TESIS DEFENDIDA POR

Juan Manuel Ramírez Alcaraz

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Andrey Chernykh

Director del Comité

Dr. José Alberto Fernández Zepeda

Miembro del Comité

Dr. Carlos Alberto Brizuela Rodríguez

Miembro del Comité

Dr. Ramin Yahyapour

Miembro del Comité

Dr. Hugo Homero Hidalgo Silva

Coordinador del programa de posgrado en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales

Director de Estudios de Posgrado

31 de agosto de 2011.

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

Estrategias de Asignación de Trabajos con Tiempos de Ejecución Estimados por el Usuario para Calendarización en Línea en Grids Jerárquicos

TESIS

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

DOCTOR EN CIENCIAS

Presenta:

JUAN MANUEL RAMÍREZ ALCARAZ

Ensenada, Baja California, México, agosto de 2011.

RESUMEN de la tesis de Juan Manuel Ramírez Alcaraz, presentada como requisito parcial para la obtención del grado de DOCTOR EN CIENCIAS en Ciencias de la Computación. Ensenada, Baja California. Agosto de 2011.

Estrategias de Asignación de Trabajos con Tiempos de Ejecución Estimados por el Usuario para Calendarización en Línea en Grids Jerárquicos

Resumen aprobado por:

_____

Dr. Andrey Chernykh

Director de Tesis

En esta tesis se aborda la calendarización en línea, sin interrupciones y no clarividente, de trabajos paralelos en un Grid. Se considera un modelo de calendarización en Grid con dos niveles. En la primera etapa los trabajos se asignan a un sitio adecuado, elegido con base en la aplicación de alguna estrategia de asignación, mientras que en la segunda etapa, se aplica una calendarización local en cada sitio de manera independiente. Se clasifican las estrategias de asignación dependiendo del tipo y cantidad de información que ellas requieren. Se lleva a cabo un estudio amplio de evaluación de rendimiento usando simulación y se demuestra que las estrategias consideradas se desempeñan bien con respecto a diferentes métricas que reflejan tanto los objetivos centrados en el usuario como los centrados en el sistema. Desafortunadamente, los tiempos de ejecución estimados por los usuarios y la información sobre los calendarios locales no ayudan a mejorar significativamente el resultado final de las estrategias de calendarización. Al examinar el rendimiento global del Grid basado en datos reales, se determinó que una distribución apropiada de los trabajos con base en sus requerimientos de procesador sobre el Grid, tiene un rendimiento más alto que una asignación de trabajos basada en la estimación de los tiempos de ejecución hechas por el usuario o de la información sobre calendarios locales. En general, los experimentos mostraron que calendarizadores bastante simples con requerimientos de información mínima pueden proveer un buen desempeño.

Palabras Clave: Computación en Grid, Calendarización, Estrategias de Asignación, Administración de Recursos.

ABSTRACT of the thesis presented by **Juan Manuel Ramírez Alcaraz** as a partial requirement to obtain the DOCTOR OF SCIENCE degree in Computer Science. Ensenada, Baja California, México. Agosto de 2011.

Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids

We address non-preemptive, non-clairvoyant, online scheduling of parallel jobs on a Grid. We consider a Grid scheduling model with two stages. At the first stage, jobs are allocated to a suitable Grid site, chosen based on some allocation strategy, while at the second stage, local scheduling is independently applied to each site. We analyze allocation strategies depending on the type and amount of information they require. We conduct a comprehensive performance evaluation study using simulation and demonstrate that our strategies perform well with respect to several metrics that reflect both user- and system-centric goals. Unfortunately, user run time estimates and information on local schedules does not help to significantly improve the outcome of the allocation strategies. When examining the overall Grid performance based on real data, we determined that an appropriate distribution of job processor requirements over the Grid has a higher performance than an allocation of jobs based on user run time estimates and information on local schedules. In general, our experiments showed that rather simple schedulers with minimal information requirements can provide a good performance.

Keywords: Grid Computing, Scheduling, Allocation Strategies, Resource Management.

Dedicatorias

A mi esposa *Azucena* y mis hijos *Arely*, *Juan Victor* y *AnaCaro* por su total comprensión y apoyo; y por haber soportado todas mis ausencias.

A mi madre, y a mi padre que en paz descanse.

Agradecimientos

Un proyecto de esta naturaleza no se logra trabajando solo, existen muchas personas e instituciones que me apoyaron para sacar adelante este trabajo. Es difícil listar aquí a todos, espero que me disculpen quienes aquí no aparezcan, pero saben que les agradezco profundamente su apoyo.

A La familia de mi esposa, que también es mi familia, por su apoyo incondicional en todo momento, sin ellos, todo abría sido más difícil.

A mis hermanos y sus familias, por estar ahí siempre que los he necesitado. Especialmente por cuidar de mi madre.

A mi asesor, Andrey Chernykh por su paciencia y comprensión durante todos estos años y por supuesto, por ser mi principal guía en esta travesía.

A los miembros de mi comité, Dr. Fernández, Dr. Brizuela y Dr. Yahyapour por sus valiosas aportaciones a esta investigación.

Al CICESE y a todo su personal, especialmente a Dolores Sarracino por su compresión y apoyo; a Gilberto López y Jesús Favela por regalarme su amistad.

A la Universidad de Colima, especialmente a las autoridades que me otorgaron todas las facilidades para poder terminar este trabajo.

Al CONACYT por el importante apoyo económico que me otorgaron (No. de registro de beca: 196406) y a Ivonne Best por la eficiente gestión de dicha beca.

A todos los compañeros que conocí durante estos años, especialmente a David, Oscar, José Luis, Ariel, Yair, Ricardo, Adán, Pablo, Daniel e Ismael, por su ayuda decisiva en momentos importantes del trayecto y por los momentos agradables que pasamos juntos.

A mis amigos y colegas: Mary, Erika, Victor y Pedro por contarme entre sus amigos y por alentarme siempre a terminar.

A todos los que en mayor o en menor medida participaron en el desarrollo del simulador *GSimula*, plataforma base para llevar a cabo los experimentos.

A todos, gracias por haberme complementado el conocimiento científico con lecciones de vida. Muchas Gracias.

CONTENIDO

Página

CONTENIDO (continuación)

LISTA DE FIGURAS

LISTA DE TABLAS

Capítulo I

Introducción

En este capítulo se presentan los conceptos más relevantes del Grid, iniciando con la definición del concepto principal: Computación en Grid (Grid Computing). También se describe el problema básico que sirve como fundamento del presente trabajo y se plantean los objetivos a lograr.

I.1 Orígenes de la *Computación en Grid*

El término *Computación en Grid* se acuñó a mitad de los 90s para denotar una propuesta de infraestructura de cómputo que permitiera utilizar recursos computacionales geográficamente distribuidos como si fueran uno solo. Esta propuesta surgió principalmente para satisfacer las necesidades surgidas en el ámbito científico cuando se abordan *problemas de gran reto*, es decir, problemas cuya demanda computacional es tan alta que la infraestructura de cómputo instalada, incluso supercomputadoras, no es suficiente (Berman y Hey, 2004; Foster y Kesselman, 2003, 1999). Dentro del tipo de problemas de gran reto se encuentran los abordados en la física de alta energía, meteorología, astronomía, biología, medicina, genética, farmacología y economía (CERN, 2008).

La idea de la Computación en Grid es análoga a la red de energía eléctrica, donde los generadores de energía están distribuidos y los usuarios son capaces de acceder a la energía eléctrica sin preocuparse acerca de la fuente de energía, del esquema de canalización, almacenamiento y de los mecanismos de regulación (Buyya, 2005). Así pues, los participantes en una comunidad de Grid podrán hacer uso de grandes recursos mediante el uso de interfaces relativamente simples, sin

tener conocimiento de las implicaciones tecnológicas que implica proporcionar un servicio. Los recursos que se pueden compartir en un contexto distribuido no se limitan a unidades de procesamiento; elementos de almacenamiento, datos, dispositivos especializados de entrada y salida, también son recursos que se pueden disponer. Cómo integrar y administrar de manera óptima un conjunto de recursos distribuidos, son temas de estudio de múltiples investigaciones.

Se pueden distinguir tres tipos de tendencias en el desarrollo de Grids, *centrados en cómputo, centrados en datos,* y *centrados en comunidad* (CERN, 2008). Los primeros son del dominio del cómputo de alto rendimiento, donde el usuario requiere alta capacidad de procesamiento; la segunda tendencia concierne a problemas que manejan grandes cantidades de datos (del orden de los Petabytes (PB)); y la tercera tendencia, también llamada *aplicaciones colaborativas*, intentan reunir comunidades o usuarios para colaboración de varios tipos sobre medios electrónicos. En esta última clasificación se encuentran principalmente los proyectos @home (California, 2009a), los cuales se pueden considerar como precursores del Grid, por ejemplo SETI@home (California, 2009b), Rosetta@home (Washington, 2009) y Climateprediction (Oxford, 2009). Los Grids se crearon principalmente para sufragar las dos primeras tendencias, y con base en esto, se pueden diferenciar los Grids Computacionales (C-Grid) y los Grids de Datos (D-Grid).

Un Grid es una colección distribuida de recursos de cómputo y almacenamiento, mantenidos para servir las necesidades de alguna organización virtual (VO) (Foster y Kesselman, 1999; Foster *et al.*, 2001; Ranganathan y Foster, 2003a). Una organización virtual se define como un conjunto de individuos y/o instituciones afines en sus actividades y que han establecido reglas para compartir sus recursos entre ellos (Foster *et al*., 2001).

Aún cuando el nacimiento de Grids se originó por las necesidades existentes en el ámbito de la investigación científica, éstos están siendo rápidamente adoptados por grandes organizaciones y empresas que buscan aprovechar las ventajas que este tipo de plataforma les ofrece para compartir sus propios recursos geográficamente distribuidos.

I.2 Planteamiento del problema

Dentro de los problemas principales que surgen en la implementación de un Grid se encuentra el de la *administración de recursos* ya que de esto depende el rendimiento del Grid. Por esta razón, es necesario contar con buenas estrategias para planificar y distribuir trabajos a los diferentes recursos que lo conforman, puesto que cuanto mejor sea el tiempo para la selección y mapeo de cada trabajo, mejor será el desempeño del sistema (ej. mejor utilización) o mayor el beneficio para el usuario (ej. menor tiempo de espera) (Ranganathan y Foster, 2003a; Shnizler *et al.*, 2004).

En general, el término *administración de recursos* se utiliza comúnmente para describir todos los aspectos del proceso de localización de diversos recursos o servicios, disponerlos para su uso, utilizarlos y monitorizar su estado (Nabrzyski *et al.*, 2003). Una de las actividades esenciales de la *administración de recursos* es la *calendarización*, la cual refiere la asignación de recursos limitados a actividades, con el objetivo de optimizar una o más medidas de rendimiento. En otras palabras, la *calendarización* es una planificación de cuándo y dónde se ejecutarán los programas de usuario de tal manera que se optimice algún criterio común (ej. utilización, tiempo de espera). La *calendarización* ha sido estudiada intensamente en las últimas seis décadas, en diferentes áreas como son la administración, la ingeniería industrial, y ciencias de la computación (Leung, 2004). En el área de computación, la *calendarización* se enfocó primeramente en sistemas de tiempo

compartido, y posteriormente, con la aparición de arquitecturas paralelas (supercomputadoras, clusters), a sistemas de espacio compartido.

Sin embargo, la calendarización de sistemas tradicionales no se puede aplicar directamente a los Grids computacionales debido a las diferencias estructurales y administrativas que existen entre ellos (Schopf, 2003). Las características básicas de los Grids computacionales y que los hacen diferentes de los sistemas tradicionales son las siguientes:

- *Diferentes dominios administrativos*. Los recursos del Grid generalmente pertenecen a diferentes instituciones.

- *Políticas locales*. Cada dominio administrativo establece sus propias políticas para el manejo de los recursos locales.

- *Heterogeneidad*. Los recursos participantes en el Grid son de muchos tipos y capacidades diferentes.

- *Recursos dinámicos*. Los recursos participantes en el Grid pueden conectarse o desconectarse del Grid sin aviso previo.

- *Gran escala*. El Grid, en su concepción básica es un sistema mucho más grande en capacidad y número de usuarios que los sistemas tradicionales.

- *Arquitectura multinivel*. En el Grid se debe considerar al menos un nivel más que en los sistemas tradicionales debido a que es necesario contar con un elemento que permita la distribución de trabajos a los diferentes recursos que participan en el Grid, sin embargo pueden existir múltiples niveles.

Tales restricciones sirven como preámbulo para el diseño de una infraestructura de calendarización Grid con múltiples niveles de calendarización, que permita

contar con un sistema automático de selección del mejor recurso que cumpla con los requerimientos de usuario y a los propietarios seguir manteniendo el control local de sus recursos.

I.3 Justificación

A pesar de que los Grids son una realidad, aún existen muchos problemas por resolver, incluyendo cuestiones de calendarización con múltiples niveles, flujos de trabajo, consumo eficiente de energía, seguridad al compartir recursos, escalabilidad, réplica de datos, ejecución simultánea en múltiples recursos (co-asignación), etc. Dentro de todas las vertientes implicadas en el Grid, la calendarización es uno de los puntos clave, debido a que ésta es una actividad principal de la cual depende el uso eficiente de los recursos que conforma el Grid y la satisfacción de los usuarios. Se puede considerar la calendarización en Grid como un proceso de dos etapas, en la primera se asigna cada trabajo al recurso más adecuado y en la segunda se aplica algún algoritmo de calendarización paralela en cada uno de los recursos. De esta manera y debido también al creciente tamaño y dinamicidad de los Grids, el diseño e implementación de estrategias de asignación de trabajos computacionales a los recursos disponibles en el Grid así como de algoritmos de calendarización local, se convierten en una necesidad constante, en la búsqueda de la optimización de los recursos tanto de propietarios como de usuarios.

Existen diversas investigaciones de calendarización en Grid que analizan estrategias para asignación de trabajos a recursos (Abraham *et al.*, 2000; Abramson *et al.*, 2000; Berten *et al.*, 2006; Buyya, 2002; Buyya *et al.*, 2005; Casanova *et al.*, 2000; Ernemann *et al.*, 2004; D. Feitelson *et al.*, 2005a; Hamscher *et al.*, 2000; Huedo *et al.*, 2003). En general podemos identificar dos categorías de estrategias de asignación de trabajos en Grid: basadas en un modelo económico y

basadas en rendimiento. La primera categoría busca reproducir el modelo económico de productores y consumidores de productos, en donde los productores son los propietarios de los recursos (procesadores, almacenamiento) y los consumidores los usuarios. Así, los usuarios podrán elegir el recurso que les ofrezca el mejor trato entre tiempo de procesamiento y costo computacional dependiendo de sus requerimientos de QoS (Buyya, 2002). Las estrategias basadas en rendimiento seleccionan los mejores recursos para la ejecución de los trabajos de usuario con base en alguna métrica de rendimiento, como puede ser: la carga de trabajo, tiempo promedio de espera, número de tareas procesadas por unidad de tiempo, porcentaje de utilización del recurso, etc.

Por otro lado, los sistemas de calendarización en Grid se pueden clasificar, con base en su arquitectura, en tres categorías: *centralizados*, *distribuidos* y *jerárquicos* (ver Sección II.1). La arquitectura más adecuada para representar un Grid real es la jerárquica, que puede constar de varios niveles de calendarización, lo cual ocasiona la necesidad de contar con una combinación adecuada de algoritmos de calendarización que soporten tales estructuras multinivel en la búsqueda de un sistema eficiente de administración de recursos para los Grids actuales.

Uno de los tipos de algoritmos locales muy utilizado en la implementación real de Grids es el de *Rellenado* (*Backfilling*). Este tipo de algoritmos se basa en el conocimiento del tiempo de ejecución de los trabajos, pero dado que obviamente este dato no se conoce sino hasta que el trabajo termina su ejecución, se utiliza una estimación que comúnmente se le solicita al usuario al momento de enviar su trabajo. Este dato también sirve como tiempo límite del trabajo, es decir, si el trabajo continúa su ejecución más allá de su tiempo estimado, el sistema detiene el trabajo. Debido a que este dato es altamente impreciso (Chiang *et al.*, 2002; Dror. G. Feitelson, 2005a; Mu'alem y Feitelson, 2001; Tsafrir *et al.*, 2006), se ha demostrado que tiene un impacto importante en los algoritmos de calendarización

local (ver Sección III.2). Sin embargo, este dato no ha sido incorporado en estrategias de selección de recursos.

En el presente trabajo se llevó a cabo un estudio extenso de evaluación del rendimiento de un modelo de calendarización en línea para un Grid jerárquico de dos niveles. En el primer nivel se asigna un trabajo a una máquina paralela adecuada utilizando un criterio de selección dado. En el segundo nivel se asigna algún algoritmo de calendarización dependiente de la máquina a los trabajos recibidos. Las estrategias aplicadas en el primer nivel utilizan diferentes tipos y cantidad de información en la toma de decisión para la asignación de trabajos a los recursos adecuados. Una de las aportaciones de este trabajo es el uso de una predicción del tiempo de ejecución por parte del sistema con base en el historial de ejecución del usuario, como base para la toma de decisión de selección de recursos. Para la evaluación del rendimiento de tales estrategias se utilizó simulación y se consideraron métricas que reflejan los objetivos de usuarios y propietarios de los recursos que conforman el Grid.

I.4 Objetivos

I.4.1 Objetivo general

Analizar de manera experimental diversas estrategias de calendarización para un Grid Computacional jerárquico de dos niveles.

I.4.2 Objetivos específicos

- Definición formal del problema de calendarización en Grid jerárquico de dos niveles.
- Diseño e implementación de una plataforma de simulación para un Grid Computacional.

- Diseño e implementación de estrategias de calendarización para un Grid Computacional.
- Generación y análisis de una carga de trabajo adecuada.
- Evaluación del impacto que tiene la imprecisión de la estimación del tiempo de ejecución en la calendarización.
- Evaluación de diversos modelos de predicción del tiempo de ejecución.

I.4.3 Preguntas de Investigación

Para lograr los objetivos propuestos, buscaremos responder las siguientes preguntas:

1. ¿Cómo se caracterizan los objetivos y los criterios de calendarización para sistemas jerárquicos de gran escala en términos de su eficiencia?

2. ¿Cómo se caracterizan las actividades del Grid que realizan los gestionadores de recursos (*Brokers*) durante el manejo de recursos computacionales?

3. ¿Están o no alineadas, en términos de su eficiencia, las evaluaciones de los calendarios con los objetivos de calendarización en Grid?

4. ¿Qué factores influyen en la calidad de la calendarización en Grid?

5. ¿Qué información es necesaria y suficiente para una asignación eficiente de las tareas en sistemas Grid de dos niveles?

6. ¿De qué manera el uso de información adicional durante la asignación de los recursos cambia el rendimiento de las estrategias de asignación?

7. ¿Cómo afecta el aumento de la complejidad de los algoritmos de asignación a la calidad de los calendarios?

8. ¿Qué efectos tiene la imprecisión de la información proporcionada por los usuarios en la eficiencia de los calendarios y rendimiento del Grid?

I.5 Organización de la tesis

Esta tesis está organizada de la siguiente manera: en el Capítulo I se muestra una introducción a la computación en Grid, así como el planteamiento del problema abordado y los objetivos que se persiguen. Se continua en el Capítulo II con la descripción formal del modelo de Grid utilizado para las simulaciones junto con una descripción de las arquitecturas de calendarización. En el Capítulo III se menciona una serie de trabajos relevantes relacionados directamente con el tema de administración de recursos en Grid, específicamente de calendarización, algunos desde el punto de vista teórico y otros desde el punto de vista experimental. También se describe en este capítulo algunos modelos de predicción del tiempo de ejecución de los trabajos. El capítulo IV describe las estrategias y los algoritmos de calendarización simulados para esta investigación. Un aspecto muy relevante en la simulación de algoritmos de calendarización es sin duda, las cargas de trabajo utilizadas, en el Capítulo V se describen y analizan las cargas de trabajo creadas para la experimentación en nuestro modelo. Por último, se muestra un análisis detallado de los resultados de los experimentos en el Capítulo VI, seguido de las conclusiones finales.

Capítulo II

Descripción del modelo

En este capítulo se muestra la arquitectura del modelo utilizado en la presente investigación, describiendo los elementos principales que lo conforman. Posterior a esto, se definen con notación matemática los parámetros considerados en el modelo, así como las métricas implementadas para su evaluación.

II.1 Arquitecturas de calendarización

Las arquitecturas de calendarización se pueden clasificar en *centralizadas*, *distribuidas*, y *jerárquicas*, las cuales no necesariamente están asociadas con la infraestructura real que forma un Grid (Hamscher *et al.*, 2000; Krauter *et al.*, 2002; Shan *et al.*, 2003).

En la arquitectura centralizada (Figura 1) todas las máquinas paralelas las calendariza una instancia central, la cual recibe todos los trabajos de usuario y recolecta información sobre el estado de todas las máquinas que conforman el Grid. Como cualquier sistema centralizado, sus principales desventajas son la escalabilidad (el sistema no tiene la capacidad de extenderse sin afectar sustancialmente su eficiencia) la propensión a fallas (el sistema tiene alta vulnerabilidad debido a que el control recae en una sola instancia) y la limitación para el manejo de diferentes algoritmos de calendarización en cada sitio. Sin embargo, debido a que el calendarizador central cuenta con toda la información necesaria para asignar y ejecutar los trabajos, esta arquitectura es capaz de producir calendarios eficientes.

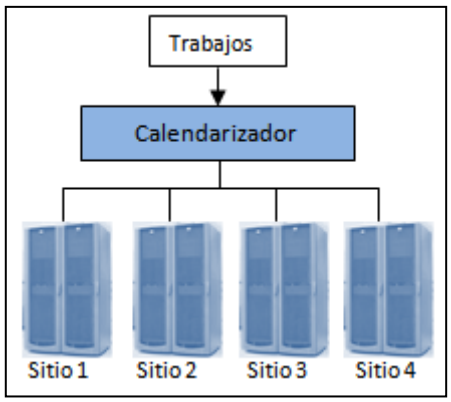


Figura 1. Arquitectura de calendarización centralizada

La arquitectura distribuida (Figura 2) considera un calendarizador por cada máquina del Grid interactuando entre ellos con el objetivo de decidir en cuál recurso se deben ejecutar los trabajos de los usuarios. Dado que no se cuenta con una instancia central, no hay cuello de botella y el sistema se vuelve más escalable y más tolerante a fallas que los sistemas centralizados, además se pueden implementar diferentes algoritmos de calendarización en los sitios locales dependiendo de las necesidades de cada uno. Por otro lado, esta arquitectura introduce una fuerte carga de comunicación entre los recursos para poder migrar trabajos entre uno y otro, además, dado que no se conoce toda la información de los trabajos y la disponibilidad de los sistemas, los calendarios pueden no ser eficientes. Aunado a esto, la ejecución de un trabajo en múltiples sitios es más difícil de lograr. Esto surge cuando un trabajo requiere más recursos de los disponibles en un sitio y se solicitan recursos de otro sitio, esto implica que el trabajo se dividirá en tantas partes como sitios requiera. Debido a que las diferentes partes del trabajo pueden necesitar comunicarse entre ellas, se requiere una sincronización o coordinación entre los calendarizadores de cada sitio para ejecutar al mismo tiempo todas las partes del trabajo. En esta arquitectura, los trabajos son enviados al recurso local y el calendarizador decide si se ejecutará en el recurso local o en uno remoto. En caso de que se tenga que ejecutar en un recurso remoto, se puede enviar el trabajo directamente a otro recurso o a una

cola global para que de ahí lo tome el recurso que primero esté disponible como en (Fölling *et al.*, 2010).

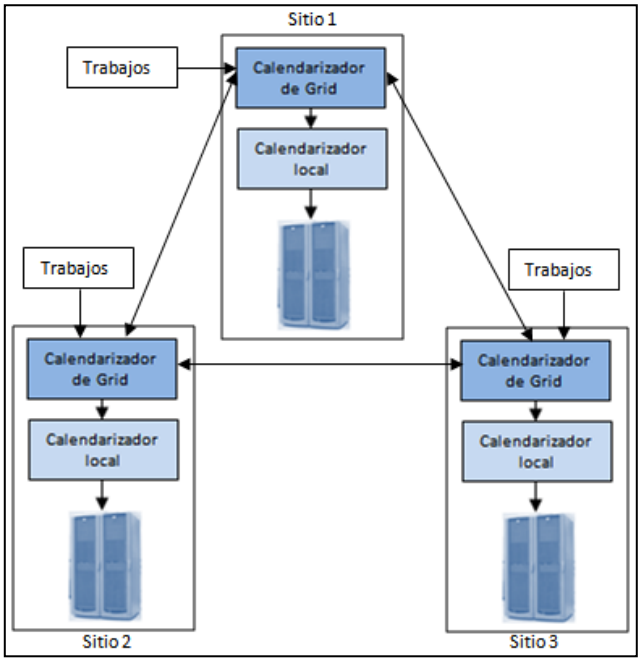


Figura 2. Arquitectura de calendarización distribuida

Sin embargo, es la arquitectura jerárquica (Figura 3) la que mejor describe la implementación real de los Grids actuales. En esta arquitectura, la administración de recursos está estructurada en múltiples niveles. Un caso común es la arquitectura jerárquica de dos niveles, en donde el nivel más alto corresponde al calendarizador Grid y típicamente tiene una vista general de los requerimientos de todos los trabajos enviados al Grid, pero no de los detalles específicos sobre el estado de los recursos locales. En el nivel bajo, el sistema local de administración de recursos (uno por cada sitio) conoce con detalle el estado del recurso y de los trabajos que ha recibido del nivel superior. En sistemas muy grandes pueden existir niveles adicionales, por ejemplo en el InterGrid propuesto por Dias de Assuncao *et al*. (2008) se considera un *InterGrid Gateway* (metacalendarizador en la Figura 3) que actúa como selector de Grid.

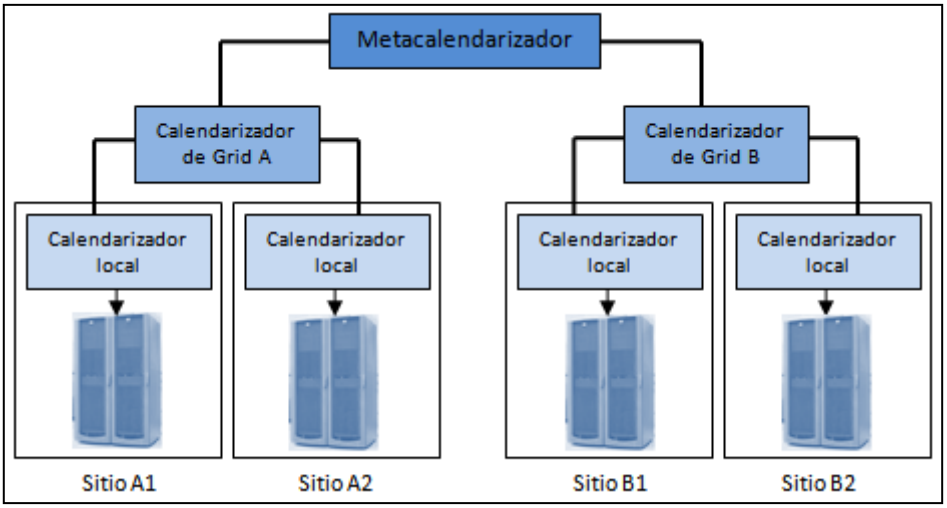


Figura 3. Arquitectura de calendarización jerárquica

La Figura 4 muestra el modelo jerárquico que se utilizó en la presente investigación. Se considera un Sistema Administrador de Recursos de Grid (GRMS) o *Broker* al cual se envian los trabajos del usuario. Todos los trabajos se reciben en una cola global donde los toma el *Calendarizador de Grid*, el cual funge como un selector de recursos. Una vez seleccionado el recurso, se envía el trabajo al Sistema Local de Administración de Recursos (*LRMS. Local Resource Management System*) de dicho recurso. El LRMS recibe los trabajos en una cola local, de la cual los tomará el calendarizador para su asignación a los procesadores que finalmente ejecutarán el trabajo. Los recursos locales se consideran como recipientes de ancho limitado por el número de procesadores y altura ilimitada correspondiente al tiempo en que está disponible el recurso. Se consideran trabajos paralelos y se representan como bloques de dos dimensiones cuya altura corresponde a su tiempo de ejecución y el ancho al número de procesadores asignados. Los problemas de este tipo se llaman comúnmente *problemas de empacamiento en una tira* (*Strip Packing*) (Lodi *et al.*, 2002; Lozano, 2005), y aplicado al Grid se consideran como *empacamiento en múltiples tiras* (*Multiple Strip Packing*) (Bougeret *et al.*, 2010b; Ye *et al.*, 2011). Las tiras

mencionadas se refieren a recipientes de un ancho específico y de una altura ilimitada.

Los trabajos paralelos se consideran *rígidos*, es decir, el trabajo no puede ejecutarse si no están disponibles los procesadores requeridos, los cuales se deben asignar exclusivamente a éste hasta finalizar su ejecución. Existen trabajos *moldeables* y *maleables*, los cuales pueden iniciar su ejecución con un número menor de procesadores que los requeridos y permanecer así hasta su finalización (moldeables) o incluso que durante su ejecución pueda cambiar el número de procesadores asignados por el sistema (maleables). Tales trabajos no se consideran en esta investigación.

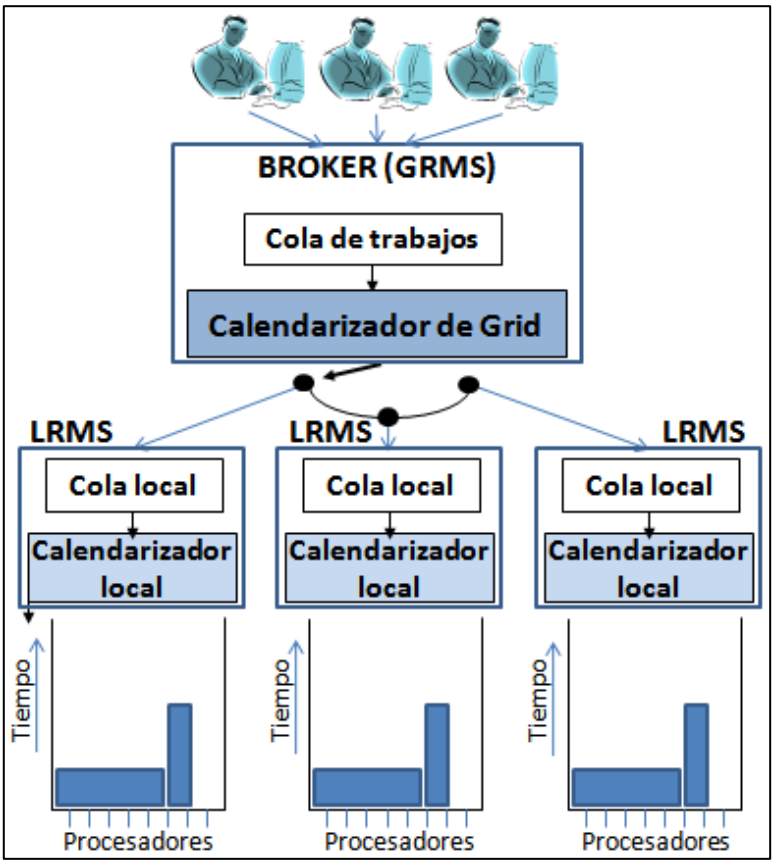


Figura 4. Modelo jerárquico utilizado

II.2 Definición formal

Se aborda un problema de calendarización en línea, en el cual se consideran n trabajos paralelos J_1, J_2, …, J_n que se deben calendarizar en m máquinas (o sitios) N_1, N_2, …, N_m. Sea m_i el número de procesadores o núcleos idénticos de la máquina N_i, se denota el número total de procesadores pertenecientes a las máquinas desde N_1 hasta N_m por $m_{1,m} = \sum_{i=1}^{m} m_i$. Se supone, sin pérdida de generalidad, que las máquinas están organizadas en orden no-decreciente respecto al número de procesadores, es decir, se cumple que $m_1 \leq m_2 \leq \cdots \leq m_m$.

Cada trabajo J_j se describe por medio de la tupla (r_j, $size_j$, p_j, p'_j, p''_j): su fecha de entrega $r_j \geq 0$, su tamaño $1 \leq size_j \leq m_m$ que significa el número de procesadores requeridos o grado de paralelismo, el tiempo de ejecución p_j, estimación del tiempo de ejecución por parte del usuario p'_j, y la predicción del tiempo de ejecución por el sistema p''_j. La fecha de entrega de un trabajo no está disponible antes de que el trabajo se envíe al sistema, y su tiempo de procesamiento no se conoce hasta que el trabajo ha finalizado su ejecución. Cuando no se conoce el tiempo de procesamiento de los trabajos sino hasta que terminan su ejecución, se dice que el sistema es *no clarividente*. El tiempo estimado p'_j provisto por el usuario para cada uno de sus trabajos se utiliza como cota de ejecución, después del cual el trabajo lo detiene el sistema si aún no ha finalizado su ejecución, esto previene el desperdicio de recursos debido a alguna falla en la ejecución de los procesos. La predicción generada por el sistema p''_j, se puede utlizar para generar mejores calendarios en algunos métodos de calendarización como Rellenado (Tsafrir *et al.*, 2007) aún si dicha predicción es imprecisa. Se Define t_w^j como el tiempo de espera del trabajo J_j desde su llegada

hasta el instante de inicio de su ejecución y s_j como el instante de inicio de su ejecución.

Se define también $w_j = p_j \cdot size_j$, $w_j' = p_j' \cdot size_j$, $w_j'' = p_j'' \cdot size_j$ como los recursos consumidos, estimación de recursos por consumir por parte del usuario, y la predicción de recursos por consumir por parte del sistema para el trabajo J_j, respectivamente. Cuando llega un trabajo al sistema, se asigna inmediata e irrevocablemente a una única máquina. Sin embargo, no se demandan procesadores específicos para que se asignen inmediatamente a un trabajo a su tiempo de llegada. Dicho de otra manera, la asignación del trabajo a los procesadores se puede postergar hasta que el número requerido de procesadores esté disponible.

Una máquina debe ejecutar un trabajo asignando de forma exclusiva y exacta el número de procesadores requerido, $size_j$, por un periodo de tiempo ininterrumpido p_j. No se considera la ejecución de un trabajo en múltiples sitios, es decir que el trabajo se divida en varias partes y cada una se ejecute en un sitio diferente (co-asignación), por lo tanto, el trabajo J_j puede solamente correr en la máquina N_i si se cumple que $size_j \leq m_i$. Se supone que los recursos involucrados son estables y dedicados al Grid, es decir, el número y tamaño de los recursos considerados son constantes, y no atienden peticiones locales.

Se utiliza $g_j = i$ para denotar que el trabajo J_j se asigna a la máquina N_i, y n_i para denotar el número de trabajos asignados a la máquina N_i. La carga actual de un sitio (número de trabajos en ejecución más número de trabajos en espera) en un tiempo dado se denota por l_i. El tiempo de finalización del trabajo J_j de la instancia (o conjunto de trabajos de entrada) I en un calendario S se denota por $c_j(S, I)$ y el tiempo de finalización del calendario (*makespan*) S y la instancia I es

$C_{max}(S,I) = max\{c_j(S,I)\}$. Se utiliza $C^*_{max}(I)$ para especificar el tiempo de finalización del calendario óptimo para la instancia I. Cuando es posible y no causa ambigüedad, se puede omitir I y S.

En la simulación de experimentos, se utilizaron 24 métricas (ver Tabla I) comúnmente empleadas para expresar los objetivos de los diferentes participantes del Grid como son los usuarios finales y los proveedores de recursos. Sin embargo, cuando se aplicó un análisis más detallado se restringieron a tres métricas de rendimiento bien conocidas: Tiempo promedio de espera (*waiting time*) $t_w = \frac{1}{n}\sum_{j=1}^{n}(c_j - p_j - r_j)$; promedio de ralentización acotado (*bounded slowdown*) (D. Feitelson *et al.*, 1997) $SD_b = \frac{1}{n}\sum_{j=1}^{n}\frac{c_j - r_j}{max\{10,p_j\}}$; y suma ponderada de tiempos de finalización (*total weighted completion time*) donde el peso corresponde a los recursos consumidos $SWCT_w = \sum_{j=1}^{n} c_j \cdot w_j$.

En la métrica ralentización acotada (bounded slowdown) se acotó los tiempos de ejecución a 10 segundos, que es un valor utilizado comúnmente (Dror. G. Feitelson, 2005a; Tsafrir *et al.*, 2007), para evitar hacer énfasis en trabajos muy cortos (tiempo de ejecución cercano a cero) dado que ese tipo de trabajos generarían una ralentización muy grande.

Schwiegelshonhn (2009) sugiere utilizar $SWCT_{weight}$ como una métrica centrada en el sistema, tomando los recursos consumidos por el trabajo J_j , $w_j = size_j \cdot p_j$, como el peso de dicho trabajo. Schwiegelshonhn demostró que esta métrica exhibe muchas propiedades que son similares a las propiedades del tiempo de finalización del calendario.

Tabla I. Métricas implementadas

	Métrica	Descripción
	Centradas en el algoritmo	
1	Factor de competitividad	$\rho = \frac{C_{max}}{C^*_{max}}, C^*_{max} \geq \hat{C}^*_{max} = max\left\{\max_j(r_j + p_j), \frac{\sum_{j=1}^{n} w_j}{m_{1,m}}\right\}$
	Centradas en el usuario	
2	Promedio del tiempo de espera	$t_w = \frac{1}{n}\sum_{j=1}^{n}(c_j - p_j - r_j)$ ó $t_w = \frac{1}{n}\sum_{j=1}^{n}(s_j - r_j)$
3	Promedio del tiempo de espera ponderado por tamaño	$t_{ww}^{size} = \frac{1}{n}\sum_{j=1}^{n}(s_j - r_j) * size_j$
4	Promedio del tiempo de espera ponderado por tiempo	$t_{ww}^{time} = \frac{1}{n}\sum_{j=1}^{n}(s_j - r_j) * p_j$
5	Promedio del tiempo de espera ponderado por recursos consumidos	$t_{ww}^{work} = \frac{1}{n}\sum_{j=1}^{n}(s_j - r_j) * w_j$
6	Promedio de ralentización (Slowdown)	$SD = \frac{1}{n}\sum_{j=1}^{n}\frac{p_j + t_w^j}{p_j}$
7	Promedio de ralentización acotado	$SD_b = \frac{1}{n}\sum_{j=1}^{n}\frac{p_j + t_w^j}{max\{10, p_j\}}$ ó $SD_b = \frac{1}{n}\sum_{j=1}^{n}\frac{c_j - r_j}{max\{10, p_j\}}$ donde $t_w^j = c_j - p_j - r_j$
8	Promedio del tiempo de permanencia (Turnaround)	$TA = \frac{1}{n}\sum_{j=1}^{n}(c_j - r_j)$
9	Promedio del tiempo de permanencia ponderado con tamaño	$TA^{size} = \frac{1}{n}\sum_{j=1}^{n}(c_j - r_j) \cdot size_j$
10	Promedio del tiempo de permanencia ponderado con tiempo	$TA^{time} = \frac{1}{n}\sum_{j=1}^{n}(c_j - r_j) \cdot p_j$
11	Promedio del tiempo de permanencia ponderado con recursos consumidos	$TA^{work} = \frac{1}{n}\sum_{j=1}^{n}(c_j - r_j) \cdot w_j$
12	Suma de tiempos de espera	$SWT = \sum_{j=1}^{n} t_w^j$
13	Suma de tiempos de espera ponderados con tamaño	$SWWT_{size} = \sum_{j=1}^{n} size_j \cdot t_w^j$
14	Suma de tiempos de espera ponderados con tiempo	$SWWT_{time} = \sum_{j=1}^{n} p_j \cdot t_w^j$
15	Suma de tiempos de espera ponderados con recursos consumidos	$SWWT_{work} = \sum_{j=1}^{n} w_j \cdot t_w^j$

Tabla I. Métricas Implementadas (continuación)

	Métrica	Descripción
	Centradas en el sistema	
16	Capacidad de procesamiento (Throughput)	$Th = \frac{n}{C_{max}}$
17	Utilización	$U = \sum_{j=1}^{n} \frac{p_j \cdot size_j}{C_{max} \cdot m_{1,m}}$
18	Balanceo de carga por tamaño	$LBal = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(PL_i^q - \overline{PL}\right)^2}$ donde $PL_{i=1..m}^q = \frac{1}{m_i}\sum_{g_k=i}(size_k + size_j^q)$ y $size_j^q$ es el tamaño del trabajo j agregado al sitio q (Kurowski *et al.*, 2008).
19	Balanceo de carga por tiempo	$LBal = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(T_i^q - \bar{T}\right)^2}$ donde $T_{i=1..m}^q = \frac{1}{m_i}\sum_{g_k=i}(p_k + p_j^q)$ y p_j^q es el tiempo de ejecución del trabajo j agregado al sitio q.
20	Balanceo de carga por recursos consumidos	$LBal = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(W_i^q - \bar{W}\right)^2}$ donde $W_{i=1..m}^q = \frac{1}{m_i}\sum_{g_k=i}(w_k + w_j^q)$ y w_j^q son los recursos consumidos del trabajo j agregado al sitio q.
21	Suma de tiempos de finalización	$SCT = \sum_{j=1}^{n} c_j$
22	Suma de tiempos de finalización ponderados con tamaño	$SWCT_{size} = \sum_{j=1}^{n} size_j \cdot c_j$
23	Suma de tiempos de finalización ponderados con tiempo	$SWCT_{time} = \sum_{j=1}^{n} p_j \cdot c_j$
24	Suma de tiempos de finalización ponderados con recursos consumidos	$SWCT_{work} = \sum_{j=1}^{n} w_j \cdot c_j$

Se elegió utilizar t_w en lugar del tiempo de permanencia (turnaround) $TA = \frac{1}{n}\sum_{j=1}^{n}\left(c_j - r_j\right)$ del trabajo J_j debido a que la diferencia entre tales métricas es

una constante independiente del calendarizador que esté siendo utilizado. La constante en este caso es simple de deducir si se considera que TA también se puede expresar como $TA = \frac{1}{n}\sum_{j=1}^{n}\left(t_w^j + p_j\right)$, y que en la simulación se utiliza la misma carga para probar diferentes algoritmos, por lo tanto, el promedio de los tiempos de ejecución de todos los trabajos es constante.

Lo mismo sucede para la métrica suma de tiempos de espera (sum waiting times) $SWT = \sum_{j=1}^{n}\left(c_j - p_j - r_j\right)$ cuya diferencia con t_w es solamente la constante $1/\mathrm{n}$.

Se utiliza $SWCT_{weight}$ sobre el factor de competitividad (competitive factor) $p = sup_I \frac{C_{max}(S,I)}{C^*_{max}(I)}$, utilización (utilization) $U = \sum_{j=1}^{n} \frac{p_j \cdot size_j}{C_{max} \cdot m_{1,m}}$, y la capacidad de procesamiento (throughput) $Th = \frac{n}{C_{max}}$ debido a que existe una relación cercana entre estas métricas. En el primer caso C^*_{max} es constante para un experimento dado y $C_{max}(S,I) = max_j\{c_j(S,I)\}$ depende exclusivamente de c_j, en el segundo caso, $\sum_{j=1}^{n} \frac{p_j \cdot size_j}{m_{1,m}}$ también es constante dentro de un experimento, puesto que éste difiere en las estrategias y algoritmos simulados y no en la carga y la configuración del Grid. En el último caso, n es constante para un experimento dado.

Debido a que no es posible determinar, en general, un tiempo de finalización del calendario óptimo, en la evaluación de los experimentos se utilizó la cota más baja posible de éste, la cual se denota como $\check{C}^*_{max}$, y se define como $C^*_{max} \geq \check{C}^*_{max} = max\left\{max_j\left(r_j + p_j\right), \frac{\sum_{j=1}^{n} w_j}{m}\right\}$.

Se denota el modelo de Grid como GP_m , y utilizando la notación de tres campos $(\alpha|\beta|\gamma)$ introducida por Graham *et al.* (1979), se caracteriza el problema de calendarización como $GP_m|r_j, size_j|\{t_w, SD_b, SWCT_w\}$, que describe el ambiente de máquinas (α), las características de los trabajos (β) y las funciones objetivo (γ). Finalmente, se utiliza la notación MPS (Multiple Parallel Scheduling, Calendarización paralela múltiple) para referir el presente problema, y la notación PS (Parallel Scheduling, Calendarización paralela) para describir la calendarización paralela en una única máquina de múltiples procesadores o núcleos.

Capítulo III

Antecedentes

Los trabajos relacionados con el tema abordado se muestran y comentan en el presente capítulo. Se mencionan por ejemplo, trabajos sobre calendarización con resultados teóricos. También se describen algunos trabajos que previamente han aplicado estrategias de asignación en Grids en diversos modelos. Así también se citan algunos trabajos que han utilizado una predicción del tiempo de ejecución de los trabajos buscando mejorar los algoritmos de calendarización.

III.1 Trabajos relacionados

La calendarización de trabajos es un aspecto importante para lograr un alto rendimiento en Grids. Se han propuesto e implementado diversos sistemas de calendarización en diferentes tipos de Grids (Avellino *et al.*, 2003; Dimitriadou y Karatza, 2010; Elmroth y Tordsson, 2005; Krauter *et al.*, 2002; Lee *et al.*, 2006; Rodero *et al.*, 2005; Rodero *et al.*, 2008; Zikos y Karatza, 2008; Zikos y Karatza, 2009). Sin embargo, todavía existen muchas cuestiones por resolver en este campo.

En general, el problema de calendarización de trabajos en multiprocesadores es un problema bien conocido y ha sido tema de estudio por décadas. Existen muchos resultados de investigación para muchas variantes diferentes de este problema. Algunos de ellos proveen ideas teóricas y otros ofrecen sugerencias para la implementación de sistemas reales.

Sin embargo, la calendarización en Grids la abordan casi exclusivamente los profesionales que buscan implementaciones adecuadas. Hay muy pocos resultados teóricos disponibles sobre calendarización en Grid (Bougeret *et al.*, 2010a; Pascual *et al.*, 2009; Schwiegelshohn *et al.*, 2008; Tchernykh *et al.*, 2006; Tchernykh *et al.*, 2010; Zhuk *et al.*, 2004).

Schwiegelshohn *et al*. (2008) demostraron que el rendimiento del algoritmo de calendarización de lista propuesto por Garey y Graham es significativamente peor en Grids que en multiprocesadores. No hay un algoritmo de tiempo polinomial que garantice calendarios con una cota de competitividad <2 para $GP_m|r_j, size_j|C_{max}$ y todas las instancias del problema a menos que $P = NP$. Por consiguiente, la cota de calendarización de lista para multiprocesador $2 - 1/m$ para problemas fuera de línea (Garey y Graham, 1975), así como para problemas en línea, no se aplica a Grids. Más aún, la calendarización de lista no puede garantizar una cota de competitividad constante para todas las instancias del problema en el caso fuera de línea (Tchernykh *et al.*, 2006). También presentaron un algoritmo en línea no clarividente que garantiza un factor competitivo de 5 para el escenario de Grid, donde todos los trabajos asignados a una maquina están disponibles para otras máquinas y se pueden utilizar para la calendarización local de cualquier máquina, es decir, se permite la migración entre las máquinas. Este algoritmo de aproximación garantiza generar un calendario con un tiempo de finalización que está dentro de una constante 5 de la solución óptima. La versión fuera de línea no clarividente de este algoritmo tiene un factor de aproximación de 3.

Tchernykh *et al*. (2010) proveen un algoritmo para calendarización clarividente en línea de trabajos paralelos rígidos en un Grid con factor de competitividad $2e + 1$, basado en el modelo de balanceo de carga de Bar-Noy y Freund.(2001).

Más resultados teóricos están disponibles para problemas fuera de línea tanto para el caso no clarividente como para el caso clarividente. Por ejemplo Pascual *et al.* (2009) modelaron un sistema clarividente fuera de línea consistente de N clusters con exactamente m procesadores idénticos cada uno y propusieron un algoritmo que garantiza un factor de aproximación en el peor caso sobre el tiempo de finalización global igual a 4. Bougeret *et al*. (2010a) consideraron el problema con k máquinas paralelas de diferentes tamaños, y trabajos con tamaños que deben caber en la máquina más pequeña. El algoritmo de calendarización propuesto logra un factor de aproximación de 5/2.

Respecto al problema de calendarización jerárquica, Tchernykh *et al*. (2006) consideraron un problema clarividente con trabajos que no se pueden ejecutar en máquinas pequeñas, lo que limita su ejecución a un conjunto de máquinas *disponibles*, es decir, máquinas cuyo tamaño es mayor o igual que el tamaño del trabajo. Esto no implica que los recursos estén disponibles en el momento de la asignación. Con la finalidad de que trabajos pequeños no sean asignados a máquinas muy grandes, se limita el conjunto de máquinas disponibles a las máquinas más pequeñas cuya suma de tamaños sea la mitad de la suma de tamaños del total de máquinas disponibles. A este nuevo conjunto se le llama *máquinas admisibles*. Los autores presentaron algoritmos fuera de línea llamados $MLB_a + LSF$ y $MCT_a + LSF$ con factores de aproximación de 10. Donde MLB_a y MCT_ason estrategias de asignación de recursos restringidas a máquinas admisibles, la primera busca la menor cota mínima del tiempo de finalización, y la segunda, la máquina con el menor tiempo de finalización actual. *LSF* (*Larger Size First*. Primero los de tamaño más grande) representa el algoritmo local utilizado.

En (Tchernykh *et al.*, 2008), estos resultados fueron extendidos introduciendo un factor de admisibilidad que parametriza la disponibilidad de los sitios del Grid para asignación de trabajos. Se obtuvo un factor de competitividad que varía entre 5 e

infinito para el algoritmo de calendarización en línea adaptable $MLB_a + LSF$ con asignación de trabajos admisibles, y para cargas de trabajo con características específicas. En (Tchernykh *et al.*, 2010) se extendió este resultado para un modelo de carga más general. El factor de competitividad del algoritmo $MLB_a + LSF$ varía entre 17 e infinito con cambio del factor de admisibilidad. Este algoritmo puede ser aplicado al caso fuera de línea, pero no se provee un factor de aproximación para este caso.

Existen estudios experimentales que han propuesto diversas estrategias o políticas para la asignación de trabajos a recursos en un ambiente Grid. Por ejemplo en (Maheswaran *et al.*, 1999) se describen y comparan ocho heurísticas, 3 para el caso por lotes (batch) y 5 para el caso en línea (on-line): *Min-Min , Max-Min* y *sufrimiento* para el primer caso y *tiempo de finalización mínimo (MCT Minimum Completion Time)*, *tiempo de ejecución mínimo (MET Minimum Execution Time), algoritmo de intercambio (MET-MCT), mejor porcentaje k* y *balanceo de carga oportunista* para el caso en línea. El trabajo considera un modelo centralizado con máquinas uniformes (el tiempo de ejecución de un trabajo puede ser diferente en cada máquina pero no depende del tipo de trabajo ejecutado) y una carga de trabajo sintética de tareas secuenciales producidas con distribuciones de Poisson y de Gauss.

En (Hamscher *et al.*, 2000) se analizan modelos de Grid con diversas arquitecturas, centralizada, distribuida y jerárquica. Utilizan cuatro estrategias de selección de recursos: *BiggestFree*, que elige la máquina con el mayor número de recursos libres; *Random*, selecciona una de las máquinas disponibles aleatoriamente; *BestFit*, toma la máquina que deja la menor cantidad de recursos disponibles si el trabajo iniciará su ejecución; *EqualUtil* elige la máquina con la menor utilización. La carga de trabajo utilizada se tomó de dos bitácoras reales donde se duplicaron los trabajos para hacer tales bitácoras más grandes. Definieron 4 Grids diferentes con 8 sitios cada uno, solamente uno de esos sitios

utiliza datos de un proyecto de Grid real. Los autores concluyen que el rendimiento de los algoritmos examinados son altamente dependientes de los parámetros, la configuración de la máquina y la carga de trabajo.

En (Ranganathan y Foster, 2003b) se plantea un escenario de arquitectura distribuida para manejo de Grids de datos. Además de un calendarizador de Grid (o *externo* como ellos le llaman) y un calendarizador local, el modelo considera un *calendarizador de datos*, el cual es responsable de determinar si se deben replicar los datos y cuándo, o si se deben borrar archivos locales. Las estrategias que utilizaron para seleccionar un sitio son: *Aleatorio* (*Random*), *Menos cargado* (*LeastLoaded*), *Aleatorio en n menos cargados* (*RandLeastLoaded*), *Datos presentes* (*DataPresent*) y *Ejecución en el mismo sitio* (*Local*). Los autores concluyen que si bien es necesario considerar el impacto de la replicación en la estrategia de calendarización no siempre es necesario acoplar estas dos actividades.

Tchernykh *et al.* (2006) abordan el problema de calendarización jerárquica de dos niveles de la misma manera que se plantea en esta tesis. Se introduce un esquema simple de admisibilidad para evitar que trabajos pequeños ocupen máquinas grandes. Analizan teóricamente, para el caso determinista (offline), el rendimiento de las estrategias *menor carga* (*MinLoad*), *menor carga paralela* (*MinParallelLoad*), *menor cota inferior de finalización* (*MinLowerBound*), *menor tiempo de finalización* (*MinCompletionTime*), y cada una de éstas con la aplicación del esquema de admisibilidad. *Larger Size First* (LSF) es el algoritmo considerado como calendarizador local. El análisis del esquema de admisibilidad se extendió en (Tchernykh *et al.*, 2010) donde introdujeron un nuevo parámetro α que define el cociente de admisibilidad utilizado para la asignación de trabajos y analizan los cocientes de competitividad para diferentes características de carga de trabajo (mayor o menor grado de paralelismo).

Zikos y Karatza (2008) utilizan una arquitectura jerárquica para evaluar experimentalmente cuatro estrategias para asignación de trabajos: *Aleatorio* (*Random*), *Diferido* (*Deferred*, se elige el sitio con la menor carga pero se aplica solamente cuando el calendarizador de Grid recibe información del calendarizador local), *Híbrida* (combinación de las dos anteriores) y *Tiempo Real* (semejante a la estrategia Diferido pero sin esperar la información del calendarizador local). Los calendarizadores locales utilizados son *Aleatorio*, *Cola más corta*, y *Aleatorio 2-Cola más corta*. El modelo considera 4 sitios con 8 procesadores cada uno. Existe una cola en el calendarizador de Grid y una cola por cada procesador en el sitio local. Los trabajos considerados son secuenciales.

Existen también otras investigaciones que aplican heurísticas naturales como en (Abraham *et al.*, 2000) que tratan el problema de calendarización para un Grid jerárquico de dos niveles con lotes de trabajos (off-line) y aplican técnicas de algoritmos genéticos, templado simulado (simulated annealing) y búsqueda tabú (tabú search) para realizar la selección de recursos. En (Fölling *et al.*, 2010) se analiza un modelo de Grid con arquitectura de calendarización distribuida (existe un GRMS en cada sitio). Los autores aplican un sistema difuso evolutivo para decidir si el trabajo se ejecutará en el equipo local o será delegado a un sitio remoto. El problema principal con estas técnicas es que toman mucho tiempo para encontrar una solución aceptable como para poder aplicarlas a sistemas on-line.

III.2 Modelos de predicción de tiempo de ejecución

Según los resultados mostrados en (Etsion y Tsafrir, 2005), los algoritmos para calendarización local más utilizados en sistemas paralelos reales son FCFS (*First Come First Serve*. Primero en llegar, primero en ser atendido) y EASY (Extensible Argonne Scheduling sYstem) *Backfilling (Rellenado)*. Este último es una implementación de FCFS con relleno (Lifka, 1995).

Para la implementación del algoritmo *Rellenado* (ver sección IV.2) es esencial la información del tiempo de ejecución del trabajo, pero obviamente, este valor se desconoce al momento en que el sistema recibe el trabajo, solamente se conoce una vez finalizada su ejecución. También existen estrategias de asignación de trabajos que toman el tiempo de ejecución como uno de sus parámetros básicos para posteriormente elegir el sitio más adecuado para la ejecución del trabajo.

Para solucionar este problema, se ha optado por solicitar al usuario un tiempo estimado de la ejecución de su trabajo al momento de enviarlo al sistema y que además sirve como su cota superior de tiempo de ejecución, es decir, si el trabajo no ha terminado para el tiempo estimado, será terminado por el sistema.

Como puede suponerse, el tiempo de ejecución estimado por los usuarios dista mucho de ser preciso, puesto que en primer lugar, los usuarios no saben exactamente cuándo va a durar su proceso en determinado sistema, y en segundo lugar, por motivos de seguridad, los usuarios buscarán estimar un tiempo holgado para que su trabajo no sea terminado por el sistema antes de que finalice su ejecución (Bailey Lee *et al.*, 2004). Esta imprecisión puede tener un gran impacto en la calendarización de las tareas y en los resultados de la evaluación preliminar del rendimiento del sistema (Chiang *et al.*, 2002; Dror. G. Feitelson, 2005a; Mu'alem y Feitelson, 2001; Tsafrir *et al.*, 2006).

Algunos estudios han encontrado que sobreestimando el tiempo de ejecución, por ejemplo multiplicando el valor estimado por un factor constante, se mejora el rendimiento (Mu'alem y Feitelson, 2001; Zotkin y Keleher, 1999). Sin embargo, recientemente, Tsafrir y Feitelson (2006) encontraron una explicación al aparente incremento del rendimiento cuando se utiliza una mayor imprecisión de la estimación del tiempo de ejecución y demostraron que es mejor hacer estimaciones más precisas del mismo.

La precisión del tiempo de ejecución estimado por el usuario se calcula como el cociente del tiempo de ejecución real entre el tiempo de ejecución estimado del trabajo j: $precisión = p_j/p'_j$. La precisión de la predicción generada por el sistema se calcula como p_j/p''_j. Para evitar la compensación por sub predicciones y sobre predicciones cuando se calcula el promedio, se define la precisión, de acuerdo a (Tsafrir *et al.*, 2007) como:

$$precisión = \begin{cases} 1 & si\ p'_j = p_j \\ p_j/p'_j & si\ p'_j > p_j \\ p'_j/p_j & si\ p'_j < p_j \end{cases}$$

Con el objetivo de encontrar mejores estimaciones del tiempo de ejecución, se han investigado otras técnicas, como el modelado y la histórica. En el primer caso, basados en información registrada en logs reales, se obtiene un modelo matemático que permita generar automáticamente valores estimados, semejantes a los registrados. Este modelo se puede utilizar después de forma independiente para generar nuevos valores estimados o para complementar logs que carecen de tal información. En (Song *et al.*, 2004; Tsafrir *et al.*, 2006) se describen modelos que permiten generar tiempos estimados de ejecución . La técnica histórica se basa en la suposición de que los usuarios ejecutan el mismo tipo de trabajos todo el tiempo (Dror G. Feitelson y Nitzberg, 1995), lo cual sugiere que la estimación del tiempo de ejecución del siguiente trabajo que envíe el usuario, se puede basar en el tiempo de ejecución de trabajos ejecutados enviados previamente por él mismo (Tsafrir *et al.*, 2007).

El problema de la estimación de tiempo se vuelve más complicado para los usuarios cuando envían su trabajo a un Grid, puesto que en general no cuenta con información suficiente o habilidades suficientes para especificar por cuánto tiempo correrá su trabajo, debido a la naturaleza heterogénea de los recursos que componen el Grid y al desconocimiento del recurso en que se ejecutará su trabajo. Lo mismo sucede cuando se trata de arquitecturas paralelas (ej. clusters) compuestas por recursos computacionales heterogéneos. Debido a esto el estudio

de técnicas de predicción de tiempo de ejecución se ha vuelto muy relevante en los últimos años (Guim *et al.*, 2007).

En (Smith, 2003), Smith aborda el problema de co-asignación en un Grid computacional incorporando reservaciones avanzadas en sistemas de calendarización. Las reservaciones avanzadas permiten a los usuarios solicitar recursos de múltiples sistemas en un tiempo específico con el fin de obtener acceso simultáneo a recursos suficientes para la aplicación. Gran parte de su trabajo utiliza como base dos técnicas de predicción del tiempo de ejecución de las aplicaciones en computadoras paralelas de espacio compartido: basada en información histórica de corridas previas similares y mediante una base de datos de experiencias (características de entradas y salidas de la ejecución de las aplicaciones).

En (Talby *et al.*, 2006) se analizan tres técnicas de predicción usando niveles decrecientes de información, enfocadas a la calendarización en Grid y a máquinas paralelas: *Session-based*, *Estimation-less*, e *Information-less*. La primera se basa en el conocimiento de que los usuarios trabajan típicamente en sesiones, es decir, periodos de intenso trabajo repetitivo, esta técnica utiliza toda la información disponible acerca de un trabajo. En la segunda analizan el caso donde la estimación del tiempo de ejecución no está disponible, lo cual permitiría quitarle la necesidad al usuario de proporcionar dicho valor. La tercera técnica analiza el caso de la predicción del tiempo de ejecución cuando no existe información alguna, lo cual es una situación común en sistemas de Grids. Los autores toman como referencia cuatro algoritmos de predicción existente: *perfect predictor*, el cual utiliza el tiempo de ejecución registrado, *constant predictor*, el cual predice el mismo tiempo de ejecución constante para todos los trabajos, *estimate predictor* que usa el tiempo estimado por el usuario, y *recent user history* (EASY++) cuya predicción es el promedio de los tiempos de ejecución de los últimos tres trabajos del mismo usuario.

Capítulo IV

Estrategias y algoritmos de calendarización

Las estrategias de asignación de trabajos, implementadas para esta investigación, fueron clasificadas con base al tipo y cantidad de información que ellas utilizan. La clasificación de los niveles de información, las estrategias de asignación implementadas, así como los algoritmos de calendarización utilizados, se describen en este capítulo.

IV.1 Estrategias de asignación de trabajos

Como se comentó antes, el esquema de calendarización en Grid utilizado en este trabajo, consta de dos partes, una asignación global y una calendarización local. Se considera *MPS* como una estrategia de calendarización de dos etapas: *MPS* = *MPS_Alloc* + *PS*. En la primera etapa se asigna una máquina adecuada para cada trabajo con base en un criterio de selección. En la segunda etapa se aplica el algoritmo PS, independientemente en cada máquina, a los trabajos asignados durante la etapa previa.

Desde el punto de vista del sistema, el objetivo del calendarizador de Grid es lograr algún tipo de balanceo de carga entre los diferentes sitios que componen el Grid y de esta forma obtener, por ejemplo, la mayor utilización de los recursos correspondientes. Sin embargo, desde el punto de vista de los usuarios, el calendarizador de Grid puede buscar objetivos diferentes a los que persiguen los propietarios de los recursos, como por ejemplo, reducir el tiempo de espera.

En este trabajo se hizo una clasificación de estrategias según la cantidad y tipo de información que requieren para seleccionar el recurso más adecuado. La información considerada corresponde al tamaño y tiempo de los trabajos así como a la información de los calendarios locales. El número total de procesadores, el número de máquinas y el número de procesadores por máquina se conocen de antemano. Se clasificaron tres niveles de información disponible para la asignación de trabajos (Tabla II):

Tabla II. Clasificación de niveles de conocimiento para estrategias de asignación

Nivel	Descripción
1	Una vez que el usuario envía su trabajo, solamente se conoce su requerimiento de procesadores. No existe información sobre el tiempo de procesamiento de los trabajos. Sin embargo, el algoritmo puede utilizar información sobre los trabajos previamente asignados a cada máquina.
2	Además de la información del nivel 1, se conoce el tiempo estimado $p_j^{'}$ o la predicción del tiempo de ejecución por parte del sistema $p_j^{''}$.
3	Se tiene acceso a la información del nivel 2 y también a todos los calendarios locales. El GRMS tiene información sobre los calendarios locales de las máquinas, y puede considerar esta información para seleccionar una máquina.

Algunas de las estrategias consideradas son estrategias básicas conocidas por su aplicación a sistemas distribuidos y paralelos, tal es el caso de el *menor número de trabajos por máquina* (*menor carga*) y la *menor desviación estándar de la carga* (considerada en este trabajo como *LBal_S*) (Kurowski *et al*., 2008).

Con la finalidad de complementar la clasificación de estrategias en los niveles 1 y 2, se introdujeron las estrategias de asignación: *Menor carga paralela (Min Parallel Load), Menor cota inferior del tiempo de finalización (Min Lower Bound), Load Balance Time (LBal_T)* y *Load Balance Work (LBal_W)*, éstas dos últimas basadas en el mismo principio que la estrategia *Load Balance Size (LBal_S)*.

En el nivel 3 de clasificación, se implementaron estrategias basadas en métricas conocidas en la literatura para la evaluación de calendarios. En términos generales, para seleccionar el mejor sitio, el gestionador de recursos elabora un calendario tentativo para cada uno, se calcula la métrica correspondiente y se elige el sitio que obtuvo el menor valor para dicha métrica. Las estrategias son: *Menor tiempo promedio de espera, Menor tiempo de espera con peso* (con tres variantes de peso: tamaño, tiempo y recursos consumidos), *Menor tiempo de finalización del calendario* (*makespan*)*, Menor tiempo de inicio (o menor tiempo de espera*) y *Menor suma de tiempos de finalización con peso* (*Tiempo de finalización total con peso)*. En este mismo nivel se implementó la estrategia *Menor tiempo de inicio*, esta es una estrategia conocida en la literatura, que no se basa en alguna métrica de evaluación de calendario, sino que simplemente busca el sitio donde el trabajo pueda iniciar antes que en los demás.

Se implementó también una estrategia de asignación aleatoria *Random* con fines meramente comparativos, ya que no se esperaba que generara buenos resultados.

En la Tabla III se listan las 14 estrategias consideradas en el presente trabajo. Estas estrategias se implementaron en el simulador de eventos discretos GSimula, desarrollado durante el presente trabajo (ver el apéndice A.6 para más información respecto al simulador). La fórmula utilizada en cada métrica toma en cuenta la información correspondiente de los trabajos previamente asignados al sitio, más el que está por asignarse. Esto significa que el valor resultante es un estimado de cuál sería el valor de la métrica en cuestión al asignar el nuevo trabajo a dicho sitio.

Estas estrategias fueron seleccionadas para soportar asignación trabajo-por-trabajo y para cubrir un amplio rango de información disponible en el momento de la asignación.

Tabla III. Estrategias de Asignación

Estrategia	Nivel	Descripción
Random (Aleatorio)	1	Asigna aleatoriamente cada uno de los trabajos a uno de los sitios disponibles. Los sitios disponibles son todos aquellos que por su tamaño, son factibles de ejecutar el trabajo j, es decir, $size_j \leq m_i$. Esto implica que para asignar un trabajo con esta estrategia se debe conocer previamente su tamaño. En promedio, esta estrategia debe proveer una distribución justa del número de trabajos entre todos los sitios del Grid, sin embargo, como la asignación se restringe al conjunto de sitios que pueden ejecutar cada trabajo, no se garantiza una adecuada distribución.
MLp (*Min Load per processor. Menor carga por procesador*)	1	Asigna el trabajo j al sitio con la menor carga por procesador en el tiempo r_j. $$min_{i=1..m}\left(\frac{l_i}{m_i}\right)$$ Al igual que la estrategia *random*, esta estrategia busca balancear adecuadamente el número de trabajos entre los diferentes sitios. Sin embargo, buscando una distribución más justa, esta estrategia normaliza el número de trabajos con el tamaño de cada sitio. Con esto, los sitios más grandes pueden recibir más trabajos que los sitios más pequeños. Por ejemplo, un sitio de tamaño 2 que tenga 4 trabajos tendrá una carga por procesador menor que un sitio de tamaño 12 con 4 trabajos; en el primer caso la carga es 2 y en el segundo la carga es 3, por lo que el trabajo j sería asignado al sitio más grande.
MPL (*Min Parallel Load. Menor carga paralela*)	1	Asigna el trabajo j al sitio con el menor grado de paralelismo por procesador considerando todos los trabajos que están en cola y en ejecución en cada sitio en el tiempo r_j. $$min_{i=1..m}\left\{\sum_{k=1}^{l_i}\frac{size_k}{m_i}\right\}$$ Cuando se asignan los trabajos en base solamente al número de ellos asignados previamente en cada sitio, no se hace distinción entre un trabajo y otro, por ejemplo, un trabajo de tamaño 100 se considera igual que otro de tamaño 1. Esta estrategia toma en cuenta el tamaño de los trabajos con la intención de ser más justos en este aspecto, buscando el sitio con el menor número de veces que son requeridos el total de sus procesadores. Un sitio de tamaño 10 que tiene una carga paralela de 2, indica que el total de sus recursos han sido requeridos 2 veces. La intuición detrás de *MPL* es mantener todos los procesadores tan ocupados como sea posible. Una ventaja de esta estrategia es su simplicidad, no toma en cuenta el tiempo de ejecución del trabajo ni su estimado.
LBal_S (*Load Balance Size. Carga balanceada por tamaño*)	1	Asigna el trabajo j al sitio con la menor desviación estándar del grado de paralelismo por procesador de todos los trabajos que están en espera y en ejecución en un momento dado (tomando en cuenta todos los sitios) cuando el trabajo se asigna a dicho sitio. Esta estrategia responde a la pregunta de que tanto se alejaría del promedio la carga paralela de un sitio si se le asigna el trabajo j.

Tabla III. Estrategias de Asignación (continuación)

Estrategia	Nivel	Descripción
		$\min_{q=1..m} \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(PL_i^q - \overline{PL}\right)^2}$ 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 del trabajo j agregado al sitio q solamente (Kurowski *et al*., 2008). El objetivo de esta estrategia es mantener la carga paralela de trabajo por procesador en cada sitio lo más cerca del promedio, y con esto mantener ocupados los procesadores tanto como sea posible.
LBal_T (*Load Balance Time. Carga balanceada por tiempo*)	2	Asigna el trabajo j al sitio con la menor desviación estándar del tiempo de ejecución por procesador de todos los trabajos que están en espera y en ejecución en un momento dado (tomando en cuenta todos los sitios) cuando el trabajo se asigna a dicho sitio. $\min_{q=1..m} \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(T_i^q - \bar{T}\right)^2}$ Donde $T_{i=1..m}^q = \frac{1}{m_i}\sum_{k=1}^{l_i}(p_k + p_j^q)$ y p_j^q es el tiempo de ejecución del trabajo j agregado al sitio q solamente. Al igual que con el tamaño de un trabajo, un trabajo que dura en ejecución 1 hora, no debería ser igual que un trabajo que dura 1 segundo. Esta estrategia le da importancia al tiempo de ejecución y busca una distribución de trabajos más justa, manteniendo los tiempos de ejecución de la carga actual por procesador en cada sitio, lo más cercanos al promedio.
LBal_W (*Load Balance Work. Carga balanceada por recursos consumidos*)	2	Asigna el trabajo j al sitio con la menor desviación estándar de los recursos consumidos por procesador de todos los trabajos que están en espera y en ejecución en un momento dado (tomando en cuenta todos los sitios) cuando el trabajo se asigna a dicho sitio. $\min_{q=1..m} \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(W_i^q - \bar{W}\right)^2}$ Donde $W_{i=1..m}^q = \frac{1}{m_i}\sum_{k=1}^{l_i}(w_k + w_j^q)$ y w_j^q son los recursos consumidos del trabajo j agregado al sitio q solamente. Esta estrategia tiene el mismo objetivo que las estrategias *LBal_S* y *LBal_T*, mantener la carga en cada sitio lo más cercana posible al promedio. La diferencia en este caso, es que la carga considerada se basa en los recursos consumidos, es decir, tamaño por tiempo de ejecución. Los recursos consumidos permiten diferenciar trabajos que por tamaño o por tiempo pueden parecer iguales. Por ejemplo, un trabajo de tamaño 1 con tiempo de ejecución de 1 hr, no puede ser considerado igual que un trabajo con tamaño 20 con el mismo tiempo de ejecución.

Tabla III. Estrategias de Asignación (continuación)

Estrategia	Nivel	Descripción
MLB (*Min Lower Bound.Menor cota inferior del tiempo de finalización*)	2	Asigna el trabajo j al sitio con los menores recursos consumidos por procesador en el tiempo r_j. $$min_{i=1..m}\left\{\sum_{k=1}^{l_i}\frac{size_k \cdot p_k^{'}}{m_i}\right\}$$ Considerar los recursos consumidos por un trabajo permite diferenciar aquellos que por su tamaño o tiempo puedan parecer iguales, esto podría dar una distribución más justa de la carga. Esta estrategia calcula cual sería la cantidad de recursos consumidos si éstos se distribuyeran equitativamente entre todos los procesadores del sitio. Esto se puede interpretar como una cota inferior del tiempo de finalización del calendario.
MCT (*Min Completion Time. Menor tiempo de finalización*)	3	Asigna el trabajo j al sitio con el menor tiempo de finalización del 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 del trabajo J_k en el sitio i. Esto ocasiona que algunas tareas sean asignadas a máquinas que no tienen el mínimo tiempo de finalización para ella. Esta estrategia intenta minimizar el tiempo de finalización del calendario. Para calcular esta métrica en cada sitio, se elabora un calendario tentativo en el gestionador de recursos con los trabajos enviados previamente a cada sitio. El algoritmo aplicado es FCFS.
MWT (*Min Waiting Time. Menor tiempo de espera*)	3	Asigna el trabajo j al sitio con el menor promedio de tiempo de espera de los trabajos. $$min_{i=1..m}\left\{\sum_{g_k=i}\frac{t_w^k}{n_i}\right\}$$ Esta estrategia basa su decisión en una de las métricas más comúnmente utilizadas en la calendarización, el tiempo promedio de espera. Esto implica que se debe elaborar un calendario con los trabajos asignados en cada uno de los sitios. El calendario considerado es un calendario tentativo, elaborado en el nivel 1, por el gestionador de recursos. Este calendario puede ser diferente al calendario real elaborado en cada sitio, debido a que los algoritmos aplicados en cada nivel pueden ser diferentes. Incluso, debido a la naturaleza del Grid, cada sitio podría estar aplicando un algoritmo diferente para calendarizar sus trabajos. El algoritmo para la elaboración del calendario tentativo es FCFS. Como se sabe, la métrica del tiempo promedio de espera es una métrica centrada en el usuario que busca minimizar el intervalo de tiempo entre el instante en que su trabajo se envia al sistema y el instante en que inicia su ejecución.
MWWT_S (*Min Weighted Waiting Time_Size. Menor tiempo de espera ponderado por tamaño*) *MWWT_T* (*Min Weighted Waiting*	3	Asignan el trabajo j al sitio con el menor tiempo promedio de espera ponderado (con peso). $$min_{i=1..m}\left\{\sum_{g_k=i}\frac{t_w^k \cdot weight_k}{n_i}\right\}$$ Donde $weight_k = \{size_k, p_k', w_k\}$

Tabla III. Estrategias de Asignación (continuación)

Estrategia	Nivel	Descripción
Time_Time. Menor tiempo de espera ponderado por tiempo)*MWWT_W* (*Min Weighted Waiting Time_Work. Menor tiempo de espera ponderado por recursos consumidos*)		Estas estrategias se basan en otra métrica bien conocida, el tiempo de espera ponderado. La ponderación representa en términos generales, prioridad. Minimizar el tiempo de espera ponderado por tamaño, implica dar prioridad a los trabajos grandes, puesto que entre menor tiempo de espera tengan los trabajos grandes, menor impacto tendrá su peso en la métrica. La ponderación por tiempo y recursos consumidos busca dar prioridad a los trabajos largos y que consumen más recursos respectivamente. Para calcular esta métrica en cada sitio, se elabora un calendario tentativo en el gestionador de recursos con los trabajos enviados previamente a cada sitio. El algoritmo aplicado es FCFS. El calendario real en cada sitio puede ser diferente debido a la aplicación de otros algoritmos de calendarización. Esto es razonable debido a que cada sitio que participa en el Grid es independiente y aplicas sus propios esquemas de administración.
MST (*Min Start Time. Menor tiempo de espera*)	3	Asigna el trabajo j al sitio que pueda iniciar más temprano su ejecución. $$min_{\ i=1..m}\{s_j^i\}$$ Para plataformas homogéneas, esta estrategia asigna cada tarea a la máquina con el menor tiempo de finalización esperado para tal tarea (*Earliest-Finish-Time*). Al disminuir el tiempo de finalización de cada tarea también se minimiza el tiempo de finalización del calendario total del Grid. Esta estrategia también requiere de la elaboración de un calendario tentativo, puesto que se debe determinar el instante estimado en que cada sitio puede tener disponibles los recursos suficientes para iniciar la ejecución del trabajo. El calendario se elabora con el algoritmo FCFS.
MSWCT_W (*Min Sum Weighted Completion Time_Work. Menor suma de tiempo de espera con peso*)	3	Asigna el trabajo j al sitio con la menor suma de tiempos de finalización ponderados con los recursos consumidos. $$min_{i=1..m}\{\sum_{g_k=i} c_k \cdot w_k\}$$ Esta estrategia se basa en una métrica muy conocida para evaluar un calendario, la *suma* (*o promedio*) *de tiempos de finalización ponderados*. Esta es una métrica centrada en el propietario del recurso, cuya minimización implica una mejor utlización de los recursos (Schwiegelshohn, 2009). A diferencia de la métrica *Tiempo de Finalización Total* (*o Suma de Tiempos de Finalización*) que favorece trabajos secuenciales sobre trabajos paralelos si tienen los mismos tiempos de procesamiento, esta métrica es neutral respecto a las características de los trabajos. El calendario se elabora con el algoritmo FCFS.

IV.2 Estrategias de calendarización local

Una vez que el trabajo se asigna a la máquina paralela, el sistema local de administración de recursos (*LRMS. Local Resource Management System)* de esta máquina generará un calendario con base en la aplicación de algún algoritmo de calendarización. Existen diferentes algoritmos que el LRMS puede aplicar; muchos sistemas reales aplican el algoritmo *FCFS* (*First Come First Serve*, *Primero en llegar, primero en ser atendido*), que calendariza los trabajos en el orden de sus tiempos de llegada. Si no hay suficientes recursos disponibles, el trabajo espera hasta que se liberen recursos y pueda iniciarse mientras los demás trabajos permanecen detenidos en la cola. Aún cuando este algoritmo se desempeña bastante bien en la práctica, tiene muy malos resultados para el peor caso. Este algoritmo cuenta con la ventaja de ser de muy fácil implementación pero puede provocar fragmentación y retardar trabajos cortos que se quedan bloqueados detrás de trabajos grandes.

Una mejora al algoritmo FCFS es el algoritmo *Rellenado* (*Backfilling*) propuesto inicialmente por David Lifka (1995; Skovira *et al.*, 1996). El algoritmo *Rellenado* ha probado ser una muy buena opción para incrementar el rendimiento tanto para el usuario como para el sistema. Este algoritmo permite que algunos trabajos puedan iniciar su ejecución antes del tiempo que les correspondería según su posición en la cola. Esto se logra, realizando una búsqueda de trabajos más pequeños en la cola de trabajos disponibles para ejecución inmediata cuando el trabajo de la cabeza de la cola no puede iniciar su ejecución debido a que no existen suficientes recursos para él inmediatamente.

El algoritmo *Rellenado* se puede implementar de varias maneras, pero existen dos versiones principales: *Conservative Backfilling* (*Rellenado conservador*) y *EASY Backfilling* (*Rellenado EASY. Extensible Argonne Scheduling sYstem*) (D. Feitelson *et al.*, 2005b; Dror G. Feitelson y Weil, 1998). En el primer caso se busca garantizar que *ninguno* de los trabajos en la cola de espera retrase el inicio de su

ejecución. Esto se logra haciendo una reservación de tiempo de inicio de ejecución para cada trabajo cuando éste se recibe, de tal manera que cuando se buscan trabajos en la cola, se establecen estas reservaciones como limitantes para poder elegir los trabajos que se pueden utilizar para rellenar. En el caso de EASY Backfilling la única limitante para iniciar otros trabajos es que no retrasen el inicio de la ejecución del trabajo de la cabeza de la cola. Es importante notar que un trabajo pequeño (menor grado de paralelismo) no necesariamente es corto (menor tiempo de ejecución), por lo que un trabajo pequeño pero largo se puede utilizar como relleno en el caso de que haya recursos libres más allá de la reservación (EASY) o reservaciones (Conservative) establecidas para los demás trabajos.

Como se puede apreciar, la base de los algoritmos *Rellenado* es el tiempo de ejecución de los trabajos, ya que para realizar una reservación de tiempo de inicio de ejecución se necesita saber cuándo finalizarán los trabajos que están en ejecución. Así mismo, para seleccionar los trabajos para relleno de la cola de espera, es necesario conocer el tiempo de ejecución de cada uno de ellos. Resulta obvio que mientras un trabajo no termine su ejecución no se puede conocer el tiempo que consumió. En estos casos, es necesario contar con una estimación del tiempo de ejecución, que puede proveer el propio usuario o generarse internamente por el sistema local de administración de recursos. Dado que este dato es esencial para el funcionamiento de los algoritmos de rellenado, se convierte en el parámetro principal que define el rendimiento de este tipo de algoritmos. Algunos trabajos relevantes que analizan la importancia de la estimación del tiempo de ejecución en el rendimiento de *Rellenado* se citan en la Sección III.2.

En el presente trabajo se utiliza el algoritmo EASY Backfilling como algoritmo de calendarización debido a su uso extendido en sistemas reales y que se ha probado que ofrece un mejor rendimiento que FCFS cuando la carga de trabajo es heterogénea (Frachtenberg y Feitelson, 2005).

Capítulo V

Cargas de trabajo

Un aspecto relevante en cualquier tipo de simulación, son los datos de entrada. En este trabajo , se hace uso de dos cargas de trabajos paralelos construidas a partir de varias bitácoras de sistemas reales. Este capítulo contiene la descripción de la metodología empleada en la generación de la carga así como un análisis de ella.

V.1 Bitácoras de sistemas en producción

Uno de los puntos claves en la simulación de sistemas es, sin duda, la selección de los datos de entrada para la ejecución de los experimentos, dado que de ellos depende la precisión de los resultados (Frachtenberg y Feitelson, 2005). Para el caso de Grid, a pesar de los esfuerzos realizados por el equipo de *The Grid Workloads Archive* (Anoep *et al.*, 2007) por ofrecer una plataforma y un estándar para cargas de Grid, no se cuenta todavía con una carga con características generales para Grid. Algunas de las cargas que se encuentran en el sitio son solamente de trabajos secuenciales, otras consideran solamente pocos sitios o una monitorización de solo unos días.

Existen trabajos que utilizan una carga generada con funciones de probabilidad como en (Maheswaran *et al.*, 1999); sin embargo, se tiene que hacer un estudio muy fuerte del comportamiento de las cargas de sistemas reales para poder proveer un sistema generador de cargas con base en probabilidad, como se demuestra en (Tsafrir *et al.*, 2006), cuyos autores generan un modelo complejo solamente para reproducir tiempos estimados de bitácoras reales. En términos generales, la reproducción de características de una carga de trabajos para un

sistema paralelo, como tiempos de llegada durante el día, la semana o incluso meses, tiempos de ejecución, grado de paralelismo, patrón de envío por usuario, etc. implica un trabajo muy fuerte que finalmente no es redituable puesto que se convierte en un carga dependiente de un sitio específico. El generar una carga para Grid de esta manera, se vuelve un trabajo mucho más complejo.

Otra opción, que se considera más adecuada, es el uso directo de bitácoras reales de sistemas en producción como en (Grimme *et al.*, 2007, 2008; Hamscher *et al*., 2000), las cuales proveen una secuencia de trabajos reales que se pueden utilizar para la evaluación del rendimiento de algoritmos basados en simulación. Para generar una carga para el Grid con base en bitácoras reales, se puede pensar inicialmente en una mezcla de ellas, intentando reproducir la participación de los usuarios de cada sitio en un ambiente de Grid.

Para los experimentos se utiliza una mezcla de trabajos (en base a su tiempo de llegada) registrados en bitácoras de sistemas en producción, tomadas de *Parallel Workloads Archive* (Dror G. Feitelson, 2005b), y para generar una carga más representativa de un Grid se aplica la siguientes pasos:

1. *Selección de bitácoras* que tengan registrado, al menos, por cada trabajo los siguientes datos válidos: el tiempo de llegada, el número de procesadores solicitado, el tiempo de procesamiento, la identificación de usuario y el tiempo de ejecución estimado, debido a que son datos necesarios para el funcionamiento de las estrategias y algoritmos de calendarización implementados
2. *Normalización por zona horaria*. Al mezclar las bitácoras es importante considerar la zona horaria en que se generó cada una así como la hora de inicio de su registro. Esto obedece a que el envío de trabajos tiene características especiales durante las diferentes horas del día, por ejemplo, existen una menor cantidad de trabajos enviados durante la madrugada que

durante el medio día. De esta manera, si el primer trabajo de una bitácora de la zona horaria GMT 0, llegó a las 12 pm y el primer trabajo de otra bitácora de zona horaria GMT -6, también llegó a las 12 pm, no pueden ser considerados que llegaron al mismo tiempo al sistema. Para elaborar la mezcla utilizada en este trabajo se tomó la zona horaria menor y se ajustaron todas las demás a dicho valor. Para lograrlo se modificaron los datos del encabezado correspondientes a la zona horaria, a la hora y a la fecha de inicio. Esto equivale a considerar que el gestionador de recursos se encuentra en la zona horaria menor.

3. *Exclusión de los primeros trabajos* de cada bitácora para descartar las condiciones iniciales del sistema, dado que éstas no son representativas de su estado estable (Frachtenberg y Feitelson, 2005).
4. *Normalización por día de la semana*. Al igual que existe un comportamiento característico del envío de trabajos durante las diferentes horas del día, también existe un comportamiento especial para los diferentes días de la semana, por ejemplo, es diferente el envío de trabajos en fines de semana que entre semana. Para respetar esta característica en las bitácoras, se ajustó el inicio de trabajos válidos a las 0 horas del primer lunes en cada bitácora y como fecha final, el último domingo registrado en cada una. Después de excluir los trabajos por las condiciones iniciales y por la normalización al primer lunes, se modifica el tiempo de llegada de todos los trabajos válidos, así, el primer trabajo de cada bitácora tendrá un tiempo de llegada relativo al primer lunes y no a la fecha real de inicio de su registro.
5. *Normalización de números de identificación de usuario*. Dado que cada bitácora registra a sus usuarios de manera independiente, se podría encontrar al usuario 1 en todas ellas, lo cual no permitiría identificar unívocamente a cada uno de los usuarios participantes en el Grid. Para solucionar este problema, se modificaron los números de identificación de usuario dándoles un número secuencial, así, los usuarios de la primer

bitácora tendrán identificaciones de 1 a *m*, los de la segunda bitácora tendrán identificadores de *m+1* a *n* y así consecutivamente.

6. *Filtrado de trabajos no válidos*. A pesar de que las bitácoras utilizadas son una versión previamente filtrada y adaptada al formato estándar (SWF. Standard Workload Format) propuesto inicialmente por Chaping *et al.* (1999) y cuya versión actualizada se encuentra en (Dror G. Feitelson, 2005b), existen trabajos que no son relevantes para la presente investigación, por lo cual se aplicaron filtros a cada bitácora para eliminar todos los trabajos con:
    a. número de trabajo menor o igual a cero (job_number <= 0)
    b. tiempo de llegada menor que cero (release_time < 0)
    c. tiempo de ejecución menor o igual a cero (runtime <= 0)
    d. número de procesadores asignados menor o igual a cero (allocated_processors <= 0)
    e. tiempo estimado menor o igual a cero (requested_time <= 0)
    f. Identificador de usuario menor o igual a cero (user_id <= 0)
    g. Estatus fallido (status = 0)
    h. Estatus fallido en la última ejecución parcial (status = 4)
    i. Estatus cancelado por el usuario (status = 5)
7. *Mezcla*. Finalmente, una vez que todas las bitácoras pasaron por el proceso anterior, se procede a tomar los trabajos en orden no decreciente de su tiempo de llegada hasta completar el número de trabajos o el número de días solicitado. Cuando se llega al final de una bitácora y no se ha completado la mezcla, la bitácora se duplica, copiando todos los trabajos válidos al final y modificando correspondientemente su tiempo de llegada.

Se debe mencionar que la unión de varias bitácoras independientes para simular la carga de un Grid computacional no garantiza una representación del Grid real con las mismas máquinas y usuarios. Por ejemplo, si un sitio pasa a ser parte de un Grid computacional donde hay máquinas más grandes disponibles, los usuarios

pueden enviar trabajos más grandes y que no están representados en la bitácora original. No obstante, éste es un buen punto de inicio para evaluar estrategias de calendarización basada en bitácoras reales dada la falta de cargas de trabajo de uso público para Grid.

Para la elaboración de la mezcla se utilizó el programa *mix_logs.m* desarrollado en Matlab por José Luis González García como parte de su tesis de maestría (García, 2009).

V.2 Configuración del Grid y análisis de la carga

Para la evaluación de las estrategias consideramos dos escenarios que denominamos Grid1 y Grid2. En cada escenario consideramos las características reales de 7 y 9 sitios, respectivamente. Para generar la carga en cada uno, se aplicó la metodología descrita en la sección V.I. La configuración de cada sitio se muestra en las Tablas IV y V.

Tabla IV. Configuración del Grid 1

	Origen	#Procs	Bitácora	#Trab	#Usua
1	KTH- Swedish Royal Institute of Technology	100	KTH-SP2-1996-2.swf	28489	204
2	SDSC-SP2 – San Diego Supercenter SP2	128	SDSC-SP2-1998-3.1-cln.swf	73496	437
3	HPC2N-High Performance Computing Center North, Sweden	240	HPC2N-2002-1.1-cln.swf	527371	256
4	CTC-Cornell Theory Center	430	CTC-SP2-1996-2.1-cln.swf	79302	679
5	LANL-Los Alamos National Lab	1024	LANL-CM5-1994-3.1-cln.swf	201387	211
6	SDSC-BLUE –San Diego Supercenter Blue Gene	1152	SDSC-BLUE-2000-3.1-cln.swf	250440	468
7	SDSC-DS – San Diego Supercenter Data Star	1368	SDSC-DS-2004-1-cln.swf (Batch 2)	96089	460
	Totales	**4442**		**1256574**	**2715**

Tabla V. Configuración del Grid 2

	Origen	#Procs	Bitácora	#Trab	#Usua
1	DAS2 - University of Amsterdam	64	Gwa-t-1-anon_jobs-reduced.swf	1124772	333
2	DAS2 - Delft University of Technology	64			
3	DAS2 – Utrecht University	64			
4	DAS2 - Leiden University	64			
5	KTH - Swedish Royal Institute of Technology	100	KTH-SP2-1996-2.swf	28489	204
6	DAS2- Vrije University Amsterdam	144	Gwa-t-1-anon_jobs-reduced.swf (cont.)	Incluidos en la primera bitácora	
7	HPC2N - High Performance Computing Center North, Sweden	240	HPC2N-2002-1.1-cln.swf	527371	256
8	CTC - Cornell Theory Center	430	CTC-SP2-1996-2.1-cln.swf , jobs, users	79302	679
9	LANL -.Los Alamos National Lab	1024	LANL-CM5-1994-3.1-cln.swf, jobs, users	201387	211
	Totales	**2194**		**1961321**	**1683**

La Tabla VI muestra los datos de cada bitácora del Grid 1 después de aplicar la metodología descrita en la Sección IV.1 para un periodo de 180 días (6 meses). La Tabla VII hace lo correspondiente con el Grid 2.

Tabla VI. Resultados del pre-proceso de las bitácoras del Grid 1 previo a la mezcla

	Número de Bitácora						
Característica	**1**	**2**	**3**	**4**	**5**	**6**	**7**
ID de Usuarios	931-1130	2218-2652	674-930	1-673	1131-1342	1343-1809	1810-2217
Días iniciales borrados	8	5	20	5	0	6	25
Trabajos utilizados	10197	13161	17812	33875	23501	29717	24133
Repeticiones de la bitácora	1	1	1	1	1	1	1
Número de trabajo <= 0	0	0	0	0	0	0	0
Fecha de envío <= 0	0	0	0	0	0	0	0
Tiempo de ejecución <= 0	8	5645	45	17	5	11010	8945
Procesadores asignados <= 0	0	0	0	0	0	8896	0
Tiempo estimado <= 0	0	35	0	6	11175	0	0
ID de usuario <= 0	0	0	0	0	0	0	0
Estatus = 0 (Falló)	7482	0	0	16183	19745	0	0
Estatus = 4 (Falló última parte)	0	0	0	0	0	0	0
Estatus = 5 (Cancelado)	0	10824	0	0	0	27802	14455

Tabla VII. Resultados del pre-proceso de las bitácoras del Grid 2 previo a la mezcla

	Número de Bitácora				
Característica	**1,2,3,4,6**	**5**	**7**	**8**	**9**
ID de Usuarios	1343-1630	931-1130	674-930	1-673	1131-1342
Días iniciales borrados	124	8	20	5	0
Trabajos utilizados	344545	10198	17812	33882	23501
Repeticiones de la bitácora	1	1	1	1	1
Número de trabajo <= 0	0	0	0	0	0
Fecha de envío <= 0	0	0	0	0	0
Tiempo de ejecución <= 0	16392	8	45	17	5
Procesadores asignados<=0	0	0	0	0	0
Tiempo estimado <= 0	651	0	0	6	11175
ID de usuario <= 0	0	0	0	0	0
Estatus = 0 (Falló)	1052	7479	0	16183	19744
Estatus = 4(Falló última parte)	0	0	0	0	0
Estatus = 5 (Cancelado)	0	0	0	0	0

La Tabla VIII muestra las características básicas de las mezclas finales para el Grid 1 y para el Grid 2. Se puede notar que el número de usuarios finales en el Grid 2 es menos de la mitad que el número de usuarios en el Grid 1; sin embargo, el número de trabajos casi se triplica en el Grid 2 respecto al Grid 1; esto puede indicar una alta actividad de los usuarios del Grid 2, lo cual puede reflejarse en una mayor utilización de los recursos.

Tabla VIII. Resumen de características de las mezclas para Grid 1 y Grid 2

Grid	#Sitios	#Procesadores	#Trabajos	#Usuarios	Zona horaria	Periodo (días)
1	7	4442	152396	2652	-8	180
2	9	2194	429938	1630	-7	180

Las Figuras 5 a 9 muestran detalles de las cargas de trabajo para el Grid 1 y el Grid 2. La Figura 5 muestra el número de trabajos por semana. La Figura 6 muestra el promedio de recursos consumidos por día del mes. Se puede ver que la demanda de recursos está distribuida en los días del mes en ambos Grids, lo cual difiere si se consideran las bitácoras originales independientemente. También se puede observar que la demanda de recursos es mayor en el Grid 1 a pesar de contar con un número menor de trabajos, lo cual indica que son trabajos muy

largos o con un alto grado de paralelismo. Esto genera dos escenarios diferentes para simulación.

Las Figuras 7 y 8 muestran una predominancia clara de trabajos de bajo grado de paralelismo en ambo Grids. La Figura 7 utiliza una escala logarítmica en el eje x para visualizar mejor el número de trabajos pequeños. El tamaño máximo de un trabajo para el Grid 1 es de 1368 y para el Grid 2 de 1024. En el Grid 1, la mayoría de los trabajos requieren a lo sumo 128 procesadores (95%). 18% de los trabajos requieren 1 procesador; 20% requieren 8 procesadores; 9% requieren 16; 14% requieren 32 y el 8% de los trabajos requieren 64 procesadores. La carga de trabajo en el Grid 2 es menos paralela. El 97% de los trabajos requieren a lo más 32 procesadores. 36% de los trabajos requieren 1 procesador; 26% requieren 2 procesadores y 9% requieren 4 procesadores.

La Figura 9 muestra que los usuarios de HPC2N (IDs de usuario 674-930) y de KTH-SP2 (IDs de usuario 931-1130) que participan en ambos Grids tienen la peor precisión de la estimación del tiempo de ejecución de sus trabajos.

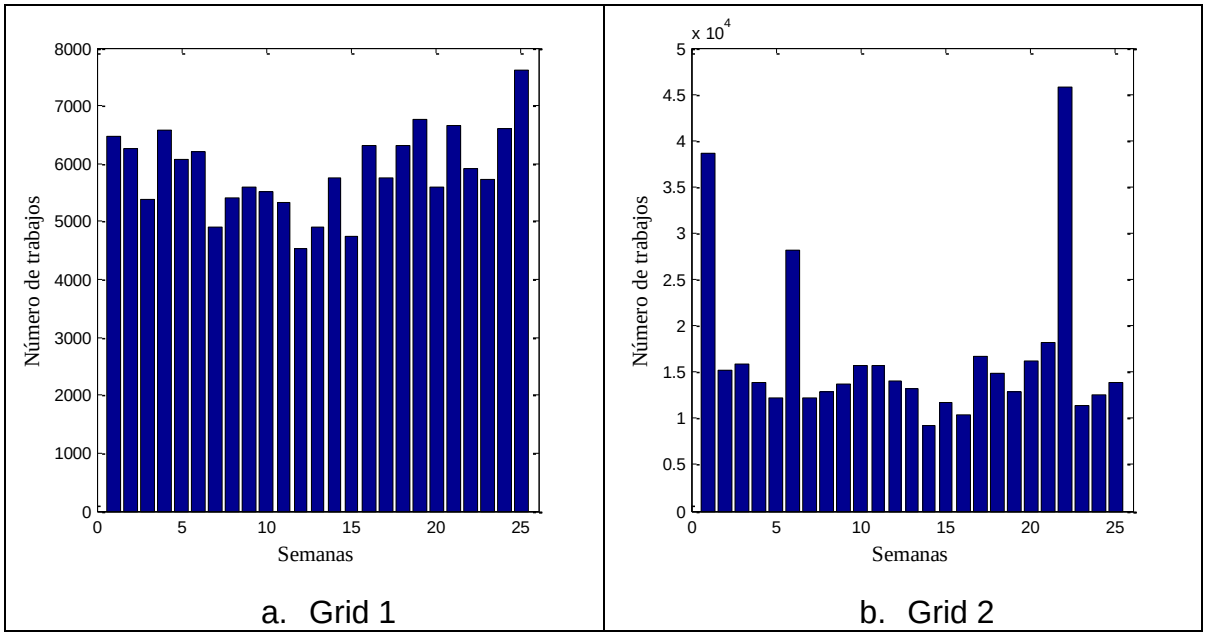


Figura 5. Número de trabajos por cada semana

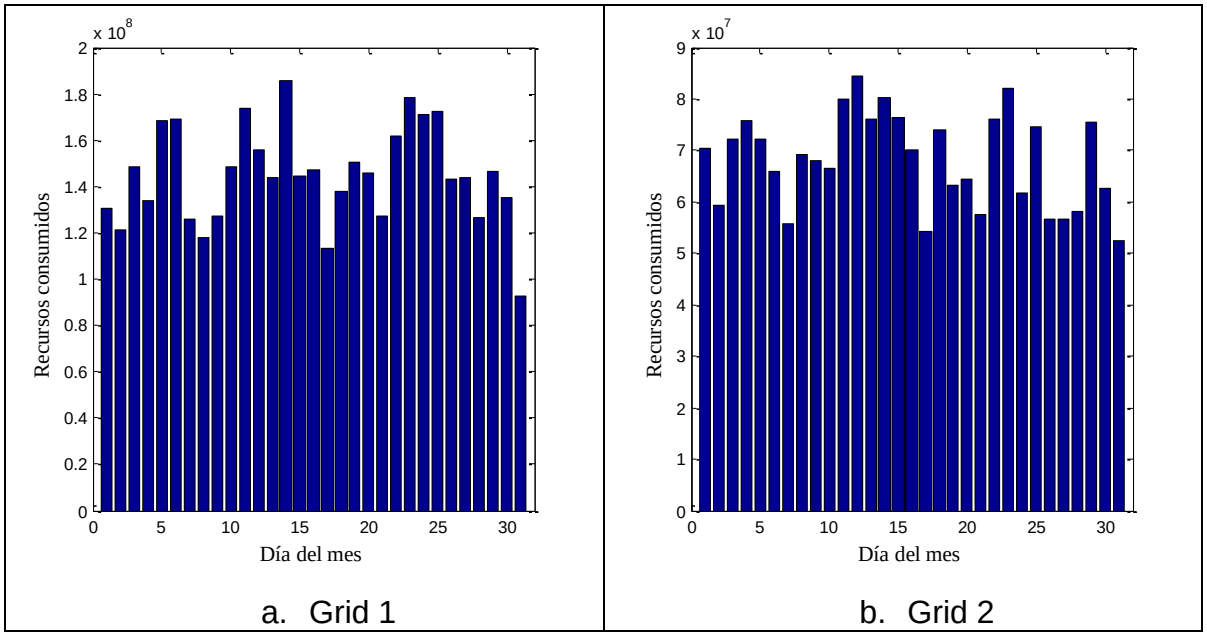


Figura 6. Promedio de recursos consumidos por día del mes

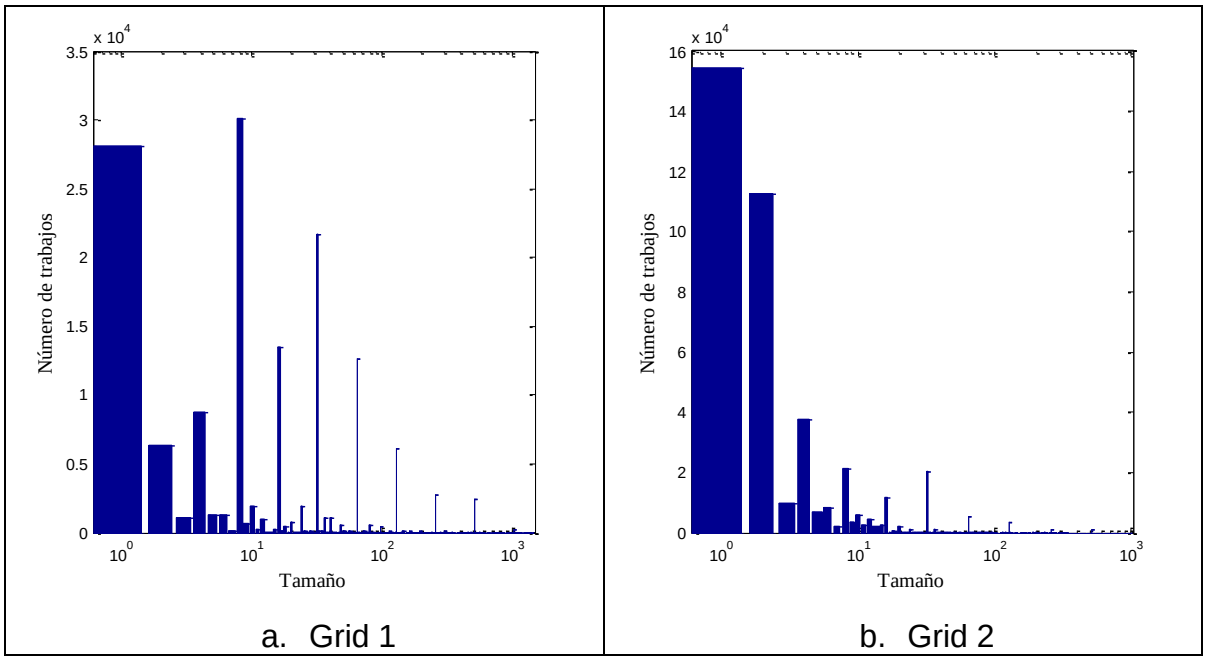
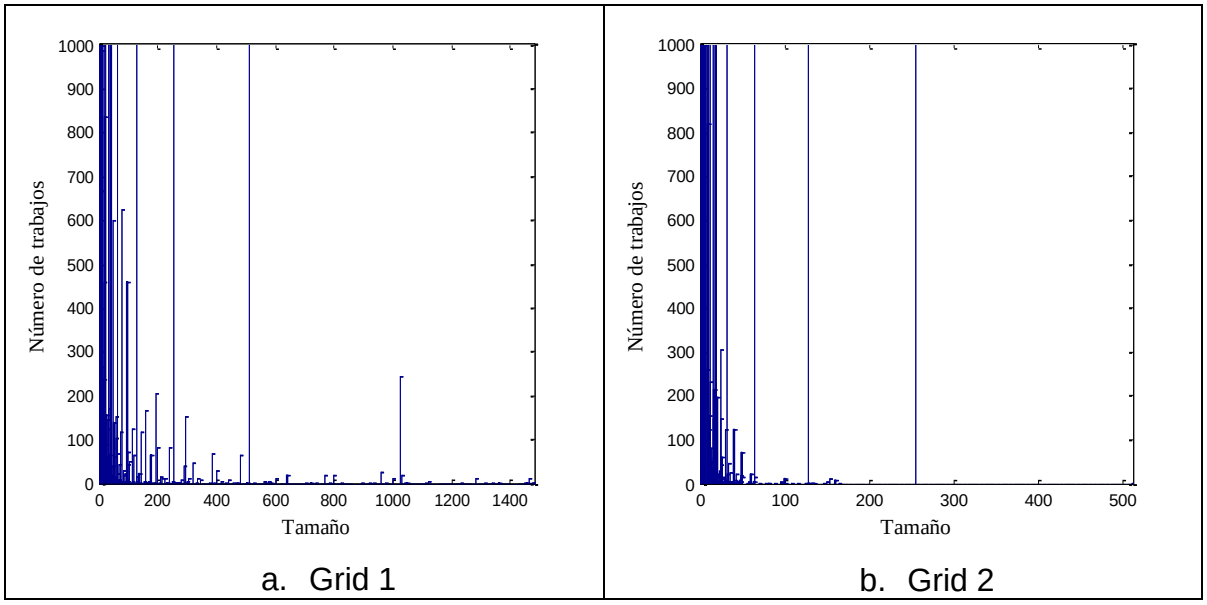
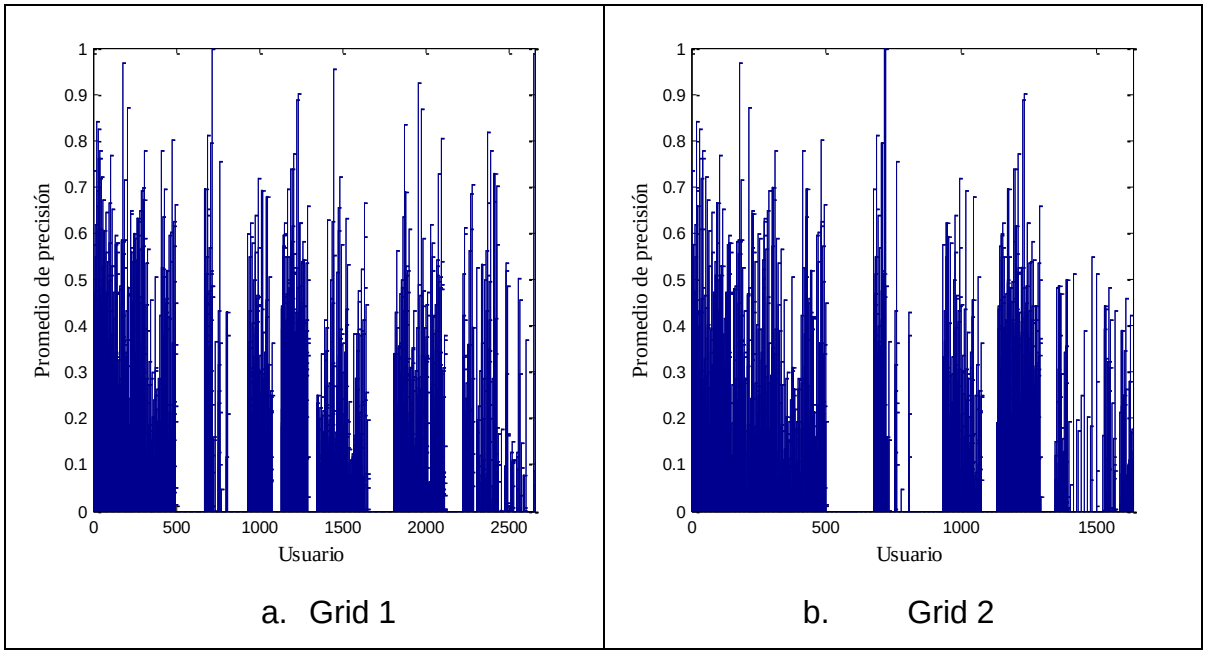


Figura 7. Número de trabajos por tamaño



a. Grid 1 b. Grid 2

Figura 8. Número de trabajos por tamaño (vista limitada a 1000 trabajos)



a. Grid 1 b. Grid 2

Figura 9. Histogramas de precisión de tiempos de ejecución estimados por el usuario

Capítulo VI

Análisis experimental

Este capítulo presenta y analiza los resultados de simulación de las estrategias de asignación de trabajos utilizando las métricas descritas en la Sección II.2. Los detalles de la configuración de Grids y las cargas de trabajos utilizadas en los experimentos se describen en la Sección V.2. Para llevar a cabo los experimentos se desarrolló el simulador de eventos discretos GSimula (Grid Simulator. Simulador de Grid) (ver Apéndice A.6).

VI.1 Método de evaluación

Dado que el problema que se aborda es un problema muti-objetivo en su formulación general, se consideraron diversos criterios para su evaluación. Un buen algoritmo de calendarización debería calendarizar trabajos para lograr un alto rendimiento del Grid mientras satisface diversas demandas. Por lo regular, los proveedores de recursos y usuarios tienen diferentes objetivos de rendimiento y comúnmente éstos están en conflicto, por ejemplo, los proveedores buscarían optimizar la utilización de recursos mientras que a los usuarios les interesaría minimizar el tiempo de respuesta. La administración de recursos del Grid puede usar soporte para decisiones multi-criterio. Una opción es la metodología general de decisión multi-criterio basada en la optimalidad de Pareto. Sin embargo, es muy difícil lograr soluciones rápidas para la administración de recursos del Grid por medio de la dominancia de Pareto. El problema es, a menudo, simplificado a un problema mono-objetivo o a diferentes métodos de combinación de objetivos. Hay

varias maneras para modelar preferencias, por ejemplo, dichas preferencias pueden ser dadas explícitamente por los interesados para especificar la importancia de cada criterio o una importancia relativa entre criterios. Esto puede llevarse a cabo mediante la definición de pesos de criterios o por clasificación de criterios según su importancia.

A fin de proveer una guía efectiva en la elección de la mejor estrategia, se realizó un análisis conjunto de varias métricas de acuerdo a la metodología propuesta en (Tsafrir *et al*., 2007) y aplicada al problema de calendarización en Grid por Ramírez-Alcaraz *et al*. (2011). Ellos introducen un método para análisis multi-criterio dando igual importancia para cada métrica. El objetivo es encontrar una estrategia de buen desempeño bajo todos los casos de prueba, con la expectativa que también tendrá buen desempeño bajo otras condiciones, por ejemplo, con diferentes configuraciones de Grid y cargas de trabajo.

El análisis se lleva a cabo como sigue. Primero, se evalua la degradación (error relativo) en rendimiento de cada estrategia en cada métrica. Esto se hace en relación a la estrategia de mejor rendimiento para cada métrica utilizando la fórmula $\left(\frac{strategy\ metric}{best\ metric} - 1\right) \cdot 100$. De esta manera, para las tres métricas principales (ver Sección II.2), cada estrategia se caracteriza por 3 números (6 para 2 Grids), reflejando su degradación de rendimiento relativo bajo los casos de prueba. Después se promedian estos 3 valores (dando igual importancia a cada métrica), y se ordenan las estrategias de cada Grid. La mejor estrategia, con el promedio de degradación de rendimiento más bajo, tiene rango 1; la peor estrategia tiene rango 14.

Como siguiente paso, se calcula el promedio de degradación de rendimiento del Grid 1 y del Grid 2. El objetivo es identificar estrategias que se desempeñan confiablemente bien en diferentes escenarios; esto es, encontrar estrategias con una posición intermedia considerando todos los casos de prueba. Por ejemplo, la

posición que ocupa la estrategia según su promedio de degradación de rendimiento podría no ser la misma para cualquiera de las métricas o escenarios de Grid si se consideran individualmente.

VI.2 Resultados de la simulación

Las Tablas XI a XIX en el Apéndice A.4 muestran los resultados de la simulación amplia y extensa de las 14 estrategias con el algoritmo de calendarización local BEFF (Backfilling EASY First Fit) con 24 métricas, todos los casos de prueba y el promedio de los escenarios Grid 1 y Grid 2. También se presentan los porcentajes redondeados de las degradaciones de rendimiento para cada estrategia y métrica. El valor 0 (cero) muestra la mejor estrategia en cada métrica. Los resultados se obtienen bajo diferentes condiciones y casos de prueba. Las condiciones en Grid 1 se caracterizan por 7 grandes sitios con 634 procesadores por sitio en promedio, y 21770 trabajos por sitio en promedio. Condiciones en Grid 2 son caracterizadas por 9 sitios de tamaño más pequeño con 243 procesadores por sitio en promedio, y 47770 trabajos por sitio en promedio.

Para un análisis más detallado, la Tabla IX muestra la degradación de rendimiento de las 14 estrategias para las 3 métricas consideradas en los escenarios Grid 1 y Grid 2. Para las estrategias que fueron comparadas, los tiempos de espera, la ralentización, y el total de tiempos de finalización ponderados varían en gran medida. La diferencia entre las estrategias es más de 136,000% para t_w, 75,000% para SD_b, y 28% para $SWCT_w$. Estas grandes desviaciones muestran que estas métricas son muy sensibles a las estrategias de asignación. Es necesaria una selección cuidadosa de estas métricas cuando se aplican a sistemas reales.

Las Figuras 10 a 12 muestran la clasificación de las seis mejores estrategias de acuerdo a la degradación del rendimiento para t_w, SD_b, y $SWCT_w$. La Figura 13

muestra las degradaciones promedio de las estrategias en Grid 1 y Grid 2 considerando los promedios de todas las métricas. La Figura 14 muestra la clasificación de la degradación de rendimiento promedio para todos los casos de prueba. Los Apéndices A.1-A.3 muestran gráficas más detalladas.

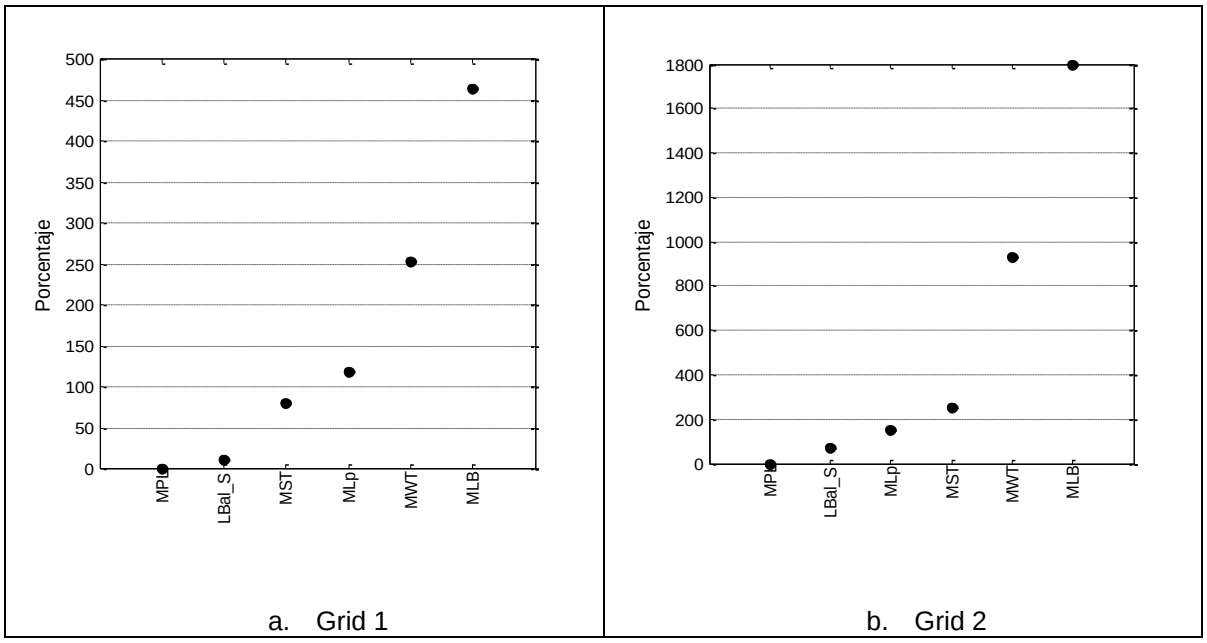
Tabla IX. Porcentajes de la degradación de rendimiento de las estrategias para Grid 1 y Grid 2

Metric / Strategy	Grid	t_w	SD_b	$SWCT_w$	Mean
MCT	1	1256	1284	0	847
	2	4833	4001	0	2945
MLB	1	463	408	0	290
	2	1799	754	0	851
LBal_S	1	11	18	0	10
	2	70	17	0	29
LBal_T	1	136322	75442	28	70597
	2	4205	2569	0	2258
LBal_W	1	1006	899	0	635
	2	5743	3346	0	3030
MLp	1	119	71	0	63
	2	153	53	0	69
MPL	1	0	0	0	0
	2	0	0	0	0
Random	1	51929	41924	3	31285
	2	18027	12196	0	10074
MST	1	80	91	0	57
	2	252	111	0	121
MSWCT_W	1	70233	47821	2	39352
	2	6515	3858	0	3458
MWWT_S	1	17303	12576	1	9960
	2	9132	7550	0	5561
MWT	1	252	204	0	152
	2	933	628	0	520
MWWT_T	1	8073	7382	1	5152
	2	30822	27638	0	19487
MWWT_W	1	15506	12889	1	9465
	2	51768	68410	0	40059

Como se esperaba, *MCT* y *MST* son más eficientes que otras estrategias para minimizar el criterio C_{max} para ambos Grids. Más aún, para estos casos de prueba, las diferencias entre *MLB*, *LBal_S*, *MLp*, *MPL*, y *MWT* son insignificantes. Ellas se desempeñan tan bien como *MCT* y *MST*.

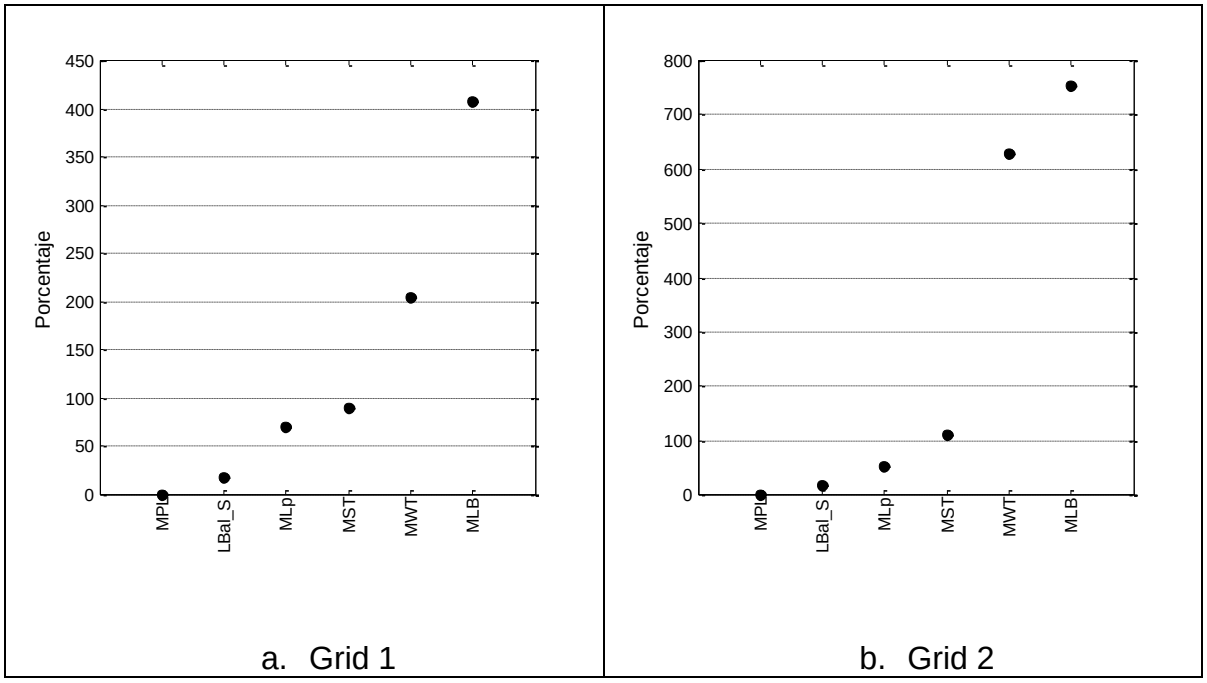
La diferencia entre estrategias bajo la métrica de suma de tiempos de finalización ponderados es menos de 0.04%. Esto es probablemente debido a que en el escenario en línea la carga puede ser ligera al final del periodo de simulación (después de la llegada del último trabajo); por lo tanto, el tiempo de finalización de los últimos trabajos puede determinar el tiempo de finalización del calendario.

Observando las Figuras 10 a 12, se encuentra que las estrategias de asignación *MPL* y *LBal_S*, que toman en cuenta solamente los tamaños de los trabajos, se desempeñan mejor que las estrategias que utilizan las métricas de tiempo de espera, ralentización acotada, y tiempos de finalización ponderados. La diferencia en el rendimiento relativo es de 19% considerando el promedio de todas las métricas.



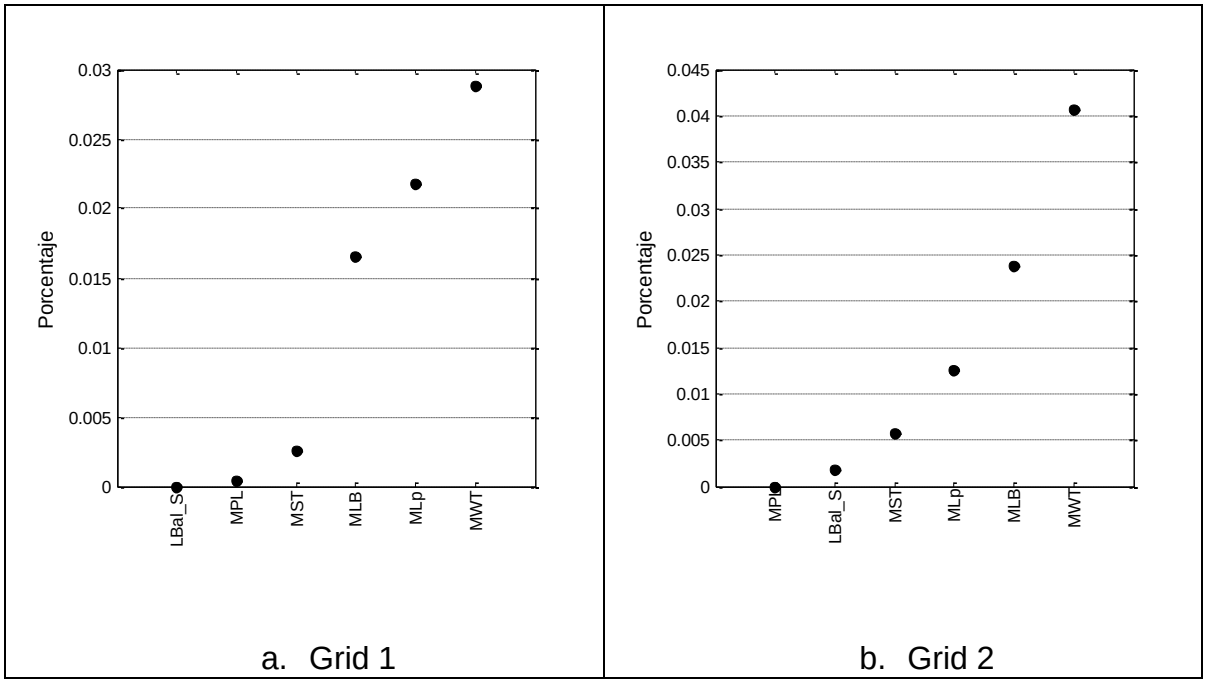
a. Grid 1 b. Grid 2

Figura 10. Degradación en t_w. Seis mejores estrategias.



a. Grid 1 b. Grid 2

Figura 11. Degradación en SD_b. Seis mejores estrategias



a. Grid 1 b. Grid 2

Figura 12. Degradación en $SWCT_w$. Seis mejores estrategias

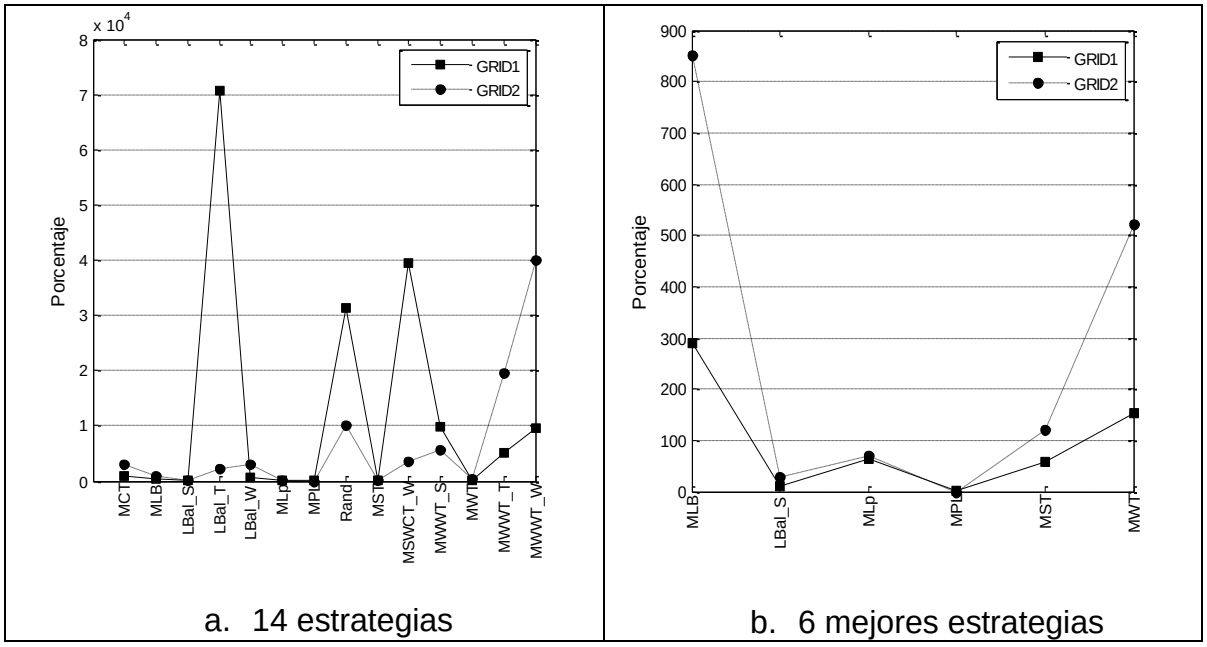


Figura 13. Degradación promedio de las estrategias de asignación en Grid 1 y Grid 2

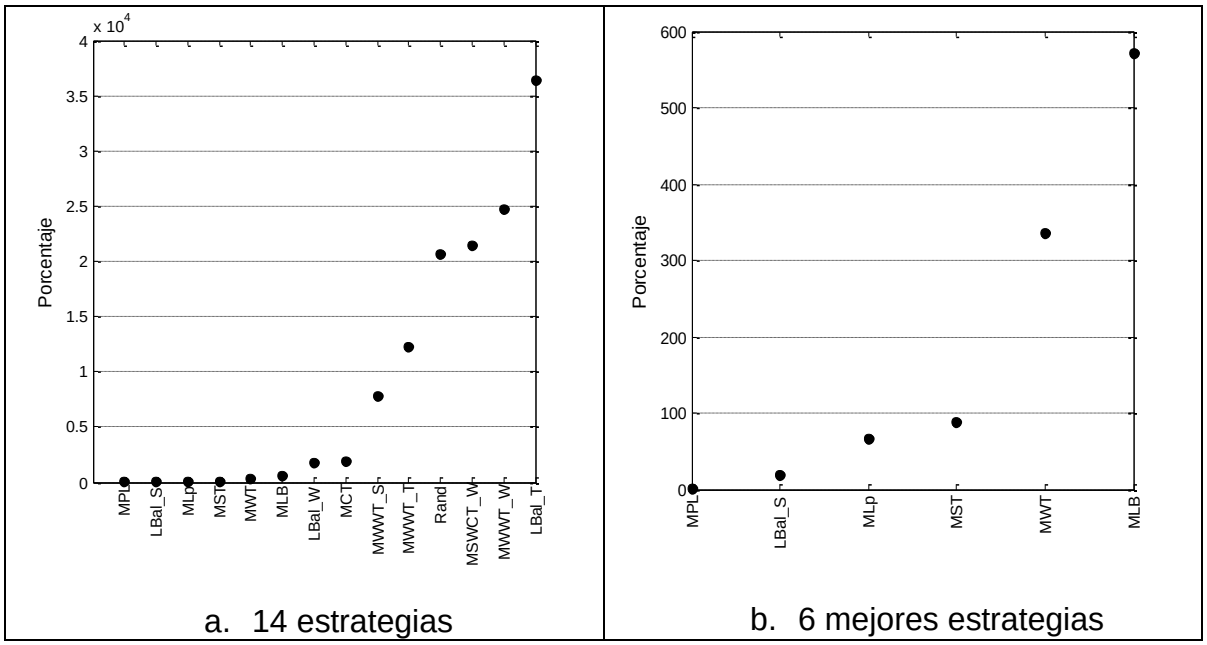


Figura 14. Clasificación de la degradación promedio de todos los casos de prueba

Se observa el mismo resultado sobre un amplio rango de parámetros de simulación. El hecho de contar con más información disponible en el momento de la asignación, tal como la estimación del tiempo de ejecución del trabajo y

calendarios locales, no ayuda en la construcción de mejores calendarios para el Grid. Una de las razones es que las estimaciones de tiempo por los usuarios son imprecisas. Las heurísticas *MLp* y *MST* también se desempeñan bien para los casos considerados, dando los segundos mejores resultados. Estas estrategias son 69.67% y 90.43% peores que *MPL* considerando el promedio de todas las métricas.

Los algoritmos *MWT* y *MLB* son más de 300% peores que *MPL* cuando consideramos el promedio de todas las métricas.

El uso de la función de ponderado en las estrategias de asignación (*LBal_T*, *LBal_W*, *MWWT_T*, *MWWT_W*, *MWWT_S*, *MSWCT_W*) con tiempo de ejecución T=p'_j y recursos consumidos W= w_j como el peso, muestra un decremento en el rendimiento como se espera para métricas sin ponderación. En contraste a *MPL* y *LBal_S*, éstas siempre tienen un pobre desempeño, clasificándose solamente en las posiciones 9 a 14.

Los resultados indican que el rendimiento relativo promedio de las estrategias de asignación *MPL* y *LBal_S* no dependen significativamente de la carga de trabajo utilizada, la configuración de Grid ni de la métrica de rendimiento. Como resultado se tiene que *MPL* y *LBal_S* son los algoritmos de mejor desempeño. Además del aspecto del rendimiento, el uso de *MPL* y *LBal_S* no requiere una carga administrativa adicional tal como la solicitud de información acerca de los calendarios locales o la construcción de calendarios preliminares en el Broker.

De la Tabla IX y la Tabla XIII (Apéndice A.4), se puede ver que los resultados varían en gran medida. Sin embargo, esto indica claramente que *MPL* y *LBal_S* son capaces de lograr un mejor desempeño que otros algoritmos en casi todos los casos de prueba. Cuando se consideran los criterios de desviación de tamaños, tiempo y recursos consumidos (*LBal_S*, *LBal_T*, *LBal_W*), existe un pequeño

número de otras asignaciones eficientes (como *MST*). Se concluye que las estrategias *MPL* y *LBal_S* son estables aún en condiciones significativamente diferentes. *MLp* y *MST* también proveen una degradación de rendimiento menor y son capaces de afrontar diferentes demandas.

Los resultados obtenidos ayudan a seleccionar la estrategia apropiada para Grids reales que dependen de las preferencias de diferentes tomadores de decisiones de Grids (usuarios finales, proveedores de recursos locales, y administradores de Grid), y de la importancia de cada criterio o una importancia relativa entre criterios.

VI.3 Estrategias de asignación con predicción del tiempo de ejecución generado por el sistema

En esta sección se presentan resultados del análisis experimental llevado a cabo sobre el impacto en el rendimiento general del Grid, debido a la incorporación de predicciones de tiempos de ejecución generadas por el sistema en las estrategias de asignación *MST* y *MWT*.

Primero, se consideran los modelos de predicción generada por el sistema con base en el historial de ejecución. Ésta predicción consiste en calcular una mejor estimación del tiempo de ejecución de un trabajo recién llegado al sistema, en base al tiempo de ejecución real de trabajos terminados previamente, propiedad del mismo usuario. Para cada estrategia de asignación, se calculan tres métricas con variación del tamaño de ventana de historial de 1 a 10 para cada uno de los 8 modelos de predicción (Tabla X). Después, se evalúa la degradación de rendimiento de cada modelo de predicción y cada tamaño de ventana bajo cada métrica. Esto se hace en relación al caso ideal con la precisión de la predicción generada por el sistema, igual a 1 ($p''_j = p'_j$). De esta manera, cada modelo de

predicción se caracteriza por 30 parámetros, reflejando su degradación de rendimiento relativo bajo los casos de prueba.

Tabla X. Modelos de predicción generada por el sistema

Modelo de predicción	Nombre	Descripción
Hrecent_k	*M1*	k trabajos recientes.
Hrecent_k_p'	*M2*	k trabajos recientes con la misma estimación de tiempo de ejecución ($p'_i = p'_j$).
Hrecent_k_size	*M3*	k trabajos recientes con el mismo tamaño ($size_i = size_j$).
Hrecent_k_size_& p'	*M4*	k trabajos recientes con $p'_i = p'_j$ y $size_i = size_j$.
Hk	*M5*	A lo más k trabajos.
Hk_p'	*M6*	A lo más k trabajos con la misma estimación de tiempo de ejecución $p'_i = p'_j$.
Hk_size	*M7*	A lo más k trabajos con el mismo tamaño ($size_i = size_j$).
Hk_size& p'	*M8*	A lo más k trabajos con $p'_i = p'_j$ y $size_i = size_j$.
C1-model	*M9*	(*C*onstante) La predicción del sistema (p''_j) se calcula multiplicando el tiempo de ejecución estimado (p'_j) por una constante $p''_j = c_1 \cdot p'_j$, donde c_1 es el promedio de precisión en la bitácora disponible.
C2-model	*M10*	(*C*onstante) La predicción del sistema (p''_j) se calcula multiplicando el tiempo de ejecución estimado (p'_j) por una
		constante $p''_j = c_2 \cdot p'_j$, donde c_2 es una constante predefinida para todos los valores en [0.01, 0.02, …, 0.99, 1] (desde 1% a 100% del tiempo estimado).
U1-model	*M11*	(Distribución *U*niforme) La predicción del sistema está distribuida uniformemente en $p''_j = U[L_1, R_1] \cdot p'_j$ donde $L_1 = \min_j (p_j/p'_j)$, y $R_1 = \max_j (p_j/p'_j)$ en la bitácora disponible. La estimación exacta del tiempo de ejecución corresponde a $L_1 = 1$ y$R_1 = 1$.
U2-model	*M12*	(Distribución *U*niforme) La predicción del sistema está distribuida uniformemente en $p''_j = U[L_2, R_2] \cdot p'_j$, donde L_2 es el promedio de la mitad de valores más pequeños de p_j/p'_j en el log disponible y R_2 es el promedio de la mitad de valores más grandes de p_j/p'_j en la bitácora. La estimación exacta del tiempo de ejecución corresponde a $L_2 = 1$ y $R_2 = 1$.

Los modelos históricos utilizados están basados en los modelos propuestos por (Tsafrir *et al.*, 2007). Los autores utilizan un parámetro designado como *tipo de ventana*, con tres configuraciones: *todos, inmediatos, extendidos.* En el primer caso, todos los trabajos recientemente ejecutados por el mismo usuario son elegibles; en el segundo caso, los trabajos recientes son utilizados si ellos son *similares* al nuevo trabajo, es decir tienen la misma estimación de tiempo de ejecución; en el tercer caso, se buscan trabajos similares en todo el historial del usuario. Sus conclusiones indican que es mejor utilizar el tipo de ventana *todos.*

Los autores analizaron también el parámetro: *métrica de predicción*, es decir, que operación debería de aplicarse al conjunto de trabajos históricos considerados. Ellos aplicaron *promedio, mediana, mínimo* y *máximo*. Los resultados muestran que se tienen mejores resultados cuando se utiliza el promedio.

El *tamaño de ventana* (número de trabajos históricos considerados) que ellos utilizaron fue de 1 a 30. Sin embargo, los autores encontraron mejores resultados con un tamaño de ventana pequeño (<7).

En la presente investigación, se implementaron los modelos que utilizan los *tipos de ventana todos* e *inmediatos*, referidos aquí como *Hrecent_k* y *Hrecent_k_p'* respectivamente, en donde *k* corresponde al *tamaño de ventana*. Se extendió el concepto de *similaridad* para considerar el tamaño del trabajo, es decir, los trabajos históricos son utilizados si tienen el mismo tamaño que el nuevo trabajo (*Hrecent_k_size*); también se implementó el modelo *Hrecent_k_size&p'*, que cosidera similares a los trabajos históricos si tienen el mismo tamaño y el mismo tiempo de ejecución que el nuevo trabajo. Estos cuatro modelos tienen la característica de considerar exactamente *k* trabajos históricos, si no hay suficientes trabajos que satisfagan las condiciones, no se calcula el promedio y en su lugar se toma como nuevo tiempo estimado el tiempo estimado originalmente por el usuario, es decir $p_j'' = p_j'$.

Derivado de esto, se implementaron cuatro modelos más: *Hk*, *Hk_p'*, *Hk_size*, *Hk_size&p'*, los cuales, a diferencia de los modelos *Hrecent*, calculan el promedio de tiempos de ejecución aún cuando no existan *k* trabajos históricos o que aun existiendo, no todos sean similares. Por ejemplo, para $k = 3$, si solamente están disponibles 2 trabajos en el historial del usuario, estos 2 trabajos se utilizan para calcular el promedio, siempre y cuando cumplan las condiciones (ej. $size_i = size_j$); si solamente uno de los dos trabajos (i) cumple la condición entonces $p_j'' = p_i$.

Con la misma finalidad de mejorar la predicción de los tiempos de ejecución, y basados en la idea de modelos estadísticos como los utilizados en (Mu'alem y Feitelson, 2001), se implementaron los modelos *C1*, *C2*, *U1*, y *U2*. *C1* y *C2*, calculan la predicción multiplicando el tiempo estimado originalmente, por una constante predefinida, mientras que los modelos *U1* y *U2*, toman como predicción un valor aleatorio uniforme dentro de un intervalo definido por la relación p_j/p'_j (ver Tabla X). En estos cuatro modelos, las constantes son calculadas previamente basadas en el análisis estadístico de bitácoras anteriores.

Como se mencionó en párrafos anteriores, en este trabajo, la predicción se calcula como el promedio del tiempo de ejecución de los trabajos precedentes seleccionados. Para un trabajo recién enviado j, el promedio del tiempo de ejecución de los trabajos precedentes i, del mismo usuario, ya finalizados, se calcula como $p_j'' = 1/k \sum_i^k p_i$, donde k es el tamaño de la ventana de historial la cual determina el conjunto de trabajos previos utilizados para predicción.

Las Figuras 15 y 16 muestran la degradación promedio de *MST* y *MWT* sobre el promedio de tres métricas variando el tamaño de la ventana de historial para los dos escenarios Grid 1 y Grid 2. La Figura 17 presenta la degradación promedio del

Grid 1 y Grid 2 para todos los casos de prueba. En todos los casos se presentan los resultados de los primeros cuatro modelos de predicción del tiempo de ejecución descritos en la Tabla X, también se muestran en conjunto los resultados obtenidos cuando utilizamos la estimación del tiempo de ejecución por el usuario, y el tiempo de ejecución real del trabajo (simulando estimación perfecta), este último mostrado como punto de referencia.

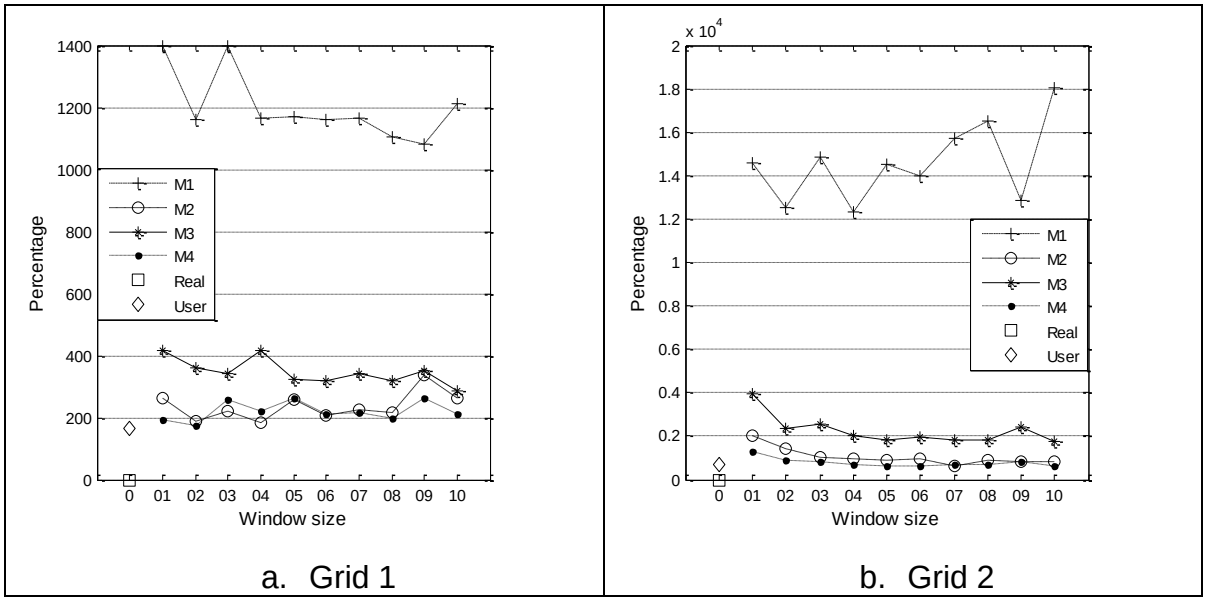


Figura 15. Degradación promedio contra el tamaño de ventana del historial. Estrategia de asignación *MST*

Para *MST* (Figura 15), se observa que *Hrecent_k_p'*, y *Hrecent_k_size&p'* muestran los mejores resultados en ambos Grids con pequeñas desviaciones de la degradación del desempeño. Para *MWT* (Figura 16), no hay dominación de un modelo específico. Sin embargo, un tamaño grande de ventana de historial (8-10) mejora los resultados del modelo *Hrecent_k_size&p'* que considera el historial de los trabajos más similares.

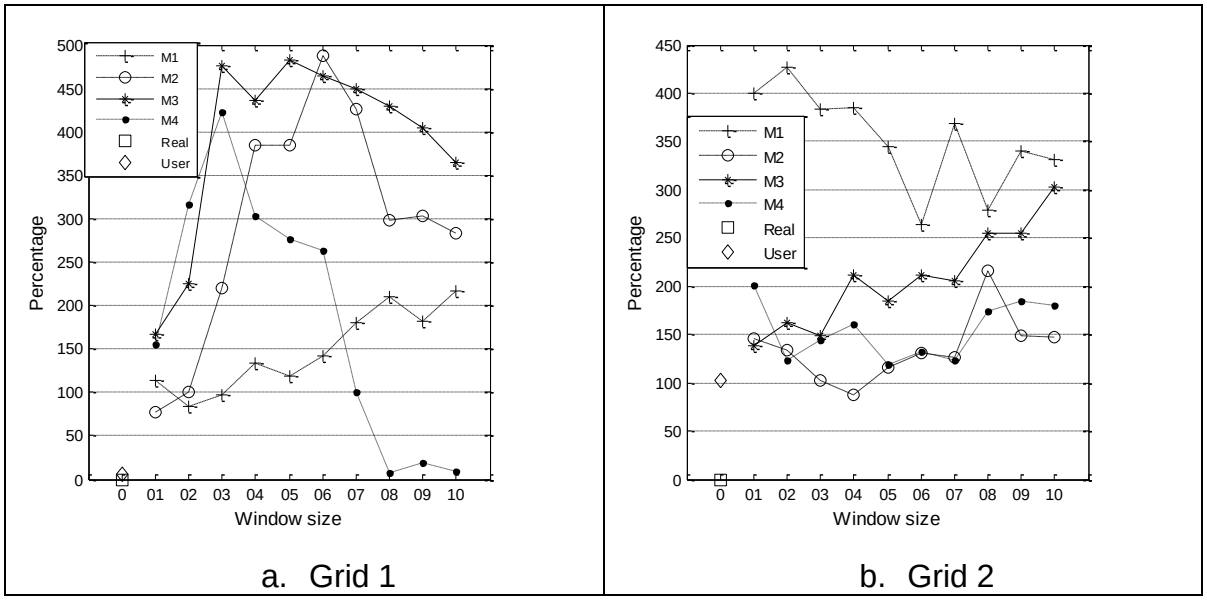


Figura 16. Degradación promedio contra el tamaño de ventana del historial. Estrategia de asignación *MWT*

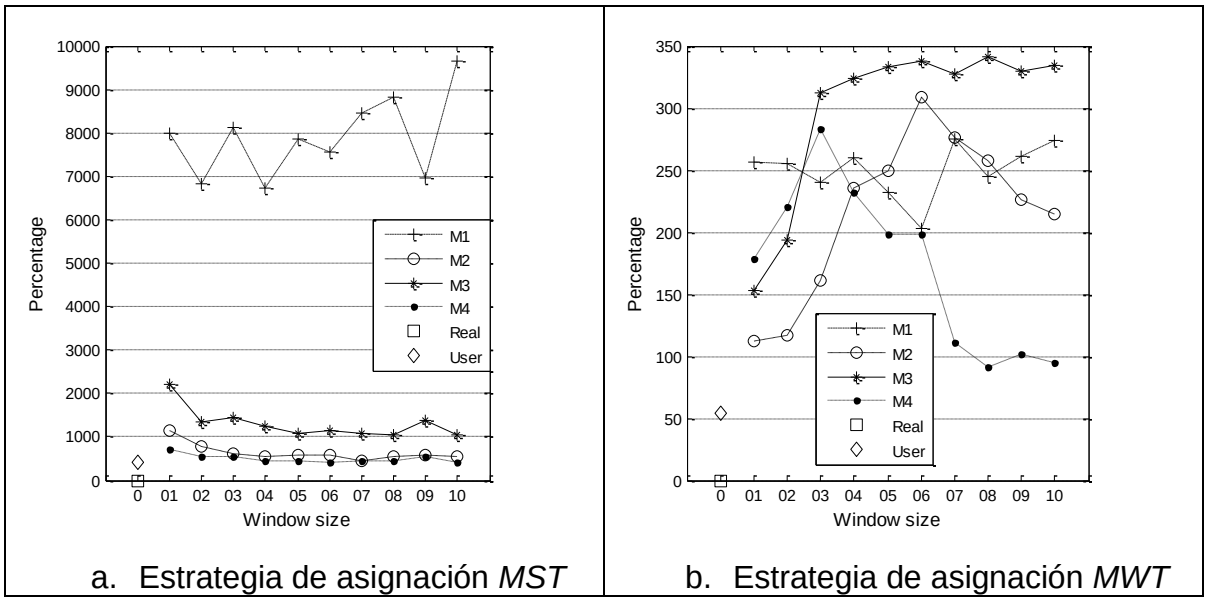


Figura 17. Degradación promedio de Grid y Grid 2 para todos los casos de prueba

Los resultados de la simulación indican claramente que es mejor utilizar los trabajos más similares, caracterizados por el mismo tiempo estimado y el tamaño (*size y p'*) dado que estos modelos dominan en todos los casos de prueba. El

incremento en el tamaño de la ventana de historial mejora ligeramente los resultados. Esto probablemente se debe a que utilizando una ventana más grande hay más oportunidad de encontrar trabajos similares. Sin embargo, no hay dominio de un tamaño de ventana específico. Usar un promedio sobre el último y hasta los diez trabajos precedentes sin condiciones (*Hrecent_k , M1*), muestra los peores resultados prácticamente en todos los casos. La Figura 17 indica que no hay mejora en el rendimiento de las estrategias de asignación *MST* y *MWT* con predicción generada por sistema basada en historial sobre la estrategia que usa los tiempos de ejecución estimados por el usuario. Esta conclusión cualitativa no depende significativamente de la carga de trabajo utilizada, ni de la métrica de rendimiento. Como resultado de esto se tiene que, la incorporación de los modelos de predicción generada por sistema basada en historial, en las políticas de asignación de trabajos y algoritmos de calendarización local, no proveen tan buenos resultados en sistemas con múltiples sitios paralelos como para políticas de calendarización con relleno en sistemas de un solo sitio. Se consideran predicciones generadas por sistema que se calculan una vez al momento de la llegada del trabajo al sistema con la información histórica disponible. Otra observación es que los trabajos asignados a un sitio específico podría reducir la precisión de las predicciones generadas por sistema para relleno porque los trabajos del mismo usuario se pueden asignar a diferentes sitios. Por ejemplo, en el escenario con sitios de Grid de diferentes tamaños, la buena asignación de trabajos no crea situaciones en la que grandes máquinas se ocupen por trabajos con pequeños requerimientos de procesadores, que ocasionarían que trabajos altamente paralelos tengan que esperar para su ejecución (Tchernykh *et al.*, 2010). Sin embargo, trabajos con pequeños requerimientos de procesadores y tiempos de ejecución estimados por el usuario son necesarios para un rellenado efectivo. La distribución de trabajos en un modelo de calendarización en línea para Grid jerárquico de dos niveles podría también limitar el rellenado en los recursos locales. Cada sitio local tiene un número de trabajos reducido comparado con el total de trabajos en Grid por lo que el desempeño del algoritmo *Rellenado* en este

sitio es menor comparando con resultados obtenidos para una máquina en (Tchernykh *et al.*, 2008). Esto se limita a nuestros escenarios de Grid: si se considera una pequeña cantidad de sitios, la predicción generada por el sistema produce resultados con mejor desempeño. Usando un orden de rellenado SJBF (shortest job backfilled first, primero rellenar con trabajos más cortos) en los calendarizadores locales podría generar una mejora en el rendimiento (Tchernykh *et al.*, 2008).

También se analizaron los modelos que están basados en la premisa de que las bitácoras previas están disponibles y se pueden analizar para aprender y mejorar los parámetros de predicción (*C1_model*, *C2_model*, *U1_model*, *U2_model*) (Tabla X). Los resultados de los modelos *Hk*, *C1*, *C2*, *U1*, y *U2* algunas veces muestran un rendimiento comparable o un poco mejor que los modelos *Hrecent*, pero no se pueden utilizar para obtener un rendimiento superior de manera consistente (Apéndice A.5).

Para elaboración de las gráficas presentadas en este trabajo se tomó como base el programa *gsimula_stats*, desarrollado en Matlab por José Luis González García como parte de su trabajo de tesis (García, 2009).

Conclusiones

Debido a que los Grids se están volviendo cada vez más habituales, las técnicas para utilizar eficientemente sus recursos se han incrementado significativamente. El problema de asignación de recursos y calendarización es crucial no solamente para lograr un alto rendimiento del Grid, sino también para satisfacer varias de las demandas de los usuarios de una manera equitativa. Mientras que los modelos de calendarización de Grid para el peor caso teórico están empezando a surgir, técnicas estadísticas rápidas aplicadas a datos reales han demostrado empíricamente ser efectivas.

En esta tesis, se aborda la calendarización en línea no clarividente y sin interrupciones de trabajos paralelos en un Grid. Se presenta una evaluación de rendimiento de algoritmos de asignación de trabajos. Este estudio redunda en varias contribuciones. Primeramente, se identifican varios niveles de información disponible para tomar las decisiones de calendarización con respecto a la asignación de trabajos. Se discuten y analizan catorce estrategias, dependiendo del tipo y cantidad de información que ellas requieren, junto con el algoritmo de calendarización local *EASY Backfilling*.

La selección de la estrategia apropiada para Grids depende de las preferencias de diferentes tomadores de decisiones (usuarios finales, proveedores de recursos locales, y administradores de Grid), y de la importancia de cada criterio o una importancia relativa entre criterios. Para proveer una guía efectiva para la elección de una buena estrategia se realiza un análisis conjunto de tres métricas basadas en la degradación en rendimiento de cada estrategia bajo cada métrica.

Los resultados de simulación presentados revelan que en términos de la minimización del tiempo de espera, de la ralentización, del total de tiempos de

finalización total ponderados, y de su promedio de degradación, las estrategias *MPL* y *LBal_S* funcionan mejor que los otros algoritmos. Estas estrategias son capaces de lograr mejor desempeño que otros algoritmos dado que dominan en casi todos los casos de prueba. Se concluye que las estrategias son estables aún en condiciones significativamente diferentes. Las estrategias *MLp* y *MST* también proveen una degradación de rendimiento pequeña y son capaces de tratar con diferentes demandas.

Se encontró que la información acerca de la estimación del tiempo de ejecución y de los calendarios locales no ayuda para mejorar significativamente las estrategias de asignación. Después de examinar el rendimiento global del Grid sobre los datos reales, se determinó que la distribución apropiada de los requerimientos de procesador sobre el Grid logra un desempeño más alto que una asignación de trabajos basada en la estimación del tiempo de ejecución por parte del usuario y de la información sobre los calendarios locales.

El resultado final sugiere calendarizadores simples, los cuales requieren mínima información y poca complejidad computacional, y no obstante, logran mejoras significativas en el rendimiento.

Trabajo futuro

Existen varias líneas en las que se puede extender el presente trabajo. Aquí se describen las más importantes.

En el presente trabajo se considera que los sitios locales utilizan el mismo algoritmo para calendarizar los trabajos que reciben de parte del gestionador de recursos del Grid. Una línea de investigación interesante es considerar que en cada sitio se aplica un calendarizador diferente, lo cual es razonable debido a la naturaleza del Grid, en la que cada sitio es independiente y libre de aplicar los algoritmos que mejor convengan al propietario. En este caso, el gestionador de recursos del Grid no elaboraría un calendario tentativo sino que solicitaría la información del calendario local a cada uno de los sitios.

Otra línea de investigación interesante es considerar las máquinas como máquinas *uniformes*, es decir, considerar que las máquinas tienen diferentes velocidades. En esta investigación se considera que la ejecución de un trabajo se puede llevar a cabo en cualquier máquina siempre y cuando $size_j \leq m_i$, y que p_j es igual en cualquier máquina sin embargo, debido a que las máquinas en un Grid pueden ser heterogéneas, el tiempo de procesamiento para un trabajo puede ser diferente en cada una de ellas. Así, se puede dar prioridad a una máquina con más velocidad que las demás aun cuando tenga un mayor número de trabajos en espera.

Dado que las características de conectividad de los sitios que conforman un Grid pueden ser diferentes, resulta importante considerar la topología del Grid. Esto implica la introducción de un parámetro que represente el ancho de banda de la conexión de cada sitio con el gestionador de recursos del Grid. Con este nuevo parámetro, la distribución de los trabajos podría dar preferencia a sitios cuya

conexión tiene un ancho de banda mayor. Un aspecto relevante en este mismo contexto es la cantidad de datos que un trabajo necesita como entrada para realizar su ejecución. Puede darse el caso que el gestionador de recursos del Grid elija un sitio porque ofrece el menor tiempo de espera, sin embargo, si el sitio tiene una conexión con un ancho de banda pequeño y el trabajo requiere una cantidad de datos muy grande, enviar el trabajo a dicho sitio puede tarde horas o incluso días, lo cual no es eficiente. En este caso es mejor optar por un sitio que, aun cuando ofrezca mayor tiempo de espera, tenga una conexión con un ancho de banda mayor.

Un tema muy importante en la actualidad es el de la conservación del medio ambiente, especialmente el ahorro de energía. En general, los sitios que componen un Grid implican recursos que consumen gran cantidad de energía eléctrica debido al elevado número de procesadores que regularmente están disponibles todo el tiempo. En este contexto, es importante considerar estrategias que provean una mejor administración de los recursos del Grid, con el objetivo de disminuir el consumo de energía. Una idea básica para reducir el consumo de energía en Grids es apagar los recursos que no estén siendo utilizados. En este sentido, la presente investigación puede extenderse agregando nuevos parámetros que consideren el consumo de energía basados en el número de procesadores que estén encendidos. Un escenario factible consiste en considerar una cota de tolerancia máxima del tiempo de espera combinado con una medida del costo de encender nuevos equipos. En este escenario, se puede decidir que una máquina no encienda nuevos recursos mientras no se exceda la cota máxima de tiempo de espera para los trabajos asignados. Esto significa que si un trabajo puede esperar 12 hrs. no es necesario encender nuevos recursos para ejecutarlo si antes de ese tiempo habrá recursos disponibles para ejecutarlo y finalizarlo.

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. Diciembre 14-16.

Abramson, D., Giddy, J., y Kotler, L. (2000). *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?* International Parallel and Distributed Processing Symposium (IPDPS 2000), Cancun, México. Mayo 1-5.

Anoep, S., Dumitrescu, C., Dick, E., Iosup, A., Jan, M., Li, H., y Wolters, L. (2007). Grid Workloads Archive. Recuperado Abril, 2011, de http://gwa.ewi.tudelft.nl

Avellino, G., Beco, S., Cantalupo, B., Maraschini, A., Pacini, F., Terracina, A., Barale, S., Guarise, A., Werbrouck, A., Sezione Di Torino, Colling, D., Giacomini, F., Ronchieri, E., Gianelle, A., Peluso, R., Sgaravatto, M., Mezzadri, M., Prelz, F., y Salconi, L. (2003). *The EU DataGrid Workload Management System: towards the second major release.* 2003 Conference for Computing in High Energy and Nuclear Physics, University of California, La Jolla. California, USA. Marzo 24-28.

Bailey Lee, C., Schwartzman, Y., Hardy, J., y Snavely, A. (2004). *Are user runtime estimates inherently inaccurate?* Job Scheduling Strategies for Parallel Processing, Columbia University, New York, NY. Junio13.

Bar-Noy, A., y Freund, A. (2001). On-Line Load Balancing in a Hierarchical Server Topology. *SIAM Journal of Computing, 31*(2): 527-549.

Berman, F., y Hey, T. (2004). The Scientific Imperative. En: I. Foster y C. Kesselman (Eds.), *The Grid 2. Blueprint for a New Computing Infrastructure*. Morgan Kaufmann. San Francisco, CA, USA. (2 ed. pp. 13-24).

Berten, V., Goossens, J., y Jeannot, E. (2006). On the Distribution of Secuencial Jobs in Random Brokering for Heterogeneous Computational Grids. *IEEE Transactions on Parallel and Distributed Systems, 17*(2): 113-124.

Bougeret, M., Dutot, P.-F., Jansen, K., Otte, C., y Trystram, D. (2010a). *A Fast 5/2 Approximation Algorithm for Hierarchical Scheduling.* 16th International European Conference on Parallel and Distributed Computing, Euro-Par 2010, Ischia, Italy. Agosto 31-Septiembre 3.

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

Buyya, R. (2002). *Economic-based Distributed Resource Management and Scheduling for Grid Computing.* Monash University, Melbourne, Australia.

Buyya, R. (2005). Grid Computing Info Centre (GRID Infoware). Recuperado Agosto, 2009, de http://www.gridcomputing.com/

Buyya, R., Murshed, M., Abramson, D., y Venugopal, S. (2005). Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost–time optimization algorithm, *Software: Practice and Experience* (pp. 491–512): John Wiley & Sons, Ltd.

California, U. o. (2009a). BOINC. Berkeley Open Infrastructure for Network Computing. Recuperado Octubre 9. 2009, de http://boinc.berkeley.edu/

California, U. o. (2009b). SETI@home. Recuperado Agosto 2009, de http://setiweb.ssl.berkeley.edu/

Casanova, H., Legrand, A., Zagorodnov, D., y Berman, F. (2000). *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments.* 9th Heterogeneous Computing Workshop, Cancun, Mexico. Mayo 1.

CERN. (2008). Grid Café. Recuperado Agosto, 2011, de http://www.gridcafe.org

Chapin, S. J., Cirne, W., Feitelson, D. G., Jones, J. P., Leutenegger, S. T., Schwiegelshohn, U., Smith, W., y Talby, D. (1999). Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. En: D. G. Feitelson y L. Rudolph (Eds.), *Job Scheduling Strategies for Parallel Processing*. Springer-Verlag (Vol. 1659: pp. 66-89).

Chiang, S.-H., Arpaci-Dusseau, A. C., y Vernon, M. K. (2002). *The impact of more accurate requested runtimes on production job scheduling performance.* 8th International Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh, Scotland. Julio 24.

Dias de Assuncao, M., Buyya, R., y Venugopal, S. (2008). InterGrid: A Case for Internetworking Islands of Grids. *Concurrency and Computation: Practice and Experience (CCPE), 20*(8): 997-1024.

Dimitriadou, S. K., y Karatza, H. D. (2010). Multi-Site Allocation Policies on a Grid and Local Level. *Electronic Notes in Theoretical Computer Science (ENTCS), 261*: 163-179.

Elmroth, E., y Tordsson, J. (2005). *An interoperable, standards-based grid resource broker and job submission service.* First International Conference on e-Science and Grid Computing, 2005., Melbourne, Australia. Diciembre 5-8.

Ernemann, C., Hamscher, V., y Yahyapour, R. (2004). *Benefits of Global Grid Computing for Job Scheduling.* Fifth IEEE/ACM International Workshop on Grid Computing (Grid '04), in Conjunction with SuperComputing 2004, Pittsburgh, Pennsylvania. Noviembre 8.

Etsion, Y., y Tsafrir, D. (2005). *A Short Survey of Commercial Cluster Batch Schedulers* (No. Technical Report 2005-13): School of Computer Science and Engineering, The Hebrew University of Jerusalem.

Feitelson, D., Frachtenberg, E., Rudolph, L., y Schwiegelshohn, U. (Eds.). (2005a). *Job Scheduling Strategies for Parallel Processing* (Vol. 3834). Cambridge, MA, USA: Springer.

Feitelson, D., Rudolph, L., y Schwiegelshohn, U. (2005b). Parallel Job Scheduling — A Status Report. En: D. Feitelson, L. Rudolph y U. Schwiegelshohn (Eds.), *Job Scheduling Strategies for Parallel Processing*. Springer Berlin / Heidelberg (Vol. 3277: pp. 1-16).

Feitelson, D., Rudolph, L., Schwiegelshohn, U., Sevcik, K., y Wong, P. (1997). Theory and practice in parallel job scheduling. En: D. Feitelson y L. Rudolph (Eds.), *Job Scheduling Strategies for Parallel Processing*. Springer Berlin / Heidelberg (Vol. 1291: pp. 1-34).

Feitelson, D. G. (2005a). Experimental Analysis of the Root Causes of Performance Evaluation Results: A Backfilling Case Study. *IEEE Transactions on Parallel and Distributed Systems, 16*(2): 175-182.

Feitelson, D. G. (2005b). Parallel Workloads Archive. Recuperado Abril, 2011, de http://www.cs.huji.ac.il/labs/parallel/workload/

Feitelson, D. G., y Nitzberg, B. (1995). *Job Characteristics of a Production parallel scientific workload on the NASA Ames iPSC/860.* 1st Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '95), Santa Barbara, California, USA. Abril 25.

Feitelson, D. G., y Weil, A. M. a. (1998). *Utilization and Predictability in Scheduling the IBM SP2 with Backfilling.* 12th. International Parallel Processing Symposium (IPPS 1998), Orlando, Florida, USA. Marzo 30 - Abril 3.

Fölling, A., Grimme, C., Lepping, J., y Papaspyrou, A. (2010). Robust Load Delegation in Service Grid Environments. *IEEE Transactions on Parallel and Distributed Systems, 21*(9): 1304-1316.

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 (1a ed. pp. 359-373).

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

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.

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: pp. 257-282).

García, J. L. G. (2009). *Calendarización en línea para Grid jerárquico de dos niveles con un esquema de asignación admisible.*, Centro de Investigación Científica y de Educación Superior de Ensenada. CICESE, Ensenada, B.C. México.

Garey, M. R., y Graham, R. L. (1975). Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing, 4*(2): 187-200.

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

Grimme, C., Lepping, J., y Papaspyrou, A. (2007). *Identifying Job Migration Characteristics in Decentralized Grid Scheduling Scenarios.* 19th Iasted International Conference on Parallel and Distributed Computing and Systems, Cambridge, MA, USA. Noviembre 19-21.

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.

Guim, F., Corbalan, J., y Labarta, J. (2007). *Prediction f Based Models for Evaluating Backfilling Scheduling Policies.* Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007), Adelaide, Australia. Diciembre 3-6.

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. Diciembre 17.

Huedo, E., Montero, R. S., y Llorente, I. M. (2003). *Experiences on Grid Resource Selection Considering Resource Proximity.* Grid Computing. First European Across Grids Conference, Santiago de Compostela, España. Febrero 13-14.

Krauter, K., Buyya, R., y Maheswaran, M. (2002). A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *International Journal of Software: Practice and Experience (SPE), 32*(2): 135-164.

Kurowski, K., Nabrzyski, J., Oleksiak, A., y Weglarz, J. (2008). A multicriteria approach to two-level hierarchy scheduling in grids. *Journal of Scheduling, 11*(5): 371-379.

Lee, L.-T., Liang, C.-H., y Chang, H.-Y. (2006). *An Adaptive Task Scheduling System for Grid Computing.* Sixth IEEE International Conference on Computer and Information Technology, 2006. CIT '06., Seoul. Septiembre 20-22.

Leung, J. Y.-T. (2004). *Handbook of Scheduling. Algorithms, Models, and Perfomance Analysis*. Chapman & Hall/CRC (1 ed.). Boca Raton, Florida, USA. pp 1120.

Lifka, D. A. (1995). *The ANL/IBM SP Scheduling System.* 1st Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, CA, USA. Abril 25.

Lodi, A., Martello, S., y Monaci, M. (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research, 141*(2): 241-252.

Lozano, J. (2005). *Un algoritmo simple de resolución del problema de strip-packing 2D con rotación.* IX Congreso de Ingeniería de Organización, Gijón, España. Septiembre 8-9.

Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., y Freund, R. F. (1999). *Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems.* Eighth Heterogeneous Computing Workshop (HCW 2009), San Juan, Puerto Rico. Abril 12.

Mu'alem, A. W., y Feitelson, D. G. (2001). Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Transactions on Parallel and Distributed Systems, 12*(6): 529-543.

Nabrzyski, J., Schopf, J. M., y Weglarz, J. (Eds.). (2003). *Grid Resource Management. State of the Art and Future Trends* (1 ed.): Kluwer Academic Publisher.

Oxford, U. (2009). Climateprediction.net. Recuperado Septiembre 2009, de http://climateprediction.net/

Pascual, F., Rzadca, K., y Trystram, D. (2009). Cooperation in multi-organization scheduling. *Concurrency and Computation: Practice & Experience, 21*(7): 905-921.

Ramírez-Alcaraz, J., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., González-García, J., y Hirales-Carbajal, A. (2011). Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids. *Journal of Grid Computing, 9*(1): 95-116.

Ranganathan, K., y Foster, I. (2003a). Computation Scheduling and Data Replication Algorithms for Data Grids. En: J. Nabrzyski, J. M. Schopf y J. Weglarz (Eds.), *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers.

Ranganathan, K., y Foster, I. (2003b). Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing, 1*(1).

Rodero, I., Corbalan, J., Badía, R. M., y Labarta, J. (2005). *eNANOS Grid Resource Broker.* Advances in Grid Computing. European Grid Conference (EGC 2005), Amsterdam, The Netherlands. Febrero 14-16.

Rodero, I., Guim, F., Corbalan, J., y Goyeneche, A. (2008). The Grid Backfilling: a Multi-Site Scheduling Architecture with Data Mining Prediction Techniques. En: D.

Talia, R. Yahyapour y W. Ziegler (Eds.), *Grid Middleware and Services. Challenges and Solutions*. Springer US (Vol. 8: pp. 137-152).

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 (1a ed.).

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. Agosto 10-12.

Schwiegelshohn, U., Tchernykh, A., y Yahyapour, R. (2008). *Online Scheduling in Grids.* IEEE International Symposium on Parallel and Distributed Processing 2008 (IPDPS 2008), Miami, FL, USA. 14-18 April 2008.

Shan, H., Oliker, L., y Biswas, R. (2003). *Job Superscheduler Architecture and Performance in Computational Grid Environments.* ACM/IEEE Conference on High Performance Computing Networking, Storage and Analysis (SC 2003), Phoenix, AZ, USA. Noviembre 15-21.

Shnizler, B., Neumann, D., y Weinhardt, C. (2004). *Resource Allocation in Computational Grids- A Market Engineering Approach.* WeB 2004, Washington.

Skovira, J., Chan, W., y Zhou, H. (1996). *The EASY-LoadLeveler API Project.* Job Scheduling Strategies for Parallel Processing, Honolulu, Hawaii. Abril 16.

Smith, W. (2003). Improving Resource Selection and Scheduling Using Predictions. En: J. Nabrzyski, J. M. Schopf y J. Weglarz (Eds.), *Grid Resource Management, State of the Art and Future Trends*. Kluwer Academic Publisher (1a ed.).

Song, B., Ernemann, C., y Yahyapour, R. (2004). *Modeling of parameters in supercomputer workloads.* Workshop on Parallel Systems and Algorithms (PASA) in conjunction with ARCS 2004, Augsburg, Germany. Marzo 23-26.

Talby, D., Tsafrir, D., Goldberg, Z., y Feitelson, D. G. (2006). *Session-Based, Estimation-less, and Information-less Runtime Prediction Algorithms for Parallel and Grid Job Scheduling* (Technical report No. 2006-77). Jerusalem, Israel: School of Computer Science and Engineering, Hebrew University of Jerusalem.

Tchernykh, A., Ramírez, J., Avetisyan, A., Kuzjurin, N., Grushin, D., y Zhuk, S. (2006). *Two Level Job-Scheduling Strategies for a Computational Grid.* 6th

International Conference on Parallel Processing and Applied Mathematics PPAM 2005, Poznan, Poland. Septiembre 11-14.

Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., y Kuzjurin, N. (2008). Online Hierarchical Job Scheduling on Grids. En: T. Priol y M. Vanneschi (Eds.), *From Grids to Service and Pervasive Computing*. Springer US (pp. 77-91).

Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., y Kuzjurin, N. (2010). Online Hierarchical Job Scheduling on Grids with Admissible Allocation, *Journal of Scheduling* (pp. 1-8): Springer Netherlands.

Tsafrir, D., Etsion, Y., y Feitelson, D. G. (2006). Modeling User Runtime Estimates. En: D. G. Feitelson, E. Frachtenberg, L. Rudolph y U. Schwiegelshohn (Eds.), *11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2005)*. Springer. Cambridge, MA, USA (Vol. LNCS 3834: pp. 1-35).

Tsafrir, D., Etsion, Y., y Feitelson, D. G. (2007). Backfilling Using System-Generated Predictions Rather than User Runtime Estimates. *IEEE Transactions on Parallel and Distributed Systems, 18*(6): 789-803.

Tsafrir, D., y Feitelson, D. G. (2006). *The Dynamics of Backfilling: Solving the Mystery of Why Increased Inaccuracy May Help.* IEEE International Symposium on Workload Characterization (IISWC 2006), San Jose, California, USA. Octubre 25-27.

Washington, U. o. (2009). Rosetta@home. Recuperado Septiembre 2009, de http://boinc.bakerlab.org/rosetta/

Ye, D., Han, X., y Zhang, G. (2011). Online multiple-strip packing. *Theoretical Computer Science, 412*(3): 233-239.

Zhuk, S., Chernykh, A., Avetisyan, A., Gaissaryan, S., Grushin, D., Kuzjurin, N., Pospelov, A., y Shokurov, A. (2004). *Comparison of Scheduling Heuristics for Grid Resource Broker.* Third International IEEE Conference on Parallel Computing Systems (PCS 2004), Colima, Colima, México. Septiembre 20-24.

Zikos, S., y Karatza, H. D. (2008). *Resource Allocation Strategies in a 2-Level Hierarchical Grid System.* Simulation Symposium, 2008. ANSS 2008. 41st Annual, Ottawa, Ont. 13-16 April 2008.

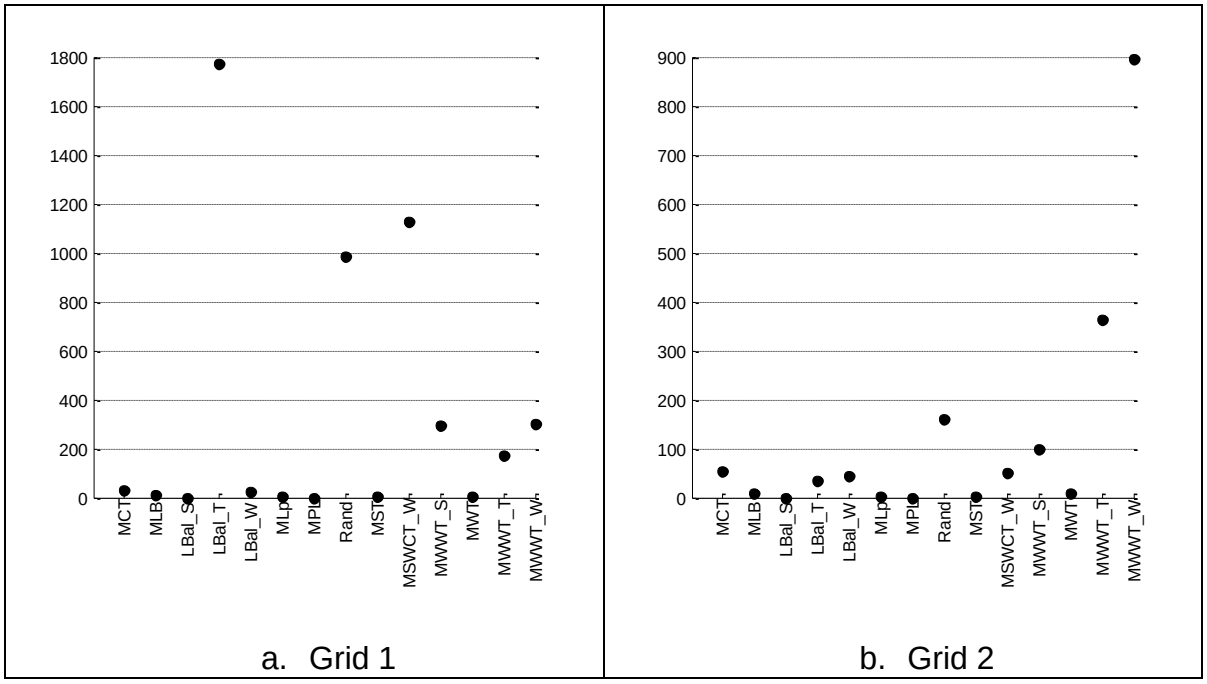
Zikos, S., y Karatza, H. D. (2009). Communication cost effective scheduling policies of nonclairvoyant jobs with load balancing in a grid. *Journal of System and Software, 82*(12): 2103-2116.

Zotkin, D., y Keleher, P. J. (1999). *Job-Length Estimation and Performance in Backfilling Schedulers.* Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8 '99), Redondo Beach, California. Agosto 3-6.

Apéndice

A.1. Rendimiento de las estrategias de asignación

En el primer paso se calcula, para cada estrategia, el promedio de ralentización acotada, el promedio de los tiempos de espera y la suma de tiempos de finalización ponderados, respecto a los *n* trabajos de entrada. Las Figuras 18-20 muestran los resultados absolutos de estas métricas, tanto para el Grid 1 como para el Grid 2.



a. Grid 1 b. Grid 2

Figura 18. Promedio de ralentización acotada

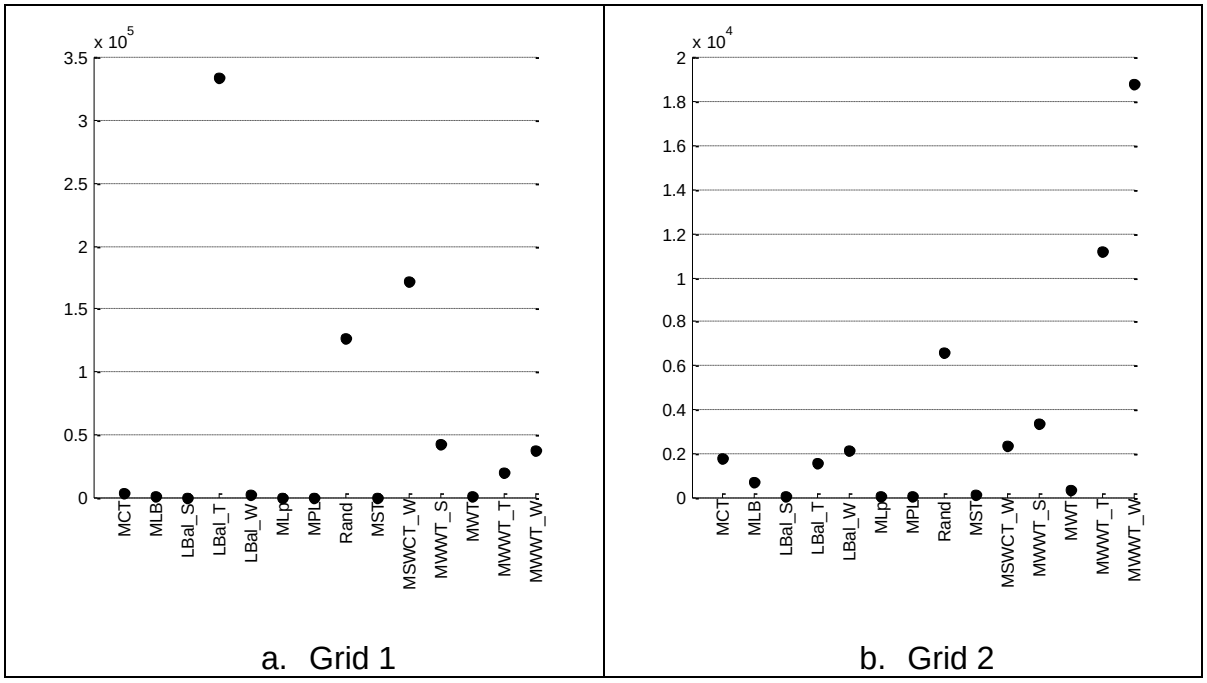


Figura 19. Tiempo promedio de espera

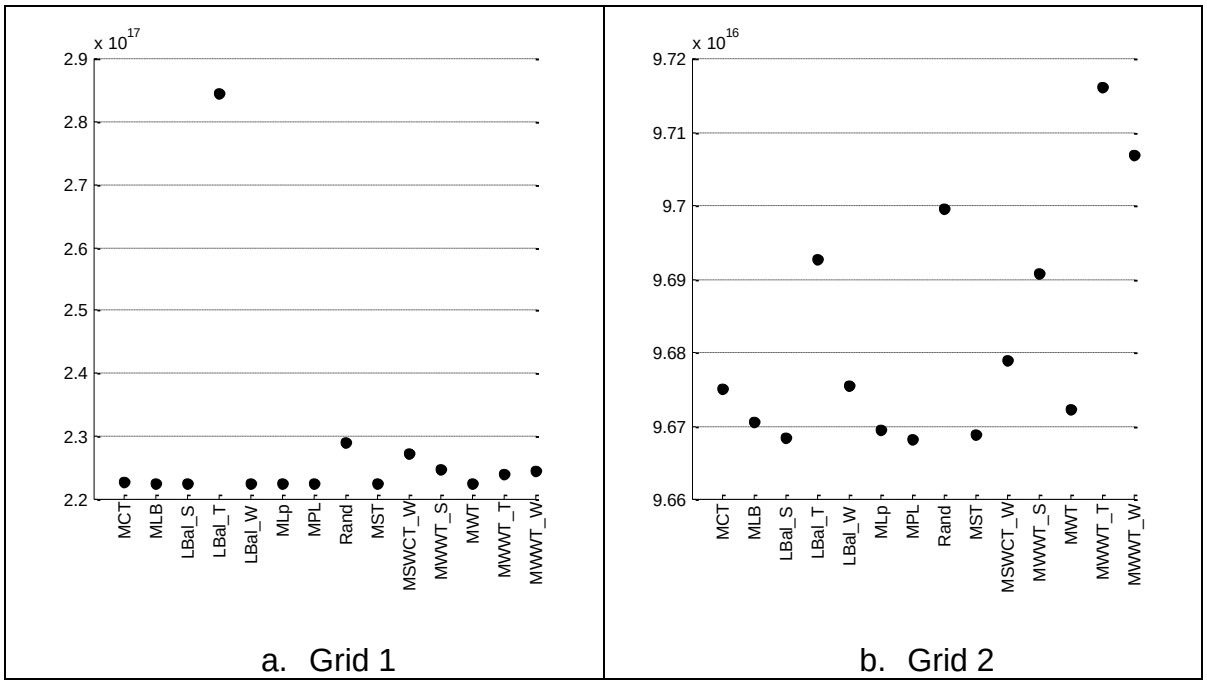


Figura 20. Suma de tiempos de finalización ponderados con recursos consumidos

A.2. Degradación de rendimiento de las estrategias de asignación

Como segundo paso, se calcula la degradación que tiene cada estrategia con respecto a la mejor de ellas. Las Figuras 21-29 muestran dicha degradación para cada una de las métricas consideradas. Dado que son métricas de minimización, la mejor estrategia es la que generó el menor valor en la métrica correspondiente. Para cada métrica se muestra primero el promedio de degradación de todas las estrategias, después se muestran los mismos valores pero en orden no decreciente, y finalmente se muestran las 6 mejores estrategias en cada métrica.

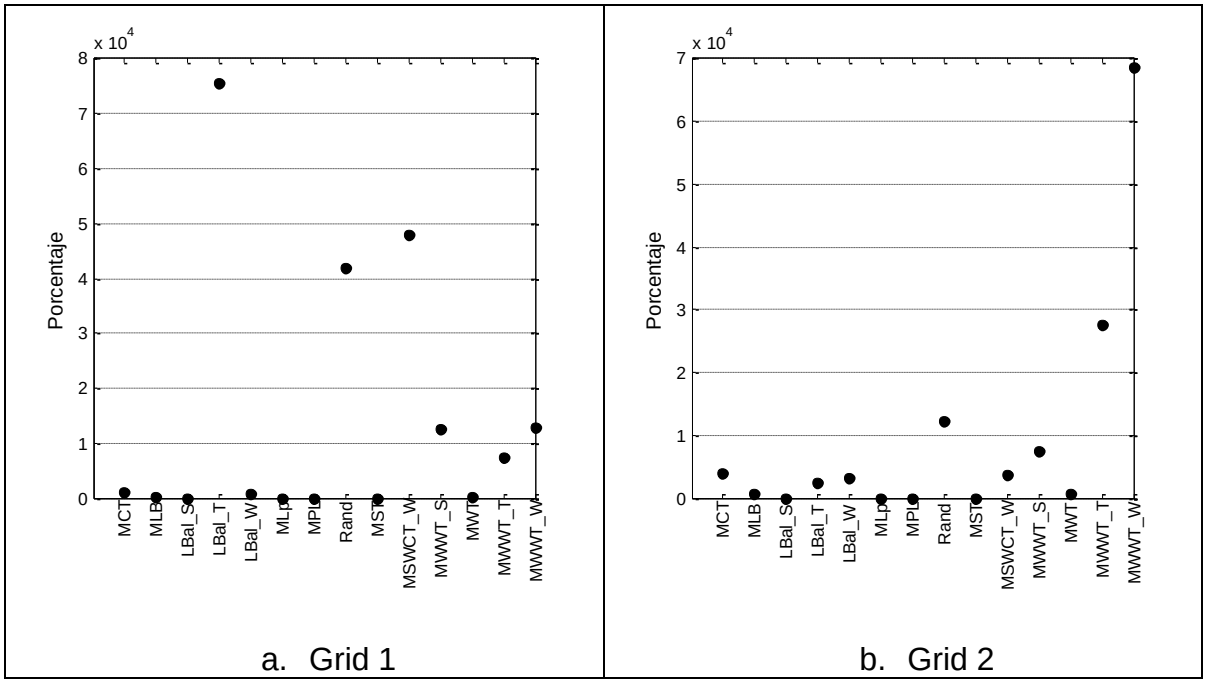


Figura 21. Degradación promedio de ralentización acotada.

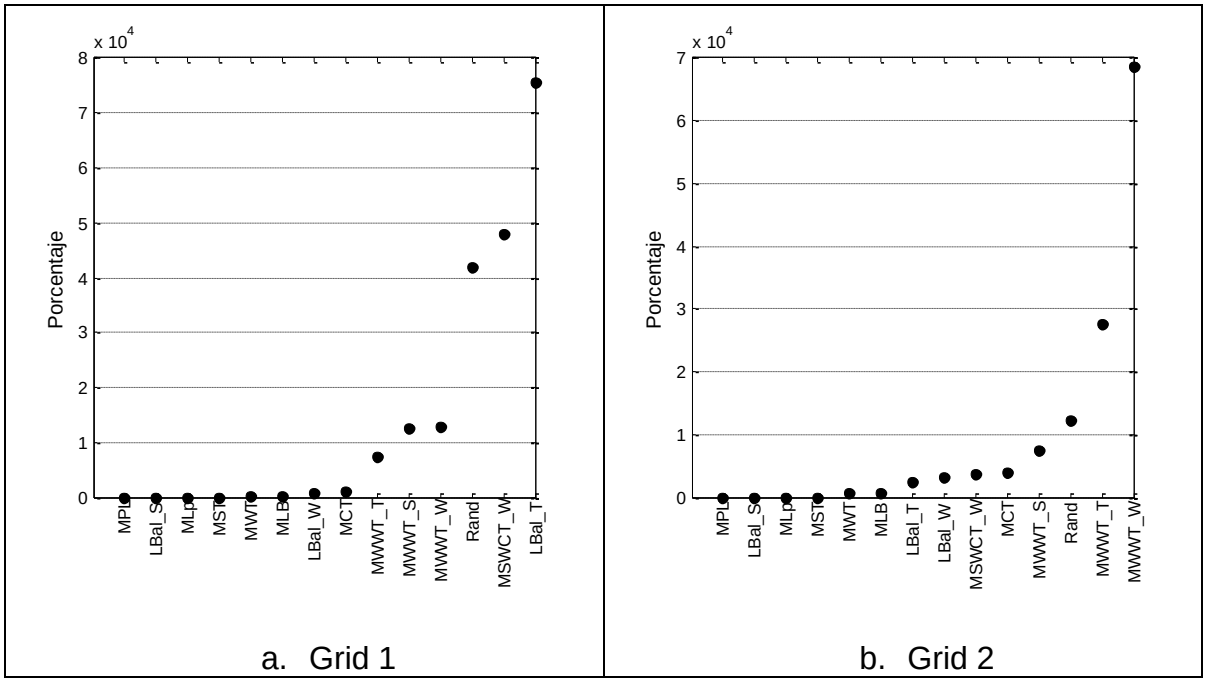


Figura 22. Degradación promedio de ralentización acotada (estrategias jerarquizadas).

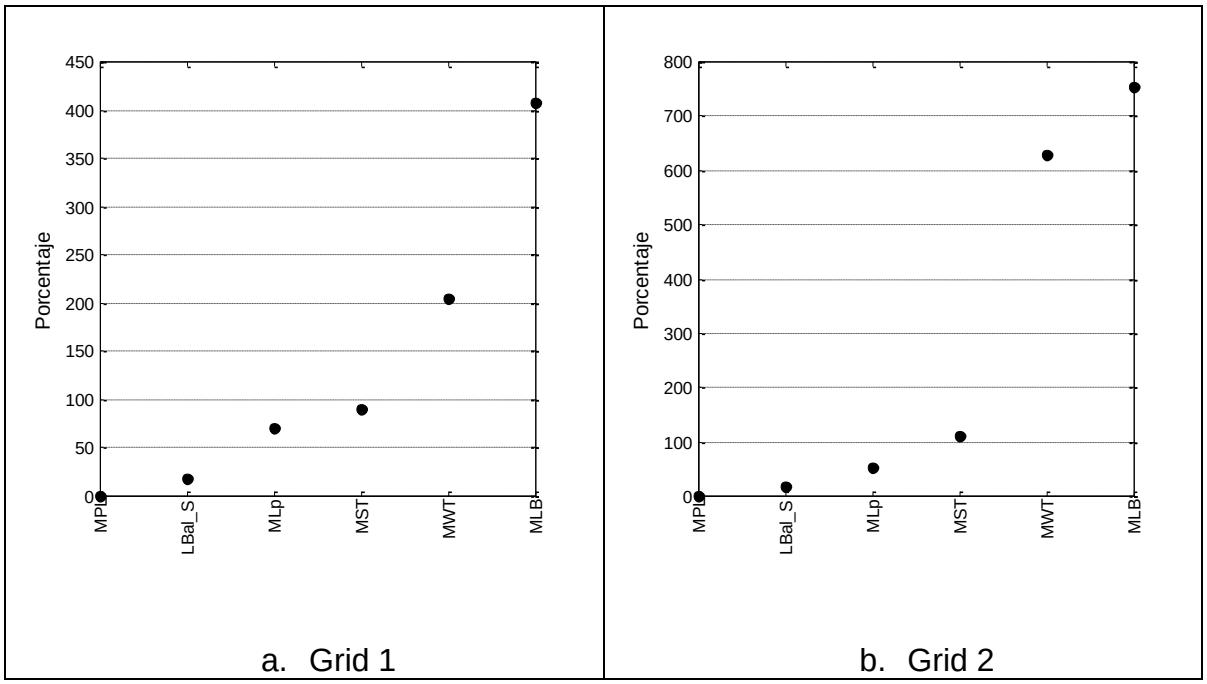


Figura 23. Degradación promedio de ralentización acotada (6 mejores estrategias).

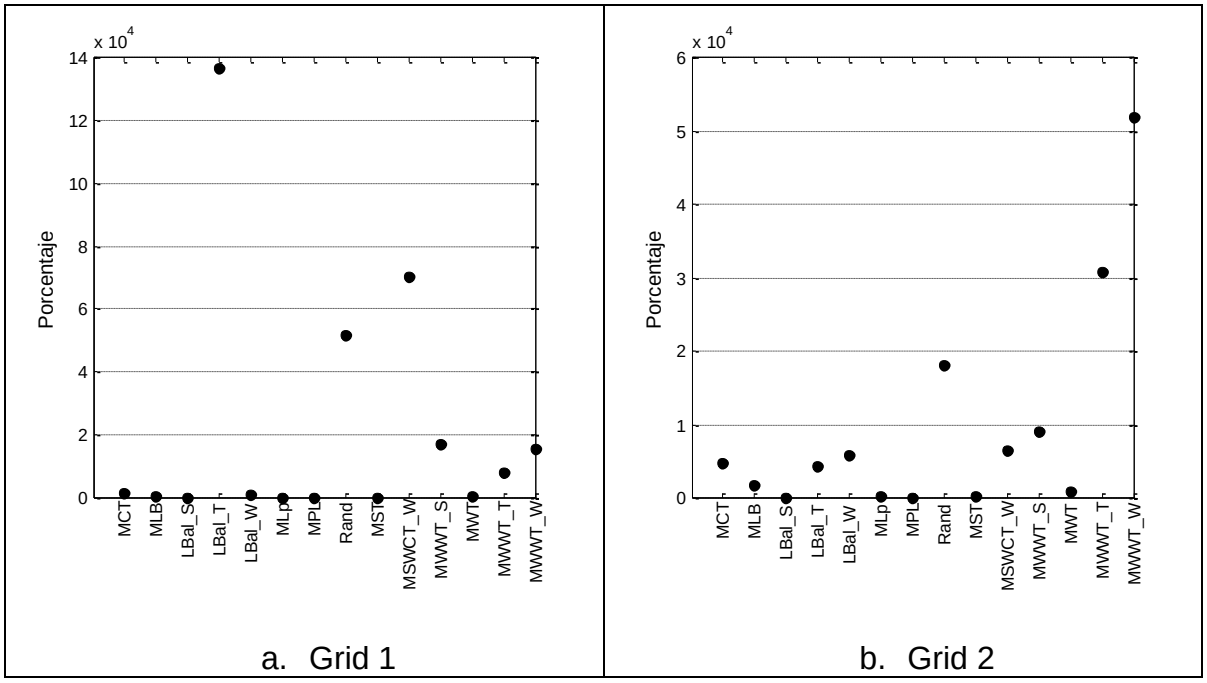


Figura 24. Degradación promedio del tiempo de espera.

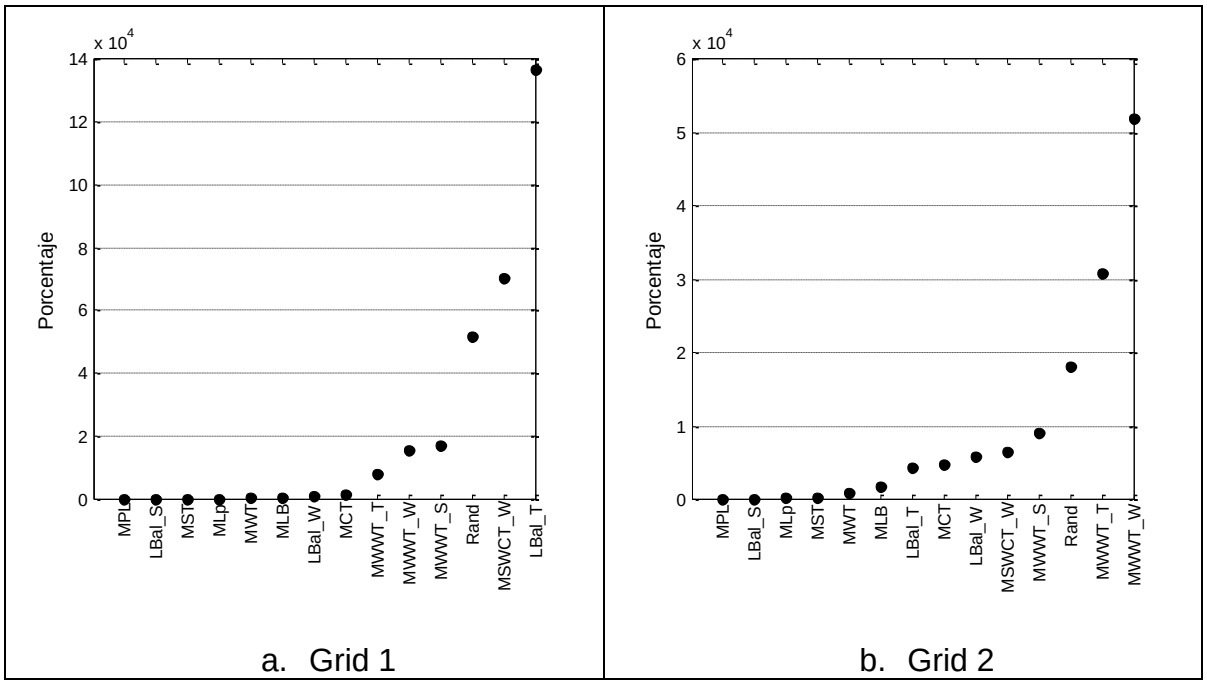
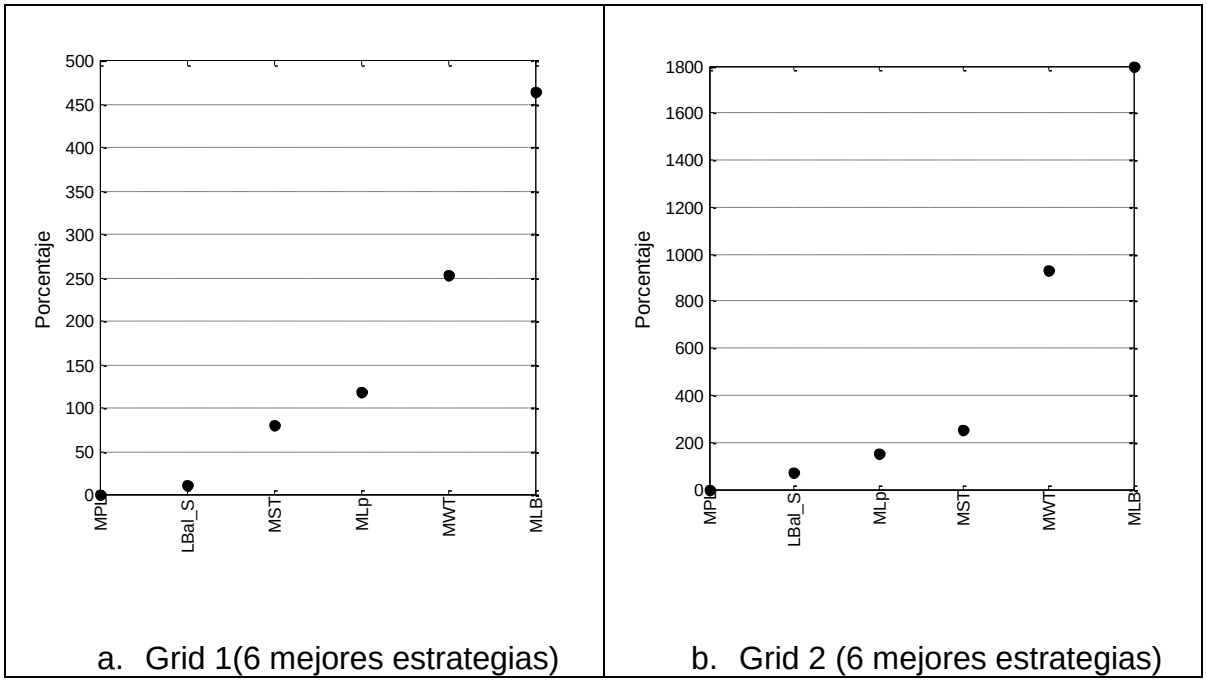
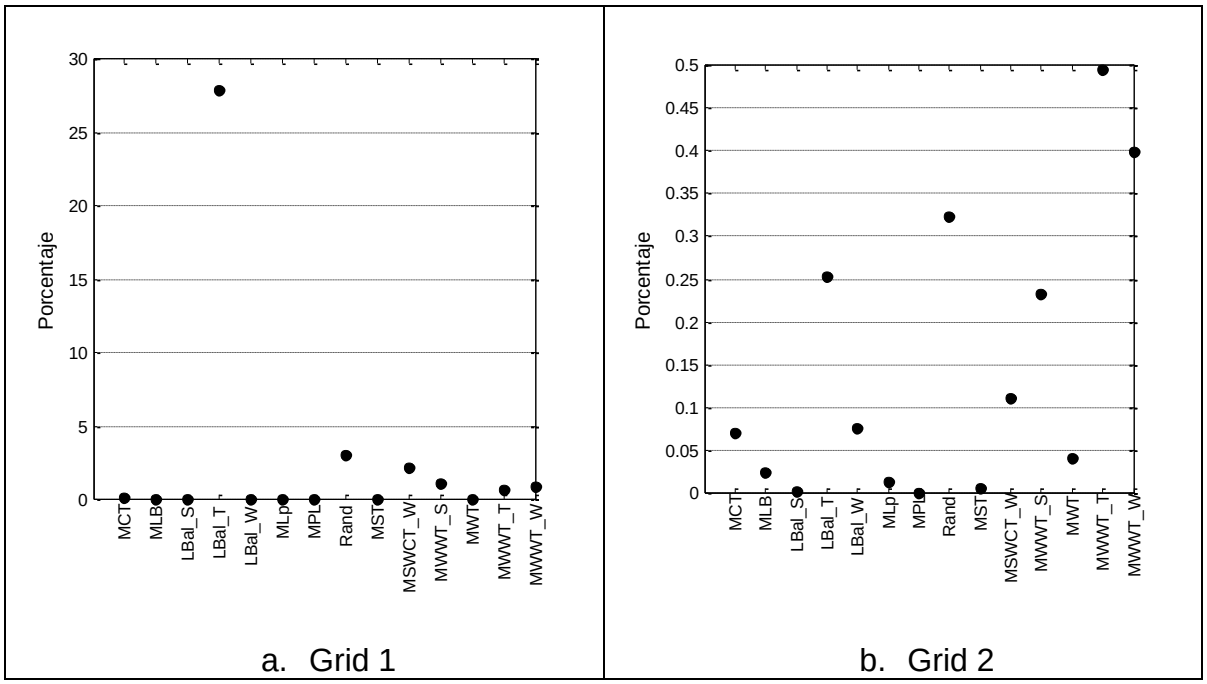


Figura 25. Degradación promedio del tiempo de espera (estrategias jerarquizadas).



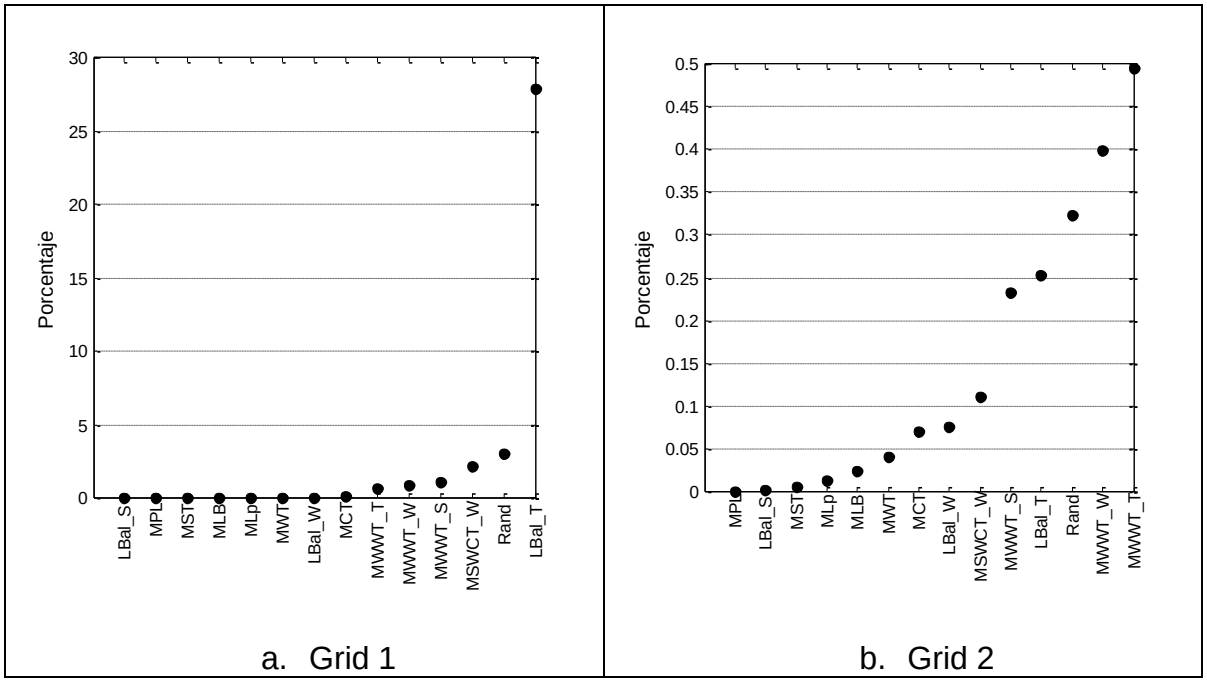
a. Grid 1(6 mejores estrategias) b. Grid 2 (6 mejores estrategias)

Figura 26. Degradación promedio del tiempo de espera (6 mejores estrategias)



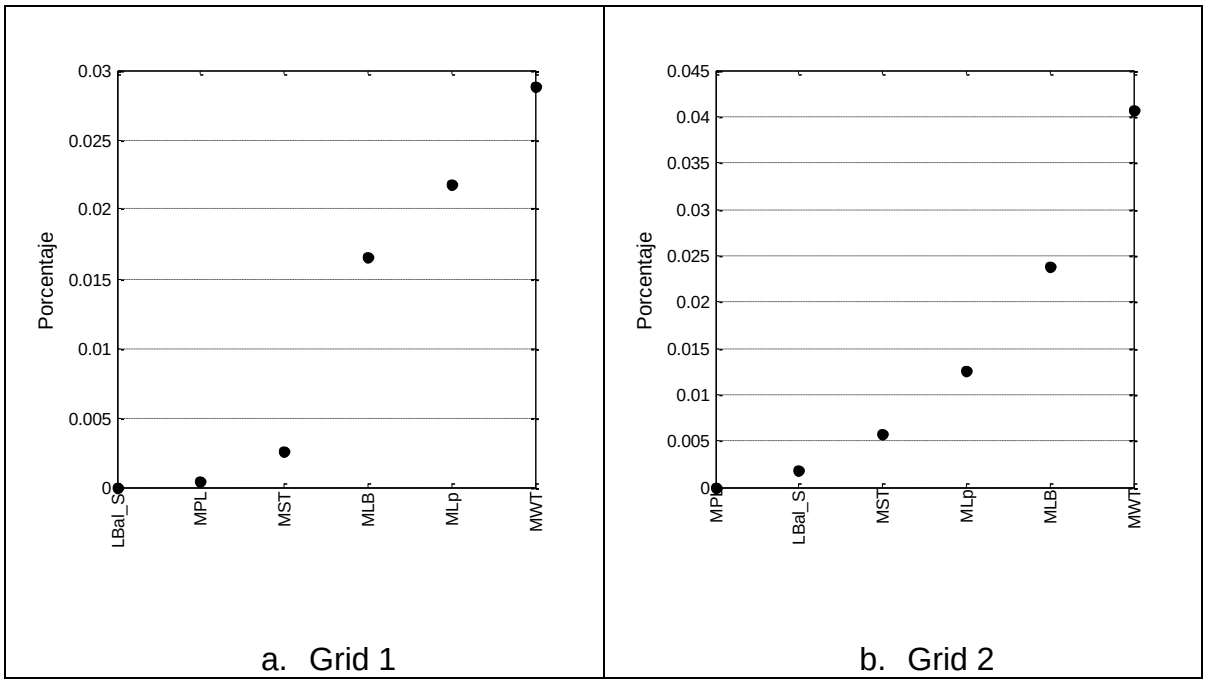
a. Grid 1 b. Grid 2

Figura 27. Degradación de la suma de tiempos de finalización ponderados con recursos consumidos.



a. Grid 1 b. Grid 2

Figura 28 Degradación de la suma de tiempos de finalización ponderados con recursos consumidos (estrategias jerarquizadas).



a. Grid 1 b. Grid 2

Figura 29. Degradación de la suma de tiempos de finalización ponderados con recursos consumidos (6 mejores estrategias)

A.3. Clasificación jerarquizada de las estrategias de asignación según el promedio de degradación de rendimiento de las 3 métricas.

En el tercer paso, se promedian los valores de la degradación de las estrategias en las 3 métricas y se ordenan las estrategias en orden no descendente. La mejor estrategia, con el promedio de degradación de rendimiento más bajo, tiene la posición 1; la peor estrategia tiene la posición 14. La Figura 24 muestra estos resultados.

Como cuarto paso, se toman los resultados de los promedios de las tres métricas por Grid y se hace un nuevo promedio. Así, cada estrategia ahora está caracterizada por 2 valores, uno es el promedio de las tres métricas para el Grid 1 y la otra corresponde al promedio de las mismas métricas para Grid 2. Se calcula el promedio por estrategia y se ordenan en orden no decreciente. Los resultados de este paso se muestran en la Figura 25

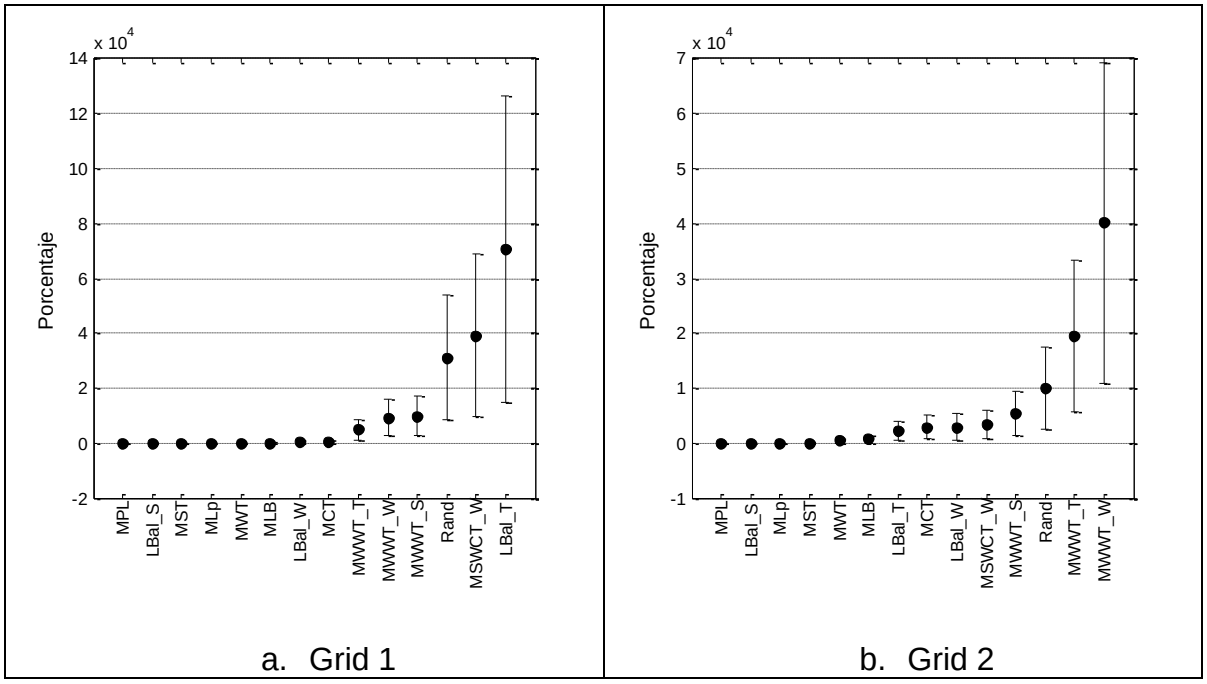
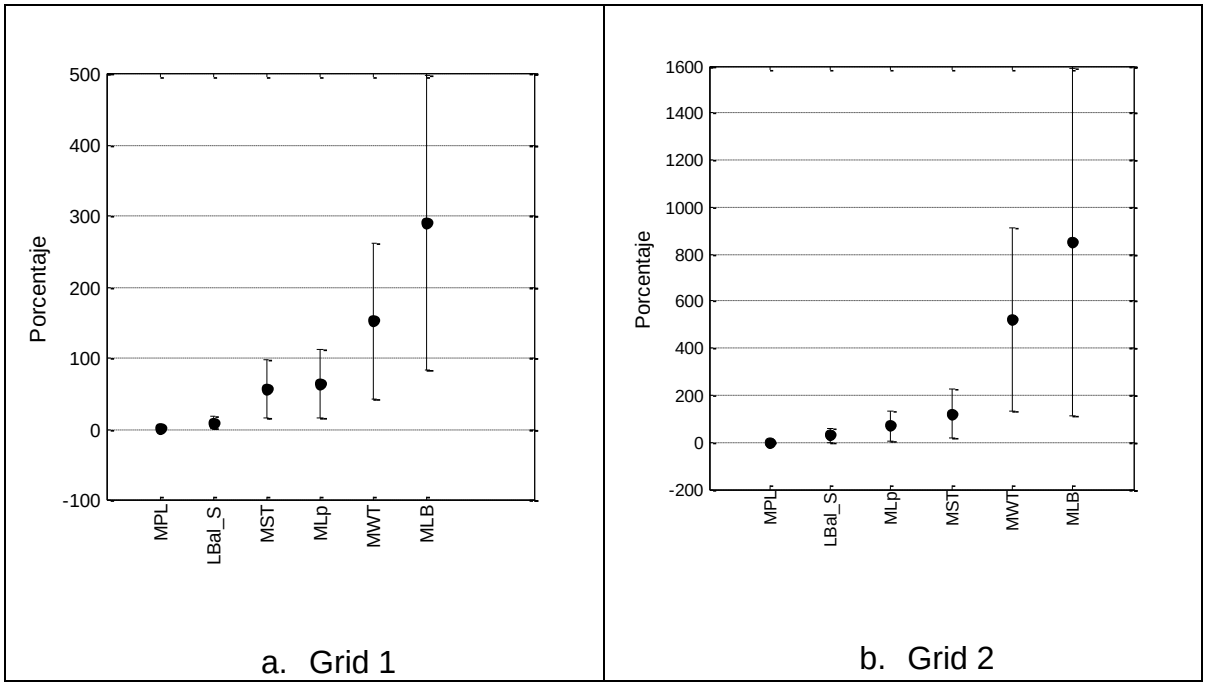
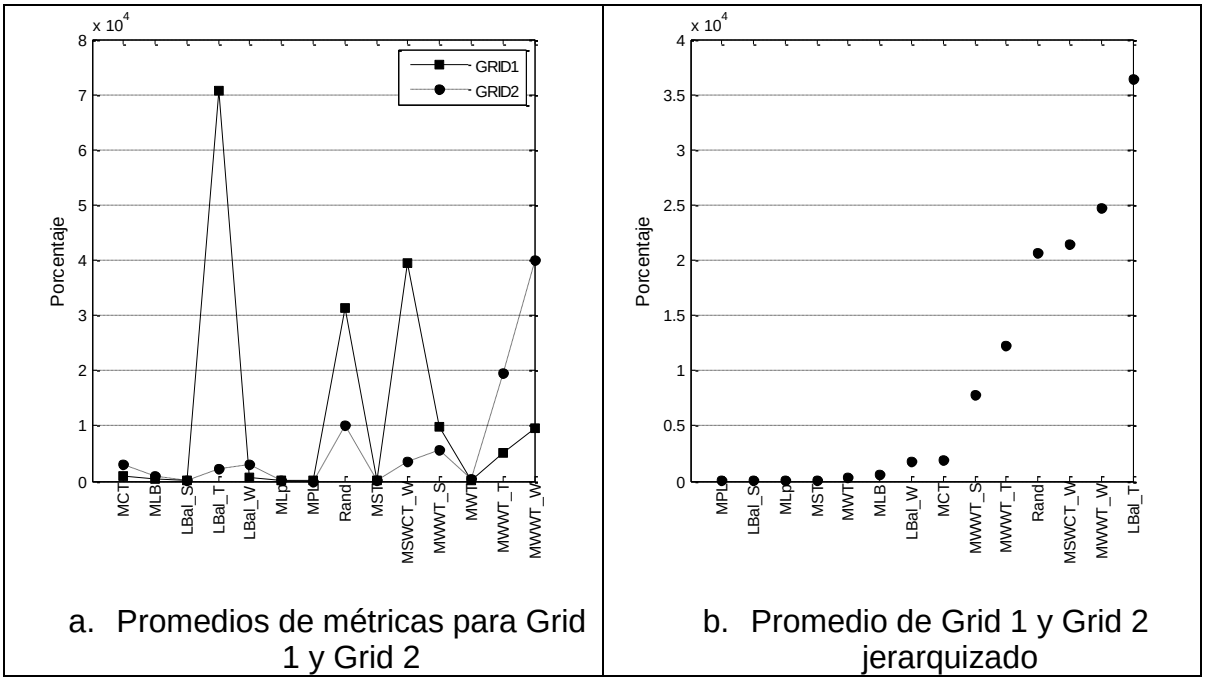


Figura 30. Clasificación jerárquica de estrategias de asignación según el promedio de degradación de rendimiento de 3 métricas.



a. Grid 1 b. Grid 2

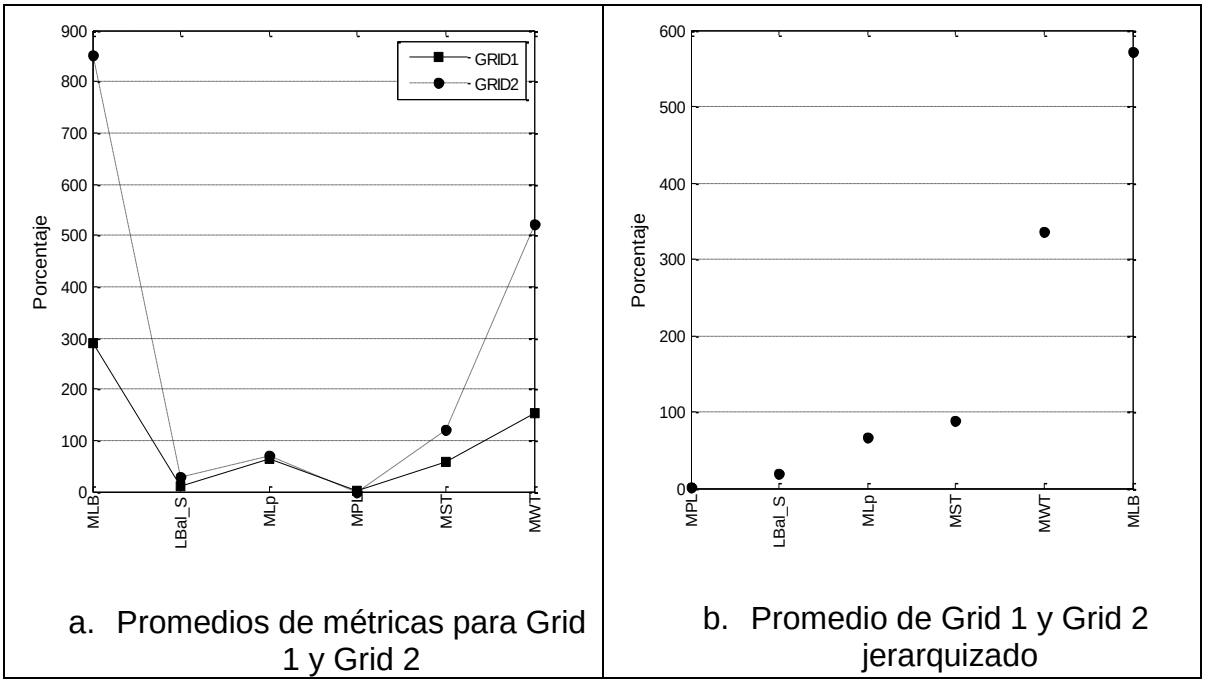
Figura 31. Clasificación jerárquica de estrategias de asignación según el promedio de degradación de rendimiento de 3 métricas (6 mejores estrategias).



a. Promedios de métricas para Grid 1 y Grid 2

b. Promedio de Grid 1 y Grid 2 jerarquizado

Figura 32. Promedios de degradación de estrategias de asignación para Grid 1 y Grid 2



a. Promedios de métricas para Grid 1 y Grid 2

b. Promedio de Grid 1 y Grid 2 jerarquizado

Figura 33. Promedios de degradación de estrategias de asignación para Grid 1 y Grid 2 (6 mejores estrategias)

A.4. Rendimiento de las estrategias de asignación para todos los casos de prueba

Las Tablas XI a XIII muestran todos los resultados obtenidos de la degradación de rendimiento de las 14 estrategias analizadas bajo todas las métricas implementadas, para Grid 1, Grid 2 y el promedio de los dos escenarios.

Tabla XI. Porcentajes de las degradaciones de rendimiento para todos los casos de prueba para Grid 1

Strategy / Metric	MCT	MLB	LBal_S	LBal_T	LBal_W	MLp	MPL	Random	MST	MSWCT_W	MWWT_S	MWT	MWWT_T	MWWT_W
ρ	0	0	0	32	1	2	0	21	0	27	7	0	2	7
SD	1501	469	20	101951	1018	93	0	53741	100	69754	17658	236	8595	15526
SD_b	1284	408	18	75442	899	71	0	41924	91	47821	12576	204	7382	12889
Th	0	0	0	24	1	2	0	18	0	21	7	0	2	7
TA	47	17	0	5047	37	4	0	1923	3	2600	641	9	299	574
U	0	0	0	24	1	2	0	18	0	21	7	0	2	7
t_w	1256	463	11	136322	1006	119	0	51929	80	70233	17303	252	8073	15506
$LBal_s$	2967	3006	171	2889	3505	588	237	1975	0	1750	267	73	3145	2380
$LBal_t$	744	2273	2650	1382	2528	0	2855	2564	2575	6487	2929	678	509	2438
$LBal_w$	47	50	31	1465	36	263	47	1141	0	1114	257	11	251	369
SCT	0	0	0	4	0	0	0	2	0	2	1	0	0	0
$SWCT_s$	0	0	0	16	0	0	0	2	0	1	1	0	1	1
$SWCT_t$	0	0	0	9	0	0	0	2	0	5	1	0	0	1
$SWCT_w$	0	0	0	28	0	0	0	3	0	2	1	0	1	1
SWT	1256	463	11	136322	1006	119	0	51929	80	70233	17303	252	8073	15506
$SWWT_s$	378	187	0	74082	364	130	6	9483	1	4626	3429	132	2884	4060
$SWWT_t$	750	427	10	213412	1006	332	0	57433	25	125756	23324	294	4586	13823
$SWWT_w$	629	121	0	203720	221	159	3	21836	19	15364	7833	210	4638	6442
WTA_s	100	49	0	19628	96	35	2	2513	0	1226	909	35	764	1076
WTA_t	6	3	0	1586	7	2	0	427	0	934	173	2	34	103
WTA_w	23	5	0	7604	8	6	0	815	1	573	292	8	173	240
WWT_s	378	187	0	74082	364	130	6	9483	1	4626	3429	132	2884	4060
WWT_t	750	427	10	213412	1006	332	0	57433	25	125756	23324	294	4586	13823
WWT_w	629	121	0	203720	221	159	3	21836	19	15364	7833	210	4638	6442

Tabla XII. Porcentajes de las degradaciones de rendimiento para todos los casos de prueba para Grid 2

Strategy / Metric	MCT	MLB	LBal_S	LBal_T	LBal_W	MLp	MPL	Random	MST	MSWCT_W	MWWT_S	MWT	MWWT_T	MWWT_W
ρ	0	0	0	0	0	0	0	2	0	1	0	0	1	4
SD	5734	1031	10	3587	4469	62	0	17688	110	4695	17609	999	37340	185601
SD_b	4001	754	17	2569	3346	53	0	12196	111	3858	7550	628	27638	68410
Th	0	0	0	0	0	0	0	2	0	1	0	0	1	4
TA	93	34	1	80	110	3	0	345	5	125	175	18	590	992
U	0	0	0	0	0	0	0	2	0	1	0	0	1	4
t_w	4833	1799	70	4205	5743	153	0	18027	252	6515	9132	933	30822	51768
$LBal_s$	1257	1312	219	938	1688	244	0	990	40	623	66	27	1282	1158
$LBal_t$	480	760	1416	596	1127	0	1237	1287	1292	2293	1258	387	313	1191
$LBal_w$	222	95	0	668	25	200	117	461	1	229	72	78	158	78
SCT	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$SWCT_s$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$SWCT_t$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$SWCT_w$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SWT	4833	1799	70	4205	5743	153	0	18027	252	6515	9132	933	30822	51769
$SWWT_s$	700	384	10	1739	1036	101	0	2378	39	865	1411	287	4216	3547
$SWWT_t$	1486	2254	47	2880	5860	167	0	19330	96	6267	7915	537	15123	26358
$SWWT_w$	613	207	16	2209	656	110	0	2826	50	975	2026	355	4320	3486
WTA_s	135	74	2	334	199	19	0	457	7	166	271	55	810	682
WTA_t	3	5	0	6	12	0	0	41	0	13	17	1	32	56
WTA_w	20	7	1	71	21	4	0	91	2	31	65	11	139	112
WWT_s	700	384	10	1739	1036	101	0	2378	39	865	1411	287	4216	3547
WWT_t	1486	2254	47	2880	5860	167	0	19330	96	6267	7915	537	15123	26358
WWT_w	613	207	16	2209	656	110	0	2826	50	975	2026	355	4320	3486

Tabla XIII. Porcentajes de las degradaciones de rendimiento para todos los casos de prueba

Metric \ Strategy	MCT	MLB	LBal_S	LBal_T	LBal_W	MLp	MPL	Random	MST	MSWCT_W	MWWT_S	MWT	MWWT_T	MWWT_W
ρ	0	0	0	16	1	1	0	12	0	14	4	0	1	6
SD	3618	750	15	52769	2743	77	0	35715	105	37224	17633	617	22968	100564
SD_b	2643	581	17	39005	2122	62	0	27060	101	25839	10063	416	17510	40650
Th	0	0	0	12	1	1	0	10	0	11	3	0	1	5
TA	70	26	1	2564	74	4	0	1134	4	1363	408	14	445	783
U	0	0	0	12	1	1	0	10	0	11	3	0	1	5
t_w	3045	1131	40	70264	3374	136	0	34978	166	38374	13217	593	19447	33637
$LBal_s$	2112	2159	195	1913	2597	416	118	1483	20	1186	166	50	2214	1769
$LBal_t$	612	1516	2033	989	1827	0	2046	1926	1933	4390	2094	532	411	1815
$LBal_w$	134	72	16	1066	30	231	82	801	0	671	165	44	204	224
SCT	0	0	0	2	0	0	0	1	0	1	0	0	0	0
$SWCT_s$	0	0	0	8	0	0	0	1	0	1	0	0	1	1
$SWCT_t$	0	0	0	4	0	0	0	1	0	3	1	0	0	0
$SWCT_w$	0	0	0	14	0	0	0	2	0	1	1	0	1	1
SWT	3045	1131	40	70264	3375	136	0	34978	166	38374	13217	593	19447	33638
$SWWT_s$	539	285	5	37910	700	115	3	5931	20	2745	2420	209	3550	3803
$SWWT_t$	1118	1341	29	108146	3433	249	0	38381	61	66011	15620	416	9855	20091
$SWWT_w$	621	164	8	102965	438	134	2	12331	35	8169	4929	283	4479	4964
WTA_s	117	62	1	9981	148	27	1	1485	4	696	590	45	787	879
WTA_t	4	4	0	796	10	1	0	234	0	474	95	2	33	79
WTA_w	22	6	0	3838	15	5	0	453	1	302	179	10	156	176
WWT_s	539	285	5	37910	700	115	3	5931	20	2745	2420	209	3550	3803
WWT_t	1118	1341	29	108146	3433	249	0	38381	61	66011	15620	416	9855	20091
WWT_w	621	164	8	102965	438	134	2	12331	35	8169	4929	283	4479	4964
Average	832	459	102	31315	1061	87	94	10565	114	12616	4324	197	4975	11331

Las Tablas XIV a XIX muestran los promedios globales generados directamente por el simulador para las 14 estrategias analizadas bajo todas las métricas implementadas, para Grid 1, Grid 2 y el promedio de los dos escenarios.

Tabla XIV. Grid 1. Promedio de rendimiento (1ª parte, 7 estrategias)

Strategy / Metric	MCT	MLB	LBal_S	LBal_T	LBal_W	MLp	MPL
ρ	0.0029	0.377	0	32.0724	1.0898	2.0803	0
SD	1501.15942	468.571429	20.1656315	101950.766	1017.72257	93.0434783	0
SD_b	1284.39626	407.695578	17.5170068	75441.7517	899.022109	70.9608844	0
Th	0	0.36987568	0	24.2782287	1.07880407	2.03431624	0
TA	46.5112993	17.1432747	0.42009954	5047.39615	37.2317699	4.39205014	0
U	0.00277064	0.37554783	0	24.2839152	1.07803133	2.0379326	0
t_w	1256.18651	463.008017	11.3404379	136322.468	1005.56949	118.641256	0
$LBal_s$	2966.99014	3006.11965	170.954797	2889.03208	3504.81932	588.35298	236.665831
$LBal_t$	743.722965	2272.7382	2650.42657	1381.69783	2527.53717	0	2855.48601
$LBal_w$	47.2674368	49.9451895	31.3818706	1465.13739	36.2227441	262.577893	47.2587762
SCT	0.03864845	0.01424515	0.00034891	4.18707957	0.03093783	0.00365018	0
$SWCT_s$	0.08256517	0.04080597	0	16.196019	0.07957595	0.02848006	0.00140523
$SWCT_t$	0.03106365	0.01769717	0.00041099	8.83628047	0.04164111	0.01373466	0
$SWCT_w$	0.08611198	0.01653283	0	27.8939322	0.03020598	0.02174043	0.00044254
SWT	1256.18624	463.008033	11.3405793	136322.451	1005.56941	118.641392	0
$SWWT_s$	377.528901	186.585113	0	74081.5294	363.860787	130.22494	6.4254095
$SWWT_t$	750.242596	427.418266	9.92605228	213411.995	1005.70727	331.716455	0
$SWWT_w$	750.242596	427.418266	9.92605228	213411.995	1005.70727	331.716455	0
WTA_s	100.037537	49.4413045	0	19627.9463	96.415062	34.5043364	1.70253854
WTA_t	5.57495021	3.17608681	0.07375907	1585.8354	7.4732729	2.46493991	0
WTA_w	23.4753112	4.50707832	0	7604.2701	8.23456612	5.9267402	0.12064328
WWT_s	377.528902	186.585113	0	74081.5277	363.860798	130.224937	6.4254105
WWT_t	750.242633	427.418267	9.92605187	213411.997	1005.70731	331.716493	0
WWT_w	628.909234	120.745694	0	203720.215	220.605981	158.778778	3.23205984

Tabla XV. Grid 1. Promedio de rendimiento (2ª parte, 7 estrategias)

Strategy / Metric	Random	MST	MSWCT_W	MWWT_S	MWT	MWWT_T	MWWT_W
ρ	21.493	0	26.9758	7.1618	0.1441	1.6332	7.1621
SD	53741.4079	100.331263	69753.9545	17657.764	235.942029	8595.44513	15526.1698
SD_b	41924.1071	90.6887755	47820.6207	12576.1905	204.039116	7381.97279	12888.7755
Th	17.6923867	0	21.247303	6.6783109	0.14384054	1.60279462	6.6783109
TA	1922.68999	2.956734	2600.4134	640.650709	9.33730875	298.896503	574.121478
U	17.6907964	0	21.2447736	6.68329051	0.14382147	1.60697194	6.68354239
t_w	51928.5474	79.8570646	70232.6819	17302.8828	252.183667	8072.68537	15506.05
$LBal_s$	1975.39984	0	1749.70772	267.109756	72.5040032	3144.68775	2379.56707
$LBal_t$	2564.39116	2574.75112	6486.53947	2929.13143	677.947377	509.064053	2438.0499
$LBal_w$	1140.84458	0	1114.13885	257.272621	10.7015354	250.920444	369.000223
SCT	1.59408353	0.00245692	2.15607425	0.5323493	0.00775881	0.24836821	0.47706693
$SWCT_s$	2.07120107	0.00027078	1.00959981	0.74983437	0.02878235	0.63082039	0.88781902
$SWCT_t$	2.37799074	0.00102761	5.20690053	0.96573131	0.01219054	0.18990139	0.57235449
$SWCT_w$	2.98980312	0.00260377	2.10364228	1.07246453	0.02878312	0.635035	0.88203449
SWT	51928.5393	79.8570395	70232.6713	17302.8804	252.183505	8072.68375	15506.0479
$SWWT_s$	9483.32595	1.23816475	4625.67065	3428.61481	131.607167	2884.42388	4059.54911
$SWWT_t$	57432.7378	24.8186359	125755.973	23324.1353	294.423356	4586.45803	13823.3841
$SWWT_w$	21835.6954	19.0163116	15363.7198	7832.62434	210.214291	4637.90701	6441.84041
WTA_s	2512.9051	0.32818304	1225.71555	908.514074	34.8714633	764.305519	1075.70163
WTA_t	426.774849	0.1844239	934.475462	173.318482	2.18781989	34.081344	102.719681
WTA_w	815.061632	0.7098224	573.481949	292.368589	7.84667392	173.11928	240.45474
WWT_s	9483.32584	1.23816875	4625.67054	3428.6149	131.607161	2884.42384	4059.54893
WWT_t	57432.7373	24.8186447	125755.982	23324.1367	294.42337	4586.45815	13823.3843
WWT_w	21835.695	19.0163099	15363.72	7832.62422	210.214294	4637.90688	6441.84027

Tabla XVI. Grid 2. Promedio de rendimiento (1ª parte, 7 estrategias)

Strategy / Metric	MCT	MLB	LBal_S	LBal_T	LBal_W	MLp	MPL
ρ	0	0.3749	0.1919	0.1565	0.2328	0	0
SD	5733.85781	1031.41026	9.55710956	3586.59674	4468.53147	61.7715618	0
SD_b	4000.91673	753.857907	17.1122995	2569.06035	3345.68373	53.2467532	0
Th	0	0.3715035	0.18939394	0.15661422	0.23310023	0	0
TA	92.594558	34.4711317	1.33348641	80.3963635	110.027179	2.92558581	0
U	0	0.37367725	0.19152337	0.15652557	0.2323082	0	0
t_w	4833.37568	1799.39518	69.6539534	4205.42959	5743.40081	153.044823	0
$LBal_s$	1257.1918	1311.60932	218.613868	937.598032	1688.43351	243.631081	0
$LBal_t$	480.131893	760.167307	1416.37007	595.916522	1126.83421	0	1236.52361
$LBal_w$	221.675811	94.506211	0	667.844319	24.6802316	199.978016	117.34419
SCT	0.02299858	0.00856203	0.00033144	0.02001148	0.02732873	0.00072823	0
$SWCT_s$	0.0690725	0.03785983	0.00098173	0.17156948	0.10223302	0.00991792	0
$SWCT_t$	0.02103442	0.03191063	0.00066638	0.040762	0.08294961	0.00236776	0
$SWCT_w$	0.07019004	0.02372094	0.0018597	0.25286421	0.07508892	0.01256161	0
SWT	4833.43131	1799.41679	69.6569888	4205.4779	5743.46452	153.047098	0
$SWWT_s$	700.112451	383.743691	9.9506708	1739.01233	1036.2245	100.527129	0
$SWWT_t$	1486.04067	2254.42479	47.0786371	2879.75609	5860.22934	167.27765	0
$SWWT_w$	613.21397	207.237589	16.2472335	2209.14342	656.012851	109.744296	0
WTA_s	134.511953	73.7335269	1.9154017	333.876372	199.093766	19.2960121	0
WTA_t	3.15742213	4.79002355	0.100029	6.11867868	12.4513527	0.35541837	0
WTA_w	19.7691942	6.68106066	0.52378896	71.219813	21.148975	3.53800839	0
WWT_s	700.112495	383.743693	9.950678	1739.01241	1036.22461	100.527137	0
WWT_t	1486.04066	2254.42473	47.0786351	2879.75613	5860.22959	167.277663	0
WWT_w	613.21399	207.237591	16.2472329	2209.14347	656.012864	109.744295	0

Tabla XVII. Grid 2. Promedio de rendimiento (2ª parte, 7 estrategias)

Strategy / Metric	Random	MST	MSWCT_W	MWWT_S	MWT	MWWT_T	MWWT_W
ρ	1.6283	0	1.1991	0.2004	0	0.8579	4.425
SD	17688.4615	110.431235	4694.93007	17608.5664	998.659674	37339.9767	185601.34
SD_b	12195.7983	110.618793	3857.8304	7550.26738	628.495034	27638.2735	68410.3896
Th	1.6025641	0	1.18371212	0.20032051	0	0.84863054	4.2358683
TA	345.334887	4.82328025	124.79907	174.936339	17.8745589	590.455975	991.745918
U	1.60245811	0	1.18496473	0.20006614	0.00027557	0.85069444	4.23748898
t_w	18027.0292	251.771659	6514.57925	9131.50046	933.024939	30821.8067	51768.4001
$LBal_s$	990.47811	39.7744639	622.541018	65.6380674	26.5269451	1282.46259	1158.41856
$LBal_t$	1287.19195	1291.63674	2292.90296	1258.15587	386.862773	312.821578	1191.0024
$LBal_w$	460.519876	0.83597828	228.608817	72.3275965	78.0312279	157.574316	78.0067729
SCT	0.08577786	0.001198	0.0309982	0.04345036	0.0044396	0.14665908	0.24632455
$SWCT_s$	0.23459343	0.00380335	0.08536455	0.13920786	0.02834148	0.41593423	0.34994405
$SWCT_t$	0.27360328	0.00136496	0.08870661	0.11203461	0.00760764	0.21405489	0.37308536
$SWCT_w$	0.3235156	0.00577079	0.11154632	0.23188241	0.04067376	0.49451771	0.39902051
SWT	18027.2293	251.775978	6514.65379	9131.60247	933.037325	30822.1471	51768.9715
$SWWT_s$	2377.81735	38.5503588	865.247212	1410.9979	287.266602	4215.87081	3547.00048
$SWWT_t$	19329.5413	96.4318942	6266.95012	7915.02878	537.465032	15122.5631	26357.7615
$SWWT_w$	2826.38812	50.4163601	974.522331	2025.83629	355.345473	4320.34451	3486.03497
WTA_s	456.875455	7.40720979	166.23698	271.116272	55.1914358	810.069238	681.552554
WTA_t	41.0698877	0.20489089	13.315522	16.8172286	1.14196343	32.1312308	56.0028867
WTA_w	91.1189459	1.62535592	31.4172884	65.3102375	11.4558611	139.282102	112.385084
WWT_s	2377.81754	38.5503618	865.24729	1410.99796	287.266633	4215.87122	3547.00077
WWT_t	19329.541	96.4318942	6266.94995	7915.02866	537.465033	15122.563	26357.7619
WWT_w	2826.38821	50.4163594	974.522315	2025.83635	355.345461	4320.34473	3486.03494

Tabla XVIII. Promedio de rendimiento Grid 1 y Grid 2 (1ª parte, 7 estrategias)

Strategy / Metric	MCT	MLB	LBal_S	LBal_T	LBal_W	MLp	MPL
ρ	0.00145	0.37595	0.09595	16.11445	0.6613	1.04015	0
SD	3617.508615	749.990842	14.8613705	52768.6814	2743.12702	77.40752	0
SD_b	2642.656494	580.776743	17.3146531	39005.406	2122.35292	62.1038188	0
Th	0	0.37068959	0.09469697	12.2174215	0.65595215	1.01715812	0
TA	69.55292865	25.8072032	0.87679298	2563.89625	73.6294743	3.65881797	0
U	0.001385321	0.37461254	0.09576168	12.2202204	0.65516977	1.0189663	0
t_w	3044.781095	1131.2016	40.4971957	70263.949	3374.48515	135.84304	0
$LBal_s$	2112.09097	2158.86449	194.784333	1913.31506	2596.62641	415.992031	118.332915
$LBal_t$	611.9274289	1516.45275	2033.39832	988.807176	1827.18569	0	2046.00481
$LBal_w$	134.4716241	72.2257002	15.6909353	1066.49085	30.4514879	231.277955	82.3014832
SCT	0.030823515	0.01140359	0.00034018	2.10354552	0.02913328	0.00218921	0
$SWCT_s$	0.075818836	0.0393329	0.00049086	8.18379423	0.09090448	0.01919899	0.00070262
$SWCT_t$	0.026049031	0.0248039	0.00053868	4.43852123	0.06229536	0.00805121	0
$SWCT_w$	0.078151012	0.02012689	0.00092985	14.0733982	0.05264745	0.01715102	0.00022127
SWT	3044.808776	1131.21241	40.4987841	70263.9643	3374.51696	135.844245	0
$SWWT_s$	538.8206757	285.164402	4.9753354	37910.2709	700.042642	115.376034	3.21270475
$SWWT_t$	1118.141635	1340.92153	28.5023447	108145.876	3432.9683	249.497053	0
$SWWT_w$	681.7282828	317.327928	13.0866429	107810.569	830.860059	220.730376	0
WTA_s	117.2747452	61.5874157	0.95770085	9980.91135	147.754414	26.9001743	0.85126927
WTA_t	4.366186171	3.98305518	0.08689403	795.977037	9.9623128	1.41017914	0
WTA_w	21.62225272	5.59406949	0.26189448	3837.74496	14.6917706	4.7323743	0.06032164
WWT_s	538.8206985	285.164403	4.975339	37910.2701	700.042705	115.376037	3.21270525
WWT_t	1118.141645	1340.9215	28.5023435	108145.877	3432.96845	249.497078	0
WWT_w	621.0616119	163.991642	8.12361644	102964.679	438.309423	134.261536	1.61602992

Tabla XIX. Promedio de rendimiento Grid 1 y Grid 2 (2ª parte, 7 estrategias)

Strategy / Metric	Random	MST	MSWCT_W	MWWT_S	MWT	MWWT_T	MWWT_W
ρ	11.56065	0	14.08745	3.6811	0.07205	1.24555	5.79355
SD	35714.9347	105.381249	37224.4423	17633.1652	617.300851	22967.7109	100563.755
SD_b	27059.9527	100.653784	25839.2256	10063.2289	416.267075	17510.1231	40649.5826
Th	9.64747541	0	11.2155076	3.43931571	0.07192027	1.22571258	5.4570896
TA	1134.01244	3.89000712	1362.60623	407.793524	13.6059338	444.676239	782.933698
U	9.64662727	0	11.2148691	3.44167833	0.07204852	1.22883319	5.46051568
t_w	34977.7883	165.814362	38373.6306	13217.1916	592.604303	19447.2461	33637.2251
$LBal_s$	1482.93898	19.8872319	1186.12437	166.373912	49.5154742	2213.57517	1768.99282
$LBal_t$	1925.79155	1933.19393	4389.72121	2093.64365	532.405075	410.942816	1814.52615
$LBal_w$	800.682229	0.41798914	671.373834	164.800109	44.3663817	204.24738	223.503498
SCT	0.8399307	0.00182746	1.09353623	0.28789983	0.0060992	0.19751364	0.36169574
$SWCT_s$	1.15289725	0.00203707	0.54748218	0.44452111	0.02856191	0.52337731	0.61888154
$SWCT_t$	1.32579701	0.00119629	2.64780357	0.53888296	0.00989909	0.20197814	0.47271993
$SWCT_w$	1.65665936	0.00418728	1.1075943	0.65217347	0.03472844	0.56477636	0.6405275
SWT	34977.8843	165.816509	38373.6625	13217.2415	592.610415	19447.4154	33637.5097
$SWWT_s$	5930.57165	19.8942618	2745.45893	2419.80636	209.436884	3550.14735	3803.2748
$SWWT_t$	38381.1396	60.6252651	66011.4617	15619.582	415.944194	9854.51055	20090.5728
$SWWT_w$	12331.0418	34.7163358	8169.12106	4929.23031	282.779882	4479.12576	4963.93769
WTA_s	1484.89028	3.86769641	695.976267	589.815173	45.0314495	787.187379	878.627091
WTA_t	233.922369	0.19465739	473.895492	95.0678551	1.66489166	33.1062874	79.3612836
WTA_w	453.090289	1.16758916	302.449618	178.839413	9.6512675	156.200691	176.419912
WWT_s	5930.57169	19.8942653	2745.45892	2419.80643	209.436897	3550.14753	3803.27485
WWT_t	38381.1391	60.6252695	66011.4661	15619.5827	415.944201	9854.51057	20090.5731
WWT_w	12331.0416	34.7163347	8169.12114	4929.23028	282.779878	4479.1258	4963.93761

A.5. Modelos *C1*, *C2*, *U1*, *U2* para predicción del tiempo de ejecución.

En esta sección se presenta el análisis experimental de la predicción del tiempo de ejecución incorporando los modelos *C1*, *C2*, *U1*, *U2* en las estrategias de asignación de trabajos. Se presenta el rendimiento de la estrategia *MST* para cada métrica. Después se evalúa la degradación en rendimiento de los modelos bajo cada métrica (mostrada como %). Estos se hace en relación al caso con la precisión de la predicción generada por el sistema igual a 1 ($p_j'' = p_j$). La constante $c2 = 0.9$ es una constante que fue obtenida con base en el análisis de la degradación de rendimiento de *Min_ST*. La Figura 26 muestra los resultados de 100 experimentos con variación de *c2* desde 1% hasta 100% de Grid 1 y Grid 2.

El valor de $c1 = 0.4816$ es el promedio de la relación p_j/p_j' de todos los trabajos en la bitácora de Grid 1. $L1 = 4.3403e^{-006}$ y $R1 = 241.733$ son calculados como $\min_j(p_j/p_j')$, y $\max_j(p_j/p_j')$ de la bitácora para Grid 1 respectivamente. $L2 = 0.0748$ es el promedio de la mitad de valores más pequeños de p_j/p_j' de la bitácora para el Grid 1, y $R2 = 0.8884$ es el promedio de la mitad de valores más grandes de p_j/p_j' de la bitácora del Grid 1. Para el Grid 2, $c1 = 0.2288$, $L1 = 9.5e^{-007}$, $R1 = 241.733$, $L2 = 0.0129$ y $R2 = 0.4447$.

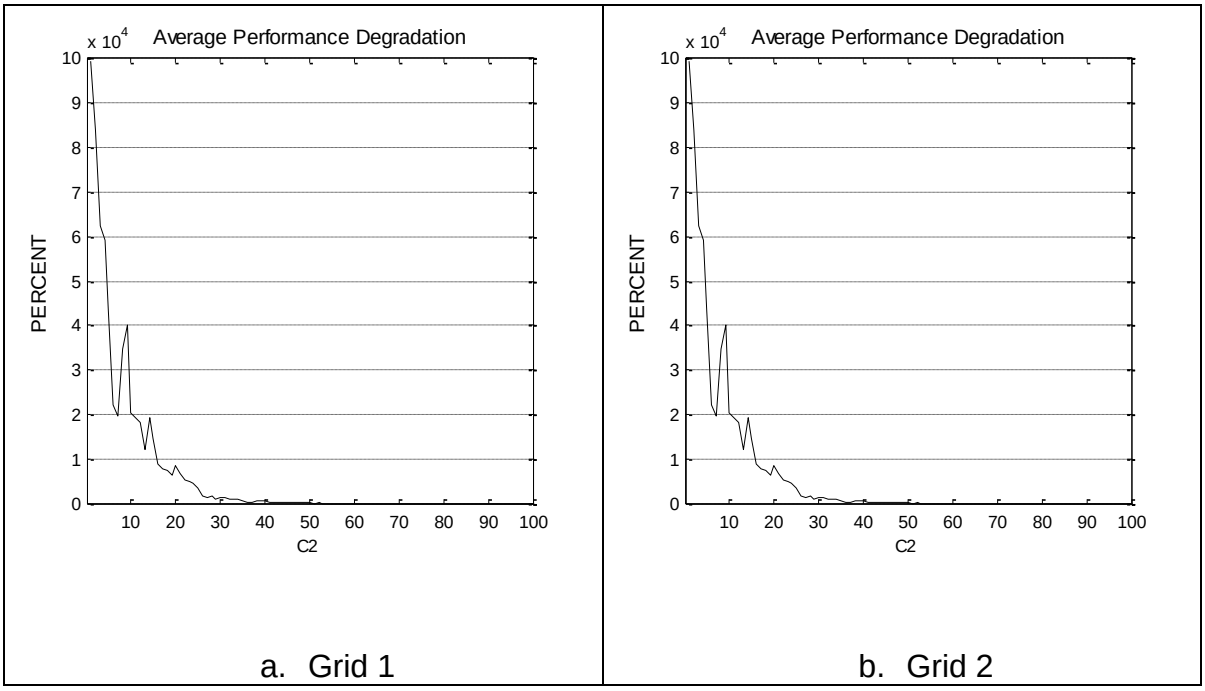


Figura 34. *MST* Degradación de rendimiento con $p_j'' = c2 \cdot p_j'$, variando *c2* de 1% a 100%

A.6. Simulador GSimula

En esta sección se describe de manera general la estructura del simulador GSimula (Grid Simulator, Simulador de Grid), desarrollado para realizar las simulaciones de la presente investigación.

GSimula es un simulador de eventos discretos desarrollado en lenguaje C y puede ser compilado con Borland C en Windows o con gcc en Linux. La Figura 27 muestra los elementos básicos del simulador.

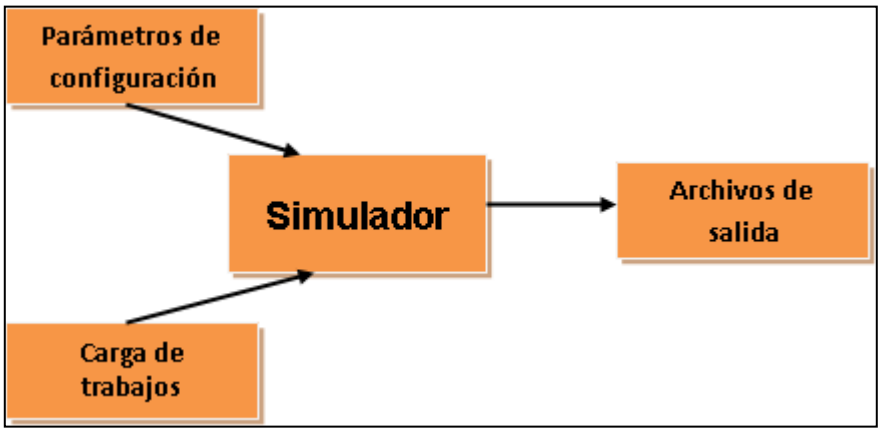


Figura 35. Componentes básicos de GSimula

Los *parámetros de configuración* se proveen al simulador mediante tres archivos de texto que se modifican manual e independientemente de la ejecución del simulador. La Tabla XX describe de manera general los tres archivos de configuración. Cada parámetro se define con el caracter "&" y en la línea siguiente al parámetro, aparece su valor correspondiente (ver Apéndice A.7), por ejemplo:

&number of experiments
1

Tabla XX. Descripción de los archivos de configuración de GSimula

Archivo	Descripción
Config.ini	Este archivo contiene los parámetros iniciales de configuración para la simulación. Los principales parámetros son: tipo de modelo (en línea o fuera de línea), número de experimentos, número de datos o días a considerar, estrategias de asignación de recursos, algoritmos de calendarización local y métricas de evaluación.
System_runtime_ prediction.ini	Este archivo contiene los parámetros para la configuración de los modelos de predicción de tiempos de ejecución. Se puede elegir, por ejemplo el modelo *Hrecent_k*, el tamaño de ventana y las condiciones que deben cumplir los trabajos históricos para ser considerados en el promedio.
Grid_config.ini	Este archivo contiene los parámetros para configurar las características de los recursos que componen el Grid considerado. Esencialmente se consideran el número de sitios y el tamaño de cada uno.

La versión actual de GSimula considera las extensiones hechas principalmente por José Luis González García y Ariel Quezada Pina, alumnos de maestría y doctorado respectivamente, para soportar admisibilidad, Grid de tres niveles y la estrategia de *robo (*stealing).

La *carga de trabajos (workload)* introducida al simulador debe estar en el formato estándar SWF (Standar Workload Format) descrita en *Parallel Workloads Archive* (Feitelson, 2005b) que considera los 18 campos mostrados en la Tabla XXI, y ordenados por tiempo de llegada.

Tabla XXI. Campos del Formato estándar para carga de trabajo (SWF)

1. Número de trabajo	Tiempo de llegada	Tiempo de espera	Tiempo de ejecución	#Procesadores asignados	Promedio de tiempo de CPU
7. Memoria utilizada	#Procesadores requeridos	#Tiempo estimado	Memoria requerida	Estado	Identificador de usuario
13. Identificador de grupo	Número de aplicación	Número de cola	Número de partición	Trabajo precedente	Tiempo entre trabajos consecutivos

El simulador se ejecuta desde la línea de comandos con la siguiente sintaxis:

>gsimula [config.ini][,interactive.txt][,system_runtime_prediction.ini][,grid_config.ini]

Los nombres y el número de parámetros pueden cambiar, si no se especifican se toman los indicados en la sintaxis. El archivo *interactive.txt* se utiliza para almacenar todas las salidas del programa cuando el simulador se configura para no requerir la interacción del usuario durante su ejecución, por ejemplo para contestar alguna pregunta que permita que el programa continúe.

La Figura 28 muestra el algoritmo general del núcleo del simulador. Los calendarios tentativos elaborados en el nivel de asignación son independientes en cada sitio. Para cada uno existe un conjunto de estructuras independientes, esto implica que cada uno puede tener diferentes relojes de ejecución. Por cada trabajo por asignar se ejecuta el algoritmo de calendarización en cada sitio y se calculan todas las métricas implementadas. Dependiendo de la estrategia considerada, será la métrica verificada para seleccionar el sitio al cual se le asignará el trabajo.

```
1  Leer archivos de configuración
2  Inicializar variables del modelo
3  Para cada experimento
4          Leer trabajos de la bitácora (Instancia I)
5          Para cada estrategia E de selección de recursos especificada en la configuración inicial
6                  Para cada algoritmo de calendarización local C especificado en la configuración inicial.
7                          Inicializar variables de calendario y métricas
8                          Inicializar reloj
9                          Para cada conjunto A de trabajos  con tiempo de llegada igual al instante actual (reloj)
10                                 Solicitar información de la carga actual en cada máquina.
11                                 Para cada trabajo  j del conjunto A (evento llegada de trabajo)
12                                         Ejecutar algoritmo de estrategia de selección
13                                         Seleccionar el sitio con el menor valor del parámetro en que se basa
                                           la estrategia.
14                                         Asignar el trabajo j a la cola del recurso elegido
15                                 FinPara
16                                 Para cada sitio del Grid
17                                         Ejecutar el algoritmo de calendarización local C.
18                                         Determinar el próximo evento de finalización.
19                                 FinPara
20                                 Encontrar el menor evento de finalización de todos los sitios.
21                                 Si evento de próxima llegada de trabajo < próxima finalización
22                                         Para cada sitio del Grid
23                                                 Tiempo del calendario en cada procesador = próxima
                                                   llegada de trabajo
24                                                 Reloj = próxima llegada de trabajo
25                                 Sino
26                                         Para cada sitio del Grid
27                                                 Finalizar trabajos en cada procesador
28                                                 Tiempo de calendario en cada procesador = próxima
                                                   finalización.
29                                                 Reloj = próxima finalización
30                                 Finsi
31                         FinPara
32                         Calcular métricas finales (del Grid y por sitio)
33                         Almacenar resultados
34                 FinPara
35         FinPara
36 FinPara
```

Figura 36. Algoritmo general del núcleo de GSimula

Los *archivos de salida* están organizados como se describe a continuación. Para cada ejecución del simulador se genera una carpeta cuyo nombre se basa en algunos de los parámetros configurados. Por ejemplo, la carpeta *MIX_5_LOGS_30_P(180_DAYS)_O_GRID1* indica que la simulación consideró la

bitácora *mix_5_logs*, *30* experimentos, 180 días por experimento, para un sistema *en línea* para el Grid con etiqueta *Grid1* (definido en el archivo *grid_config.ini*). Dentro de esta carpeta se crea una carpeta por cada una de las estrategias seleccionadas en el archivo de configuración, por ejemplo la carpeta *CT-FCFS-User(C_1)-BEFF,* almacena los resultados para la métrica *Min completion time*, con calendarizador de nivel de asignación *FCFS*, predicción del tiempo de ejecución basada en el tiempo estimado por el usuario (*User*) multiplicado por la constante 1 (*C_1*) (implica que la predicción es igual a la estimación del usuario); el algoritmo de calendarización local utilizado es el algoritmo de relleno *EASY* (*BEFF, Backfilling EASY First Fit*). Cada una de las carpetas correspondientes a las estrategias almacena un archivo de texto por cada una de las métricas seleccionadas en el archivo de configuración *config.ini*. Cada archivo almacena un dato por cada experimento. Se puede elegir almacenar también las métricas para cada sitio, en cuyo caso, cada archivo almacena una matriz de valores con columnas igual al número de sitios y renglones igual al número de experimentos realizados. Para el análisis de estos archivos se generan gráficas basadas en el programa de graficación *gsimula_stats* elaborado en Matlab por José Luis González García.

Diversas personas colaboraron en diferentes etapas del desarrollo de GSimula, se agradece su colaboración a: Elisa Lizeth Salazar Ricarte, Manuel Valenzuela Arce, y Daniel Mendoza Camacho, en su momento, alumnos de la Universidad de Sonora, México; a David Madrigal Angulo, Luis Manuel Galaviz Flores, Roberto Sánchez Sánchez, y Ana Isabel Rodríguez Barragán, en su momento, alumnos del Tecnológico de Colima, México; a José Luis González García y Yair Castro García, exalumnos de la Maestría en Computación del CICESE; y a Ariel Quezada Pina actual alumno del Doctorado en Computación.

A.7. Archivos de configuración para el programa de simulación

En esta sección se muestran los archivos de configuración que especifican las características de los escenarios, así como estrategias, algoritmos y métricas que se simularán durante los experimentos. Dividimos los parámetros en 3 archivos: *config.ini*, *system_runtime_prediction.ini* y *grid_config.ini*. Los nombres de los parámetros configurables en cada archivo inician con un "&". El valor asignado a cada parámetro viene inmediatamente después del nombre del parámetro sin algún caracter precedente. El caracter ";" indica una línea de comentario.

CONFIG. INI

```
&number of experiments
1

&first experiment
1
;set the number of experiment to begin the simulation,
;1 to perform all the number of experiments from the beginning of the ;log
; "&first experiment" must be less than or equal to "&number of ;experiments"

&period of time
yes
;no : parameter "&number of jobs or days" is fixed number of jobs per ;experiment
;yes: parameter "&number of jobs or days" is number of days per ;experiment

&number of jobs or days
180
;It can express number of jobs or number of days, dependig on "&period ;of time" parameter value

&period alignment unit
0
; This parameter define whether "&number of jobs or days" parameter is ;part of a week, month or year
; e.g. If we define "&period_of_time"=yes, "number_of_jobs_or_days"=4 ;and "&period alignment unit"=1,
; the experiments will take the first 4 days of each week, starting in ;the first day of the log.
; 0 = No alignment
; 1 = Week
; 2 = Month     (Note: Considering each month as 30 days)
; 3 = Year

&jobs not considered
0
; This parameter define the percent of jobs that doesn't will be
; included in metrics computation.
; 0 = All jobs will be included in metrics computation
```

CONFIG. INI

```
; 1 = First and last 1% of "&number of jobs" scheduled jobs doesn't be
; included in metrics.
; 10 = First and last 10% of "&number of jobs" scheduled jobs doesn't be
; included in metrics.

&scheduling type
0
;    0 = online
;    1 = batch

&system runtime prediction
yes
;    yes = Jobs runtimes using in allocation level will be predicted
; based on different parameters, these parameters are specified on
; system_runtime_prediction.ini file
;    no  = Jobs runtimes using in allocation level will be the same that
; requested runtimes submitted by user originally.
; For execution level: Jobs runtimes = Real runtimes.


&first job selection
1
;    1 = FIFO
;    2 = max size first (widest job first,Max_SF)
;    3 = min size first (narrowest job first,Min_SF)
;    4 = Max exec time first (longest job first,Max_TF)
;    5 = Min exec time first (shortest job first,Min_TF)
;    6 = Max work first (biggest job first,Max_WF)
;    7 = Min work first (smallest job first.Min_WF)

&last job selection
1
;    1 = FIFO
;    2 = max size first (widest job first,Max_SF)
;    3 = min size first (narrowest job first,Min_SF)
;    4 = Max exec time first (longest job first,Max_TF)
;    5 = Min exec time first (shortest job first,Min_TF)
;    6 = Max work first (biggest job first,Max_WF)
;    7 = Min work first (smallest job first.Min_WF)

&total selection strategies
36
; Total number of resource selection strategies

&selection strategies list
1,2,3,4,5,6,7,8,9,10,18,21,30,31
; List of choosen selection strategies from the following options
; (e.g  4,5,8,3,2  to select Min_LBal,Min_LB,Min_WT,Min_PL y Min_Lp)
; values are in the range from 1 to "&total selection strategies"
;    1 = random
;    2 = min load per-proc (least loaded proc site,Min_Lp)
;    3 = min threaded load-per-proc (min-parallel-load-per-proc,Min_PL)
```

CONFIG. INI

```
;    4 = min load balance, Min_LBal (SIZE)
;    5 = min lower bound, Min_LB
;    6 = min completion-time, Min_CT
;    7 = min waiting time, Min_WT
;    8 = min weighted waiting time (weight is the job size, Min_WSWT)
;    9 = min weighted waiting time (weight is the job runtime, Min_WTWT)
;    10 = min weighted waiting time (weight is Work=size_j*p_j,
;        Min_WWoWT)
;    11 = min sum waiting time, Min_SWT
;    12 = min sum weighted waiting time (weight is the job size,
;        Min_SWSWT)
;    13 = min sum weighted waiting time (weight is the job runtime,
;        Min_SWTWT)
;    14 = min sum weighted waiting time (weight is Work=size_j*p_j,
;        Min_SWWoWT)
;    15 = min sum completion time, Min_SCT
;    16 = min sum weighted completion time (weight is the job size,
;        Min_SWSCT)
;    17 = min sum weighted completion time (weight is the job runtime,
;        Min_SWTCT)
;    18 = min sum weighted completion time (weight is Work=size_j*p_j,
;        Min_SWWoCT)
;    19 = min utilization, Min_U
;    20 = max utilization, Max_U
;    21 = min start-time, Min_ST
;    22 = min turnaround, Min_TA
;    23 = min weighted turnaround, (weight is the job size, Min_WSTA)
;    24 = min weightedTime turnaround, (weight is job runtime, Min_WTTA)
;    25 = min weightedWork turnaround, (weight is Work=size_j*p_j,
;        Min_WWoTA)
;   26 = stealing AHB-First Fit
;   27 = stealing AHB-Best Fit
;   28 = List Common Pool
;   29 = List Common Pool Best Fit
;   30 = min load balance work, Min_LBal (WORK)
;   31 = min load balance time, Min_LBal (TIME)
;   32 = biggest free, BFree        (Not implemented)
;   33 = best fit, BF       (Not implemented)
;   34 = min-first-fit, Min_FF  (Not implemented)
;   35 = min-best-fit, Min_BF   (Not implemented)
;   36 = best size, BS (Not implemented)

&selection scheduler
1
;    1 = FCFS
;    2 = LSF     (Largest  size first)
;    3 = SSF     (Smallest size first)
;    4 = LETF    (Largest exec-time- first)
;    5 = SETF    (Smallest exec-time-first)
;    6 = LWF     (Largest work first)
;    7 = SWF     (Smallest work first)
;    8 = BackFill-Easy-FirstFit
```

CONFIG. INI

```
;    9 = BackFill-Easy-BestFit                  (Not applicable at this moment)
;   10 = BackFill-FirstSizeFit                  (Not applicable at this moment)
;   11 = Backfill_flexible_first_fit    (Not implemented)
;   12 = Backfill_flexible_best_fit     (Not implemented)
;   13 = Backfill_conserv_first_fit     (Not implemented)
;   14 = Backfill_conserv_best_fit      (Not implemented)

&admissible flag
0
; 0 = Admissible will not be used.
; 1 = Admissible is applied

&first admissible selection
100
; percent of available range

&last admissible selection
100
; percent of available range

&step admissible selection
10
; step of the percent incrementing of admissible range

&first admissible strategy
1
;    1 = constant
;    2 = random
;    3 = not implemented
;    4 = not implemented

&last admissible strategy
1
;    1 = constant
;    2 = random
;    3 = not implemented
;    4 = not implemented

&first local scheduling
9
;    1 = FCFS
;    2 = LSF     (Largest  size first)
;    3 = SSF     (Smallest size first)
;    4 = LETF   (Largest exec-time- first)
;    5 = SETF   (Smallest exec-time-first)
;    6 = LWF    (Largest work first)
;    7 = SWF    (Smallest work first)
;    8 = BackFill-Easy-FirstFit (When It is looking for job to
;        backfilling, It takes job real runtime)
;    9 = BackFill-Easy-FirstFit_reqtime (When It is looking for job to
;        backfilling, It takes job requested runtime)
;   10 = BackFill-Easy-BestFit
```

CONFIG. INI

```
;   11 = BackFill-FirstSizeFit
;   12 = Backfill_flexible_first_fit    (Not implemented)
;   13 = Backfill_flexible_best_fit  (Not implemented)
;   14 = Backfill_conserv_first_fit (Not implemented)
;   15 = Backfill_conserv_best_fit   (Not implemented)

&last local scheduling
9
;    1 = FCFS
;    2 = LSF     (Largest  size first)
;    3 = SSF     (Smallest size first)
;    4 = LETF   (Largest exec-time- first)
;    5 = SETF   (Smallest exec-time-first)
;    6 = LWF    (Largest work first)
;    7 = SWF    (Smallest work first)
;    8 = BackFill-Easy-FirstFit (When It is looking for job to
;        backfilling, It takes job real runtime)
;    9 = BackFill-Easy-FirstFit_reqtime (When It is looking for job to
;        backfilling, It takes job requested runtime)
;   10 = BackFill-Easy-BestFit
;   11 = BackFill-FirstSizeFit
;   12 = Backfill_flexible_first_fit    (Not implemented)
;   13 = Backfill_flexible_best_fit  (Not implemented)
;   14 = Backfill_conserv_first_fit (Not implemented)
;   15 = Backfill_conserv_best_fit   (Not implemented)

&number of metrics
29
; See "&output metrics list" parameter

&output metrics type
1
; There are two types of metrics, execution and allocation, this option
; select which type of metrics
; will be generated and the next parameter (&output metrics) specify
; what metrics will be generated.
; 1 = Execution  (Only metrics from execution level will be generated)
; 2 = Allocation (Only metrics from allocation level will be generated)
; 3 = Both               (Boty type of metrics, execution and allocation, will
;     be generated)
; NOTE: No allocation metric are generated for strategies that do not use tentative schedule (e.g. Min_PL).

&scope of metrics
3
; 1 = Grid (Metrics for Grid will be generated)
; 2 = Node (Metrics for each node will be generated)
; 3 = Both (Both scope of metrics, node and Grid, will be generated)
;
; e.g. if &output metrics type=1 and &scope of metrics=1 only execution
; metrics by node will be generated
; NOTE 1: if any of metrics: competitive ratio, idle, utilization or
; throughput were selected in &output metrics list,
```

CONFIG. INI

```
; and &scope of metrics=1 an extra metric file will be generated. This
; extra file calculate the metric by node
;       using the Grid_Cmax value.
; NOTE 2: There are not metrics by node for Load Balance (size, time or
; work).

&output metrics list
3,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27
;    Output metrics list, the type must be selected in the previous
;    parameter (&output metrics type)
;           List of metrics, separated by comma, to be generated as
;    output(Choosed from the following list, in any order).
;           NOTE 1: Only selected metrics will be generated
;
;           -1 = All metrics will be generated (This value must be used alone)
;           1 = Work
;           2 = Cmax
;           3 = Competitive ratio
;           4 = Idle
;           5 = Utilization
;           6 = Throughput
;           7 = Turnaround
;           8 = Weighted(size) turnaround
;           9 = Weighted(Time) turnaround
;           10 = Weighted(Work) turnaround
;           11 = Waiting time
;           12 = Weighted(size) waiting time
;           13 = Weighted(Time) waiting time
;           14 = Weighted(Work) waiting time
;           15 = Sum waiting time
;           16 = Sum weighted(size) waiting time
;           17 = Sum weighted(Time) waiting time
;           18 = Sum weighted(Work) waiting time
;           19 = Sum completion time
;           20 = Sum weighted(size) completion time
;           21 = Sum weighted(Time) completion time
;           22 = Sum weighted(Work) completion time
;           23 = Slowdown
;           24 = Bounded slowdown
;           25 = Load Balance (size)
;           26 = Load Balance (time)
;           27 = Load Balance (work)
;           28 = Response time
;           29 = System response ratio



&workload source
5
;    1 = LOGS
;    2 = MODELS
;    3 = EXAMPLES
```

CONFIG. INI

```
;    4 = RANDOM_WORKLOAD
;    5 = MIXED_WORKLOAD

&workload name
1
;    If Workload Source = 1          If Workload Source = 2
;     1 = NASA_AMES_IPSC_860            1 = FEITELSON96
;     2 = SDSC_PARAGON_1995_B2          2 = LUBLIN99
;     3 = SDSC_PARAGON_1996_B2
;     4 = CTC_SP2
;     5 = LANL_CM_5                 If Workload Source = 3
;     6 = SDSC_SP2                  1 = EXAMPLE1
;     7 = OSC_CLUSTER                   2 = EXAMPLE2
;     8 = SDSC_BLUE_HORIZON             3 = EXAMPLE3
;     9 = DAS2_FS0_2003
;    10 = DAS2_FS3_2003
;    11 = DAS2_FS4_2003             If Workload Source = 4
;    12 = LPC_EGEE_2004                1 = RANDOMLOG1
;    13 = SDSC_DS_2004_B2              2 = RANDOMLOG2
;    14 = SDSC_DS_2004_B3

;   If Workload Source = 5
;    1 = mix_7_logs
;    2 = mix_5_logs
;    3 = mix_6_logs
;    4 = mix_8_logs
;    5 = mix_13_logs

&grid number
1
; 1 = Grid1
; 2 = Grid2
; 3 = Grid3
; 4 = Grid4
; 5 = Grid5
; Configuration of each Grid is described in file grid_config.ini

;---------------
;begin - 3 level hierarchical grid scheduling

&step stealing
100
;          50, FOR ORIGINAL STEALING ALGORITHM OR SET VALUE AS
;          0 =< VALUE <= 100

&grid scheduling levels
2
;          2 = traditional one/two level hierarchical grid scheduling
;          3 = 3 level hierarchical grid scheduling (stealing in super-
; clusters is used)

&SC resource selection
```

CONFIG. INI

```
1
;   1 = stealing AHB-First Fit
;   2 = stealing ABH-First Fit
;   3 = stealing AHB-Best Fit
;   4 = stealing ABH-Best Fit

&number of super-clusters
0
; super-cluster is a group of nodes (clusters) that acts like a sub-grid
; and stealing will be used to schedule its jobs

&nodes in super-clusters
0,1,2,3,
,
; each line represents a super-cluster and its nodes (nodes sorted in
; non decreasing order in each super-cluster)
; the last one is a fake super-cluster and has all nodes that don't
; belong to a super-cluster
; each line must finish with a comma (,). Cluster numbers begin in zero.

&path local directory
Results
; the relative path where the results will be saved in

&interactive
1
;           1 = TRUE:       GSimula will show information in the command window,
;              or ask the user to answer questions.
;           0 = FALSE:      GSimula will run without interaction with the user.
;                           All messages will be saved in a file in the directory
;              specified by "&path local directory"
;                           All questions will be answered with "YES"
;              automatically

&save workload file
0
;           1 = TRUE:       GSimula will create the file "workload.txt".
;
;           0 = FALSE:      GSimula will not save the workload file.

&save events log
0
;           1 = TRUE:       GSimula will create the files with all events logs.
;
;           0 = FALSE:      GSimula will not save the events log files.

&save queue size log
0
;           1 = TRUE:       GSimula will create the files with all queue sizes
;              logs.
;
;           0 = FALSE:      GSimula will not save the queue sizes logs.
```

CONFIG. INI

```
;
;
;end - 3 level hierarchical grid scheduling
;---------------

&save jobs per node
0
;           1 = TRUE:         GSimula will create the file
;              "allocated_jobs_per_node.txt".
;           0 = FALSE:        GSimula will NOT save the index of jobs executed on
;              each node.

&save executed jobs runtimes
0
;           1 = TRUE:         GSimula will create the file(s)
;                             "user_executed_jobs_runtimes_<Type of
;              estimation>.txt".
;           0 = FALSE:        GSimula will NOT save runtimes of user executed jobs.

&save executed jobs history
0
;           1 = TRUE:   GSimula will create the file
;              "user_executed_jobs_history.txt" that contain
;                             index of all executed jobs by each user
;           0 = FALSE:        GSimula will NOT save history of user executed jobs.

&save jobs size histogram
0
;           1 = TRUE:         GSimula will create the file "size_histogram.txt" that
;              contain the number of jobs by size in each experiment.
;           0 = FALSE:        GSimula will NOT save the job size histogram.

&save work by jobs size
0
;           1 = TRUE:         GSimula will create the file "RC_by_each_job_size.txt"
;              that contain the sum of work by jobs size in each
;              experiment.
;           0 = FALSE:        GSimula will NOT save the sum of work by job size.
```

SYSTEM_RUNTIME_PREDICTION.INI

```
&estimated time source
0
;   0 = Estimated by user (requested_time field will be used)
;   1 = Runtime (Real runtime will be used)
;   2 = Model k_last
;       This model take in account only execution runtimes of k last
;       executed jobs by the same user, if they exist, if not,
;       requested_runtime is used.
```

SYSTEM_RUNTIME_PREDICTION.INI

```
;    3 = Model k_at_most
;        This model take in account execution runtimes of at most k
;        executed jobs by the same user.
;              The average of executed runtimes is computed with at most k
;        jobs, even with one job.
;    4 = Not exist estimation-k_last. You must provide the initial
;        value. Below "&initial_estimation"
;              After initial estimation, model k_last will be applied.
;    5 = Not exist estimation-k_at_most. You must provide the initial
;        value. Below "&initial_estimation"
;              After initial estimation, model k_at_most will be applied.
;
; NOTE: For options 2 and 3, jobs can be conditioned by some
;       characteristic(s). (See "&k model conditions")
;       For k_last, conditions are applied to the last k jobs, only. All
;       last k jobs must fulfil conditions to compute the average, if
;       not, requested_time will be used.
;       For k_at_most, conditions are applied to last w (See "&window
;       size historic jobs") executed jobs, from last to first, until
;       get k match jobs. Average compute is applied to these match
;       jobs, even if are less than k.
;       For option 4 and 5, only conditions = 0 or 2 could be applied
;       because we don't have a valid value for requested_time.
;
; This parameter work together with "&k model conditions", "&num jobs
; history prediction", "&min bound for similars jobs" and "&max bound
; for similars jobs" parameters.

&k model conditions
0
;  0 = No condition will be applied to executed jobs.
;  1 = Requested_time condition (It computes average with runtimes of
;      executed jobs only
;      if their requested_time is similar to requested_time of current
;      job).
;  2 = Size condition (It computes average with runtimes of executed
;      jobs only if their size is similar to size of current job).
;  3 = Requested_time & size condition (It computes average with
;      runtimes of executed jobs only
;      if their requested_time and size are similar to requested_time
;      and size of current job).
;  4 = Work condition (It computes average with runtimes of executed
;      jobs only if their work (requested_time * size) is similar to
;      work of current job). (NOT IMPLEMENTED YET)
; NOTE: For interpretation of "similar" see "&constant for range width"

&num jobs history prediction
0
; This is the value for k constant on k models.
; Number of historic jobs execution to be considered for average
; computation.
;    0 = all jobs
```

SYSTEM_RUNTIME_PREDICTION.INI

```
;    1 = one job
;    2 = two jobs
;       ...
;  n-1 = n-1 jobs

&min bound for similar jobs
0
; This parameter define a percentage for lower bound of range for
; consider "similiar jobs",
; based on condition field (see "&k model conditions").
; Instances:
;   0 --> <condition field of historical jobs> must be equal to
;         <condition field of current job>
;   10 --> lower bound = <condition field of current job> * (1 - 10/100)
;   80 --> lower bound = <condition field of current job> * (1 - 80/100)
;
; <condition field of historical jobs> must be into [min_bound,
; max_bound] range.
;
;   This parameter must be into [0,100]

&max bound for similar jobs
0
; This parameter establish a max bound of range for consider "similar
; jobs",
; based on condition field (see "&k model conditions").
; Instances:
;   0 --> <condition field of historical jobs> must be equal to
;         <condition field of current job>
;   20 --> upper bound = <condition field of current job> * (1 + 20/100)
;   100 -> upper bound = <condition field of current job> * (1 +100/100)
;   300 -> upper bound = <condition field of current job> * (1 +300/100)
;
; <condition field of historical jobs> must be into [min_bound,
; max_bound] range.
;
;   This parameter must be >= 0

&window size historic jobs
0
; This parameter is applied on model k_at_most only and it define the
; number of jobs in which the search will continue
; to reach k jobs (defined in "&num jobs history prediction" parameter)
; that fulfil condition(s) established in "&k model condition"
; parameter when these number of jobs were not completed into the last k
; historic jobs.
;            -1 = Seek in all executed jobs
;            0 = No one job more, stop searching.
;            1 = Seek in 1 last job (after k jobs)
;            2 = Seek in 2 last jobs (after k jobs)
;            10 = Seek in 10 last jobs (after k jobs)
```

SYSTEM_RUNTIME_PREDICTION.INI

```
&type of estimated time fluctuation
0
; when "estimated time" = 0 or 1
;    0 = Constant
;    1 = Uniform
;    2 = Normal
;    3 = Rule1
;    4 = Rule2

&constant fluctuation coefficient
1
; when "type of estimated time fluctuation" = 0
; instances:
;  1.2 = increase 20% of original value
;    1 = no change
; 0.75 = decrease 25% of original value

&mean fluctuation coefficient
0.75
; used to calculate the mean of the distribution, mean=q*x,
; where  x = user estimated runtime or real runtime
; and q is this coefficient
; 0-2

&percent of fluctuation range
25
; used to calculate the range of changing,
; for normal distribution  range=x*p/100, where  mean-range =3*sigma
; (normal distribution)
; standard deviation= sigma = (x * percent of fluctuation range/100)/3
; for uniform distribution range = mean +/- x*percent of fluctuation
; range/100
; 0-100

&initial estimation
160000
; When "estimated time source" = 4 (Not exist estimation)
; This value will be the initial value to the first estimation of each
; job after that
;
; 160000 is the max runtime of the 10000 jobs from
; mix_7_logs_10000(NewMixer)_cln2.swf

;&rule1
; For required parameter of some strategy proposed
; To use this parameter, uncomment the line "&rule2", define values here
; and define variable in "global.h" to assign this value in the program.

;&rule2
; For required parameter of some strategy proposed
; To use this parameter, uncomment the line "&rule2", define values here
; and define variable in "global.h" to assign this value in the program.
```

GRID_CONFIG.INI

```
;GRID nodes size must be ordered in non decreasing order

;--------------  GRID 1  ---------------------------
&max job size allowed_GRID1
1368

&number of nodes_GRID1
7

&size of nodes_GRID1
100
128
240
430
1024
1152
1368

;node 1: The Swedish Royal Institute of Technology (KTH) IBM SP2 ;        100 processors
;node 2: IBM SP2, San Diego Supercomputer Center (SDSC)128 processors
;node 3: HPC2N - High Performance Computing Center North, Sweden ;        240 processors
;node 4: IBM SP2, Cornell Theory Center (CTC) 430 processors
;node 5: CM-5, Los Alamos National Lab (LANL) 1024 processors
;node 6: IBM SP3 Blue Horizon, Diego Supercomputer Center (SDSC) ;        1152 processors
;node 7: SDSC DataStar IBM p655/p690 Partition: 2(Batch partition) ;        1368 processors

;---------------    GRID 2  ----------------------------------
&max job size allowed_GRID2
1024

&number of nodes_GRID2
9

&size of nodes_GRID2
64
64
64
64
100
144
240
430
1024

;node 1: DAS2 - Leiden University Pentium-III Linux cluster (fs1) ;        64 processors
;node 2: DAS2 - University of Amsterdam (UvA) Pentium-III Linux;        cluster(fs2)        64 processors
;node 3: DAS2 - Delft University of technology. Pentium-III Linux ;        cluster(fs3)64 processors
;node 4: DAS2 - Utrecht University. Pentium-III Linux cluster (fs4);        64 processors
;node 5: The Swedish Royal Institute of Technology (KTH) IBM SP2    ;        100 processors
;node 6: DAS2 - Vrije University Amsterdam (fs0).Pentium-III Linux;        cluster.            144 processors
;node 7: HPC2N - High Performance Computing Center North, Sweden;        240 processors
```

GRID_CONFIG.INI

```
;node 8: Cornell Theory Center (CTC). IBM SP2 ;        430 processors
;node 9: Los Alamos National Lab (LANL). Thinking Machines CM-5;        1024 processors

;------------------ GRID 3  -----------------------------
&max job size allowed_GRID3
1368

&number of nodes_GRID3
13

&size of nodes_GRID3
64
64
128
128
140
144
178
224
352
430
1024
1152
1368

;node 1: DAS2 - Delft University of technology (fs3). Pentium-III Linux;        cluster. 64 processors
;node 2: DAS2 - Utrecht University (fs4). Pentium-III Linux cluster ;        64 processors
;node 3: IBM SP2, San Diego Supercomputer Center (SDSC), 128 processors
;node 4: Intel iPSC/860, NASA Ames Research Center, 128 processors
;node 5: LPC(Laboratoire de Physique Corpusculaire.3GHz Pentium-IV Xeon;        Linux Cluster 140
processors
;node 6: DAS2 - Vrije University Amsterdam (fs0).Pentium-III Linux;        cluster, 144 processors
;node 7: Linux Cluster, Ohio Supercomputer Center  (OSC) 178 processors
;node 8: SDSC DataStar. IBM p655/p690. Partition: 3 Batch partition;        224 processors
;node 9: Intel Paragon,San Diego Supercomputer Center (SDSC);        352 processors
;node 10: IBM SP2, Cornell Theory Center (CTC),  430 processors
;node 11: CM-5, Los Alamos National Lab (LANL), 1024 processors
;node 12: IBM SP3 blue horizont, Diego Supercomputer Center (SDSC);        1152 processors
;node 13: SDSC DataStar. IBM p655/p690. Partition: 2 Batch partition
;        1368 processors

;-----------------    GRID 4   ------------------------

&max job size allowed_GRID4
10

&number of nodes_GRID4
2

&size of nodes_GRID4
5
10
```

GRID_CONFIG.INI

```
; For test

;------------------    GRID 5  --------------------------

&max job size allowed_GRID5
1368

&number of nodes_GRID5
8

&size of nodes_GRID5
100
128
140
224
430
1024
1152
1368

;node 1: The Swedish Royal Institute of Technology (KTH) IBM SP2;        100 processors
;node 2: SDSC IBM SP2, 128 processors
;node 3: LPC(Laboratoire de Physique Corpusculaire) Part of the LCG ;       140 processors
;node 4: SDSC DataStar IBM p655/p690. Partition: 3 batch partition;        224 processors
;node 5: CTC IBM SP2, 430 processors
;node 6: LANL CM-5, 1024 processors
;node 7: SDSC IBM SP3 Blue Horizon, 1152 processors
;node 8: SDSC DataStar IBM p655/p690. Partition: 2 batch partition;        1368 processors
```