PS. En este caso, se contempló un modelo de Grid con máquinas de diferentes tamaños, donde las tareas arriban en línea, son de hasta el mismo tamaño que la máquina más grande.

Capítulo 3

Estrategias de calendarización

3.1 Calendarización local

Cada uno de los sitios que componen el Grid cuenta con un calendarizador local. Las tareas pueden ser sometidas localmente, a un bróker de Grid, a una cola global o a otro sitio, dependiendo del modelo de Grid utilizado y del algoritmo de calendarización de Grid. Sin embargo, el calendarizador local se encargará de la ejecución de la tarea en el sitio a donde finalmente la tarea fue asignada.

El algoritmo *First Come First Serve* (primero en llegar, primero en ser servido, *FCFS*) es un algoritmo muy primitivo. Simplemente consiste en ir ejecutando las tareas en los recursos disponibles en el orden en que arriban al sitio. Si ya no hay recursos disponibles, las tareas se colocan en una cola local a la espera de su ejecución. La implementación de este algoritmo local de calendarización es relativamente sencilla. Sin embargo, presenta el problema de comúnmente presentar recursos ociosos al calendarizar las tareas en un orden estricto de llegada. El algoritmo *BackFilling* (retrollenado, *BF*; Lifka 1995)da una solución a este problema. Este algoritmo elabora un calendario preliminar con las tareas que están disponibles para ejecución en los recursos del sitio, asignando un tiempo de inicio de ejecución para cada tarea. Después verifica si en los espacios disponibles de recursos ociosos dentro del calendario, puede ejecutar alguna tarea antes del tiempo de inicio de ejecución asignado originalmente. Existen diferentes versiones del algoritmo BF. Por ejemplo, la versión *First Fit* (primero en ocupar), adelanta la ejecución de la primera tarea que encuentre en el calendario, que pueda ejecutarse en el espacio de recursos ociosos. La versión *Best Fit* (mejor en ocupar), coloca en el espacio de recursos ociosos la tarea que "llene" la mayor cantidad de área disponible en el calendario. En esta tesis se utilizó *Backfilling Easy First Fit* (*BEFF*) como calendarizador local para el modelo jerárquico.

Esta versión busca la primera tarea que pueda asignarse al espacio disponible, sin que al hacerlo se atrase el inicio de ejecución de las tareas restantes en el calendario.

3.2 Estrategias de calendarización en Grid

3.2.1 Estrategias de asignación de tareas a recursos para Grid de dos niveles

3.2.1.1 Aleatoria (*Random*)

Asigna tarea a tarea a un sitio elegido aleatoriamente de un conjunto de sitios permitidos. Los sitios permitidos son aquellos sitios *i* que cumplen con $m_i \geq size_j$. La información requerida de la tarea es entonces su tamaño. La idea general de esta estrategia es lograr una distribución uniforme de número de tareas en los diferentes sitios. Sin embargo, es común que esta estrategia no logre un buen balanceo de carga. El tamaño de la tarea es utilizado solamente para determinar si un sitio es parte del conjunto permitido o no, para su asignación. Pero esta información no es utilizada para definir una mejor distribución de la carga en todos los sitios.

3.2.1.2 Carga por procesador mínima (*MLp*)

Asigna la tarea *j* en el tiempo r_j al sitio con menor carga por procesador. De manera formal, el mejor sitio para la tarea *j* siguiendo esta estrategia es:

$$min_{i=1..m}\left(\frac{l_i}{m_i}\right) \qquad \text{Ec. (4)}$$

Mejora la asignación aleatoria, al elegir el sitio con menor número de tareas, pero tomando en cuenta el tamaño del sitio. Por lo tanto, al momento de realizar la asignación y al tomar en cuenta el número de tareas en el sitio por procesador, logra un mejor balanceo de la carga que la estrategia aleatoria.

3.2.1.3 Carga paralela por procesador mínima (*MPL*)

De forma semejante que la estrategia anterior, el parámetro que define la asignación se obtiene al dividir entre el número de procesadores por máquina. Sin embargo, toma en

cuenta no solo el número de tareas pero el tamaño de cada una de ellas. Las tareas utilizadas para el cálculo son aquéllas que se encuentran tanto en la lista de espera como en ejecución. La definición formal es:

$$min_{i=1..m}\left\{\sum_{k=1}^{l_i}\frac{size_k}{m_i}\right\} \quad \text{Ec. (5)}$$

La idea general es lograr un balanceo de cargas entre los sitios al uniformizar la carga paralela de cada procesador en todas las máquinas.

3.2.1.4 Balanceo de carga mínimo con tamaño (*LBal_S*)

Asigna la tarea *j* al sitio con la menor desviación estándar de carga paralela por procesador, tomando en cuenta las tareas que se encuentran en espera y las que se encuentran en ejecución. Esto es:

$$\min_{q=1..m}\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(PL_i^q-\overline{PL}\right)^2} \quad \text{Ec. (6)}$$

Donde $PL_{i=1..m}^{q}=\frac{1}{m_i}\sum_{k=1}^{l_i}(size_k+size_j^q)$ y $size_j^q$ es el tamaño de la tarea *j* si es agregada al sitio *q*. La intención de esta estrategia es disminuir las desviaciones en el promedio de la carga paralela por sitio de todo el Grid. *PL* puede estar dado en función de diferentes parámetros como tamaño, tiempo o trabajo.

3.2.1.5 Cota inferior mínima (*MLB*)

Asigna la tarea *j* en el tiempo r_j al sitio con la menor cantidad de recursos consumidos. El sitio elegido queda definido por:

$$min_{i=1..m}\left\{\sum_{k=1}^{l_i}\frac{size_k\cdot p'_k}{m_i}\right\} \quad \text{Ec. (7)}$$

Esta estrategia requiere de una estimación del tiempo de ejecución de cada tarea para poder llevarse a cabo. La idea general es asignar la tarea al sitio en el cual haya menor cantidad de recursos consumidos divididos de forma uniforme entre todos los procesadores del sitio.

3.2.1.6 Tiempo de finalización mínimo (*MCT*)

Asigna la tarea j en el tiempo r_j al sitio en el cual se tenga el tiempo de finalización mínimo. Grid $min\{C_{max}^{i}\}$, donde $C_{max}^{i} = max_{g_k=i}(C_k^i)$ y C_k^i es el tiempo de finalización de la tarea j_k en el sitio i. En esta estrategia el bróker elabora un calendario en nivel de asignación, de todos los sitios en el Grid.

3.2.1.7 Tiempo de inicio mínimo (*MST*)

Asigna la tarea *j* al sitio donde su tiempo de inicio de ejecución sea mínimo. De manera formal, el sitio elegido es:

$$min_{\ i=1..m}\{s_j^i\} \qquad \text{Ec. (8)}$$

El bróker elabora un calendario a nivel de asignación, de todos los sitios del Grid.

3.2.1.8 Tiempo de espera promedio mínimo (*MWT*)

Asigna la tarea *j* al sitio con el menor promedio de tiempo de espera. El sitio elegido se define formalmente como:

$$min_{i=1..m}\left\{\sum_{g_k=i}\frac{t_w^k}{n_i}\right\} \qquad \text{Ec. (9)}$$

También en esta estrategia el bróker elabora un calendario a nivel de asignación, de todas las máquinas que forman el Grid. Esta estrategia está basada en la métrica de tiempo de espera. La idea general es mantener el tiempo promedio de espera de ejecución tan pequeño como sea posible. Y por consiguiente, el tiempo de finalización también disminuye para cada tarea, esperando que el tiempo de finalización del calendario de Grid disminuya también.

3.2.1.9 Tiempo de espera promedio ponderado por tamaño mínimo (*MWWT_S*)

Asigna la tarea j al sitio con el menor tiempo promedio de espera ponderado. En nuestro trabajo dimos por peso a cada tarea, su tamaño correspondiente. Formalmente el sitio elegido es:

$$min_{i=1..m}\left\{\sum_{g_k=i}\frac{t_w^k \cdot size_k}{n_i}\right\} \qquad \text{Ec. (10)}$$

También puede utilizarse como peso el tiempo de ejecución de la tarea o su trabajo. Aquí también, el bróker elabora un calendario en nivel de asignación. La idea general es disminuir el tiempo promedio de espera de las tareas, como en la estrategia anterior. El parámetro $size_k$ en el numerador evita que se tome por igual tareas de alto o bajo paralelismo.

3.2.2 Estrategias asignación de tareas en Grid de un nivel

3.2.2.1 Intercambio de tareas en cola global

Grimme et al. (2007), aplicaron el concepto de intercambio de tareas entre los sitios que conforman el Grid. Los objetivos de dicho esquema son incrementar la utilización de los recursos, disminuir el tiempo de finalización de calendario y disminuir el tiempo de respuesta promedio. La comunicación entre los diferentes sitios es a través de una cola global, tal como se ve en la Figura 11.

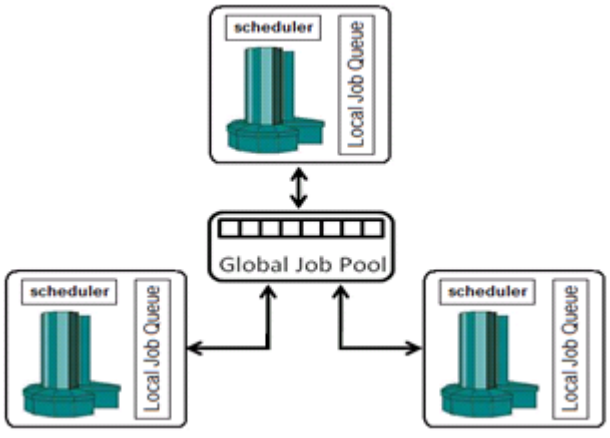


Figura 11. Esquema de calendarización con cola global

Ellos propusieron el Algoritmo 1.

Algoritmo 1

Condiciones: una cola local *l*, una cola global *g*

1: algoritmo $\in$ {FCFS, EASY, LIST}
2: aplicar algoritmo a *l*
3: **for** todas las tareas en *l* pero no ejecutables localmente
4: enviar tareas a *g*
5: **end for**
6: **if** no hay tarea para ejecutar localmente **then**
7: aplicar algoritmo a *g*
8: **end if**

La idea general del algoritmo es ejecutar localmente las tareas sometidas por cada sitio utilizando el calendarizador local. Cuando una tarea no puede ejecutarse de forma inmediata en la máquina local, la tarea es ofrecida a la cola global de tareas y removida de la cola local. En caso de que una máquina tenga procesadores ociosos, trata de utilizarlos ejecutando alguna tarea que se encuentre en la cola global.

Grimme et al. (2007) probaron diferentes algoritmos de calendarización local. Ellos utilizaron FCFS, BEFF y una versión modificada de List. Este List utiliza una lista de tareas la cual está ordenada en base a pesos de las mismas. El peso está dado por el tiempo de sometimiento de la tarea. El calendarizador revisa la lista de forma ascendente de los pesos hasta encontrar la tarea que pueda ejecutarse inmediatamente en los procesadores ociosos de la máquina.

3.2.2.2 Algoritmo de Robo de Tareas (ST, *Stealing*)

Schwiegelshohn et al. (2008) presentaron un algoritmo para distribución de tareas y balanceo de carga en un Grid. Tal como en Grimme et al. (2007), el algoritmo de robo de tareas promueve el intercambio de trabajos entre los diferentes sitios que conforman el

Grid. Pero aquí, la comunicación se considera punto a punto y por lo tanto no es necesaria una cola común de tareas.

La idea general del algoritmo es mantener ocupados el mayor número de procesadores en la ejecución de las tareas. Los autores demostraron que esto no se puede garantizar con *List*, no importando si el orden de las tareas es ascendente o descendente.

El algoritmo considera diferentes conjuntos de tareas en cada máquina. Cada tarea pertenece exclusivamente a uno de dos conjuntos, *A* o *B*, los cuales se definen como:

$$A_i = \{ j \mid \max\{\frac{s_i}{2}, s_{i-1}\} < size_j \leq s_i \}$$ Ec. (11)

$$B_i = \{ j \mid s_{i-1} < size_j \leq \frac{s_i}{2} \}$$ Ec. (12)

El conjunto, o cola, A_i contiene las tareas que requieren al menos el 50% de los procesadores de la máquina *i*. Y B_i, las tareas que requieren menos del 50% de los procesadores de la máquina *i*. Las tareas de los conjuntos A_i o B_i, solo pueden calendarizarse en las máquinas *i* o más grandes. El valor de frontera de 50% entre las colas *A* y *B* puede cambiarse. En este trabajo nos referiremos al algoritmo original que utiliza el valor de frontera de 50% como *ST50*.

La descripción completa del algoritmo considera otro conjunto definido como:

$$H_i = \{ j \mid \frac{s_i}{2} < size_j \leq s_{i-1} \}$$ Ec. (13)

El conjunto *H* es utilizado por el algoritmo para ejecutar las tareas que requieren más del 50% de los procesadores de la máquina *i*, pero que pertenecen al conjunto A_{i-1}. Puede advertirse que *B* o *H* siempre estará vacío para cada máquina, dependiendo del tamaño de *i* y de *i-1*. En Algoritmo 2.A y Algoritmo 2.B se muestra la descripción completa de *ST50*.

Algoritmo 2.A

Condiciones iniciales: $s_i \leq s_j$, for $i<j$

1: **for** $i \leftarrow 1$ to m **do**

2: $L_i \leftarrow A_i$

3: $S_i \leftarrow H_i$

4: **endfor**

```
5:    Procedimiento Update
6:    repeat
7:        for i←1 to m do
8:            while haya suficiente procesadores ociosos en la máquina i do
9:                calendarizar una tarea de L_i en la máquina i
10:               remover la tarea calendarizada de todas las listas L_k
11:               if cualquier lista L_k se vacía
12:                   Procedimiento Update
13:               endif
14:           endwhile
15:       endfor
16:   until calendarizar todas las tareas
```

Algoritmo 2.B Procedimiento Update

```
1:    for i ← 1 to m do
2:        if L_i = ϕ
3:            if i≠1 and S_i≠ϕ
4:                L_i← L_{i-1}∩S_i
5:            elseif B_i= ϕ
6:                L_i← B_i
7:            endif
8:            if i≠1 and L_{i-1}≠ϕ and S_i=ϕ
9:                L_i← L_{i-1}
10:           endif
11:       endif
12:   endfor
```

La ejecución de *ST50* es como sigue. Cada máquina i ejecuta tareas de la cola A_i, una a la vez. Cuando la última tarea de la cola A_i ha iniciado su ejecución y hay procesadores

ociosos, la máquina *i* ejecuta tareas de H_i y luego de B_i. Cuando no hay más tareas en las colas A_i, H_i y B_i y hay procesadores ociosos, la máquina *i* "roba" tareas de las colas *A*, *H* y *B* de máquinas más pequeñas que puedan iniciar su ejecución de forma inmediata

El objetivo de este algoritmo es mantener al menos el 50% de los procesadores disponibles al tiempo de la ejecución de la última tarea. Los autores demostraron un factor de aproximación de 3, para el caso fuera de línea, y un factor de competitividad de 5, para el caso en línea para este algoritmo.

3.2.2.3 Admisibilidad adaptable

Tchernykh (2006) presentó un esquema de admisiblidad que puede ser implementado eficientemente en sistemas reales de Grid. La idea principal del esquema es obtener un subconjunto de las máquinas disponibles para ejecutar una tarea. Esto se lleva a cabo para evitar que trabajos secuenciales puedan ocupar procesadores de máquinas grandes, causando que tareas con alto grado de paralelismo tengan que esperar para su ejecución.

La estrategia trabaja como sigue. Suponga que las máquinas están ordenadas de forma no descendente de acuerdo al tamaño de las mismas ($s_1 \leq s_2 \leq ... \leq s_m$). El conjunto de máquinas admisibles para la tarea *Jj* se define como las máquinas con los índices $\{f_j, ..., l_j\}$ (o simplemente $\{f,...,l\}$ si no existe ambigüedad), donde f_j es el menor índice *i* tal que $m_i \geq size_j$ y l_j es el menor índice de máquina tal que $s_{fj,lj} \geq a\ s_{fj,m}$, donde $s_{fj,lj}$ es el número total de procesadores que pertenecen a las máquinas que van de N_{fj} a N_{lj}. Se toma en cuenta que $l_j \leq m$ siempre se mantiene. Un factor de admisibilidad $0<a\leq 1$ define el rango de admisibilidad usado para la asignación de la tarea. Al menos una máquina es admisible ya que *a* es estrictamente mayor que 0. Y con $a=1$, todas las máquinas del Grid son admisibles. Si *f* es el índice menor tal que $m_f \geq size_j$ entonces la tarea puede ser asignada a máquinas con índices que van desde *f* hasta *m*. El factor de admisibilidad reduce las máquinas disponibles a las máquinas con índices que van de *f* a *l*, como se muestra en la Figura 12. Se supone que la tarea *Jj* se asigna a la máquina del conjunto $f,...,l$ que contiene $a_{sf,m}$ procesadores. Así, $(1-a)s_{f,m}$ procesadores son excluídos de $s_{f,m}$ procesadores elegibles inicialmente para la asignación de *Jj*.

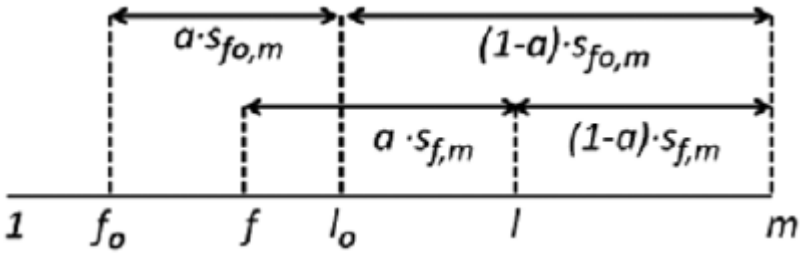


Figura 12. Esquema de admisibilidad

Existen varias propiedades del esquema de admisibilidad adaptable que resultan atractivas para su utilización. Primero que nada, puede utilizarse para calendarizar tareas tanto en línea como fuera de línea. También, puede implementarse de forma eficiente lo cual es particularmente importante para el caso en línea. Su utilización no está restringida a algún modelo de calendarización en particular. Puede utilizarse en modelos centralizados, jerárquicos y descentralizados. El esquema de admisibilidad adaptable puede usarse por sí sola como una estrategia de asignación de tareas. Sin embargo, también puede ser utilizada en combinación con otras estrategias de asignación, *MST* y *MCT* por ejemplo, para mejorar su desempeño de las estrategias originales.

3.2.2.4 Teoremas de admisibilidad (Quezada-Pina et al. 2012)

Teorema 1. La calendarización fuera de línea de tareas paralelas rígidas en Grids con procesadores idénticos usando el algoritmo *MCTa+PS* con rango de asignación admisible de $0 < a \leq 1$ tiene un factor de aproximación de

$$\rho \leq \begin{cases} 1+\dfrac{2}{a^2} & if \quad a \leq \dfrac{s_{f,l}}{s_{f_0,m}} \\ 1+\dfrac{2}{a(1-a)} & if \quad a > \dfrac{s_{f,l}}{s_{f_0,m}} \end{cases} \qquad \text{Ec. (14)}$$

con $1 \leq f_0 \leq f \leq l \leq m$, parámetros que dependen de la configuración de máquinas y la carga de trabajo.

Demostración. Suponga que el tiempo de finalización de calendario de la máquina k es también el $C_{\max}$ del Grid, entonces $C_{\max} = C_k$ se mantiene. Aún más, sea la tarea J_d la última tarea agregada a esta máquina.

Ya que J_d se agregó a la máquina k, MCT_a garantiza que $C_k \le C_i$ antes de agregar J_d. Así, basados en la propiedad de que *PS* es 2-competitivo, $C_k \le \min C_i = \min\{2 \cdot \max\{p_{\max}, \frac{W_i}{m_i}\}\}$, $C_k \le \max\{\min\{2p_{\max}\}, \min\{2\frac{W_i}{m_i}\}\}$. Sea la máquina e la máquina con la menor razón $\frac{W_i}{m_i}$: $\frac{W_e}{m_e} = \min_{f \le i}\left\{\frac{W_i}{m_i}\right\}$. Entonces, $C_k \le \max\{2p_{\max}, 2\frac{W_e}{m_e}\}$.

Al agregar J_d obtenemos $C_{\max} \le C_k + p_d \le C_k + p_{\max}$ y $C_{\max} \le \max(2\frac{W_i}{m_i} + p_{\max}, 3p_{\max})$ (1)

Ahora consideremos las propiedades de una solución óptima.

$$W_{f,l} = \sum_{i+f}^{l} W_i = \sum_{i=f}^{l} \frac{W_i}{m_i} \cdot m_i \ge \sum_{i=f}^{l} \frac{W_e}{m_e} \cdot m_i = \frac{W_e}{m_e} \sum_{i=f}^{l} \cdot m_i = \frac{W_e}{m_e} \cdot m_{f,l} \quad (2)$$

Sea J_b la tarea de menor tamaño de todos y que puede ser ejecutado en la máquina N_f de nuestro calendario, que es $l_b \ge f$. Así las tareas calendarizadas en el rango de $N_f, ..., N_l$ no pueden ser asignadas a una máquina con índice menor a f_b. Como J_b puede ser ejecutado en la máquina N_f, tenemos $C^*_{\max} \ge \frac{W_{f,l}}{m_{f_0,m}}$.

Substituyendo (2) en esta ecuación, tenemos que $C^*_{\max} \ge \frac{W_e}{m_e} \cdot \frac{m_{f,l}}{m_{f_0,m}}$.

Con $m_{f,l} \ge a \cdot m_{f,m}$, obtenemos $m_{f,l} \ge a^2 \cdot m_{f_0,m}$ si $a \le \frac{m_{f,m}}{m_{f_0,m}}$ se mantiene. Esto nos lleva $C^*_{\max} \ge \frac{W_e}{m_e} \cdot \frac{m_{f,l}}{m_{f_0,m}} \ge \frac{W_e}{m_e} \cdot a^2$

Debido a que $C^*_{\max} \ge p_{\max}$, tenemos con (1) que $\rho \le \max\{\frac{2 \cdot W_e}{m_e \cdot C^*_{\max}} + \frac{p_{\max}}{C^*_{\max}}, \frac{3p_{\max}}{C^*_{\max}}\} \le \max\{1 + \frac{2}{a \cdot (1-a)}, 3\}$ y $\rho \le 1 + \frac{2}{a \cdot (1-a)}$, ya que $a < 1$.

Con $m_{f_0,m} \le m_{f,m} + a \cdot m_{f_0,m}$ (ver en Figura 12), y si $a > \frac{m_{f,m}}{m_{f_0,m}}$, tenemos

$C^*_{\max} \ge \frac{W_e}{m_e} \cdot \frac{m_{f,l}}{m_{f_0,m}} \ge \frac{W_e}{m_e} \cdot \frac{a \cdot m_{f,m}}{\frac{m_{f,m}}{1-a}} = \frac{W_e}{m_e} \cdot a \cdot (1-a)$ y $\rho \le \max\{\frac{2 \cdot W_e}{m_e \cdot C^*_{\max}} + \frac{p_{\max}}{C^*_{\max}}, \frac{3p_{\max}}{C^*_{\max}}\} \le \max\{1 + \frac{2}{a \cdot (1-a)}, 3\}$

y $\rho \le 1 + \frac{2}{a \cdot (1-a)}$, ya que $a < 1$.

Teorema 2. La calendarización fuera de línea de tareas paralelas rígidas en Grids con procesadores idénticos usando el algoritmo *MLB$_a$+PS* con un rango de asignación admisible de $0 < a \le 1$ tiene el factor de aproximación de

$$\rho \le \begin{cases} 1 + \frac{2}{a^2} & si \quad a \le \frac{m_{f,l}}{m_{f_0,m}} \\ 1 + \frac{2}{a(1-a)} & si \quad a > \frac{m_{f,l}}{m_{f_0,m}} \end{cases} \qquad \text{Ec. (15)}$$

con $1 \le f_0 \le f \le l \le m$ siendo un parámetro que depende de la configuración de las máquinas y la carga de trabajo.

Demostración. Suponga que el tiempo de finalización de calendario de la máquina k es el mismo que el del Grid, entonces $C_{\max} = C_k$ se mantiene.

Aún más, sea la tarea J_d la última tarea asignada a esta máquina.

Las máquinas $f = f_d, .., l = l_d$ constituyen el conjunto de $M_{admissible}(d)$. Ya que J_d se agregó a la máquina k, *MLB$_a$* garantiza que $\frac{W_k}{m_k} \le \frac{W_i}{m_i}$ para toda $i = f, ..., l$. Note que W_k no incluye la tarea J_d.

Así, basado en la propiedad de 2-competitivo de *PS* $C_k \le \max\{2 \cdot \frac{W_k}{m_k}, 2 \cdot p_{\max}\}$. Sabemos que

$W_{f,l} = \sum_{i+f}^{l} W_i = \sum_{i=f}^{l} \frac{W_i}{m_i} \cdot m_i \ge \sum_{i=f}^{l} \frac{W_k}{m_k} \cdot m_i \quad = \frac{W_k}{m_k} \sum_{i=f}^{l} \cdot m_i = \frac{W_k}{m_k} \cdot m_{f,l}$

Sea J_b la tarea de menor tamaño entre todas las tareas ejecutadas en las máquinas $N_f, ..., N_l$

Usamos la notación $f_0 = f_b$. De esta forma, las tareas en $N_f, ..., N_l$ no pueden ser asignadas

a una máquina con un índice menor a f_0. Como J_b es ejecutada en una de las máquinas de $N_f,...,N_l$, se tiene que $l_b \geq f$ y $C^*_{\max} \geq \frac{W_{f,l}}{m_{f_0,m}}$.

Sustituyendo (4), se tiene $C^*_{\max} \geq \frac{W_k}{m_k} \cdot \frac{m_{f,l}}{m_{f_0,m}}$.

Como la estrategia *MLB*$_a$ usa W_k sin que se incluya la tarea J_b se considera la tarea J_b en adición. En el peor caso C_k es aumentado por $p_d \leq C^*_{\max}$ dando por resultado que $C_{\max} \leq C_k + C^*_{\max}$ y $\rho \leq 1 + \frac{C_k}{C^*_{\max}}$. Debido a (3), se tiene que $\rho \leq 1 + \frac{\max\{2\frac{W_k}{m_k}, 2p_{\max}\}}{C^*_{\max}}$.

Con $m_{f,l} \geq a \cdot m_{f,m}$, se obtiene que $m_{f,l} \geq a^2 \cdot m_{f_0,m}$ si $a \leq \frac{m_{f,m}}{m_{f_0,m}}$. Esto lleva a que $C^*_{\max} \geq \frac{W_k}{m_k} \cdot \frac{m_{f,l}}{m_{f_0,m}} \geq \frac{W_k}{m_k} \cdot a^2$ y $\rho \leq 1 + \frac{\max\{2\frac{W_k}{m_k}, 2p_{\max}\}}{C^*_{\max}}$. Ya que $\frac{2}{a^2} \geq 2$, se tiene que $\rho \leq 1 + \frac{2}{a^2}$.

Con $m_{f_0,m} \leq m_{f,m} + a \cdot m_{f_0,m}$ (ver Figura 12), y si $a > \frac{m_{f,m}}{m_{f_0,m}}$, se tiene que

$$C^*_{\max} \geq \frac{W_k}{m_k} \cdot \frac{m_{f,l}}{m_{f_0,m}} \geq \frac{W_k}{m_k} \cdot \frac{a \cdot m_{f,m}}{\frac{m_{f,m}}{1-a}} = \frac{W_k}{m_k} \cdot a \cdot (1-a) \text{ y } \rho \leq 1 + \frac{\max\{2\frac{W_k}{m_k}, 2p_{\max}\}}{C^*_{\max}} \leq 1 + \frac{2}{a \cdot (1-a)}$$

Note que ambos algoritmos, *MLB*$_a$*+PS* y *MCT*$_a$*+PS* producen cotas de aproximación con el mismo resultado $\rho \leq 9$ para $a = 0.5$.

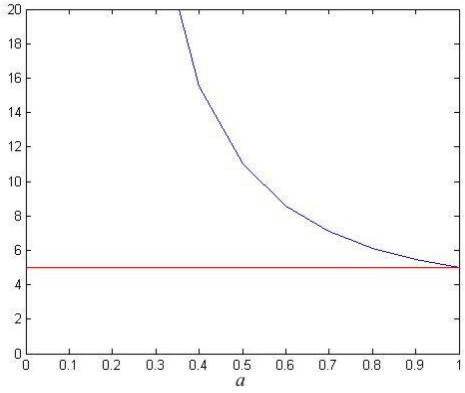


Figura 13. El factor de competitividad de las estrategias *MCT$_a$+PS* y *MLB$_a$+PS* si $a \le \frac{m_{f,l}}{m_{f_0,m}}$

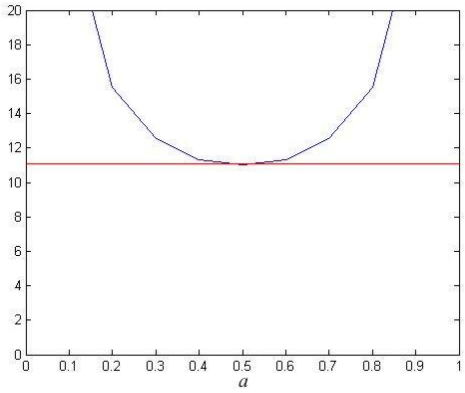


Figura 14. El factor de competitividad de las estrategias *MCT$_a$+PS* y *MLB$_a$+PS* si $a > \frac{m_{f,l}}{m_{f_0,m}}$

Teorema 3. La calendarización en línea de tareas paralelas rígidas en Grids con máquinas con procesadores idénticos usando los algoritmos MCT_a+PS and MLB_a+PS con un rango de asignación admisible de $0 < a \le 1$ tiene el factor de competitividad de:

$$\rho \le \begin{cases} 3+\frac{2}{a^2} & si \quad a \le \frac{m_{f,l}}{m_{f_0,m}} \\ 3+\frac{2}{a(1-a)} & si \quad a > \frac{m_{f,l}}{m_{f_0,m}} \end{cases} \qquad \text{Ec. (16)}$$

con $1 \le f_0 \le f \le l \le m$ siendo parámetros que dependen de la configuración de las máquinas y carga de trabajo.

Demostración. Suponga que r es el tiempo de sometimiento de la última tarea en el calendario. Después del tiempo r, un algoritmo de calendarización fuera de línea puede ser utilizado. Sin embargo, tiene que iniciar su ejecución con todos los procesadores en estado ocioso. El tiempo de finalización está limitado por el tiempo de procesamiento máximo de cualquiera de las tareas. Además, esta fase del algoritmo termina en el tiempo $r + p_{\max} + C_{\max}$ cuando mucho. Debido a las cotas de los teoremas 1 y 2, las inecuaciones $C^*_{\max} \ge r$ y $C^*_{\max} \ge p_{\max}$ se mantienen, y el teorema queda demostrado.

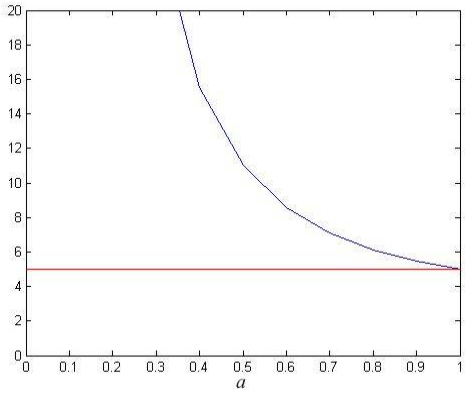


Figura 15. El factor de competitividad de las estrategias *MCT$_a$+PS* y *MLB$_a$+PS* si $a \le \frac{m_{f,l}}{m_{f_0,m}}$.

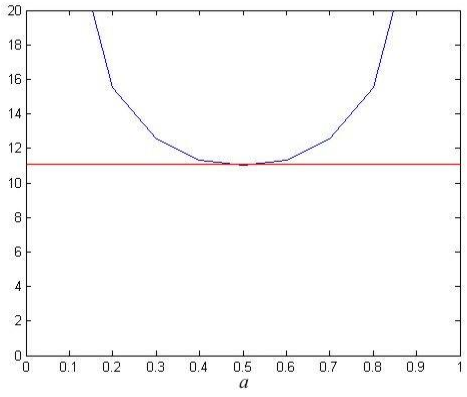


Figura 16. El factor de competitividad de las estrategias *MCT$_a$+PS* y *MLB$_a$+PS* si $a > \frac{m_{f,l}}{m_{f_0,m}}$.

Las Figuras 15 y 16 muestran las cotas de los factores de competitividad de las estrategias *MCT*$_a$*+PS* and *MLB*$_a$*+PS* si los valores de factor de admisibilidad son $a \le \frac{m_{f,l}}{m_{f_0,m}}$ y $a > \frac{m_{f,l}}{m_{f_0,m}}$, respectivamente.

Se observa que si $a \le \frac{m_{f,l}}{m_{f_0,m}}$ la cota para el peor caso cambia de ∞ a 5 en función de a. Si $a > \frac{m_{f,l}}{m_{f_0,m}}$ la cota para el peor caso va de ∞ a ∞ con valor mínimo para $a = 0.5$. La Figura 17 muestra la cota resultante que son los máximos para los factores de competitividad para los peores casos mostrados en Figura 15 y Figura 16. Se observa que la cota da por resultado $\rho \le 11$ para $a = 0.5$.

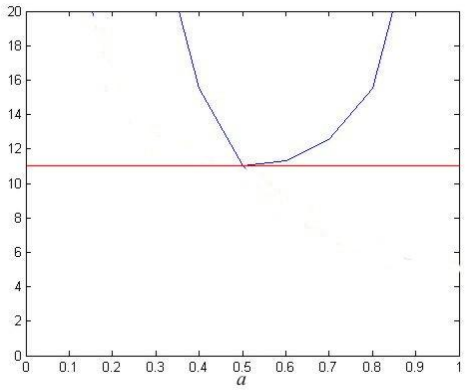


Figura 17. El factor de competitividad par alas estrategias *MCT*$_a$*+PS* y *MLB*$_a$*+PS.*

Teorema 4. La calendarización de tareas paralelas rígidas en Grids con procesadores idénticos usando los algoritmos *MCTa+PS*, *MLBa+PS* y las tareas son tan grandes como el tamaño de la máquina más pequeña del Grid, tiene como factor de aproximación de 3 y factor de competitividad de 5.

Demostración. La prueba es directa considerando la restricción respecto al tamaño de las tareas. Si todas las tareas son de tamaño no mayor que la máquina más pequeña $size_j \le m_1$, entonces todas las tareas pueden ser asignadas a todas las máquinas, de 1 a m. De esta forma se tiene que $f_0 = f = 1$. Considerando que $l = m$, tenemos que $m_{f,l} = m_{f_0,m}$, $a = 1$.

Siguiendo los Teoremas 1, 2 y 3 se tiene que $\rho \leq 3$ para el caso fuera de línea y $\rho \leq 5$ para el caso en línea.

El análisis teórico presentado en esta sección tiene especial relevancia ya que viene a cubrir diferentes espacios en el estado del arte. Como se comentó en la sección de trabajo previo, los estudios teóricos a la fecha de este estudio presentaban siempre un modelo con algún tipo de deficiencia en comparación al modelo que se utilizó en este trabajo. Las deficiencias podían ser respecto al tamaño de las máquinas, el tamaño de las tareas y/o clarividencia. Algunos estudios restringen el modelo a que todas las máquinas tienen igual cantidad de procesadores. Otros estudios determinan que las máquinas son de diferente tamaño pero que todas las tareas pueden ejecutarse en la máquina más pequeña. Por último, algunos estudios suponen conocer el tiempo de ejecución de las tareas antes de ejecutarlas. El modelo utilizado en este trabajo define que las máquinas pueden tener diferente tamaño, que las tareas pueden calendarizarse hasta en las máquinas más grandes del Grid y que no se conoce el tiempo de ejecución de las tareas hasta haber completado su ejecución. A pesar de superar estas deficiencias, se presenta un estudio teórico completo determinando el factor de aproximación y el factor de competitividad, para los casos fuera de línea y en línea, respectivamente

Capítulo 4

Análisis experimental

En este capítulo se definen los experimentos que se llevaron a cabo, se muestran los resultados obtenidos y se realiza un análisis de los mismos.

4.1 Configuraciones de Grid

Se formaron varios Grid virtuales. Esto se debe a la falta de cargas de trabajo de Grid reales que cuenten con todas las características para poder llevar a cabo los experimentos requeridos en esta tesis. Estos Grid se hicieron utilizando la información disponible de instalaciones reales de centros de supercómputo distribuidos en el mundo.

Se formaron los Grid 1, Grid 2 y Grid 3. Las características del Grid 1 y del Grid 2, se muestran en la Tabla 1 y la Tabla 2.

Tabla 1. Características de Grid 1

Localidad	Procs.	Bitácora
KTH- Swedish Royal Institute of Technology	100	KTH-SP2-1996-2.swf, 28489 tareas, 204 usuarios
SDSC-SP2 – San Diego Supercenter SP2	128	SDSC-SP2-1998-3.1-cln.swf, 73496 tareas,437 usuarios
HPC2N-High Performance Computing Center North, Sweden	240	HPC2N-2002-1.1-cln.swf , 527371 tareas, 256 usuarios
CTC-Cornell Theory Center	430	CTC-SP2-1996-2.1-cln.swf , 79302 tareas, 679 usuarios
LANL-Los Alamos National Lab	1024	LANL-CM5-1994-3.1-cln.swf, 201387 tareas, 211 usuarios
SDSC-BLUE –San Diego Supercenter Blue Gene	1152	SDSC-BLUE-2000-3.1-cln.swf, 250,440 tareas, 468 usuarios
SDSC-DS – San Diego Supercenter Data Star	1368	SDSC-DS-2004-1-cln.swf, 96089 tareas, 460 usuarios

Estas tablas muestran el nombre del centro de supercómputo, el número de procesadores y la bitácora de tareas, o carga de trabajo, relacionado con cada uno de los sitios. Las

bitácoras de tareas fueron utilizadas para simular la carga de trabajo de cada Grid virtual y en la siguiente sección de este capítulo se explica la forma en que se utilizaron.

Para los experimentos con el modelo de un nivel se consideraron los escenarios de Grid 1 y Grid 2. Para el modelo jerárquico, se utilizaron los 3 escenarios de Grid virtual.

Para el escenario del Grid 3, se considera un Grid mucho más pequeño formado por 11, sitios con un total de 136 procesadores, con los siguientes tamaños de sitio: 4, 4, 4, 4, 8, 8, 8, 16, 16, 32 y 32 procesadores. Se utilizó la carga de trabajo del Grid 2, tomando en cuenta únicamente las tareas que requieren a lo más 32 procesadores. Se ideó esta configuración de Grid para simular una carga de trabajo densa con tareas de alto y bajo paralelismo. Este escenario de prueba simula los casos considerados en los teoremas 1 al 3, donde no todas las tareas pueden ejecutarse en la máquina más pequeña. En esta carga de trabajo, tareas que requieren un solo procesador son el 37%; 2, 4, 8, y 32 procesadores son requeridos por el 23%, 10%, 6% y 5% de las tareas, respectivamente.

En las configuraciones de Grid 1 y Grid 2, la mayoría de los trabajos pueden ejecutarse en la máquina más pequeña, como se considera en el Teorema 4. Por ejemplo, en el Grid 2 donde la mayoría de las tareas requieren a lo más 32 procesadores (97%), éstas pueden ser ejecutadas en los sitios más pequeños del Grid, de 64 procesadores.

Tabla 2. Características de Grid 2

Localidad	Procs.	Bitácora
DAS2 - University of Amsterdam	64	Gwa-t-1-anon_tareas-reduced.swf , 1124772 tareas, 333 usuarios
DAS2 - Delft University of Technology	64	
DAS2 – Utrecht University	64	
DAS2 - Leiden University	64	
KTH - Swedish Royal Institute of Technology	100	KTH-SP2-1996-2.swf, 28489 tareas, 204 usuarios
DAS2- Vrije University Amsterdam	144	Gwa-t-1-anon_tareas-reduced.swf (cont.)
HPC2N - High Performance Computing Center North, Sweden	240	HPC2N-2002-1.1-cln.swf , 527371 tareas, 256 usuarios
CTC - Cornell Theory Center	430	CTC-SP2-1996-2.1-cln.swf , 79302 tareas, 679 usuarios
LANL -.Los Alamos National Lab	1024	LANL-CM5-1994-3.1-cln.swf, 201387 tareas, 211 usuarios

4.2 Bitácora de tareas

Para realizar los experimentos fue necesario crear de forma sintética una bitácora de tareas para cada una de las configuraciones de Grid utilizadas. Éstas consisten en un listado de tareas, cada una con las características necesarias para realizar su calendarización en un Grid virtual, como son tiempo de sometimiento o número de procesadores requeridos para la tarea. Para simular de la forma más cercana a la realidad se utilizaron las bitácoras de tareas de cada sitio individual que se utilizaron para cada uno de los dos Grid de prueba. Es decir, para el Grid 1, formado por 7 sitios, se utilizaron las correspondientes 7 bitácoras de cada uno de los sitios. Las bitácoras se tomaron de PWA (*Parallel Workload Archive*) y GWA (*Grid Workload Archive*). Las premisas para la integración de diferentes bitácoras de tareas de los distintos sitios son las siguientes: la bitácora de Grid contiene tareas sometidas por usuarios de diferentes sitios en el Grid. El Grid está formado por los distintos sitios desde los cuales pueden ser sometidas las tareas. La conformación de un Grid a partir de estos sitios reúne a los usuarios y sus tareas. Sin embargo, conformar una bitácora única a partir de bitácoras de sitios independientes no garantiza una representación exacta de la realidad de un Grid en cuanto a sus usuarios y las tareas que someten se refiere. Por ejemplo, si un sitio se relaciona con otros sitios para formar un Grid, donde máquinas más grandes están disponibles, los usuarios podrían someter tareas de mayor tamaño que no estarían representadas en la bitácora original del sitio independiente. Aún así, es un punto más que adecuado para evaluar estrategias de calendarización de Grid a partir de bitácoras de tareas reales debido a la falta de bitácoras de Grid públicas disponibles y adecuadas para su análisis.

Para conformar cada bitácora de Grid, se llevó a cabo un proceso de filtrado de cada una de las bitácoras por sitio. Las tareas filtradas de cada bitácora fueron aquéllas que cumplían con:

-etiqueta de tarea menor o igual a cero

-tiempo de sometimiento menor a cero

-tiempo de ejecución menor o igual a cero

-número de procesadores requeridos menor o igual a cero

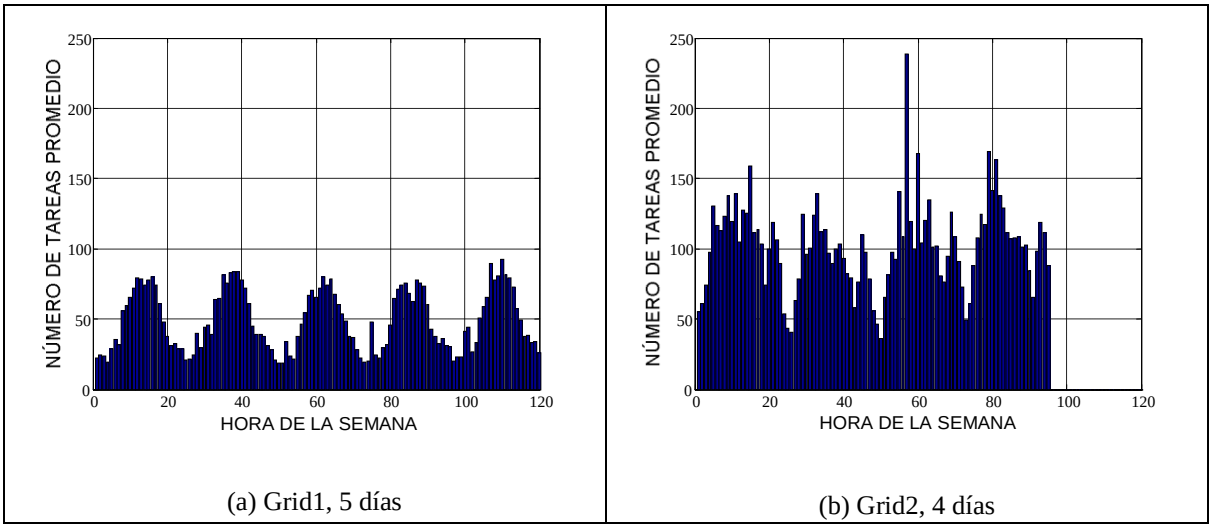
-tiempo solicitado menor o igual a cero

-número de identificación del usuario menor o igual a cero

4.3 Análisis de las cargas de trabajo

A continuación se muestra un análisis de las cargas de trabajo utilizadas. Se realizó desde dos perspectivas. Una de forma general y otra de forma específica para el algoritmo de robo de tareas.

En la Figura 18 puede apreciarse el número de tareas promedio que arriban al Grid, por hora de la semana para el Grid 1 y el Grid 2. Con el fin de utilizar condiciones distintas para los dos Grids, se optó por hacer bitácoras de 5 días, para el caso del Grid 1, y de 4 días para el caso del Grid 2. Esto se hizo con la idea de tener realmente problemas de calendarización donde el tiempo de finalización del calendario esté determinado más por la estrategia de calendarización que por la configuración de Grid o la carga de trabajo utilizada. Por esta razón se utilizaron las tareas de los días lunes a viernes, para el Grid 1, y de lunes a jueves para el Grid 2.



(a) Grid1, 5 días

(b) Grid2, 4 días

Figura 18. Número promedio de tareas sometidas por hora

De haber utilizado las tareas correspondientes a los días sábado y domingo, los sitios estarían mayormente desocupados para esos días; de tal forma que el tiempo de finalización del calendario de Grid estaría más relacionado con alguna tarea en específico

que determinara este tiempo y no con la estrategia de calendarización, que es lo que se está evaluando con estos experimentos.

Ya que la mayor parte de los sitios utilizados se encuentran en América del Norte, puede observarse que la mayor cantidad de tareas arriban al Grid en el horario diurno correspondiente de América del Norte, el cual se utilizó para normalizar el horario de todos los sitios de cada Grid.

La Figura 19 muestra la cantidad de tareas promedio, para el Grid 1 y Grid 2, clasificadas por su tamaño según la definición de los conjuntos A y B del algoritmo de robo de tareas. Esta figura muestra la distribución inicial de la carga. Equivale a ejecutar el algoritmo ST50 sin robo de tareas. Se observa que la mayoría de las tareas, son tareas pequeñas. Para el Grid 2, los primeros 4 sitios tienen el mismo tamaño, por lo que las tareas correspondientes a estos sitios se presentan solo en A1 y B1 en la figura.

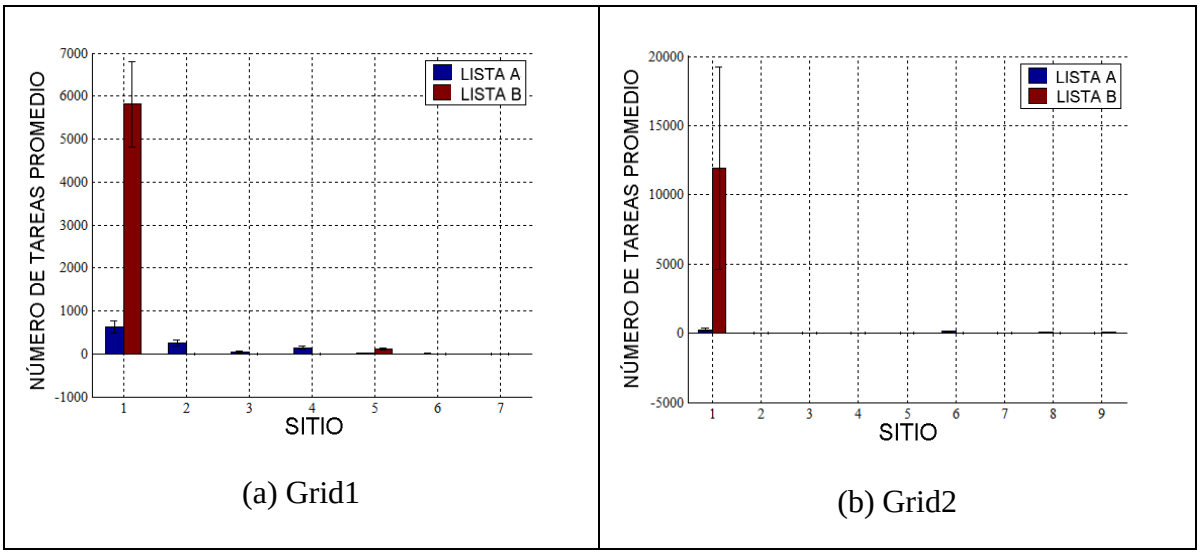
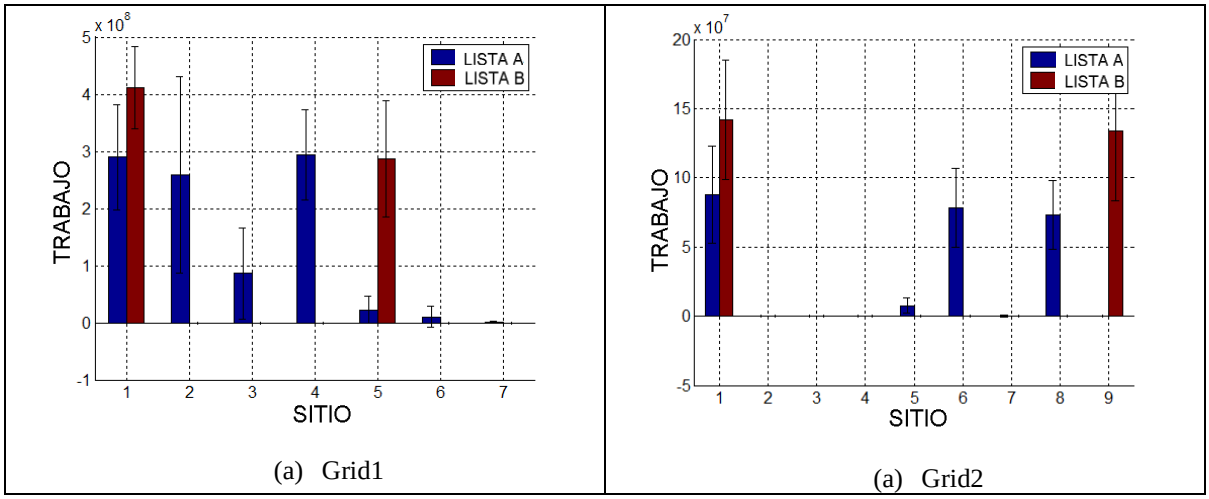


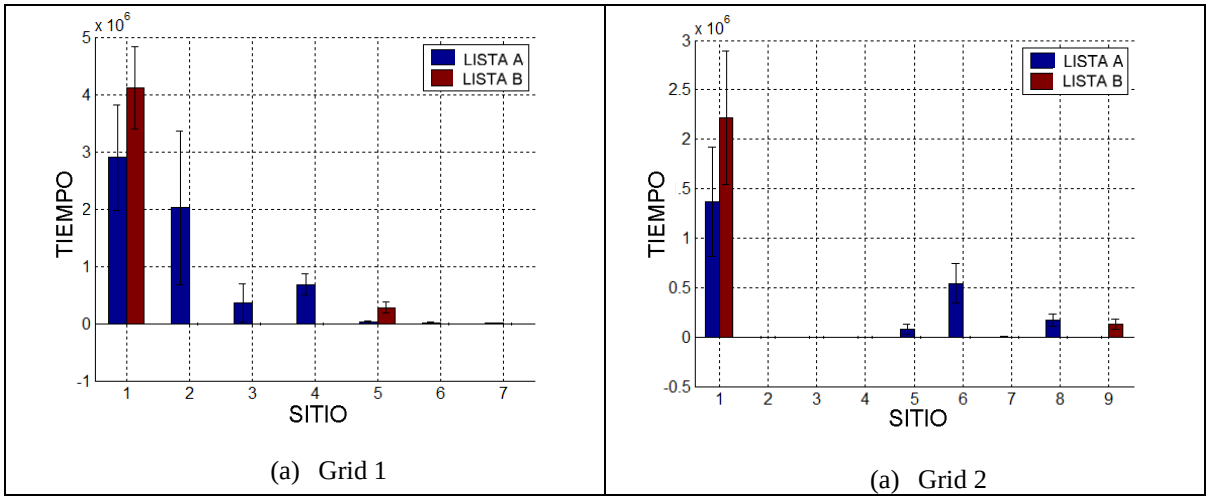
Figura 19. Número promedio de tareas en conjuntos A y B (ST50)



(a) Grid1 (a) Grid2

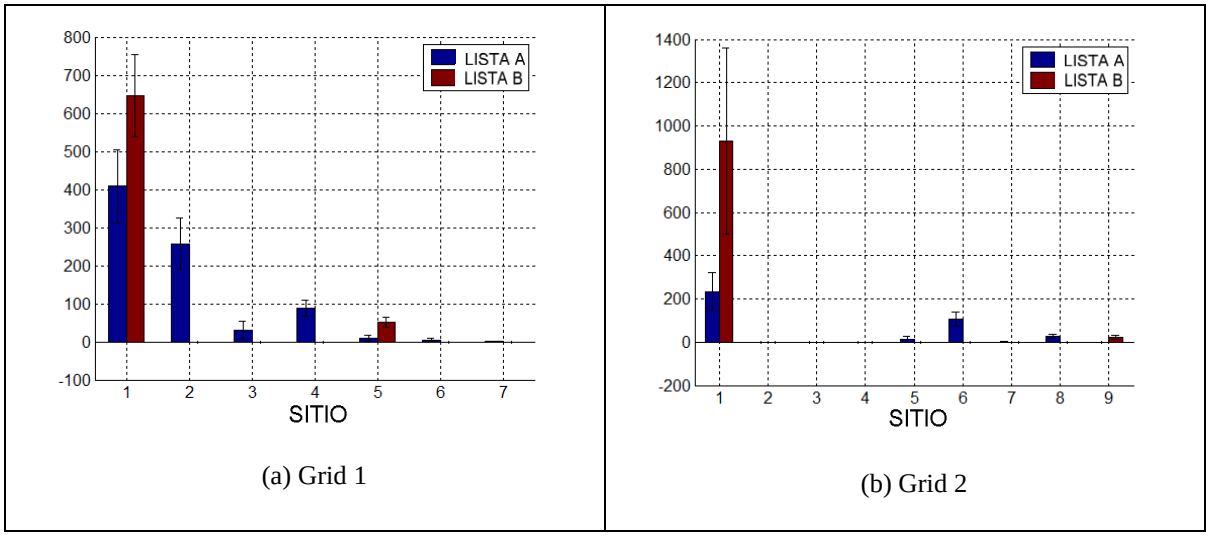
Figura 20. Consumo de recursos promedio por conjuntos A y B (*ST*50).

Sin embargo, la Figura 20 proporciona mejor visión de la carga de trabajo. En esta figura, la cantidad de trabajo se calcula como $W=size_j \,.\, p_j$. Se muestran los promedios por conjuntos A y B. Nuevamente se trata de la distribución inicial, como si ST50 no realizara robo de tareas y esta información es útil para analizar el balanceo de la carga llevada a cabo por el algoritmo. Se observa que para el Grid 1 la mayor cantidad de trabajo lo aportan los sitios más pequeños del Grid.



(a) Grid 1 (a) Grid 2

Figura 21. Consumo de recursos promedio por procesador por conjuntos A y B (*ST*50)



(a) Grid 1

(b) Grid 2

Figura 22. Carga paralela promedio de tareas por procesador en conjuntos A y B (*ST*50)

4.4 Metodología de evaluación de estrategias

La evaluación de los resultados obtenidos en las simulaciones es en base a una metodología propuesta por Tsaffrir et al. en 2007 y aplicada al problema de calendarización en Grid por Ramírez-Alcaraz et al. 2011. Este tipo de evaluación permite un análisis multicriterio basado en las distintas métricas utilizadas a la cuales se le asigna una importancia igual entre cada una de ellas.

El análisis se lleva a cabo de la siguiente manera. Primero se calcula la degradación (error relativo) de cada estrategia en cada métrica. Para esto se utiliza como referencia el valor obtenido por la estrategia con mejor desempeño en dicha métrica, como sigue: 100* métrica_estrategia/ mejor_estrategia_métrica – 100. De esta forma el valor asignado a cada estrategia en cada métrica, representa que tan malo es su desempeño con respecto al resultado con mejor desempeño en la medida de desempeño que está siendo analizada. A cada estrategia se le asigna un valor absoluto y un ranking en cada métrica de cada uno de los Grid. El objetivo de esta metodología es identificar qué estrategias tienen buen desempeño en distintas métricas para diferentes escenarios, es decir, el compromiso que existe en todos los casos de prueba utilizados. Al final, cada estrategia tendrá un valor para cada métrica para cada Grid. Estos valores son promediados y se obtiene un ranking general para todas las estrategias.

4.5 Resultados de estrategias para Grid Jerárquico

La Tabla 3 muestra los resultados de las tres métricas utilizadas para los 3 escenarios de Grid. El valor muestra la degradación promedio para los 30 experimentos que se hicieron con cada escenario de Grid, para cada métrica. Las estrategias mostradas por columna, se trata de las estrategias sin utilizar el esquema de admisibilidad.

Tabla 3. Valores redondeados de degradaciones promedio de estrategias en Grid 1, Grid 2 y Grid 3

Estrategia / Métricas		MCT	MLB	LBal_S	MLp	MPL	Random	MST	MWWT_S_a	MWT
ρ	Grid1	0	0	0	2	0	21	0	7	0
	Grid2	0	0	0	0	0	2	0	0	0
	Grid3	2	22	19	87	17	83	0	45	26
SD_b	Grid1	1284	408	18	71	0	41924	91	12576	204
	Grid2	4001	754	17	53	0	12196	111	7550	628
	Grid3	1	33	10	65	18	46	0	50	42
t_w	Grid1	1256	463	11	119	0	51929	80	17303	252
	Grid2	4833	1799	70	153	0	18027	252	9132	933
	Grid3	1	32	9	65	16	49	0	47	40

En la Tabla 4, se muestra el promedio de las tres métricas por Grid, por estrategia. Al final de la tabla se muestra un ranking donde MPL es la mejor estrategia. Salta a la vista cómo MCT, MWWT_S, MWT y MLB, tienen el peor desempeño, después de Random. Estas estrategias utilizan los niveles más altos de información para realizar la asignación de tareas. El resultado esperado sería que llevaran a cabo una mejor asignación, sin embargo estas estrategias utilizan como información, el tiempo estimado por el usuario, el cual es conocido por ser inexacto. La excepción es MST, la cual, a pesar de usar esta información, tiene muy buen desempeño. Los resultados indican, que en la práctica, asignar una tarea para su ejecución lo antes posible en el calendario, tiene un buen resultado.

Tabla 4. Porcentajes promedio de degradación de desempeño y ranking para Grid 1, Grid 2 y Grid 3

		MCT	MLB	LBal_S	MLp	MPL	Random	MST	MWWT_S	MWT
Promedio	Grid1	846.7	290.3	9.7	64.0	0.0	31291.3	57.0	9962.0	152.0
	Grid2	2944.7	851.0	29.0	68.7	0.0	10075.0	121.0	5560.7	520.3
	Grid3	1.3	29.0	12.7	72.3	17.0	59.3	0.0	47.3	36.0
Promedio para todos los casos		1264.23	390.10	17.13	68.33	5.67	13808.53	59.33	5190.00	236.10
Ranking	Grid1	7	6	2	4	1	9	3	8	5
	Grid2	7	6	2	3	1	9	4	8	5
	Grid3	2	5	3	9	4	8	1	7	6
Ranking para todos los casos		7	6	2	4	1	9	3	8	5

La Tabla 5 muestra los resultados para las estrategias de la Tabla 3 pero con esquema de admisibilidad. Se muestran los promedios por métrica, para los 30 experimentos y los 3 escenarios de Grid. En este caso, MLPa es la mejor estrategia. Aún cuando el orden en el ranking entre estrategias es completamente diferente al obtenido con las estrategias sin esquema de admisibilidad, pueden observarse que el valor promedio de las 3 métricas para los tres escenarios de Grid son mejorados respecto a las estrategias sin esquema de admisibilidad.

Tabla 5. Grid3. Porcentajes de mejora de desempeño para las estrategias con asignación admisible (3 métricas)

Estrategia / Métricas	MCT_a	MLB_a	$LBal\_S_a$	MLp_a	MPL_a	$Random_a$	MST_a	$MWWT\_S_a$	MWT_a
ρ	1.25	2.51	0.37	24.79	1.54	22.68	0.78	2.54	3.06
SD_b	0.96	0.80	3.12	62.69	5.20	7.76	1.78	4.49	11.64
t_w	1.06	1.11	2.75	57.88	4.80	10.24	1.80	0.67	8.29
Promedio	1.09	1.47	2.08	48.45	3.85	13.56	1.45	2.57	7.66
Ranking	9	7	6	1	4	2	8	5	3

Tabla 6. Grid3. Ranking de degradación de desempeño de estrategias con y sin asignación de tareas con admisibilidad

Estrategia / Rank	MCT	MLB	LBal_S	MLp	MPL	Random	MST	MWWT_S	MWT
a=1	2	5	3	8	4	9	1	7	6
a=0.5	3	7	4	1	5	8	2	9	6

En las Tabla 6 y Tabla 7, se muestran los resultados para el Grid 3. Hay que recordar que para este Grid virtual, la carga de trabajo es representativa para estudiar el caso relacionado con los teoremas 1 a 3, en el cual hay gran cantidad de tareas secuenciales, pero también gran cantidad de tareas paralelas y que pueden ocupar el máximo de procesadores de las máquinas más grandes del Grid. En la Tabla 6 pueden observarse los resultados del ranking para el Grid 3 exclusivamente, para estrategias con y sin admisibilidad. Es notable la mejora que se tiene en la estrategia MLp al agregar el esquema de admisiblidad, la cual no tenía un buen desempeño al compararse con las otras estrategias en su definición original.

Tabla 7. Grid 3. Porcentajes de degradación de desempeño con a=0.5 y a=1 (3 Métricas)

Estrat. / Métrics	MCTa	MCT	MLBa	MLB	LBal_Sa	LBal_S	MLp_a	MLp	MPLa	MPL	Random_a	Random	MSTa	MST	MWWT_Sa	MWWT_S	MWT_a	MWT
ρ	1	2	20	23	20	19	41	87	15	17	41	83	0	0	43	45	22	26
SD_b	62	63	114	115	73	79	0	168	81	91	118	136	59	62	142	143	112	130
t_w	44	45	87	90	52	57	0	137	58	66	93	115	41	44	110	111	85	102
Prom.	36	37	74	76	48	52	14	131	51	58	84	111	33	35	98	100	73	86
Rank	4	5	11	12	6	8	1	18	7	9	13	17	2	3	15	16	10	14

La Tabla 7 muestra un ranking para las 18 estrategias, con y sin admisibilidad. Puede observarse que el valor de las métricas SD_b y t_w tienen valores varias veces más grandes que los obtenidos en ρ. Ya que la metodología utilizada da un mismo peso a cada métrica, los

resultados se ven mayormente influidos por el tiempo de espera en la ejecución de las tareas, de forma absoluta y relativa a su tiempo de ejecución.

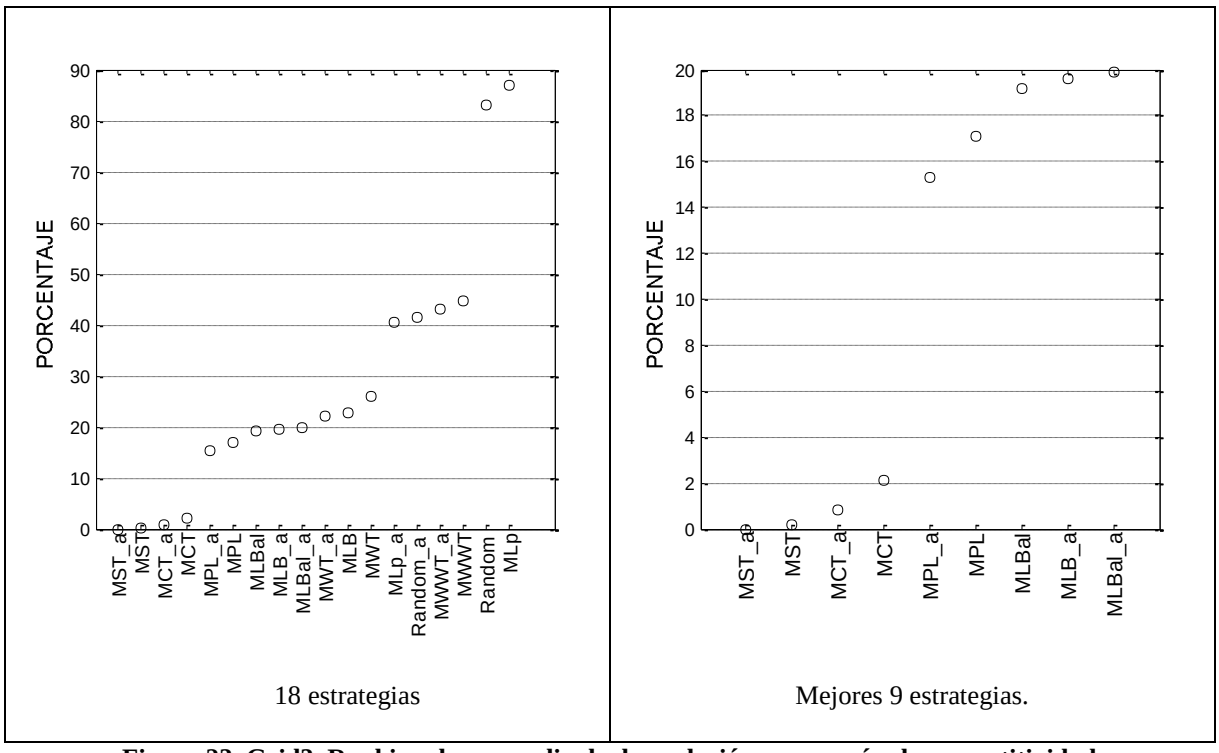


Figura 23. Grid3. Ranking de promedio de degradación para razón de competitividad

La Figura 23 muestra los mismos resultados de la Tabla 7, pero de forma gráfica, para la métrica de razón de competitividad. La Figura 24 y la Figura 25 hacen lo propio para las otras dos métricas. Todos estos resultados son para el Grid 3. Aún cuando el Grid 3 es el más artificial de los 3 escenarios, pues se construyó utilizando bitácoras de tareas filtradas del Grid 2, es el escenario que refleja lo que puede esperarse en la vida real en un Grid, respecto a la cantidad y tipo de tareas. Esto es, gran cantidad de tareas secuenciales y también gran cantidad de tareas que requieran a lo más, el máximo de procesadores disponibles de las máquinas más grandes del Grid.

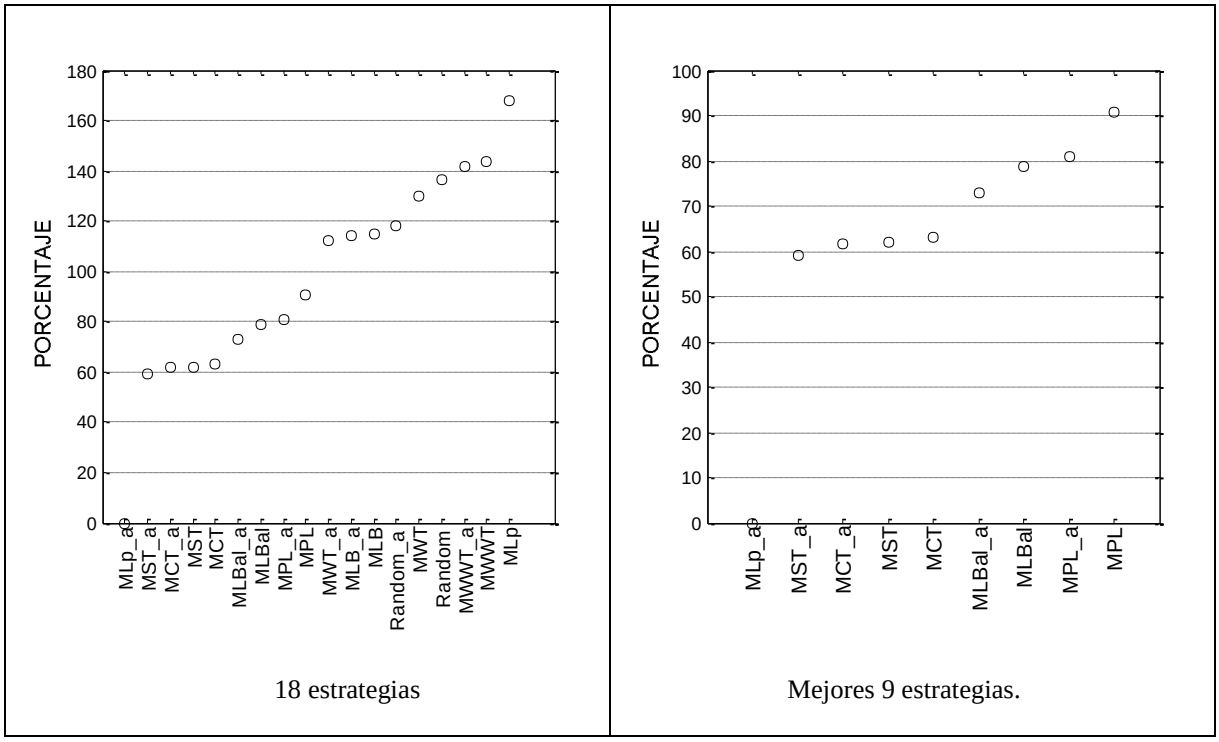


Figura 24. Grid3. Ranking de promedio de degradación de desaceleración acotada promedio

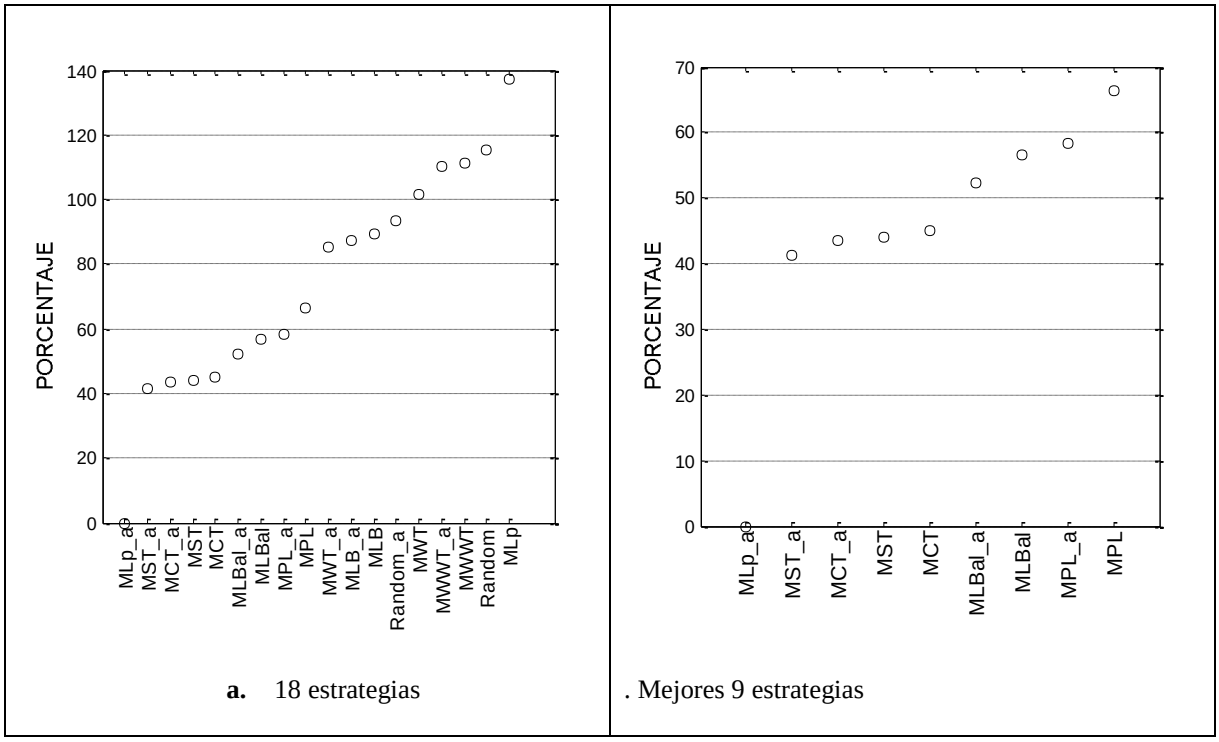
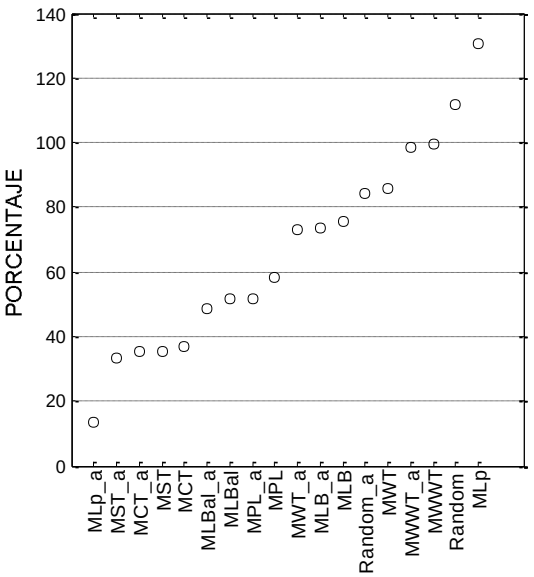
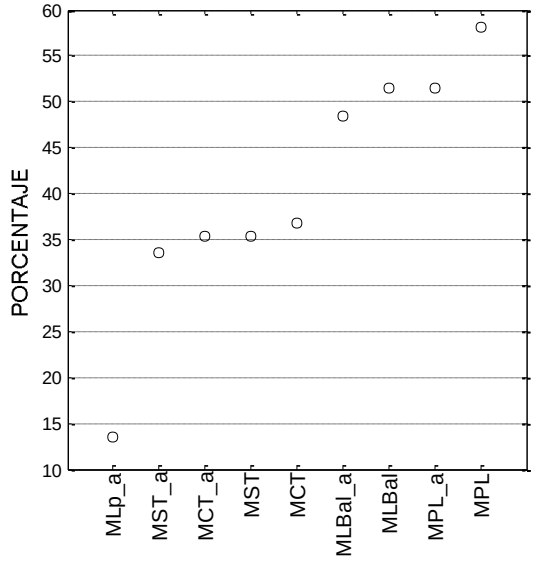


Figura 25. Grid3. Ranking de promedio de degradación de tiempo de espera promedio



a. 18 estrategias



Mejores 9 estrategias.

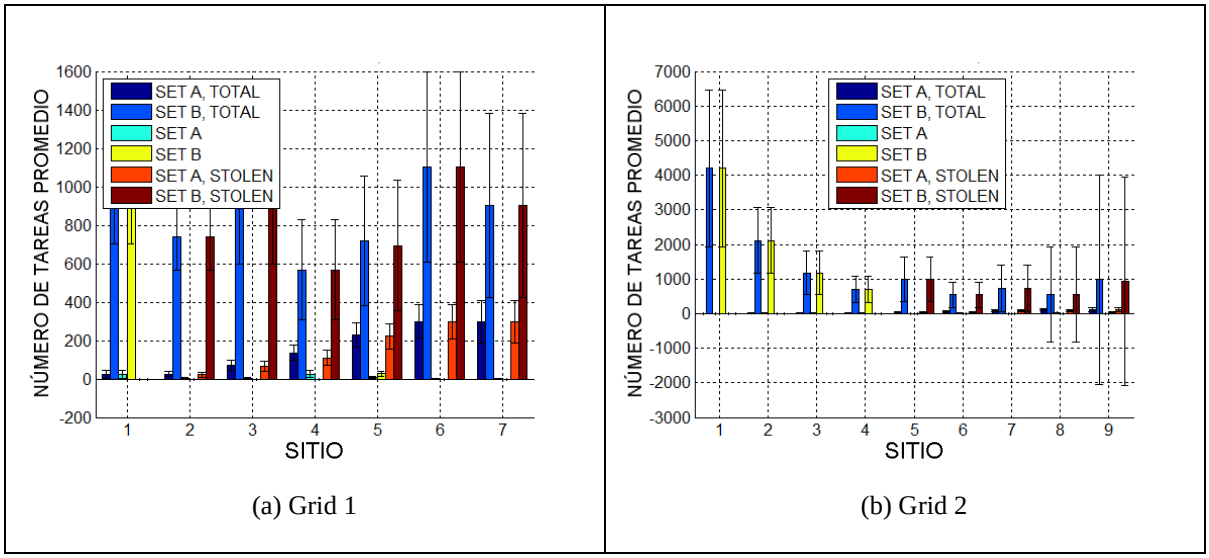
Figura 26. Grid3. Ranking de promedio de degradaciones para todos los casos

La Figura 26 muestra el resultado final, al promediar todas las degradaciones para todas la métricas para el Grid 3.

4.6 Resultados de estrategias de un nivel

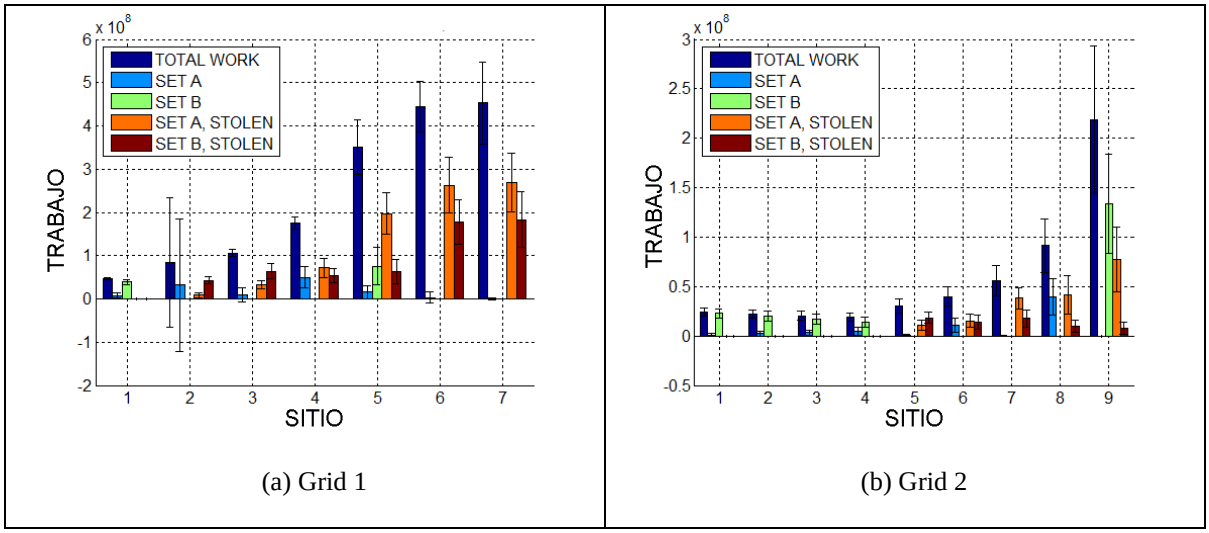
Se muestran y analizan a continuación los resultados de las estrategias para el modelo descentralizado. Tanto las figuras como las tablas, están separadas para los dos escenarios de prueba, Grid 1 y Grid 2.

En la Figura 27 se muestra el número de tareas por sitio, por conjunto A y B, según son ejecutadas. Nos muestra también, qué cantidad de estas tareas fueron robadas de otros sitios. En el Grid 1, puede observarse que mientras más grande sea el sitio, es mayor el número de tareas robadas. Para el Grid 2, los primeros 4 sitios son del mismo tamaño y es por eso que no muestran ninguna tarea robada, al considerar que ejecutaron solo tareas que les pertenecía orignalmente.



(a) Grid 1 (b) Grid 2

Figura 27. Número de tareas promedio ejecutadas en conjuntos A y B por sitio (*ST*50)



(a) Grid 1 (b) Grid 2

Figura 28. Consumo de recursos promedio por tarea ejecutada en conjuntos A y B por sitio (*ST*50)

La Figura 28 muestra la información respecto a la cantidad de trabajo ejecutado por sitio, ya fuera por tareas propias o por tareas robadas. Se observa que en el Grid 1 y Grid 2, los sitios más grandes ejecutan hasta más del 50% de trabajo por tareas robadas. Esto indica que la migración de tareas por medio del robo de las mismas resulta benéfico para el tiempo de finalización de calendario. También nos indica que la distribución inicial de las tareas en colas A y B está pobremente balanceada.

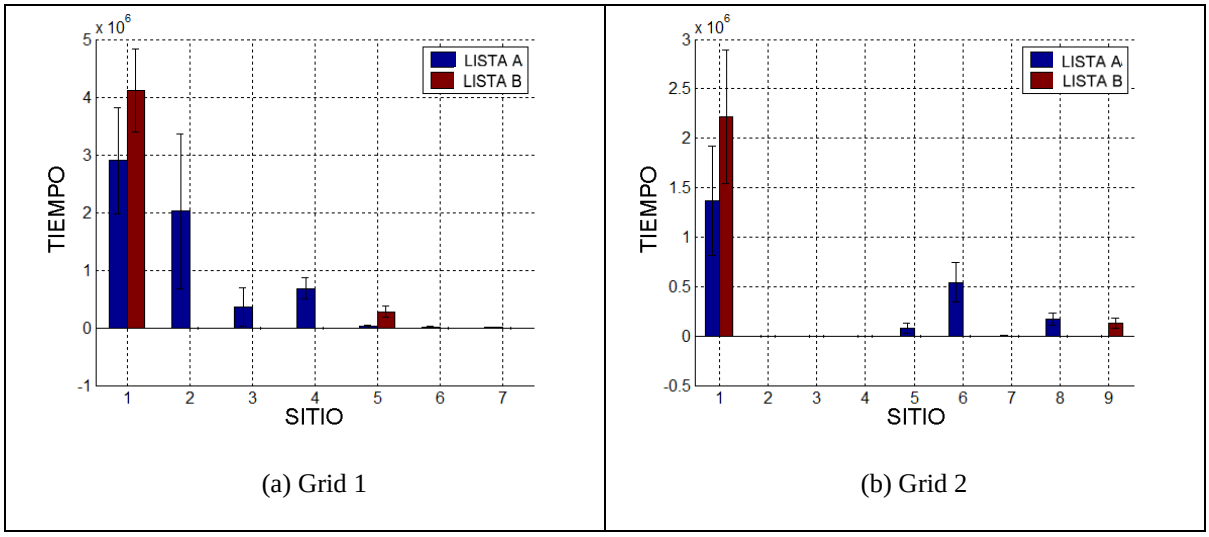


Figura 29. Consumo de recursos promedio de tareas por procesador por sitio en conjuntos A y B (*ST*50), distribución inicial

En la Figura 30 se muestra la carga paralela promedio ejecutada en cada uno de los sitios. Al tratarse de tareas ya ejecutadas, los conjuntos Ay B, representan las tareas no solo propias si no las ajenas. Es decir, todas las tareas de cualquier conjunto A y que se ejecutaron en el sitio 4, están representados por la barra A4. Y de la misma forma para el conjunto B, para cualquiera de los sitios en las dos configuraciones de Grid 1 y Grid 2.

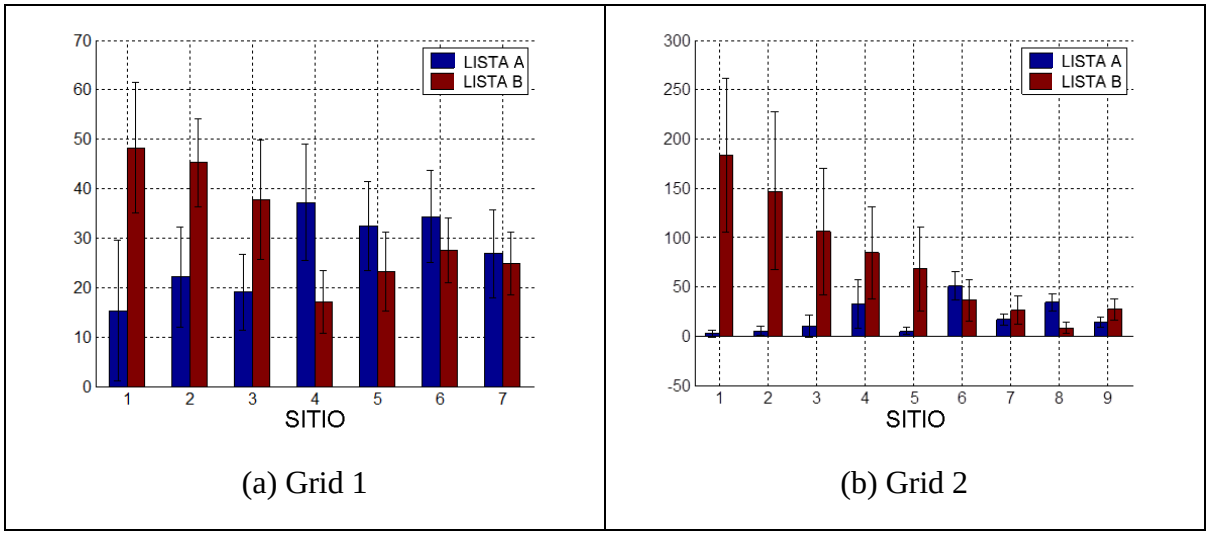


Figura 30. Carga paralela promedio por procesador por tarea ejecutada por conjuntos A y B (*ST*50)

En el Grid 1 puede observarse que el algoritmo de robo de tareas realiza un muy buen balanceo de la carga, especialmente si la comparamos con la distribución inicial de las tareas en los sitios del Grid. En el Grid 2 hay una particularidad en el balanceo de la carga

en los primeros 4 sitios. Estos sitios tienen el mismo tamaño. El algoritmo eligió el sitio para la ejecución de una tarea respetando el orden dentro del Grid. Por resta razón, se observa que se ejecutaron más tareas de la clase B en el sitio 1 y que la cantidad de carga paralela por procesador disminuye en cada sitio hasta llegar al sitio 4. Y sucede exactamente lo contrario con la clase A. La carga paralela aumenta conforme se cambia de sitio hasta llegar al sitio 4. Esto se debe a dos razones. La primera es que las tareas de la clase A bloquean la ejecución de cualquier otra tarea, sin importar si es de la clase A o B. La segunda razón es que al ser el primer sitio la primera opción también para la ejecución de las tareas B, era común durante la ejecución de un calendario que el sitio 1 estuviera más ocupado, o con procesadores no suficientes para ejecutar tareas de la clase A en comparación con los sitios 2, 3 y 4.

El análisis de las Figuras 27 a 30 indican que el algoritmo de robo de tareas realiza un buen balanceo de la carga, al comparar la distribución inicial de la carga en las clases A y B. Es decir, la migración de tareas resulta muy útil para balancear la carga entre las distintas máquinas. Sin embargo no nos dicen nada sobre el tiempo de finalización de las mismas o del calendario. Para ello, se realiza el siguiente análisis utilizando las métricas de razón de competitividad, tiempo de espera promedio y desaceleración acotada promedio.

En la Figura 31 se muestra el factor de competitividad absoluto promedio con su desviación estándar para los 30 experimentos, por Grid. Puede observarse que para el Grid 1 y el Grid 2, las soluciones generadas por las distintas estrategias están dentro del 0.5% en valor promedio. Es decir, las soluciones producidas son de muy buena calidad. También, puede observarse que la desviación estándar es mucho mayor para los casos donde la división de las colas A y B están más uniformemente distribuidas. Esto siginifica que en la práctica, el algoritmo de robo de tareas en su versión original, ST50, no tiene el mejor desempeño en comparación con otras versiones del mismo algoritmo donde la división entre las colas A y B está completamente desproporcionado hacia alguna de las dos colas, para esta métrica.

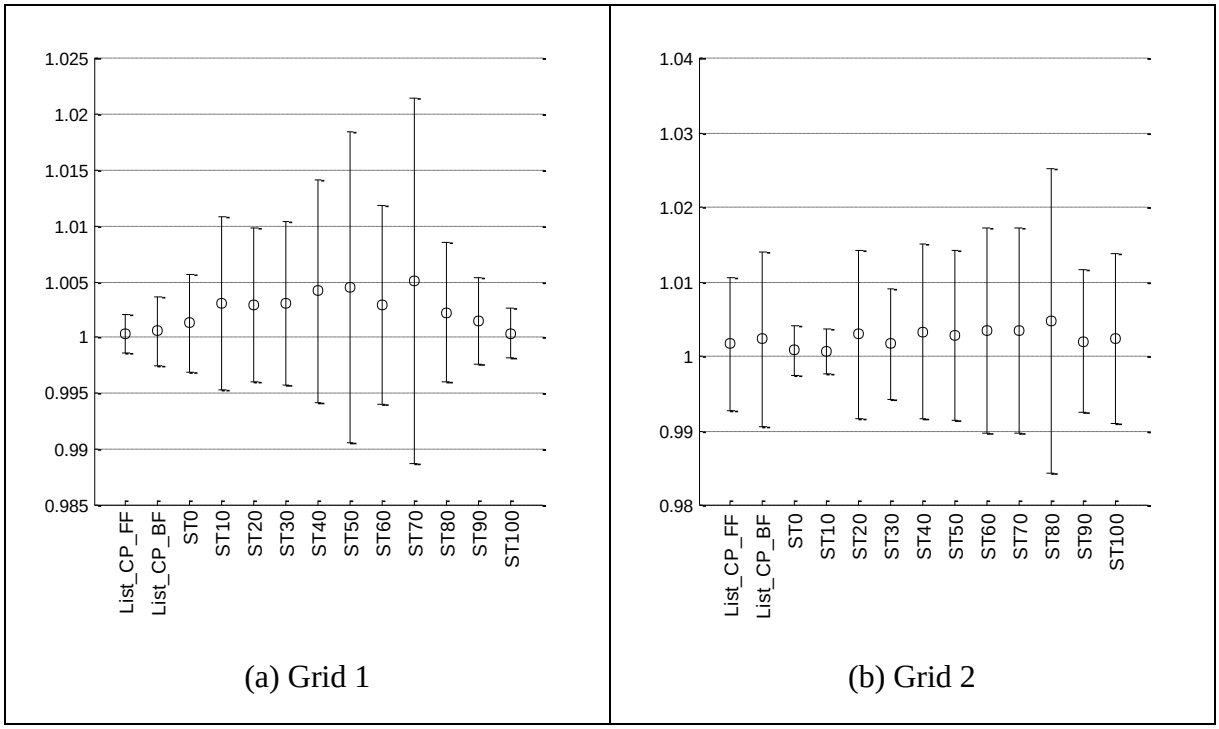


Figura 31. Razón de competitividad promedio y desviación estándar

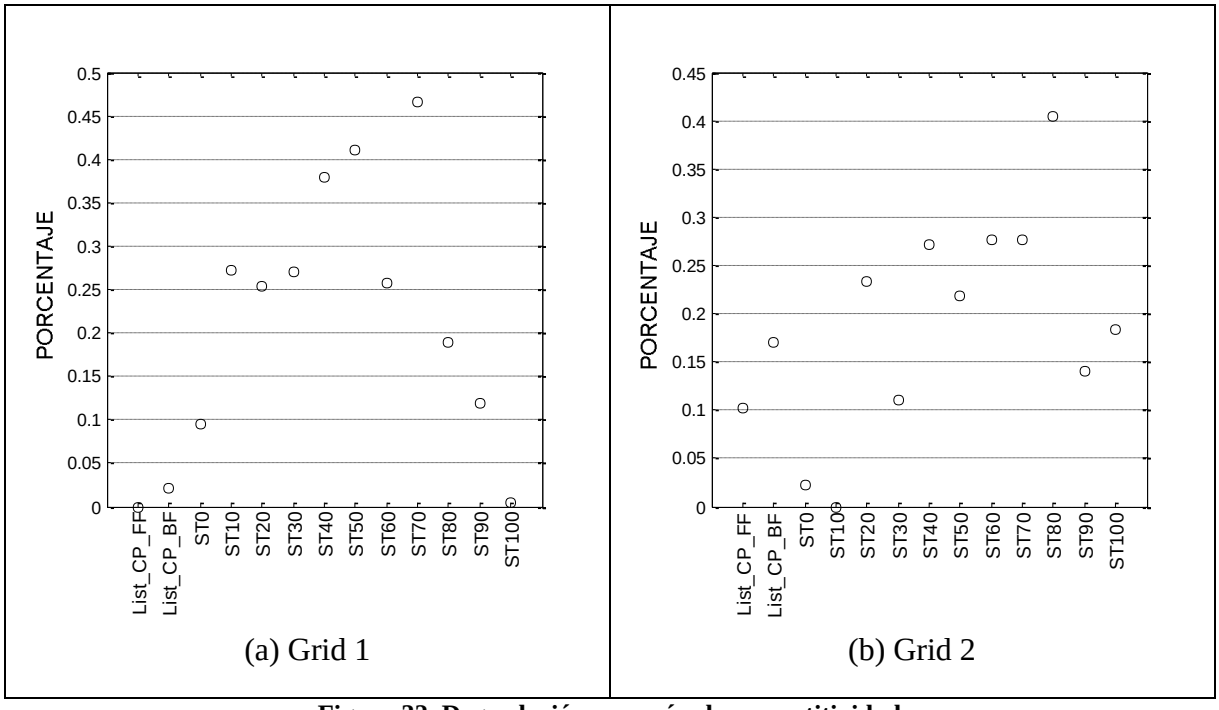


Figura 32. Degradación en razón de competitividad

La degradación para la razón de competitividad se muestra en la Figura 32. Esta figura nos ayuda a ver a mayor detalle las diferencias entre las distintas estrategias respecto al mejor

valor obtenido. Se observa que *List* en sus diferentes versiones obtiene muy buenos resultados junto con algunas versiones del algoritmo de robo de tareas donde prácticamente existe una sola cola por máquina.

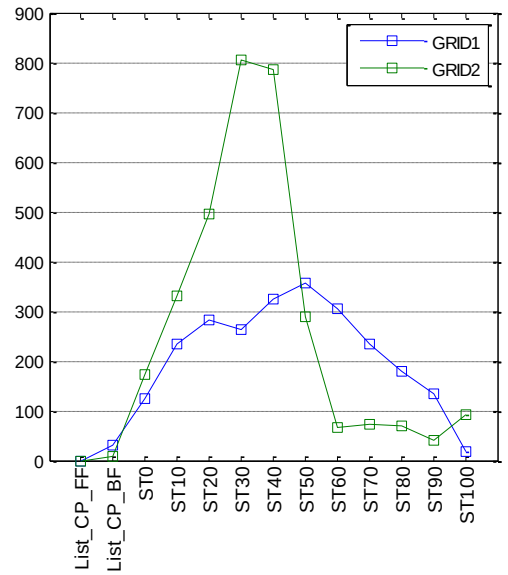


Figura 33. Promedio de degradación de tres métricas, dos Grid

La Figura 33 muestra la degradación promedio para las très métricas y los dos escenarios de Grid. Se observa nuevamente que para versiones de robo de tareas donde hay solo una cola por máquina, ST0 y ST100, se obtienen mejores resultados que para las versiones donde existen dos colas por máquina y el porcentaje de umbral entre las colas A y B, es aproximadamente 50%. La Figura 34 muestra el resultado final, un ranking para todas las estrategias, métricas y escenarios de Grid. Las distintas versiones de List muestran el mejor desempeño. De las versiones del algoritmo de robo de tareas se observa que las mejores son aquellas en las que prácticamente se tiene una cola de tareas por máquina. El análisis del algoritmo y los restultados obtenidos muestran que las tareas ubicadas en la cola A de una máquina retrasan la ejecución de tareas en las colas B de la misma máquina, o en cualquier cola de cualquier otra máquina, de tal forma que se tiene un impacto muy negativo en la ejecución del calendario. Es decir, al menos en los escenarios de prueba, no existen tantos casos en los que tareas secuenciales atrasen la ejecución de tareas paralelas que provoquen que el algoritmo de robo de tareastenga un mejor desempeño que List..

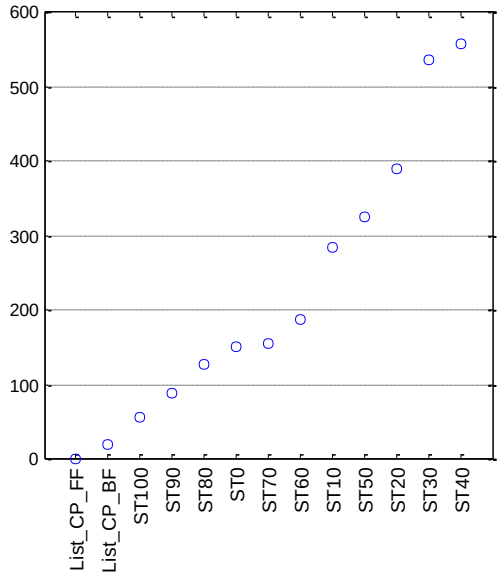


Figura 34. Ranking, promedio de 3 métricas en dos Grid

En las tablas 8 (a) y (b) muestran el número promedio de tareas que fueron robadas por sitios más grandes a sitios más pequeños. El sitio original de sometimiento de las tareas es el de la fila *j* y el sitio que robó las tarea se muestra en las columnas *i*. Puede observarse que un gran número de tareas son migradas de los sitios más pequeños. Al sitio más grande del Grid no le roban ninguna tarea, ya que la distribución original de las tareas se hace de forma tal que solo puede ejecutarse en ese sitio o en sitios más grandes. Sin embargo, el número de tareas promedio pudiera ser un tanto subjetivo. Esto es porque se agrupan tareas independientemente de si son secuenciales o paralelas, y puede existir una gran diferencia en la cantidad de trabajo que representan. Por esta razón se muestran las tablas 10 (a) y (b), donde puede observarse la cantidad de trabajo robado por cada uno de los sitios, expresado como un porcentaje del trabajo total del sitio original. Es decir, que parte del total de trabajos del sitio de la columna *j* fue robado por el sitio de la columna *i*. Hay que recordar que para el Grid 2, se considera que las tareas para los sitios 1,2,3 y 4, se consideran que originalmente fueron sometidos al sitio 1 solamente, ya que estos sitios son del mismo tamaño.

Tabla 8. Número promedio de tareas migradas de sitios originales de sometimiento

Sitio *i* / *j*	1	2	3	4	5	6	7
1	0	762	967	648	821	1229	1010
2	0	0	9	24	57	81	82
3	0	0	0	3	7	13	12
4	0	0	0	0	28	39	48
5	0	0	0	0	0	36	41
6	0	0	0	0	0	0	4
7	0	0	0	0	0	0	0
total	0	762	976	675	913	1398	1197

(a) Grid 1

Sitio *i* / *j*	1	2	3	4	5	6	7	8	9
1	0	2121	1184	710	1025	583	784	591	981
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	2	4	4	3
6	0	0	0	0	0	0	24	35	48
7	0	0	0	0	0	0	0	0.3	0.6
8	0	0	0	0	0	0	0	0	18
9	0	0	0	0	0	0	0	0	0
total	0	0	0	0	1025	585	812	630	1050

(b) Grid2

Uno podría esperar que el número de tareas robadas de cada sitio fuera disminuyendo conforme se analiza de un sitio más pequeño a uno más grande, ya que mientras más pequeña es la máquina hay más máquinas dispuestas a robar tareas. Sin embargo, esto no sucedió y la razón es debido a la distribución inicial de la carga de trabajo. El hecho de que haya más máquinas con recursos ociosos no necesariamente implica que se robarán más tareas, pues es posible que el sitio original de sometimiento de tareas pueda ejecutar el mayor número de ellas, sin dejar tareas disponibles para ser robadas por sitios más grandes.

Lo mismo sucedería al observar la cantidad de trabajo. El resultado esperado era que la cantidad de porcentaje de trabajo robado fuera aumentando, al ir de sitios más pequeños a sitios más grandes, en la columna del total de porcentaje de trabajo robado. Sin embargo no sucede así, por ejemplo en el Grid 1, el sitio 4 robó un total promedio de 81% de trabajo y el sitio 5, un sitio más grande, solo robó el 69.4%.

Tabla 9. Porcentaje promedio de trabajo migrado desde el sitio original de sometimiento

Sitio *i* / *j*	1	2	3	4	5	6	7	Total
1	0	11.8	15.0	10.1	12.8	19.1	15.7	84.5
2	0	0	3.5	9.3	22.1	31.4	31.8	98.1
3	0	0	0	7.3	17.1	31.7	29.3	85.4
4	0	0	0	0	19.7	27.5	33.8	81.0
5	0	0	0	0	0	32.4	36.9	69.4
6	0	0	0	0	0	0	80.0	80.0
7	0	0	0	0	0	0	0	0.0

(a) Grid1

Sitio *i* / *j*	1	2	3	4	5	6	7	8	9	Total
1	0	17.4	9.7	5.8	8.4	4.8	6.4	4.9	8.1	65.5
2	0	0	0	0	0	0	0	0	0	0.0
3	0	0	0	0	0	0	0	0	0	0.0
4	0	0	0	0	0	0	0	0	0	0.0
5	0	0	0	0	0	12.5	25	25	18.8	81.3
6	0	0	0	0	0	0	19.8	28.9	39.7	88.4
7	0	0	0	0	0	0	0	30	60	90.0
8	0	0	0	0	0	0	0	0	39.1	39.1
9	0	0	0	0	0	0	0	0	0	0.0

(b) Grid2

Capítulo 5

Conclusiones

El uso de Grids es cada vez más generalizado. Por esta razón, el problema de asignación de recursos y calendarización es crucial, no solo para lograr alto desempeño en el Grid, sino también para satisfacer diferentes demandas. Cada vez se utilizan más los modelos teóricos de calendarización de Grid basados en análisis de peor caso; técnicas estadísticas de rápida ejecución se aplican a datos reales y han mostrado ser efectivas para calendarización.

El presente trabajo aborda el problema de calendarización en línea para Grid. Se utilizaron dos modelos de calendarización, jerárquico y descentralizado, de los cuales se obtuvieron varios resultados.

Para el modelo jerárquico se realizó un estudio teórico y experimental. Usando el esquema de admisibilidad se derivaron cotas de factor de competitividad para dos estrategias de calendarización: MLBa+PS y MCTa+PS. Las cotas se obtuvieron para el caso en línea y fuera de línea y se establecieron para dos tipos de carga, dependiendo del tamaño de la tarea. Para la carga donde las tareas tienen tamaño de hasta la máquina más pequeña del Grid, las cotas fueron de 3 para el caso fuera de línea y de 5 para el caso en línea. Para la carga en la cual las tareas son de hasta el tamaño de la máquina más grande, las cotas obtenidas fueron de 9 para el caso fuera de línea y de 11 para el caso en línea. El modelo que utilizan comúnmente presenta deficiencias en las características de las máquinas o de la carga de trabajo, si se comparan con los teoremas presentados en este trabajo. Algunos ejemplos de estas deficiencias son que todas las máquinas del Grid son de un mismo tamaño o bien, que las tareas son de hasta el tamaño de la máquina más pequeña.

Se estudiaron 9 estrategias de asignación diferentes, dependiendo del tipo y cantidad de información que requieren junto a BEFF como calendarizador local. Se llevó a cabo una avaluación completa de su desempeño práctico usando simulación de tres escenarios de Grid en 3 métricas.

Se presentó un estudio detallado del impacto del factor de admisibilidad, incorporado a las estrategias de asignación para calendarizacion en Grid, sobre las medidas de desempeño de Grid presentadas en este trabajo. Se mostró que los algoritmos se benefician bajo circunstancias definidas.

La selección de estrategias apropiadas de Grid depende de las preferencias de distintos actores decisivos en el Grid (usuarios finales, proveedores de recursos locales y administradores de Grid). Para proveer una guía efectiva en la selección de una buena estrategia, se llevó a cabo un análisis de tres métricas basadas en la degradación del desempeño de cada una de ellas en cada métrica para todos los casos de prueba.

Los resultados en simulación presentados en este trabajo revelan que en términos de los criterios considerados, las estrategias de asignación con admisibilidad superan a los algoritmos que utilizan todos los sitios disponibles para la asignación de tareas. Se concluye que la estrategias de asignación con admisibilidad adaptable mantienen un buen desempeño en diferentes condiciones de carga real, período de prueba y configuraciones de máquinas que constituyen un Grid.

En Grids reales, el factor de admisibilidad puede ser ajustado dinámicamente para enfrentar las circunstancias cambiantes de la ejecución de una carga de trabajo. Para tal fin, la carga de tareas ya ejecutada debe de ser analizada por algún intervalo de tiempo para poder determinar el factor de admisibilidad apropiado. El tiempo de intervalo debe de ser ajustado de acuerdo a características de la carga de tareas y la configuración del Grid.

Cuando se examina el desempeño general del Grid con datos reales, se encontró que las estrategias que se desempeñan mejor son aquellas que toman en cuenta el tamaño de la tarea (número de procesadores requeridos) para llevar a cabo su asignación. Estrategias que utilizan tiempos de ejecución estimados por el usuario y calendarios locales para llevar a cabo la asignación de tareas, mostraron peor desempeño.

El resultado esperado era que estrategias que utilizaban mayor información tuvieran mejor desempeño que las que usaban menor información. Esto no fue así y al analizar los resultados se llegó a la conclusión de que tiempo estimado de ejecución proporcionado por el usuario es poco exacto. Y, al utilizar esta información, se promueve la creación de calendarios erróneos a nivel de asignación. Por erróneos se refiere a calendarios donde se atrasan innecesariamente la ejecución de tareas por una mala estimación de la finalización de tareas anteriores o un balanceo inadecuado de la carga de trabajo en todo el Grid.

Para el modelo descentralizado se realizó un estudio experimental. Se implementaron diferentes versiones del algoritmo de robo de tareas, con variaciones en el orden de ejecución de tareas según los conjuntos A, H o B y también variando la selección de las mismas, con *FF* y *BF*. Los resultados obtenidos mostraron que no existe diferencia significativa entre esas variantes y por esta razón se utilizó la versión original del algoritmo para analizarlo de manera profunda. Se varió el umbral que existe entre las colas A y B para obtener distintas versiones del algoritmo. Estas versiones se compararon con algoritmos que realizan el intercambio de tareas a través de una cola global y tienen una sola cola local. El resultado esperado era que el algoritmo de robo de tareas, en el cual existen dos colas locales por máquina y las tareas se ejecutan según la prioridad definida para las colas, tuviera un mejor desempeño, sin embargo no fue así. Los algoritmos con una sola cola local tuvieron mejor desempeño. Incluso las versiones de robo de tareas, donde el valor de frontera forza a la existencia de una sola cola local por máquina, tuvieron mejor desempeño que aquellas versiones del algoritmo que tenían dos colas con una distribución por tamaño de tareas más uniforme. Esto se corroboró al observar mejores resultados con List Common

Pool. También ST0 y ST100, donde todas las tareas pertenecen a una sola cola. Al analizar los resultados se concluyó que el algoritmo, al atrasar la ejecución de tareas de los conjuntos B, siempre que hay tareas de los conjuntos A en ejecución, tiene un impacto negativo en las métricas utilizadas para evaluar el desempeño de las estrategias.

Trabajo Futuro

Una línea de investigación tiene que ver con la forma de analizar las medidas de desempeño. En este trabajo se utilizaron 3 medidas de desempeño, de las cuales se obtenía un valor absoluto para cada experimento, que después se promediaban con los valores obtenidos en 30 experimentos. Sin embargo, el valor de una métrica puede aumentar o disminuir a lo largo de la calendarización de un conjunto de tareas. Esto dependerá de las características de la carga de trabajo y del Grid. De esta forma, el valor de una métrica puede ser mayor o menor, dependiendo del punto donde se interrumpa el experimento. Por lo tanto, puede resultar interesante analizar qué es lo que sucede con estas métricas en línea, ya que el valor obtenido al final de un experimento no detalla durante su ejecución el comportamiento de determinada métrica.

Otra línea de investigación es el uso de las estrategias de calendarización en Grid estudiadas en esta tesis a otros tipos de problema. Por ejemplo, la calendarización de contratos de energía eléctrica. El modelo utilizado para el Grid es muy semejante al de la red eléctrica para estudiar dicho problema. Las máquinas del Grid pueden verse como las máquinas generadoras de electricidad de una central eléctrica. Y los contratos pueden identificarse con las tareas que son asignadas al Grid. La potencia de generación se puede relacionar directamente con el número de procesadores de una máquina paralela. Sin embargo, varios ajustes en el modelo son necesarios para abordar el problema. Si bien en el modelo utilizado en esta tesis no se considera la co-asignación de tareas en varias máquinas, esto es algo común en la administración de la energía eléctrica. Los contratos se proporcionan a centrales eléctricas las cuales comúnmente reparten la carga en varias máquinas diferentes. También intervienen otro tipo de métricas. En nuestro estudio no se consideraron métricas

de tipo económico, las cuales resultan indispensables para la asignación de contratos de energía eléctrica. Estos contratos intervienen en un mercado internacional, de costos variables. Las operaciones de compra/venta de energía se realizan en un mercado nacional y también con EUA. El problema resultante sigue siendo NP-Difícil y por lo tanto no se conoce una solución óptima para todos los casos del problema. Vale la pena estudiar si el tamaño de entrada del problema da por resultado un espacio de soluciones tan grande que no se pueden atacar con fuerza bruta y entonces poder proponer alguna solución aproximada que mejore los algoritmos que actualmente se utilizan en la industria.

Referencias

Abraham, A., Buyya, R., y Nath, B. (2000). *Nature's Heuristics for Scheduling Jobs on Computational Grids*. The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), Cochin, India.

Bar-Noy, A., Freund, A. (2001) On-Line Load Balancing in a Hierarchical Server Topology. *SIAM Journal of Computing* 31. 527-549 p.

Bougeret, M., Dutot, P.-F., Jansen, K., Otte, C., Trystram, D. (2010) *A Fast 5/2 Approximation Algorithm for Hierarchical Scheduling*. In: 16th International European Conference on Parallel and Distributed Computing, Euro-Par 2010. Ischia, Italy.

Bougeret, M., Dutot, P., Jansen, K., Otte, C. & Trystram, D. (2010) *Approximation Algorithms for Multiple Strip Packing*. In: E. Bampis y K. Jansen (Eds.), Approximation and Online Algorithms. Springer Berlin / Heidelberg. Vol. 5893, 37-48 p.

Dong, F. y Akl, S. G. (2006) Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. *Technical Report No. 2006-504 School of Computing, Queen's University* Kingston, Ontario. 55 p.

Foster, I., y Kesselman, C. (2003) The Grid in a Nutshell. En: J. Nabrzyski, J. M. Schopf y J. Weglarz (Eds.), *Grid Resource Management, State of the Art and Future Trends*. Kluwer Academic Publisher. Norwell, MA, USA. 359-373 p.

Foster, I., y Kesselman, C. (1999) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

Foster, I., Kesselman, C., y Tuecke, S. (2001) *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International Journal of High Performance Computing Applications, 15(3): 200-222 p.

Frachtenberg, E., y Feitelson, D. (2005) Pitfalls in Parallel Job Scheduling Evaluation. En: D. Feitelson, E. Frachtenberg, L. Rudolph y U. Schwiegelshohn (Eds.), *Job Scheduling Strategies for Parallel Processing*. Springer Berlin / Heidelberg Vol. 3834. 257-282 p.

Garey, M.R., Graham, R.L. (1975) Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing* 4. 187-200 p.

Graham, R.L. (1969). Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, Vol. 17, No. 2. 416-429 p.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In: Hammer, P.L., Johnson, E.L., Korte, B.H. (eds.) *Annals of Discrete Mathematics 5. Discrete Optimization II*. North-Holland. 287-326 p.

Grimme, C., Lepping, J., y Papaspyrou, A. (2007). Identifying Job Migration Characteristics in Decentralized Grid Scheduling Scenarios. 19th International Conference on Parallel and Distributed Computing and Systems, Cambridge, MA, USA. 124-129 p.

Grimme, C., Lepping, J., y Papaspyrou, A. (2008). Prospects of collaboration between compute providers by means of job interchange, *Proceedings of the 13th international conference on Job scheduling strategies for parallel processing*. Seattle, WA, USA: Springer-Verlag. Volume 4942/2008. 132-151 p.

GWA Grid Workloads Archive, TU Delft, http://gwa.ewi.tudelft.nl

Hamscher, V., Schwiegelshohn, U., Streit, A., y Yahyapour, R. (2000). Evaluation of Job-Scheduling Strategies for Grid Computing. First IEEE/ACM International Workshop on Grid Computing (GRID 2000), Bangalore, India.

Hirales-Carbajal, A.,Tchernykh, A., Yahyapour, R., Röblitz, T.,Ramírez-Alcaraz, J. M., González-García, J. (2012) Multiple Parallel Workflow Scheduling Strategies on a Grid. Journal of Grid Computing, Springer-Verlag, Netherlands, DOI: 10.1007/s10723-012-9215-6, ISSN:1570-7873.

Hurink, J.L., Paulus, J.J. (2008). Online scheduling of parallel jobs on two machines is 2-competitive, *Operations Research Letters*, Volume 36, Issue 1. Elsevier 51-56 p.

Kamalan, G.K. y Murali, V. (2011). An effective approach to job scheduling in decentralized Grid environment. *International journal of computer applications*. Vol 24. Número 1.

Krauter, K., Buyya, R., Maheswaran, M. (2002). A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *International Journal of Software: Practice and Experience (SPE)* 32, 135-164

D.A. Lifka, "The ANL IBM SP Scheduling System", Lecture Notes in Computer Science, Vol. 949, p. 303, 1995.

Naroska, E., Schwiegelshohn, U. (2002). On an on-line scheduling problem for parallel jobs. *Information Processing Letters* 81, 297-304

PWA Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/

Pascual, F., Rzadca, K., Trystram, D. (2009). Cooperation in multi-organization scheduling. *Concurrency and Computation: Practice & Experience* 21. 905-921 p.

Quezada-Pina, A., Tchernykh, A., González-García, J.L., Hirales-Carbajal, A., Miranda-López, V., Ramírez-Alcaraz, J.M., Schwielgelshohn, U., Yahyapour, R. (2012) Adaptive job scheduling on hierarchical Grids. *Future Generation of Computer Systems*, Elsevier Science. DOI: 10.1016/j.future.2012.02.004,ISSN:0167-739X

Ramírez-Alcaraz J.-M., Tchernykh, A. Yahyapour,R., Schwiegelshohn, U., Quezada-Pina, A., González-García,J.-L., Hirales-Carbajal, A. (2011). Job Allocation Estrategias with User Run Time Estimates for Online Scheduling in Hierarchical Grids. *Journal of Grid Computing*, 9:95-116.

Schopf, J. M. (2003). Ten Actions When Grid Scheduling. En: J. Nabrzyski, J. M. Schopf y J. Weglarz (Eds.), *Grid Resource Management. State of the Art and Future Trends*. Kluwer Academic Publisher.

Schwiegelshohn, U. (2009). An Owner-centric Metric for the Evaluation of Online Job Schedules. *Multidisciplinary International Conference on Scheduling. Theory and Applications (MISTA 2009)*, Dublin, Ireland. 557-569 p.

Schwiegelshohn, U., Tchernykh, A., Yahyapour, R. (2008). On-line Scheduling in Grids. In: *IEEE International Symposium on Parallel and Distributed Processing 2008 (IPDPS 2008)*, Miami, FL, USA. 1-10 p.

Shmoys, D., Wein, J. y Williamson, D. (1995). Scheduling parallel machines on-line. *SIAM Journal on Computing*, Vol. 24(6). 1313–1331 p.

Tchernykh, A. (2006). Admissible Resource Selection Strategies in Two Level Job-Scheduling for a Computational Grid. *Scheduling Algorithms for new Emerging Applications (book of abstracts)*. Marseille - Lumini - CIRM., France

Tchernykh A., Schwiegelsohn U., Yahyapour R., Kuzjurin N. (2010). On-line Hierarchical Job Scheduling on Grids with Admissible Allocation, *Journal of Scheduling, Vol. 13(5)*, Springer-Verlag, Netherlands. 545-552 p.

Tchernykh, A., Ramírez, J., Avetisyan, A., Kuzjurin, N., Grushin, D., Zhuk, S. (2005). Two Level Job-Scheduling Strategies for a Computational Grid. En: Wyrzykowski, R., Dongarra, J., Meyer, N., Wasniewski, J. (eds.) 6th International Conference on Parallel Processing and Applied Mathematics PPAM 2005, vol. LNCS 3911. Poland. 774-781 p.

Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N. (2008). On-line Hierarchical Job Scheduling on Grids. In: Priol, T., Vanneschi, M. (eds.) From Grids to Service and Pervasive Computing, Springer (EUA). 77-91 p.

Tsafrir, D., Etsion, Y., Feitelson, D.G. (2006). Modeling User Runtime Estimates. In: Feitelson, D.G., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) *11th Workshop on Job Scheduling Estrategias for Parallel Processing (JSSPP 2005)*, vol. LNCS 3834, Springer, Cambridge, MA, EUA. 1-35 p.

GWF The Grid Workload Format.

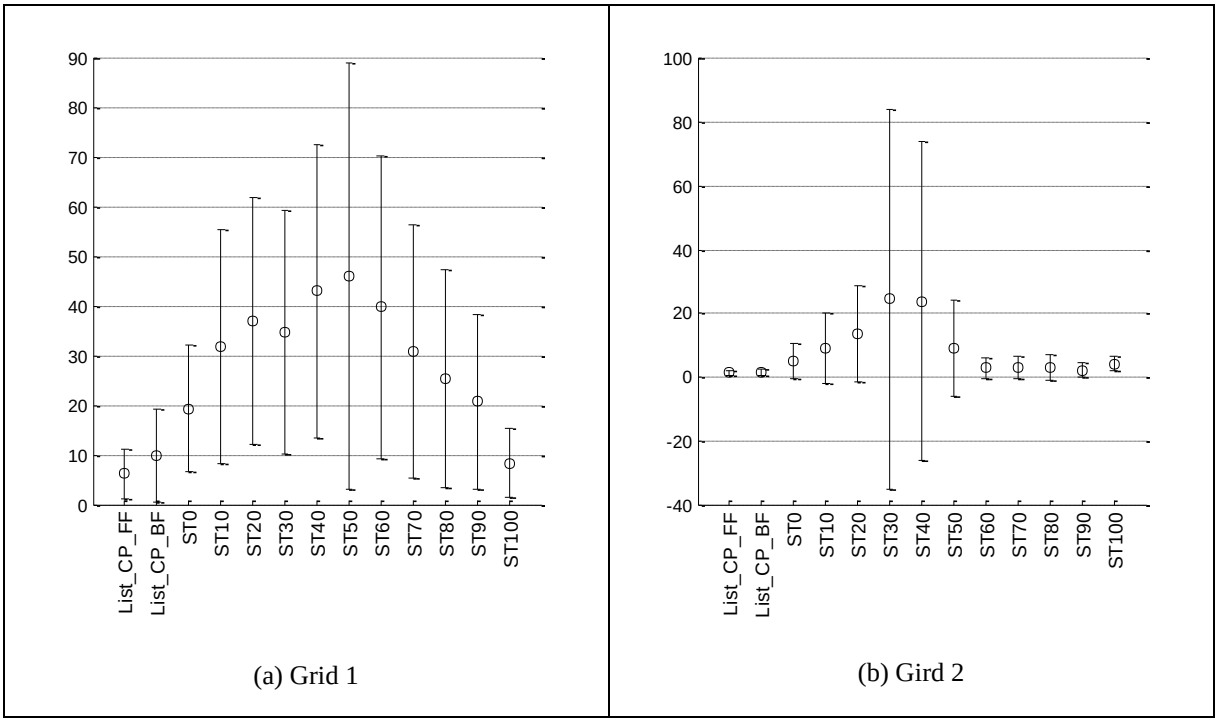
Ye, D., Han, X., y Zhang, G. (2011). Online multiple-strip packing. *Theoretical Computer Science, Combinatorial Optimization and Applications, COCOA2009*. Vol. 412(3). 233-239 p.

Zhuk, S., Tchernykh, A., Avetisyan, A., Gaissaryan, S., Grushin, D., Kuzjurin, N., Pospelov, A., Shokurov, A. (2004). Comparison of Scheduling Heuristics for Grid Resource Broker. In: *Third International IEEE Conference on Parallel Computing Systems (PCS 2004), IEEE*, Colima, Colima, México. 388-392 p.

Zikos, S., y Karatza, H. D. (2009). Communication cost effective scheduling policies of nonclairvoyant jobs with load balancing in a grid. *Journal of System and Software*, Vol. 82(12). 2103-2116 p.

Apéndices

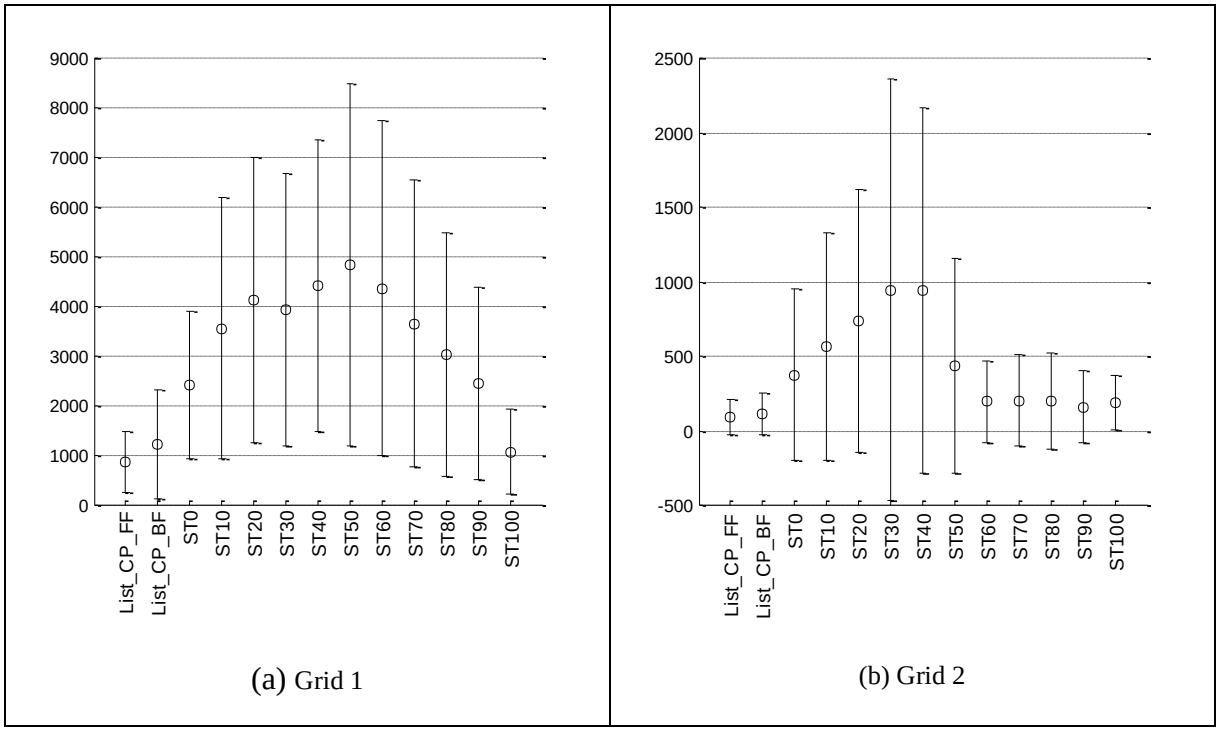
A. Resultados Completos de Estrategias de un Solo Nivel



(a) Grid 1

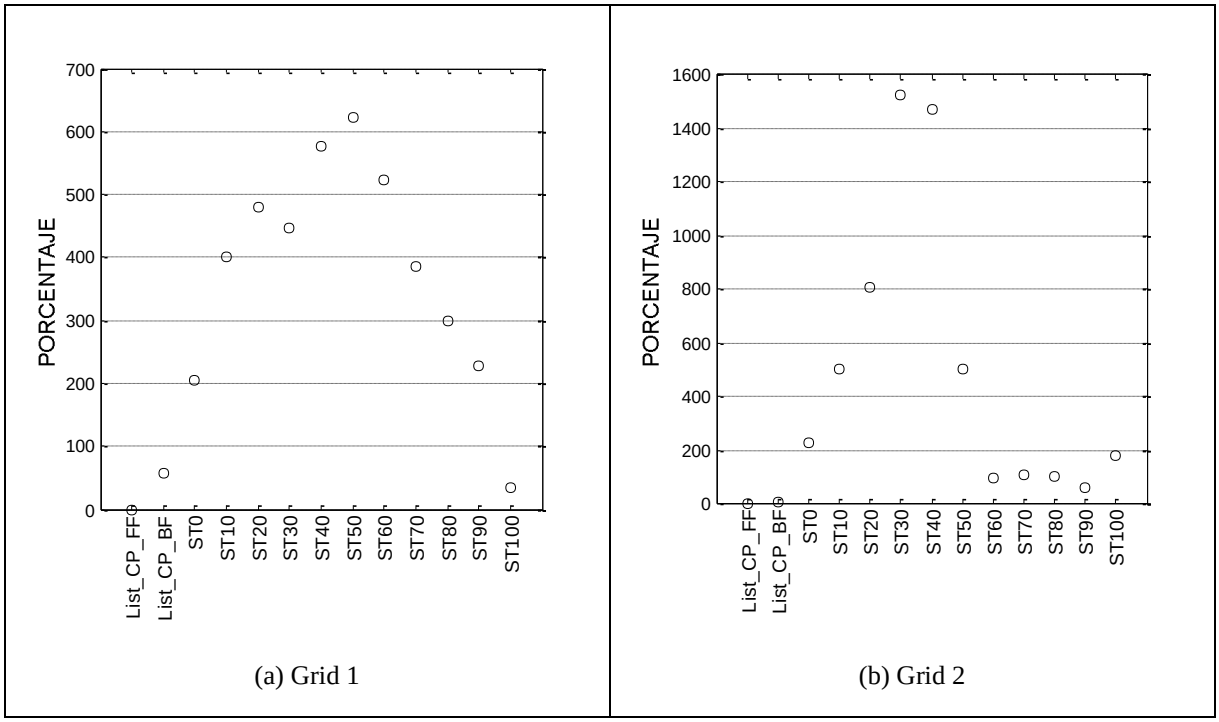
(b) Gird 2

Figura 35. Desaceleración acotada promedio



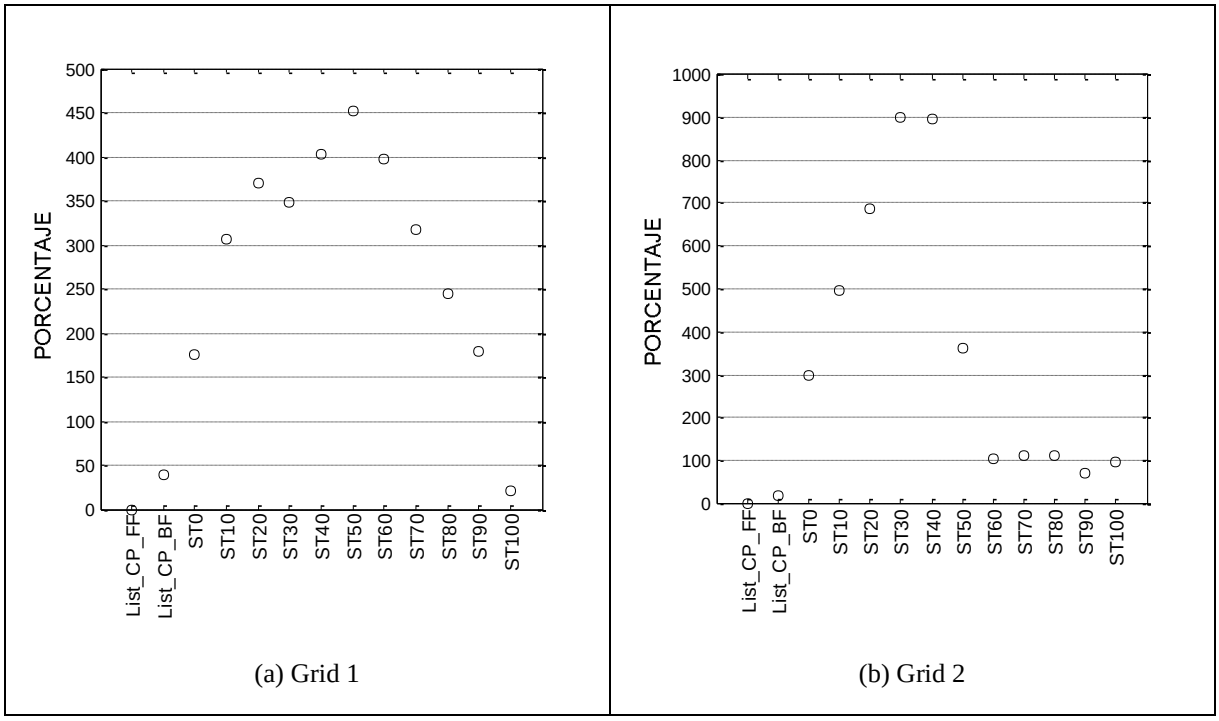
(a) Grid 1 (b) Grid 2

Figura 36. Tiempo de espera promedio



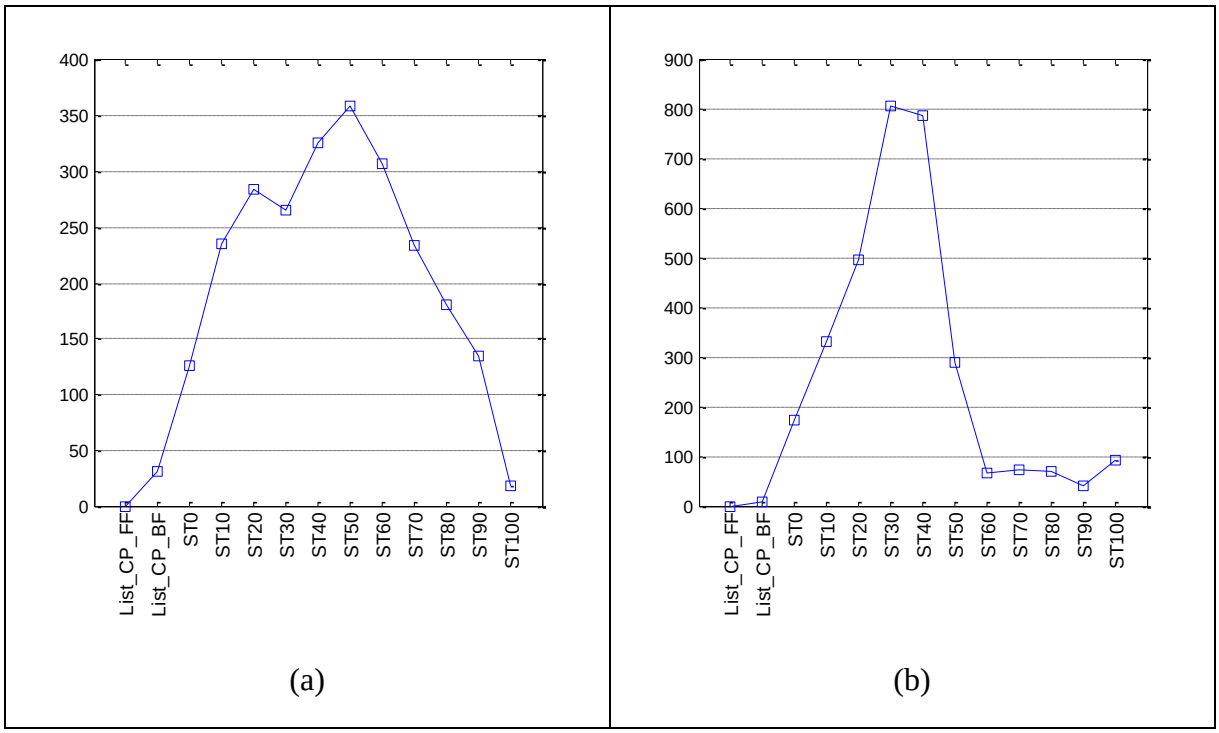
(a) Grid 1 (b) Grid 2

Figura 37. Degradación en desaceleración acotada promedio



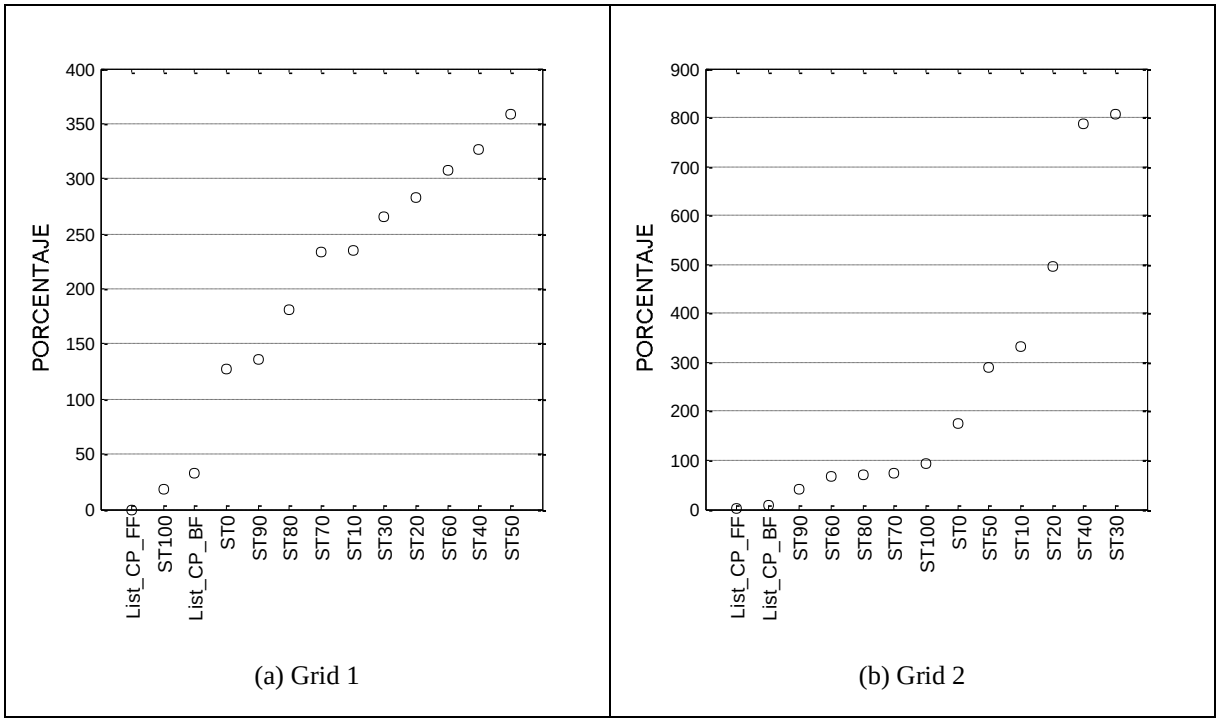
(a) Grid 1 (b) Grid 2

Figura 38. Degradación de tiempo de espera promedio



(a) (b)

Figura 39. Promedio de tres métricas



(a) Grid 1

(b) Grid 2

Figura 40. Ranking de promedio de tres métricas

B. Resultados Completos de Estrategias de Dos Niveles

Tabla 10. Porcentajes de degradación de desempeño para Grid 1, Grid 2 y Grid 3

Estrategia / Métricas		MCT	MLB	LBal_S	MLp	MPL	Random	MST	MWWT_S_a	MWT
ρ	Grid1	0	0	0	2	0	21	0	7	0
	Grid2	0	0	0	0	0	2	0	0	0
	Grid3	2	22	19	87	17	83	0	45	26
SD	Grid1	1501	469	20	93	0	53741	100	17658	236
	Grid2	5734	1031	10	62	0	17688	110	17609	999
	Grid3	1	33	13	63	19	41	0	54	43
SD_b	Grid1	1284	408	18	71	0	41924	91	12576	204
	Grid2	4001	754	17	53	0	12196	111	7550	628
	Grid3	1	33	10	65	18	46	0	50	42
Th	Grid1	0	0	0	2	0	18	0	7	0
	Grid2	0	0	0	0	0	2	0	0	0
	Grid3	1	18	17	47	14	44	0	27	19
TA	Grid1	47	17	0	4	0	1923	3	641	9
	Grid2	93	34	1	3	0	345	5	175	18
	Grid3	1	32	9	65	15	49	0	46	40
U	Grid1	0	0	0	2	0	18	0	7	0
	Grid2	0	0	0	0	0	2	0	0	0
	Grid3	2	18	16	46	15	46	0	30	20
t_w	Grid1	1256	463	11	119	0	51929	80	17303	252
	Grid2	4833	1799	70	153	0	18027	252	9132	933
	Grid3	1	32	9	65	16	49	0	47	40
WTA	Grid1	100	49	0	35	2	2513	0	909	35
	Grid2	135	74	2	19	0	457	7	271	55
	Grid3	2	28	8	112	16	111	0	13	34
WTA_w	Grid1	23	5	0	6	0	815	1	292	8
	Grid2	20	7	1	4	0	91	2	65	11
	Grid3	2	24	10	102	16	110	0	11	30
WWT_s	Grid1	378	187	0	130	6	9483	1	3429	132
	Grid2	700	384	10	101	0	2378	39	1411	287
	Grid3	2	28	8	112	16	112	0	13	34

Tabla 11. Grid 3. Porcentaje de mejora de desempeño por métrica y el factor de admisibilidad correspondiente

Estrategia / Métricas	MCT_a	MLB_a	$LBal\_S_a$	MLp_a	MPL_a	$Random_a$	MST_a	$MWWT\_S_a$	MWT_a
ρ	1.25	2.51	0.37	24.79	1.54	22.68	0.78	2.54	3.06
	0.5	0.5	0.6	0.5	0.5	0.5	0.6	0.6	0.5
SD	1.06	0.37	4.08	65.75	5.68	5.44	1.78	6.18	12.80
	0.5	0.6	0.5	0.5	0.5	0.5	0.5	0.2	0.2
SD_b	0.96	0.80	3.12	62.69	5.20	7.76	1.78	4.49	11.64
	0.5	0.2	0.5	0.5	0.5	0.5	0.5	0.2	0.2
Th	0.51	3.14	0.00	34.68	0.0	25.62	1.57	3.35	1.19
	0.5	0.5	0.6	0.5	-	0.5	0.6	0.5	0.5
TA	1.06	1.11	2.75	57.83	4.79	10.22	1.80	0.67	8.28
	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.2	0.2
U	1.12	2.55	0.32	31.25	1.64	29.43	0.67	2.44	3.01
	0.5	0.5	0.6	0.5	0.5	0.5	0.6	0.6	0.5
t_w	1.06	1.11	2.75	57.88	4.80	10.24	1.80	0.67	8.29
	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.2	0.2
WTA	1.41	4.13	2.46	36.06	4.78	29.81	0.67	0.81	7.09
	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
WTA_w	1.83	4.22	1.91	29.87	4.31	30.58	0.76	0.53	7.13
	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.6	0.5
WWT_s	1.41	4.13	2.46	36.09	4.79	29.84	0.67	0.81	7.10
	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Tabla 12. Grid 3. Porcentaje de degradación de desempeño de estrategias con a=1 y a=0.5

Estrategia / Métricas	MCTa	MCT	MLB_a	MLB	$LBal\_S_a$	LBal_S	MLp_a	MLp	MPL_a	MPL	$Random_a$	Random	MST_a	MST	$MWWT\_S_a$	MWWT_S	MWT_a	MWT
ρ	1	2	20	23	20	19	41	87	15	17	41	83	0	0	43	45	22	26
SD	79	81	139	139	94	102	0	192	102	114	140	154	76	80	174	176	137	156
SD_b	62	63	114	115	73	79	0	168	81	91	118	136	59	62	142	143	112	130
Th	1	2	16	19	19	17	29	47	17	14	30	45	0	1	25	28	19	19
TA	43	45	87	89	52	56	0	137	58	66	93	115	41	44	110	111	85	101
U	1	2	16	18	16	16	29	46	13	15	30	46	0	0	29	30	18	20
t_w	44	45	87	90	52	57	0	137	58	66	93	115	41	44	110	111	85	102
WTA	1	3	24	29	6	9	36	113	11	17	49	113	0	1	12	13	25	35
WTA_w	1	3	19	25	8	11	43	104	12	17	47	112	0	1	11	12	22	31
WWT_s	1	3	24	29	6	9	37	114	11	17	49	113	0	1	12	13	25	35
average	23.4	24.9	54.6	57.6	34.6	37.5	21.5	114.5	37.8	43.4	69	103.2	21.7	23.4	66.8	68.2	55	65.5

Tabla 13. Grid 3. Porcentajes de degradación de desempeño de estrategias con a=0.5

Estrategia / Métricas	MCT_a	MLB_a	$LBal\_S_a$	MLp_a	MPL_a	$Random_a$	MST_a	$MWWT\_S_a$	MWT_a
ρ	1	20	20	41	15	41	0	43	22
SD	79	139	94	0	102	140	76	174	137
SD_b	62	114	73	0	81	118	59	142	112
Th	1	16	19	29	17	30	0	25	19
TA	43	87	52	0	58	93	41	110	85
U	1	16	16	29	13	30	0	29	18
t_w	44	87	52	0	58	93	41	110	85
WTA	1	24	6	36	11	49	0	12	25
WTA_w	1	19	8	43	12	47	0	11	22
WWT_s	1	24	6	37	11	49	0	12	25
average	23.4	54.6	34.6	21.5	37.8	69	21.7	43	55

C. Ejecución de Experimentos en GSimula

El simulador GSimula es un simulador de eventos discretos desarrollado en CICESE para llevar a cabo experimentos de calendarización en Grid. Tiene como entrada 4 archivos. Un archivo es la carga de trabajo en el formato SWF. Otro archivo de configuración de experimentos en donde se especifica por ejemplo, el número de experimentos a realizar y qué estrategias se van a utilizar. Otro archivo establece la forma en que será definido el tiempo de estimación de ejecución de cada tarea. Y por último, un archivo define el número de sitios y la cantidad de procesadores de cada uno.

Para los experimentos con modelo jerárquico ya existía una estructura en GSimula, para llevar a cabo los experimentos. Se utilizó la estructura existente para ejecución de tareas en modelo jerárquico de dos niveles para simular la ejecución de tareas en un modelo descentralizado de un solo nivel.

El GSimula lleva a cabo dos calendarios. Un calendario a nivel de asignación se realiza utilizando la información de las tareas de llegada y el tiempo de estimación de ejecución de la tarea. Otro calendario se realiza en nivel de ejecución utilizando el tiempo de ejecución real de cada tarea. Sin embargo, para las estrategias de un solo nivel no existen dos calendarios, ya que no existe el nivel de asignación, todo se realiza a nivel de ejecución.

El simulador registra dos tipos de eventos en los cuales realiza la calendarización de tareas. El primer tipo de evento es llegada o sometimiento de tareas y el segundo tipo es finalización de tareas. El primer tipo de evento, implica que al menos una tarea llega al sistema y está disponible para ser ejecutada. En el modelo jerárquico, una estrategia de asignación de tareas define en qué sitio se hará la ejecución de la tarea. Entonces la tarea es tomada en cuenta por el calendarizador local de dicho sitio y define el momento en que se ejecutará dicha tarea, dependiendo del número de tareas en espera y los recursos disponibles en el sitio. El evento de finalización de tareas se utiliza para liberar los recursos del sitio donde haya terminado su ejecución y entonces posiblemente iniciar la ejecución de una o más tareas en el mismo sitio y que se encuentran en la cola de espera del sitio.

Las estrategias de un solo nivel carecen de estrategia de asignación de tareas. La estrategia de un solo nivel es en sí el algoritmo de calendarización local y dispone de la posibilidad de

interactuar con los demás sitios del Grid. El simulador asigna un sitio a cada tarea según van siendo sometidas. Para simular un solo nivel, lo que se implementó fue que para cada evento de llegada de tarea, se asignaba una tarea a un sitio, sin que necesariamente fuera la tarea que había disparado el mecanismo de asignación de tareas. En cambio, se verifican todas las tareas disponibles y el estado de los sitios, respecto al número de procesadores disponibles. Si hay suficientes procesadores como para ejecutar una tarea en alguno de los sitios siguiendo la estrategia definida de un solo nivel, entonces se reportaba al nivel de asignación cuál tarea se "asignaba" a cuál sitio. Si no había suficientes procesadores disponibles para definir en qué sitio se ejecutaría una tarea, se ejecutaba el calendario hasta el siguiente evento, ya fuera la llegada de una nueva tarea o la finalización de alguna tarea en el calendario, utilizando siempre los tiempos reales de ejecución. Si alguna de las tareas recién llegadas se podían calendarizar, se definía cuál y qué sitio. O bien, si al finalizar una de las tareas se liberaban suficientes recursos como para calendarizar una de las tareas, se reportaba al nivel de asignación cuál tarea sería ejecutada en qué sitio. Hay que observar que el hecho de finalizar tareas significa utilizar información de lo que sucederá en el tiempo posterior al que actualmente en el nivel de asignación se tiene. Sin embargo, esto no afecta, ya que el reloj que va disparando los eventos de llegada de tareas y ejecución de las mismas, no es modificado. De tal forma que las estrategias de un solo nivel solo definen qué tareas se ejecutan en qué sitio, como una estrategia de asignación de dos niveles, pero la diferencia es que siempre se utiliza el tiempo de ejecución real de las tareas y que se van definiendo sitios de ejecución de tareas que pueden ser diferentes a las tareas que disparan el evento de llegada de tarea. En el modelo de dos niveles, la tarea que llega al sistema es siempre la que es asignada.

Ya que las estrategias de un solo nivel definen en qué sitio se ejecutarán cada una de las tareas, fue posible utilizar la estructura ya existente de GSimula para obtener los resultados respecto a las métricas de evaluación. Lo que fue necesario agregar fue la generación de una bitácora de ejecución de tareas. Esto permitió hacer análisis respecto al número de tareas y cantidad de trabajo, sobre el sitio de sometimiento original de una tarea y el sitio de la ejecución de la tarea. También, hacer los cálculos de migración de tareas.

A continuación se lista el archivo CONFIG.INI y se explican las opciones en los parámetros para la ejecución de un experimento.

Los parámetros están definidos por el carácter "&". En la siguiente línea se especifica el valor de dicho parámetro. Si una línea inicia el el carácter ";", se trata de un comentario y será ignorado por el configurador de GSimula al leer el archivo.

Parámetro	Significado
&number of experiments	El número de experimentos a realizarse: 30.
&first experiment	Número de experiment a partir del cual se inicia la simulación. Es decir, es posible dejar pasar una cantidad de tareas que representen un determinado número de experimentos antes de iniciar con la simulación
&period of time	Define si el parámetro &number of jobs or days sera un número de días o si será un número fijo de tareas
&number of jobs or days	Define si el número de tareas o número de días por experimento
&period alignment unit	Define el tipo de período utilizado para los experimentos, generalmente semanas.
&jobs not considered	Define el porcentaje de tareas que no serán considerados en las métricas al realizar una simulación.
&scheduling type	Define si la calendarización será en línea o fuera de línea.
&system runtime prediction	Establece si el tiempo de estimación será el proporcionado por el usuario o si será alguna de las opciones elegidas en el archivo de configuración de estimación de tiempo de ejecución.
&first job selection	Define la forma en que se tomarán las tareas que llegan en un determinado tiempo. Es un rango y este parámetro define el inicio del rango. Generalmente FIFO.
&last job selection	Define cuál será la última estrategia del rango para seleccionar la tarea.
&total selection strategies	Define el número de estrategias de asignación disponibles
&selection strategies list	Define la lista de estrategias que se utilizarán para los experimentos
&selection scheduler	Define el tipo de selección de tarea que se utilizará en la cola de espera
&admissible flag	Define si se utilizará admisibilidad

&first admissible selection	Define el inicio del rango a partir del cual se generarán los pasos de admisibilidad
&last admissible selection	Define el final del rango hasta donde se generarán los pasos de admisibilidad
&step admissible selection	Define la forma la cantidad del paso en que se incrementará el rango de admisibilidad
&first admissible strategy	Define la primer estrategia de admisibilidad que se utilizará dentro del rango de opciones disponibles
&last admissible strategy	Define la última estrategia de admisibilidad
&first local scheduling	Define el primer tipo de calendarización local que se utilizará dentro del rango de opciones diponibles
&last local scheduling	Define el ultimo tipo de calendarización local que se utilizará
&number of metrics	Define cuántas métricas están disponibles para generarse durante la simulación
&output metrics type	Define si las métricas serán solo de nivel de asignación, de nivel de ejecución o de ambos.
&scope of metrics	Define si los archives de resultados de las métricas serán por sitio, Grid o ambos
&output metrics list	Define la lista de métricas que se generarán en los archivos de resultados
&workload source	Define si la carga de trabajo se origina en un modelo, carga de un sitio, proviene de una mezcla de sitios, es aleatoria o simplemente un ejemplo.
&workload name	Define el nombre de la carga de trabajo
&grid number	Define el número de Grid que será utilizado desde el archivo de configuración de Grid grid_config.ini
&step stealing	Define el umbral en porcentaje entre las colas A y B
&grid scheduling levels	Define si habrá un nivel con súperclúster
&SC resource selection	Define cuál estrategia se utilizará para cada superclúster
&number of super-clusters	Define el número de superclústeres en el Grid
&nodes in super-clusters	Define el número de nodos en los superclústeres
&path local directory	Define la ruta para almacenar los archives de

&interactive	resultados
&save workload file	Indica si se guarda un log con la carga de trabajo de tareas que sí fueron usadas para la simulación
&save events log &save queue size log &save jobs per node &save executed jobs runtimes &save executed jobs history &save jobs size histogram &save work by jobs size	Dan opciones a generación de archives para el análisis de la ejecución del calendario