TESIS DEFENDIDA POR

Jorge Mario Cortés Mendoza

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Andrey Chernykh
Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez
Miembro del Comité

Dr. José Alberto Fernández Zepeda
Miembro del Comité

Dr. Vassili Spirine
Miembro del Comité

Dr. Víctor Hugo Yaurima Basaldúa
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

11 de Marzo de 2011.

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACIÓN

Optimización extrema generalizada para la calendarización de trabajos paralelos en sistemas Grid de dos niveles.

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de
MAESTRO EN CIENCIAS

Presenta:

Jorge Mario Cortés Mendoza

Ensenada, Baja California, México, Marzo del 2011.

RESUMEN de la tesis de **Jorge Mario Cortés Mendoza,** presentada como requisito parcial para la obtención del grado de MAESTRO EN CIENCIAS en Ciencias de la Computación. Ensenada, Baja California. Marzo del 2011.

Optimización extrema generalizada para la calendarización de trabajos paralelos en sistemas Grid de dos niveles.

Resumen aprobado por:

_____

Dr. Andrey Chernykh
Director de Tesis

Los sistemas Grid se utilizan cada vez más debido a los beneficios mostrados cuando es necesario trabajar con una gran cantidad de datos o poder de cómputo. Los retos iniciales se han superado (ejecución de trabajos, transferencia de archivos, múltiples cuentas de usuarios, etc.). Actualmente, se trabaja en la búsqueda de mecanismos que permitan mejorar la utilización del Grid, uno de los campos más estudiados es la calendarización de trabajos.

Se han utilizado varias técnicas para afrontar el problema de calendarización en sistemas Grid. En los últimos años, la utilización de heurísticas inspiradas por la naturaleza ha tenido un gran auge debido a la efectividad para solucionar problemas difíciles. El presente trabajo propone la heurística de optimización extrema generalizada (GEO) para solucionar el problema de calendarización de trabajos paralelos en sistemas Grid de dos niveles. La heurística GEO se basa en el proceso de optimización extrema observado en sistemas complejos, donde avalanchas generacionales evolucionan un ecosistema hasta un estado crítico. Se ha implementado la abstracción de este modelo para solucionar diversos problemas como: TSP, grafo bipartito y calendarización en sistemas multiprocesadores.

Con el objetivo de evaluar el desempeño de GEO como estrategia de asignación de trabajos, se define un modelo para el problema de calendarización y se realizan una serie de simulaciones. Los resultados se analizan y comparan con estrategias existentes en la literatura. La evaluación de los resultados muestra que es apropiado aplicar optimización extrema generalizada al problema de calendarización de trabajos paralelos en sistemas Grid de dos niveles porque supera el desempeño de las estrategias clásicas de calendarización.

Palabras Clave: Calendarización en Grid, optimización extrema generalizada, heurística, calendarización de dos niveles.

ABSTRACT of the thesis presented by **Jorge Mario Cortés Mendoza** as a partial requirement to obtain the MASTER OF SCIENCE degree in Computer Science. Ensenada, Baja California, México, March 2011.

Generalized extremal optimization for parallel job scheduling in two levels hierarchical Grid systems.

Grids are becoming almost commonplace nowadays because of the benefits shown when it is necessary to work with a large amount of data or computing power. The initial challenges of Grid computing have been overcome to first order (running a job, transferring files, managing multiple user accounts, etc.). Researchers can now address the issues involved to improve the uses of Grid computing, one of the most studied field is job scheduling.

Several techniques have been used to address the problem of scheduling. In recent years the use of heuristics inspired by nature has boomed owing the effectiveness to solve hard problems. This work presents the generalized extremal optimization heuristic (GEO) to solve parallel job scheduling problem in two levels hierarchical Grid systems. GEO heuristic is based on the extremal optimization observed in complex systems, where generational avalanches evolve an ecosystem to a critical state. The abstraction of this model has been successfully implemented to solve various problems such TSP, bipartite graph and scheduling in multiprocessors systems.

In order to evaluate the performance of GEO as job allocation strategy, we defined a model for the scheduling problem and a series of simulations. The results are analyzed and compared with existing strategies in the literature. Evaluation result shows that it is appropriate to apply generalized extremal optimization to the parallel job scheduling problem in two levels hierarchical Grid systems because it exceeds the performance of traditional scheduling strategies.

Keywords: Grid scheduling, generalized extremal optimization, heuristic, two levels hierarchical scheduling.

Dedicatorias

A mis padres

Por darme la vida y por todo lo que me han enseñado.

A mis hermanos

Por compartir este camino llamado vida.

A mis familiares y amigos

Por los momento que pasamos junto y el apoyo que siempre me han brindado.

Agradecimientos

A mi asesor:

Dr. Andrei Tchernykh

A los miembros del comité de tesis:

Dr. Carlos Alberto Brizuela Rodríguez
Dr. José Alberto Fernández Zepeda
Dr. Vassili Spirine
Dr. Víctor Hugo Yaurima Basaldúa

A mis compañeros en el CICESE

Al

Centro de Investigación Científica y de Educación Superior de Ensenada

Al

Consejo Nacional de Ciencia y Tecnología

CONTENIDO

Página

CONTENIDO (continuación)

Página

CONTENIDO (continuación)

Página

LISTA DE FIGURAS

Figura Página

LISTA DE FIGURAS (continuación)

Figura Página

LISTA DE TABLAS

Capítulo I

Introducción

I.1 Ambiente Grid

Los avances tecnológicos en diferentes áreas de la computación (hardware y software) han permitido percibir cambios substanciales en la forma de utilizar los recursos de cómputo. Una consecuencia de estos cambios es la capacidad de utilizar recursos ampliamente distribuidos para hacer frente a una gama de problemas a gran escala en distintos campos de la ciencia (Attanasio *et al.*, 2005).

Los primeros sistemas distribuidos se concibieron a partir de la posibilidad de conectar y comunicar equipos de cómputo. Las características de diseño, construcción y desarrolló de estos sistemas continúan siendo objeto de estudio por un numeroso grupo de investigadores. Sin duda, esta nueva tecnología atrajo el interés por los beneficios mostrados, entre los que se encuentra una alta capacidad de cómputo que no se había visto anteriormente.

La última generación de sistemas distribuidos se basa en la popularidad del internet y la disponibilidad de una cantidad de recursos computacionales distribuidos geográficamente (Nebro *et al.*, 2007). Un modelo sobresaliente en los sistemas distribuidos fue aquel que permitió la utilización eficiente de recursos heterogéneos débilmente acoplados (Haynos, 2004).

Se han utilizado varios nombres para distinguir este modelo de cómputo: meta computadora, cómputo escalable, cómputo global, cómputo por internet, etc. Sin embargo, en los últimos años, se aceptó el término Grid de manera general para denotar esta nueva tecnología.

Las redes eléctricas inspiraron significativamente el concepto de computación Grid, en éstas la generación de electricidad se realiza a distancia del punto de consumo y de forma transparente al usuario (Foster *et al.*, 2001).

I.2 Paradigma Grid

El termino Grid fue acuñado a mediados de la década de los 90's (Foster y Kesselman, 2004). Este término visualiza una infraestructura con poder de cómputo compartido y distribuido para áreas de investigación e ingeniería. Posteriormente, esta visión se refinó dando lugar a una idea más amplia: "infraestructura para almacenamiento, cómputo compartido y control de recursos, los cuales están orientados a resolver problemas dinámicos en organizaciones multi-institucionales y organizaciones virtuales[1]" (Kesselman y Foster, 1998).

El término Grid se ha modificado constantemente para adaptarse a las circunstancias actuales. La definición más reciente la muestra CoreGrid (2010): "es una infraestructura plenamente distribuida, reconfigurable dinámicamente, escalable, autónoma, de múltiple acceso eficiente e independiente, confiable y segura, que coordina un conjunto de servicios virtuales encapsulados (poder de cómputo, almacenamiento, instrumentos, datos, etc.) en pro de la generación del conocimiento" (Figura 1).

El Grid puede entenderse como una infraestructura de cooperación entre instituciones, donde una serie de recursos de cómputo de uso colectivo están disponibles. Ciertas instituciones como universidades, empresas, centros de investigación y otros grupos, se unen para formar una comunidad que permite a cada usuario trabajar con un mayor número de recursos de cómputo a un bajo costo (Ramírez *et al.*, 2007).

La infraestructura de recursos (máquinas, dispositivos de almacenamiento, instrumentos especializados, software, etc.) heterogéneos (diferentes hardware y/o software) plenamente distribuidos (posicionados en varias partes del mundo) permiten

[1] Una organización virtual es un conjunto de individuos y/o instituciones regidas por reglas para compartir recursos.

manejar una gran cantidad de datos y aplicaciones. Aunado a las ventajas de escalabilidad (capacidad de aumentar y disminuir recurso), autonomía (control independiente) y seguridad (proteger datos de manipulación, pérdida, destrucción o acceso por personas no autorizadas) hacen del Grid una herramienta indispensable en la solución de problemas modelados por computadora.

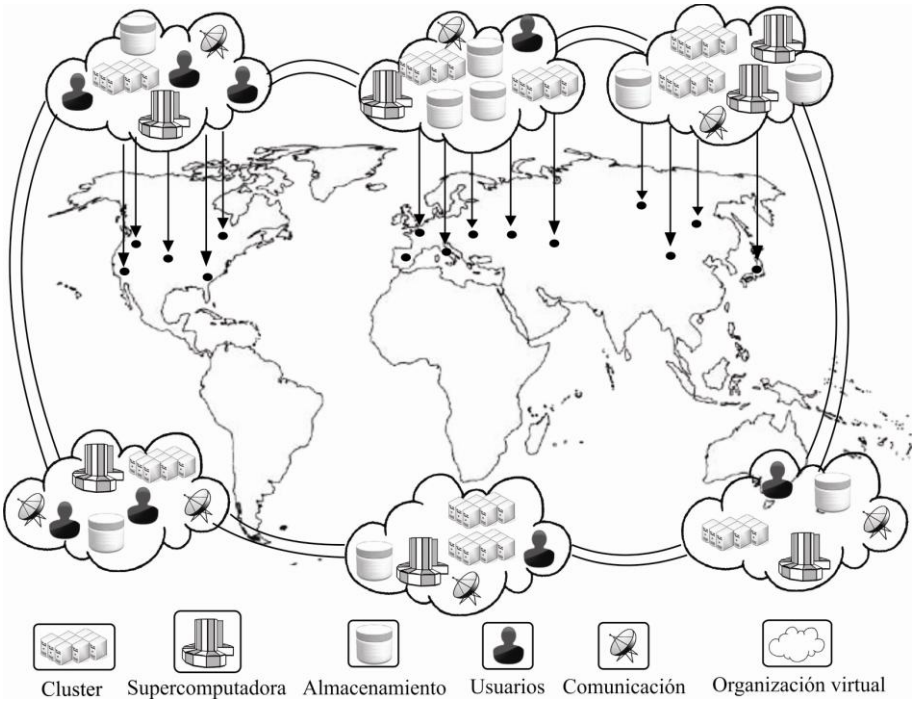


Figura 1. Grid computacional.

La computación Grid emergió para satisfacer el incremento de la demanda de la comunidad científica por un mayor poder de cómputo. Las instituciones académicas estuvieron a la vanguardia en el desarrollo de la tecnología y arquitectura, formando las bases del paradigma Grid. Algunas instituciones y organizaciones como la Alianza Globus (the Globus Alliance), el Grid Chino (the China Grid), el programa del Reino Unido de e-Ciencia para el núcleo y el Grid (UK e-Science Grid core program and the Grid) fueron algunas de las primeras en incubar y desarrollar esta tecnología hasta la madurez, preparándola para la adopción comercial (Haynos, 2004).

En los último años, la computación Grid ha contribuido en el alcance de avances en Meteorología (Bernholdt *et al.*, 2007), Física (Ernst *et al.*, 2006), Biología (Garzón *et al.*, 2007) y en otros campos que requieren de cómputo intensivo. Algunos beneficios mostrados por los sistemas Grid son: procesamiento de aplicaciones en paralelo, reducción de los costos en infraestructura y manejo de una gran cantidad de información.

I.3 Clasificación Grid

La tecnología Grid ha beneficiado diferentes áreas de la ciencia. Para entender el alcance de esta tecnología en la resolución de problemas, se define una clasificación Grid basada en la orientación (problemas que pueden solucionar) y aplicaciones:

1. *Supercómputo distribuido:* las aplicaciones de supercómputo distribuido utilizan al Grid para conectar (de forma física y lógica) recursos, con la finalidad de solucionar problemas que no se pueden resolver en un solo sistema. Estos recursos están compuestos por supercomputadoras (computadora con capacidad de cálculo superior a las comúnmente disponibles) pertenecientes a una o varias compañías en distintas ciudades del mundo. A esta clase de Grid se le conoce como Grid computacional (*C-Grid*).

2. *Cómputo de alto rendimiento:* Con el objetivo de aumentar la utilización de las máquinas del Grid (reducir el número de procesadores ociosos), el cómputo de alto rendimiento calendariza un gran número de aplicaciones, el resultado puede ser similar al supercómputo distribuido, pero los recursos se enfocan en un solo problema. Por ejemplo, el sistema Condor, de la Universidad de Wisconsin, administra lotes (conjuntos o cantidades definidas) de computadoras en universidades y laboratorios alrededor del mundo, los recursos se emplean para estudiar y realizar simulaciones (experimentación) de cristal líquido, estudios en la penetración de la radiación, y el diseño de motores a diesel (Kesselman y Foster, 1998).

3. *Cómputo bajo demanda:* las aplicaciones bajo demanda utilizan las capacidades del Grid para reunir recursos, localmente no rentables o inconvenientes, a corto plazo. Los recursos pueden ser desde computadoras, software, repositorios de datos, sensores hasta dispositivos especializados. La característica de esta clase de Grid es la cantidad de bienes o servicios que los consumidores están dispuestos a adquirir dado un precio determinado. Contrario a las aplicaciones de supercómputo distribuido, estas aplicaciones a menudo se manejan por la relación costo-desempeño en lugar del desempeño absoluto.

   Por ejemplo, la computadora MRI (Computer-Enhaced Machine) y el microscopio STM (Scanning Tunneling Microscope) (Potter *et al.*, 1996), desarrollados en el Centro Nacional para Aplicaciones de Supercómputo, usan computadoras para el procesamiento de imágenes en tiempo real.

4. *Manejo de datos en forma intensa:* este tipo de Grid se centra principalmente en el manejo de datos en forma intensa. Gestión, intercambio y acceso a grandes cantidades de información distribuida geográficamente (repositorios distribuidos, bibliotecas digitales y bases de datos) (Kesselman y Foster, 1998). Muchas aplicaciones científicas y de ingeniería requieren acceso a grandes cantidades de datos distribuidos (los datos pueden tener un formato propio), realizar una consulta o síntesis de la información a menudo requiere de cómputo intensivo y alto grado de comunicación.

   El instrumento científico más grande del planeta, el gran colisionador de hadrones (LHC), produce aproximadamente 15 Petabyte (15 millones de Gigabytes[2]) de datos anualmente; esta información la consultan y analizan cientos de científicos alrededor de todo el mundo (CERN, 2010); ciertos mecanismo más complejos de solicitudes permitirán acceder a una gran fracción de datos que contengan información interesante. Los colaboradores científicos que accedan a dichos datos,

[2] Unidad de almacenamiento, equivale a 2^{30} bytes.

estarán distribuidos sobre la faz de la tierra, como consecuencia, los sistemas donde están localizados los datos también estarán distribuidos.

5. *Cómputo colaborativo:* el cómputo colaborativo resulta de un proceso que involucra el trabajo de varias personas en conjunto. Las aplicaciones colaborativas permiten mejorar la interacción humano-humano, dichas aplicaciones se estructuran en términos de un espacio virtual (sistema o interfaz que genera entornos sintéticos en tiempo real) compartido (Kesselman y Foster, 1998). Muchas aplicaciones están interesadas en compartir el uso de recursos computacionales, tales como archivos de datos y simulaciones; en este caso, también tienen características de las clases descritas anteriormente. Un ejemplo de dicha aplicación es el sistema BoilerMaker desarrollado en los Laboratorios Nacionales de Argonne (Diachin *et al.*, 1996). Este sistema permite colaborar a múltiples usuarios en el diseño de un sistema controlador de emisiones en la industria incineradora. Los diferentes usuarios interactúan unos con otros y simultáneamente con un simulador de incineración.

A pesar de las ventajas ofrecidas por los sistemas Grid en diferentes modalidades, todavía hay aspectos por mejorar. Por ejemplo, los desafíos del *supercómputo distribuido y del cómputo de alto rendimiento* son: la necesidad de planificación (calendarización) de recursos, la escalabilidad (incrementar o disminuir el número de recursos disponibles), los protocolos de comunicación (reglas que especifican la comunicación entre sistemas de cómputo), algoritmos tolerantes a latencia (suma de retardos dentro de una red de computadoras) y mantener altos niveles de desempeño comparado con sistemas heterogéneos (Kesselman y Foster, 1998).

Los desafíos de aplicaciones que manejan *grandes cantidades de datos* son: la planificación de recursos, configuración de sistemas complejos y flujos (movimiento o circulación) de datos por medio de una jerarquía (forma de organización, generalmente por etapas). Las *aplicaciones colaborativas* requieren de un sistema de tiempo real impuesto

por las capacidades de los humanos y una gran variedad de interacciones que pueden surgir.

En la literatura existen diferentes taxonomías para caracterizar sistemas Grid, por ejemplo, considerar el diseño del Grid y el tipo de aplicaciones (Figura 2), utilizar el número de modos de ejecución y modelos de las máquinas (Ekmecic *et al.*, 1996), los algoritmos de calendarización en el sistema, interfaces externas, diseño de sistemas internos, la clase de hardware y software (Kapadia *et al.*, 1998), por mencionar algunos.

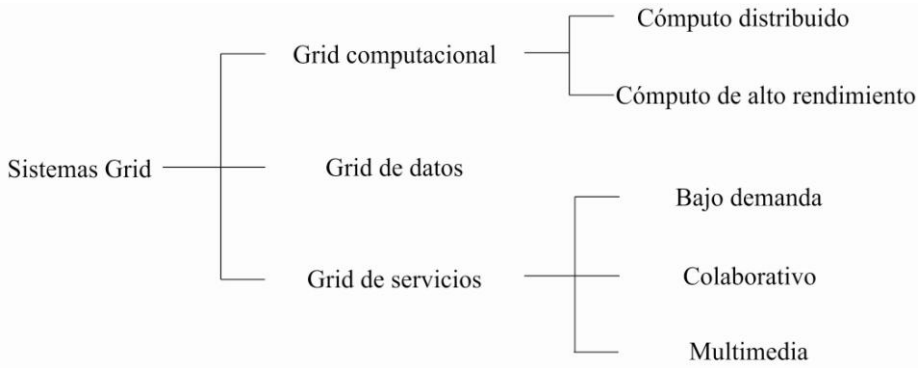


Figura 2. Clasificación de sistemas Grid (Krauter *et al.*, 2001).

Actualmente se sigue trabajando en la búsqueda de mecanismos que permitan mejorar la utilización del Grid computacional, la calendarización de trabajos es uno de los campos más estudiados.

I.4 Calendarización en Grid computacional

Los algoritmos de calendarización de trabajos han sido estudiados ampliamente para sistemas distribuidos y paralelos, tales como máquinas con múltiples procesadores simétricos (SMP), computadoras con procesadores paralelos masivos (MPP) y clusters de estaciones de trabajo (COW) (Dong y Akl, 2006). La estructura del calendarizador tiene influencia en la arquitectura del Grid computacional (Foster y Kesselman, 2004), ésta

dependerá de la cantidad de recursos y el número de dominios donde estén distribuidos, con base en esto, existen tres modelos de calendarizador para sistemas Grid.

I.4.1 Calendarización centralizada

En un ambiente centralizado, una instancia central calendariza en todas las máquinas (Figura 3). La instancia central recolecta la información del estado de todos los sistemas disponibles, por lo tanto, el calendarizador es conceptualmente viable para producir calendarios eficientes porque la instancia central contiene toda la información de los recursos disponibles. Sin embargo, esta arquitectura puede provocar un cuello de botella en algunas situaciones (pérdida de comunicación entre el calendarizador y los recursos disponibles en el sistema). No es adecuado para sistemas manejadores de recursos de Grid debido a que en ellos hay políticas impuestas por los propietarios de los recursos.

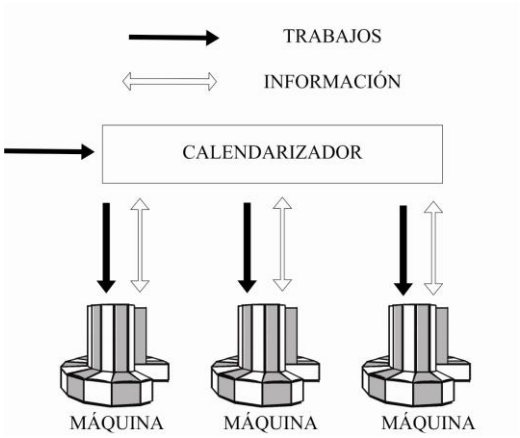


Figura 3.Calendarización centralizada (Hamscher *et al.*, 2000). Las líneas oscuras especifican el flujo de los trabajos y las líneas blancas el flujo de la información.

I.4.2 Calendarización descentralizada

En un ambiente descentralizado, los calendarizadores distribuidos interactúan entre ellos con el objetivo de seleccionar un recurso para ejecutar los trabajos, no existe un líder central responsable de la calendarización, por tanto este modelo es altamente escalable y tolerante a fallas. Este esquema es adecuado para sistemas Grid debido a las políticas de calendarización local definidas por los propietarios de los recursos. Sin embargo, la

generación de calendarios óptimos no es segura ya que la información de trabajos y recursos remotos no está disponible en una sola ubicación. Además, es un modelo difícil de implementar en ambientes Grid, porque los dueños de los recursos no están de acuerdo en políticas globales para el manejo de recursos.

Comunicación directa. Los calendarizadores locales envían y reciben trabajos desde y hacia otros calendarizadores directamente (Figura 4). Cada calendarizador tiene una lista de los calendarizadores remotos o un directorio provee la información de los demás sistemas. Si un trabajo no puede iniciar su ejecución en la máquina local, el calendarizador local busca una máquina alternativa. Al encontrar una máquina adecuada, donde el trabajo puede iniciar su ejecución inmediatamente, el trabajo y toda su información se transfiere a la otra máquina.

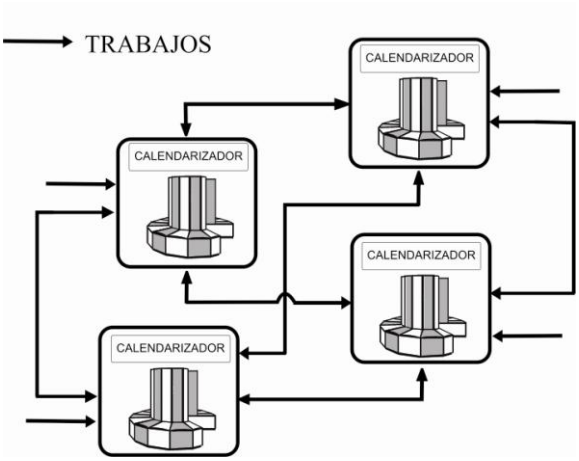


Figura 4. Calendarización descentralizada con comunicación directa (Hamscher *et al.*, 2000). En este modelo las máquinas están compuestas por un calendarizador y los procesadores.

Comunicación a través de una pila central de trabajos. Los trabajos se envían a una pila central de trabajo cuando no se pueden ejecutar inmediatamente, en lugar de enviarse a una máquina remota (Figura 5). En contraste con la comunicación directa, el calendarizador local puede tomar trabajos adecuados para su calendario, en este escenario, los calendarizadores colocan o retiran trabajos de la pila. Un problema en este esquema es la espera indefinida de los trabajo en la pila de trabajo, por lo tanto es necesario crear políticas para ejecutar todos los trabajos de la pila en cierto tiempo. Una variante de este

modelo permite asignar los trabajos directamente a la pila una vez que los trabajos entren al sistema. Los trabajos que requieren pocos recursos se pueden utilizar para rellenar recursos libres en todas las máquinas.

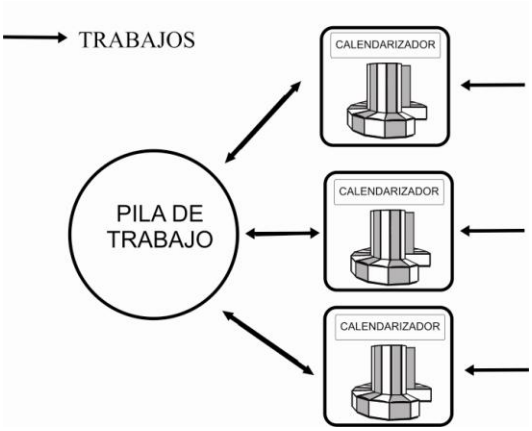


Figura 5. Calendarización descentraliza con pila de trabajos (Hamscher *et al.*, 2000).

I.4.3 Calendarización jerárquica

Este modelo parece un modelo híbrido (una combinación de los modelos centralizado y descentralizado), pero se asemeja más un modelo centralizado y funciona bien para sistemas Grid. En este esquema el bróker (meta-calendarizador) interactúa con los calendarizadores locales para seleccionar los recursos (Figura 6). Un modelo jerárquico es adecuado para sistemas Grid porque permite a los propietarios de recursos remotos aplicar sus propias políticas sobre usuarios externos.

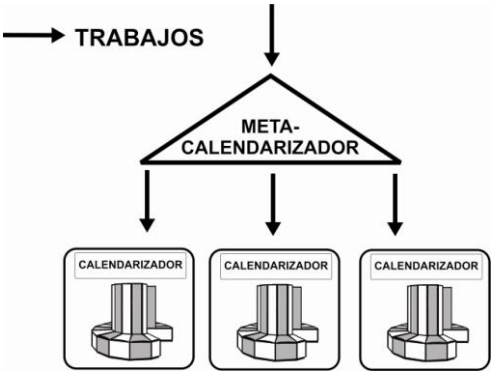


Figura 6. Calendarización jerárquica (Hamscher *et al.*, 2000).

I.5 Problema de calendarización en Grid

En los últimos años, la popularización de los ambientes Grid ha permitido solucionar los retos iniciales (ejecución de trabajos, transferencia de archivos, múltiples cuentas de usuarios, etc.), los usuarios e investigadores ahora pueden abordar problemas para utilizar eficientemente los recursos. La utilización de buenas herramientas de gestión de recursos en ambientes Grid aun no es posible debido a la existencia de varios campos abiertos.

La calendarización es uno de los campos abiertos más estudiados en sistemas Grid. Ésta se puede ver como una familia de problemas debido a la cantidad de características que intervienen para definir las necesidades de las aplicaciones ejecutadas. Una aplicación puede generar varios trabajos, cada trabajo puede estar compuesto de diferentes tareas, la función del sistema Grid es enviar cada tarea a los recursos para ser solucionada, en algunos casos es suficiente con que el usuario seleccione el recurso para ejecutar las sub-tareas. En general, un Grid dispone de un calendarizador que automáticamente y de forma eficiente encuentra los recursos más apropiados para ejecutar los trabajos.

La calendarización de trabajos en sistemas distribuidos no es completamente nueva, es uno de los problemas más estudiados en optimización. En ambientes Grid existen ciertas características que hacen del problema diferente de la versión tradicional de los sistemas distribuidos, algunas de estas características son (Xhafa y Abraham, 2010):

- *Estructura dinámica del Grid*.- En ambientes Grid no existe un control sobre los recursos. Diferente a los sistemas tradicionales (como clusters), los recursos en un Grid pueden estar disponibles o no en cualquier momento de forma impredecible, debido a la pérdida de conexión o al apagado del sistema.
- *Recursos heterogéneos*.- El sistema Grid funciona como una gran organización virtual, los recursos computacionales pueden ser muy variados, desde laptops hasta supercomputadoras (se puede contar con pequeños dispositivos con capacidades muy limitadas). Las estructuras Grid aun no son muy versátiles pero la heterogeneidad de sus recursos es una de las características más importantes.

- *Trabajos Heterogéneos*.- Los trabajos suministrados son diversos y pueden tener diferentes requerimientos (cómputo intensivo o gran cantidad de datos). Algunos trabajos pueden tener una gran cantidad de requerimientos mientras que otros pueden ser más flexibles. Un Grid no tiene control sobre el tipo de aplicaciones, trabajos o tareas que llegan al sistema.

- *Conexiones heterogéneas*.- Las conexiones existentes entre los recursos de un Grid son diferentes, el costo de transmisión juega un papel importante en el rendimiento general y por lo tanto es necesario tomar en cuenta la diferencia entre los enlaces.

- *Calendarizadores locales*.- Un Grid está conformado por las contribuciones de organizaciones como universidades, industrias y aportaciones individuales. Cada una puede estar ejecutando aplicaciones locales por lo que es evidente la utilización de calendarizadores locales. Una buena decisión es utilizar el calendarizador local en lugar de uno de dominio.

- *Políticas locales*.- Debido a los diferentes propietarios de los recursos, no es posible asumir el control total del Grid. Una compañía puede tener un conjunto de recursos asignados; sin embargo, en cualquier momento podrían reducir su contribución. También se deben tomar en cuenta otras políticas como derechos de acceso, almacenamiento, etc.

- *Sistema a gran escala*.- Se espera que los sistemas Grid sean de gran escala, teniendo millones de nodos en el mundo además de aplicaciones, trabajos, etc. Por este motivo, se busca la eficiencia en la asignación de los trabajos y la necesidad de diferentes tipos de calendarizador (súper-calendarizador, meta-calendarizador, calendarizadores descentralizados, calendarizadores locales, agentes de recursos, etc.) y su posible combinación jerárquica.

- *Seguridad*.- Esta característica, inexistente en la calendarización tradicional, tiene una gran importancia en sistemas Grid y puede verse como un objetivo de dos niveles. Por un lado, la asignación de un trabajo en un recurso seguro, se debe garantizar que el nodo no intentará visualizar o acceder a la información utilizada

por el trabajo. Por otro lado, el nodo debe contar con requerimientos de seguridad, cualquier tarea o trabajo que haya sido asignado a un nodo no tratará de visualizar o acceder a otros datos.

- *Conflicto de objetivos*.- A menudo los administradores de los recursos y los usuarios tienen diferentes objetivos; los administradores intentan maximizar la utilización de los recursos, mientras que los usuarios solo esperan ser atendidos de manera rápida (Kurowski *et al.*, 2008); satisfacer las necesidades de ambos resulta en ocasiones imposible, por lo tanto, se busca la manera de encontrar una solución que considere las necesidades de ambos.

El nivel de existencia de cada uno de los temas mencionados anteriormente depende en gran medida del escenario de uso específico y, muy a menudo, de la infraestructura Grid. Por lo tanto, es muy difícil proporcionar una estrategia (método) de calendarización única que pueda utilizarse en diferentes entornos.

I.6 Planteamiento del problema

La administración de recursos es un problema bien estudiado en sistemas tradicionales (calendarizadores por lotes, motores de flujo de trabajo y sistemas operativos), sin embargo, existen estrategias utilizadas en estos sistemas que no pueden aplicarse directamente al Grid (Foster y Kesselman, 2004). Una de las principales diferencias entre un *calendarizador local* y un *calendarizador Grid* radica en el control de los recursos, un calendarizador Grid no "posee" los recursos en un sitio y por consiguiente no tiene control sobre ellos. Esta ausencia de propiedad y control es la fuente de muchos de los problemas a ser resueltos en esta área (Buyya *et al.*, 2005).

Actualmente, la investigación de calendarización en Grid recae en la distribución de trabajos (selección de recursos); esta etapa es una de las más importantes en la calendarización Grid (Montero *et al.*, 2003). Varios factores o parámetros se utilizan para la selección de recursos, como *utilización de recursos* (Shan *et al.*, 2003), *sitio menos cargado* (Ranganathan y Foster, 2004), *mejor ajuste* (Hamscher *et al.*, 2000), entre otros.

El presente trabajo propone la utilización de una estrategia evolutiva para solucionar el problema de calendarización en Grid computacional de dos niveles. La estrategia llamada optimización extrema generalizada (Sección III.5) permite la asignación de trabajos a los sitios del Grid (estrategia de asignación en la primera capa del calendarizador Grid). A continuación se describen las características del sistema Grid y los parámetros del calendarizador considerados para la solución del problema.

El modelo utilizado es un Grid centrado en cómputo, caracterizado por la necesidad de una alta capacidad de procesamiento. Un calendarizador jerárquico de dos niveles se emplea para calendarizar los trabajos; en el primer nivel (nivel de asignación) los trabajos se asignan a las máquinas (sitios); en el segundo nivel (nivel de calendarización), los trabajos se calendarizan en los procesadores de la máquina local.

El problema de calendarización de trabajos en Grid se plantea de la siguiente manera: un conjunto de *n* trabajos paralelos independientes $J_1, J_2, \ldots, J_n$ se requieren calendarizar en *m* máquinas paralelas representadas por $N_1, N_2, \ldots, N_m$, donde m_i denota el número de procesadores idénticos de la máquina N_i. Las máquinas paralelas están ordenas de forma no descendente por su tamaño $m_1 \leq m_2 \leq \cdots \leq m_m$.

Cada trabajo J_j se describe mediante una tupla $\{ r_j, size_j, p_j, p'_j \}$, donde r_j denota el tiempo de llegada del trabajo *j*, el tamaño $1 \leq size_j \leq m_m$ describe el grado de paralelismo del trabajo *j*, p_j es el tiempo de procesamiento del trabajo *j* (este tiempo se desconoce hasta la finalización del trabajo) y p'_j es el tiempo de procesamiento estimado por el usuario del trabajo *j*. El tiempo de llegada para todos los trabajos $r_j = 0$, debido a que cualquier trabajo está disponible para ejecutarse desde el inicio (calendarización offline o batch).

El trabajo J_j sólo se puede ejecutar en la máquina N_i si se satisface $size_j \leq m_i$; no se permiten asignaciones en diferentes máquinas o co-asignación de los procesadores en diferentes máquinas; una vez que el trabajo se inicia, no es posible detener la ejecución hasta finalizar el trabajo.

La notación *MPS* (*Multi-Parallel Scheduling*) describe el proceso de calendarización empleado; esta notación define los dos niveles del Grid ($MPS = MPS_{alloc} + PS$). La notación *modelo de la máquina | características de los trabajos y limitaciones | objetivos* (Graham *et al.*, 1979) especifica las características del problema para cada nivel.

La presente investigación se enfoca en el problema $GP_m \left| r_j = 0, size_j, p_j, p'_j \right| (\rho, SD_b, T_w)$ para el primer nivel de calendarización Grid (MPS_{alloc}) y $P_m \left| r_j = 0, size_j, p_j, p'_j \right| C_{max}$ para los calendarizadores locales (PS), donde:

GP_m: Grid de máquinas paralelas con procesadores idénticos.

P_m: máquinas paralelas con procesadores idénticos.

Las métricas consideradas son (Sección II.5):

- Factor de competitividad (ρ).
- Desaceleración acotada promedio (SD_b).
- Tiempo de espera promedio (T_w).

I.6.1 Objetivo general

Desarrollar, implementar y analizar la heurística de optimización extrema generalizada para solucionar el problema de calendarización en un Grid computacional de dos niveles. Evaluar el desempeño del algoritmo mediante simulaciones, utilizando cargas de trabajo reales.

I.6.2 Objetivos específicos

- Definir la representación del calendario para el algoritmo de optimización extrema generalizada.

- Definir los operadores de mutación.
- Implementar el algoritmo de optimización extrema generalizada para el problema de calendarización en sistemas Grid.
- Establecer un umbral de equilibrio entre la calidad de la solución y el tiempo utilizado para encontrar tal solución.
- Realizar simulaciones de la heurística implementada.
- Comparar el desempeño de la heurística propuesta con estrategias existentes.

I.7 Metodología

Para llevar a cabo este trabajo y cumplir con los objetivos marcados se empleó una metodología de varias etapas descritas a continuación:

REVISIÓN DE LITERATURA. En esta etapa se revisa la literatura (artículos y libros) relacionada con el problema de calendarización en sistemas distribuidos, calendarización en sistemas Grid, estrategias evolutivas para la solución de problemas de optimización y la heurística de optimización extrema generalizada con la finalidad de entender los aspectos relacionados a la investigación.

DESARROLLO DEL SISTEMA. Esta etapa se divide en dos partes: la primera parte se centra en el estudio del simulador Teikoku y la implementación de modificaciones para la correcta interacción entre la aplicación GEO y el simulador. El simulador Teikoku es una herramienta que permite evaluar los calendarios. La segunda etapa se enfoca en la implementación de la heurística de optimización extrema generalizada para solucionar el problema de calendarización.

SELECCIÓN DE LA CARGA DE TRABAJO. En esta etapa se definen y analizan las características de la carga de trabajo. La selección de la carga de trabajo resulta fundamental, una carga de trabajo real permite evaluar el algoritmo de una forma muy apegada a los modelos empleados para calendarizar trabajos en ambientes reales.

DEFINICIÓN DE PARÁMETROS. En esta etapa se identifican los valores que permiten un mejor desempeño de la estrategia GEO; estos parámetros se definen mediante experimentos.

EVALUACIÓN DEL SISTEMA. En esta etapa se realizan un conjunto de experimentos (simulaciones) para evaluar el algoritmo de optimización extrema generalizada, posteriormente, por medio de un estudio, es posible evaluar de manera cuantitativa el desempeño del algoritmo. Finalmente, se emiten conclusiones sobre el trabajo realizado.

I.8 Organización del trabajo

El contenido de la tesis está organizado de la siguiente manera:

El Capítulo 2 presenta los fundamentos de la teoría de calendarización, el modelo jerárquico más utilizado en la calendarización en sistemas Grid de dos niveles, las métricas para evaluar el desempeño de los calendarios, y los procedimientos o heurísticas empleados para solucionar el problema de asignación de recursos y calendarización.

El Capítulo 3 describe la heurística de optimización extrema generalizada, el origen de la teoría de equilibrio puntuado y optimización extrema; una breve reseña de los problemas resueltos utilizando esta heurística. Finalmente, se describe el modelo implementado para solucionar el problema de calendarización en sistemas Grid.

El Capítulo 4 especifica la metodología utilizada para evaluar el algoritmo de optimización extrema generalizada, la definición de los experimentos, la selección y análisis de las cargas de trabajo y la configuración Grid, entre otras cosas.

El Capítulo 5 lista las conclusiones del trabajo derivadas del análisis experimental y las posibles líneas de investigación a seguir.

Capítulo II

Calendarización

II.1 Introducción

El problema de calendarización se encuentra en sistemas donde es necesario organizar y/o distribuir el trabajo entre un conjunto de entidades. La calendarización es un problema de toma de decisiones, se encarga de la asignación de los diferentes recursos a las entidades, con el objetivo de optimizar un conjunto de medidas de desempeño.

La teoría de calendarización surgió a mediados de los años 50s, investigadores de diferentes ramas de las ciencias afrontaron el problema de administrar varias actividades que ocurrían en un flujo de trabajo; durante este periodo, algunos algoritmos permitieron reducir el costo de la producción en procesos de manufactura. A inicios de los años 60s, el paradigma de la computación encontró el problema de calendarización en el desarrollo de sistemas operativos, en aquellos tiempos, los recursos computacionales (como CPU, memoria y dispositivos de entrada y salida) eran escasos y la utilización eficiente de esos recursos era indispensable para obtener un bajo costo a la hora de ejecutar programas; lo anterior provocó una razón económica para impulsar el estudio de la calendarización.

Los primeros problemas de calendarización resultaron relativamente sencillos de resolver, se desarrollo un número importante de algoritmos eficientes para generar soluciones óptimas (Jackson, 1955; Johnson, 1953; Smith, 1955). Con el tiempo, los problemas se volvieron más sofisticados y los investigadores no fueron capaces de encontrar algoritmos eficientes para solucionar los problemas. Con la aparición de la teoría de complejidad (Garey y Johnson, 1977), se llegó a la conclusión de que muchos de los problemas eran inherentemente difíciles de resolver, no fue sino hasta los años 70s cuando se demostró que muchos problemas de calendarización son NP-difíciles (Brucker, 2001;

Lenstra y Kan, 1978; Lenstra *et al.*, 1977); para más información sobre teoría de complejidad se puede consultar el Apéndice B.

Durante la década de los 80s, la industria y el mundo académico llevaron a cabo esfuerzos para solucionar este tipo de problemas. Algunas de estas vertientes fueron: el desarrollo y análisis de algoritmos de aproximación, y la creciente atención prestada a los problemas de programación estocástica. Con la aparición de nuevas tecnologías, como los ambientes Grid, el proceso de calendarización se volvió más complejo (Schwiegelshohn *et al.*, 2008). Después de casi 50 años, el conocimiento en este campo es muy amplio, sin embargo, los esfuerzo aun persisten por encontrar algoritmos eficientes para solucionar estos problemas.

II.2 Problema de calendarización

Existen diversas definiciones para el problema de calendarización, a continuación se presentan dos definiciones de calendarización:

1) "Calendarización es pronosticar el procesamiento de un trabajo, asignando recursos a trabajos y determinar sus tiempos de inicio. Los componentes de un problema de calendarización son los trabajos, las restricciones, los recursos y la función objetivo. Las trabajos se deben programar con la finalidad de optimizar un objetivo especifico" (Carlier y Chrétienne, 1988).
2) "Calendarización se refiere a la asignación de recursos limitados a trabajos en el tiempo. Es un proceso de toma de decisiones que tiene como meta la optimización de uno o más objetivos" (Pinedo, 2008).

El problema de calendarización consta de tres conjuntos: un conjunto finito $J = \{J_1, J_2, \ldots, J_n\}$ de *n* trabajos, un conjunto finito $P = \{P_1, P_2, \ldots, P_m\}$ de *m* procesadores (máquinas o sitios) y un conjunto finito $R = \{R_1, R_2, \ldots, R_s\}$ de *s* tipos de recursos adicionales. La calendarización, hablando en términos generales, trata el problema de asignación de los procesadores *P* y los posibles recursos de *R* a los trabajos pertenecientes

al conjunto J con el objetivo de completar todos los trabajos bajo ciertas limitaciones impuestas.

La teoría de calendarización clásica contempla dos tipos de limitaciones, cada trabajo se ejecuta por a lo más un procesador en cada momento (más los recursos adicionales) y cada procesador es capaz de ejecutar a lo más un trabajo en cada momento. En la actualidad, existen modelos que permiten flexibilizar la primera limitación, cada trabajo requiere de un número fijo de procesadores paralelos durante su ejecución, los procesadores necesarios se denotan por $size_j$.

Los parámetros para caracterizar un trabajo $J_j \in J$ son:

1) Vector de tiempo de procesamiento (p_{ij}): Especifica el tiempo necesario para ejecutar el trabajo *j* en un procesador *i*.
2) Tiempo de llegada (r_j): Especifica el tiempo en el cual el trabajo *j* está listo para iniciar su ejecución.
3) Fecha de vencimiento (d_j): Especifica el tiempo en el cual el trabajo *j* debería ser completado. Generalmente se emplea una función de penalización si no se cumple con este requisito.
4) Plazo ($\tilde{d}_j$): Especifica el tiempo límite en el cual el trabajo *j* se debe terminar.
5) Peso (w_j): Especifica la importancia del trabajo *j*.

El conjunto P contempla dos tipos de procesadores: *paralelos*, realizan la misma función, o *dedicados*, especializados para la ejecución de ciertos trabajos. Los procesadores paralelos consideran tres categorías dependiendo de la velocidad con la que ejecutan los trabajos. Si los procesadores del conjunto P tardan el mismo tiempo en procesar un trabajo se les conoce como *procesadores idénticos*. Si los procesadores difieren en su velocidad pero ésta no depende del trabajo, entonces, se conocen como *procesadores uniformes*. Finalmente, aquellos procesadores que dependen del trabajo que están procesando se conocen como *procesadores no relacionados* (unrelated).

Dos conceptos muy importantes en calendarización son: *la restricción de precedencia y la interrupción de trabajos*. La restricción de precedencia permite definir la relación existente entre dos trabajos J_i y J_j, donde $J_i \prec J_j$ significa que el procesamiento de J_i se debe completar antes de iniciar J_j; en otras palabras, es una relación de dependencia entre ambos trabajos. La interrupción de trabajos permite detener la ejecución de trabajos en cualquier momento para reiniciarla posteriormente, en otras palabras, ocurre un cambio de contexto.

Finalmente, el trabajo J_i está disponible para ejecutarse en el tiempo t si $r_j \leq t$ y todos sus predecesores (con respecto a la limitación de precedencia) se completaron para el tiempo t. Una vez establecidas las características de los conjuntos involucrados en calendarización, es posible definir el término calendario y los criterios de optimización:

Un *calendario* es una asignación de los procesadores del conjunto P (y posibles recursos del conjunto R) a los trabajos del conjunto J en el tiempo, cuando las siguientes condiciones se cumplen:

- En cualquier momento, a cada procesador se le asigna a lo más un trabajo y cada trabajo se procesa por $size_j$ procesadores; para el modelo clásico cada trabajo lo ejecuta a lo más un procesador.
- El trabajo J_i se procesa en el tiempo de llegada [r_i, ∞).
- Todos los trabajos se completan o terminan.
- Si los trabajos J_i y J_j están relacionados $J_i \prec J_j$, el procesamiento del trabajo J_j no inicia hasta el momento que el trabajo J_i se haya completado.
- La interrupción de trabajos es finita (calendario con interrupción de trabajos), de otra forma, el calendario se denomina calendario sin interrupción.
- Las limitaciones de los recursos se deben satisfacer.

La notación de tres campos $\alpha| \beta | \gamma$ (Graham *et al.*, 1979) se introdujo para definir las características de los trabajos, las máquinas y el calendario; el campo α describe el ambiente de la máquina utilizada, máquina con un procesador, procesadores paralelos

idénticos (denotado con una letra P), procesadores paralelos uniformes (Q) y procesadores paralelos no relacionados (R). El campo β provee detalles de las características de los trabajos y las limitaciones del calendario (como limitaciones de precedencia, interrupción de trabajos entre otras). Puede contener múltiples entradas o ninguna. El campo γ contiene la función objetivo que se optimiza, usualmente contiene una entrada simple, una extensión al modelo original permite considerar múltiples entradas independientes o combinadas (Tkindt y Billaut, 2006). Para más información sobre la notación de tres campos se puede consultar el Apéndice A.

Principalmente se emplean tres criterios o métricas de optimización para evaluar los calendarios:

Tiempo total del calendario (C_{max}): Define el tiempo en que termina la ejecución del último trabajo.

$$C_{max} = \max\{C_j\} \tag{1}$$

Promedio de permanencia en el sistema (F): Define el tiempo comprendido entre la llegada del trabajo al sistema y su finalización.

$$F = \frac{1}{n}\sum_{j=1}^{n}(c_j - r_j) \tag{2}$$

Promedio de permanencia en el sistema con pesos:

$$F_w = \sum_{j=1}^{n}(w_j F_j) \,/\, \sum_{j=1}^{n} w_j \tag{3}$$

Retraso máximo de un trabajo (L_{max}): Define el trabajo con la mayor diferencia entre el tiempo de terminación y la fecha vencimiento.

$$L_{max} = \max\{c_j - d_j\} \tag{4}$$

En algunas ocasiones, se pueden utilizar otros criterios relacionados, por ejemplo:

Tardanza promedio (D): Define la diferencia entre el tiempo de terminación y la fecha de vencimiento, si el trabajo termina su ejecución antes de la fecha de vencimiento, la tardanza de la tarea es cero.

$$D = \frac{1}{n}\sum_{j=1}^{n} \max\{c_j - d_j, 0\} \quad (5)$$

Tardanza promedio con pesos:

$$D_w = \sum_{j=1}^{n} (w_j D_j) / \sum_{j=1}^{n} w_j \quad (6)$$

Promedio de anticipación (E): Define la diferencia entre el tiempo de terminación y la fecha de vencimiento; si el trabajo termina su ejecución después de la fecha de vencimiento, la anticipación de la tarea es cero.

$$E = \frac{1}{n}\sum_{j=1}^{n} \max\{d_j - c_j, 0\} \quad (7)$$

Promedio de anticipación con pesos:

$$E_w = \sum_{j=1}^{n} (w_j E_j) / \sum_{j=1}^{n} w_j \quad (8)$$

Promedio de trabajos tardíos (U): Define el número de trabajos que finalizan su ejecución después de su fecha de vencimiento.

$$U = \frac{1}{n}\sum_{j=1}^{n} U_j \quad (9)$$

donde U_j = 1 si $c_j > d_j$ y 0 en otro caso.

Promedio de trabajos tardíos con pesos:

$$U_w = \sum_{j=1}^{n} w_j U_j / \sum_{j=1}^{n} w_j \quad (10)$$

Dependiendo de la cantidad de información que se tiene sobre los trabajos que se van a procesar, se pueden distinguir varias direcciones en teorías de calendarización.

1. Calendarización determinista, estática o fuera de línea (offline o batch): se supone que se conoce toda la información requerida para desarrollar el calendario es conocida antes de que el proceso de calendarización se realice.

2. Calendarización no determinista: ésta es menos restrictiva, solo se conoce información parcial de los trabajos.
3. Calendarización en línea (online): En muchas situaciones, se conocen algunos detalles de la naturaleza de los trabajos, pero se desconoce el tiempo en el cual ocurren los trabajos.
4. Calendarización no clarividente: Considera el problema de calendarización de trabajos con tiempo de ejecución desconocido.
5. Calendarización estocástica: La información de los parámetros disponibles es probabilística y la proporciona el sistema.

II.3 Tipos de calendarización en Grid

Actualmente se estudian diferentes tipos de calendarización en sistemas Grid debido a las necesidades que puede tener cada una de las aplicaciones. Por otra parte, las características del ambiente Grid imponen restricciones que se deben considerar al momento de calendarizar. Para poder alcanzar el desempeño deseado, tanto las especificaciones del problema como las características del ambiente se deben tomar en cuenta por el calendarizador. A continuación se describen los principales tipos de calendarización surgidos en ambientes Grid (Xhafa y Abraham, 2010).

Calendarización de trabajos independientes y de flujo de trabajos.- Una de las características más atractivas del Grid es la capacidad de cómputo paralelo masivo. Usualmente, los trabajos paralelos se pueden dividir en tareas, las cuales se pueden calendarizar de forma independiente. El modelo de trabajos paralelos parece estar bien adaptado a sistemas Grid, es posible distinguir tres tipos de trabajos paralelos: trabajos rígidos, trabajos moldeables y trabajos maleables (Dutot *et al.*, 2004). Sin embargo, existen problemas donde no es posible calendarizar los trabajos de forma independiente, ya que para solucionar estos problemas, se requiere la combinación y orquestación de procesos variados (actores, servicios, etc.).

La calendarización de flujos de trabajo surge debido a la dependencia entre trabajos, determinados por el control y la dependencia de los datos. Estas aplicaciones pueden tomar ventaja del poder de cómputo del Grid, pero las características del ambiente Grid hacen la coordinación de su ejecución compleja (Yu y Buyya, 2005; Lee *et al.*, 2006), la calendarización de flujos de trabajo debe lidiar con la robustez. Ciertamente, un flujo de trabajo no se puede ejecutar por un tiempo prolongado en un modelo dinámico, porque la posibilidad de falla se incrementa, causando la suspensión de todo el flujo de trabajo si no se utiliza un mecanismo de recuperación de fallas.

Calendarización dinámica y estática: Existen dos aspectos primordiales para determinar el dinamismo de la calendarización en Grid: (a) El dinamismo de la ejecución de los trabajos, la ejecución de los trabajos puede fallar o la ejecución se detiene debido a la llegada en el sistema de un trabajo con mayor prioridad (modo con interrupción de trabajos). (b) El dinamismo de los recursos, los recursos pueden entrar o dejar el Grid de forma impredecible, la carga de trabajo puede variar significativamente en el tiempo, las políticas locales usadas por los recursos pueden cambiar en el tiempo. Ambos factores deciden el comportamiento del calendarizador Grid (desde estático hasta altamente dinámico).

Considerando el caso estático, no existen fallas en la ejecución de los trabajos y los recursos están disponibles en todo momento, aunque estos casos son poco realistas para la mayoría de los Grid. Al utilizar calendarización por lotes (batch), el número de trabajos y recursos se consideran estables durante intervalos de tiempo y la capacidad de cómputo también se considera inalterable. Otra variación de este modelo es considerar sólo el dinamismo de los recursos, pero no el de los trabajos.

Calendarización inmediata o por lotes (online u offline). Calendarización inmediata y por lotes son dos técnicas bien estudiadas, altamente exploradas en sistemas distribuidos, también se utilizan para sistemas Grid. En la calendarización inmediata, los trabajos se calendarizan tan pronto éstos entran al sistema, sin tener que esperar por el siguiente

intervalo de tiempo cuando el calendarizador se active nuevamente o la tasa de llegada de los trabajos es pequeña, por lo que se dispone de recursos para ejecutar los trabajos de forma inmediata. En el modo por lotes, los trabajos se agrupan por lotes y se calendarizan en grupos. A diferencia del modelo inmediato de calendarización, el modelo por lotes puede tomar mejores ventajas de los trabajos y las características de los recursos, al disponer de un intervalo entre dos activaciones sucesivas, el calendarizador decide dónde mapear los trabajos entre los recursos disponibles.

Calendarización adaptativa o estática. Los cambios en el ambiente computacional Grid durante el tiempo requieren de técnicas adaptativas en el calendarizador (Lee *et al.*, 2006). Estas técnicas toman en cuenta tanto el estado actual de los recursos y predicciones sobre sus estados futuros, con el objetivo de detectar y evitar el deterioro del desempeño. Recalendarizar se puede ver como una forma adaptativa de calendarización en la cual los trabajos ejecutados se migran a más recursos disponibles.

II.4 Calendarización en Grid de dos niveles

El objetivo del Grid es coordinar un conjunto de recursos heterogéneos para maximizar el desempeño combinado de los recursos e incrementar su costo-eficiencia. La habilidad del Grid para solucionar el problema de calendarización se puede atribuir a la capacidad de asignar recursos a las necesidades computacionales, en este caso, asignar los trabajos. El objetivo es mapear los trabajos a los diferentes recursos para minimizar las funciones objetivo.

La Figura 7 ilustra un Grid computacional jerárquico de dos niveles; los trabajos enviados al Grid permanecen en la cola de trabajos donde esperan hasta ser asignados a las diferentes máquinas. El meta-calendarizador (broker) utiliza una estrategia para asignar los trabajos en los recursos, una vez que los trabajos han sido asignados por el meta-calendarizador, éstos se envian a las máquinas donde se ejecutan. Los trabajos permanecen en la cola de trabajos local (cada máquina tiene su propia cola de trabajo), hasta que el

calendarizador local asigne los recursos (los procesadores de la máquina) mediante un algoritmo de calendarización. Finalmente, los trabajos se ejecutan en los procesadores donde se asignaron.

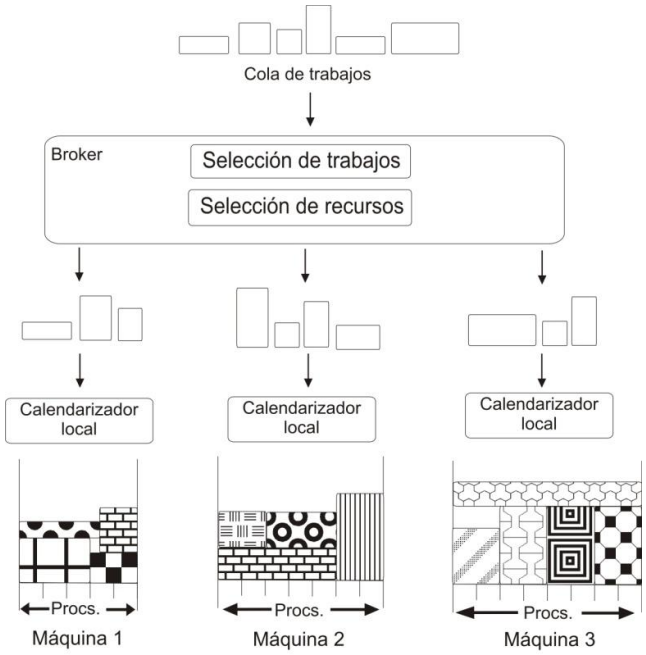


Figura 7. Calendarización en sistemas Grid jerárquicos de dos niveles.

El Grid jerárquico de dos niveles utiliza estrategias en ambos niveles para calendarizar los trabajos. A continuación se describen las estrategias de asignación de recursos a los trabajos, las cuales corresponden al primer nivel y posteriormente se presentan las estrategias de calendarización de los trabajos en las máquinas locales, segundo nivel del Grid computacional.

II.4.1 Estrategias de asignación

Las estrategias de asignación son las técnicas o métodos que asignan recursos a los trabajos propuestos por los usuarios. Las estrategias de asignación se agrupan de acuerdo a la manera en cómo trabajan, las estrategias que no necesitan un calendario previo basan su decisión en los parámetros de los trabajos, las estrategias que requieren un calendario

previo efectúan varios pasos:(i) el calendarizador Grid envía el trabajo (trabajo que se quiere asignar) a todas las máquinas; (ii) cada una de las máquinas construye un calendario tomando en cuenta la carga que tienen asignada y el trabajo recién enviado, (iii) cada máquina regresa la información generada al meta-calendarizador y (iv) éste toma la decisión de asignar el trabajo a una máquina en base a la estrategia empleada.

Las estrategias que no crean un calendario previo para decidir la asignación de los trabajos son (Madrigal *et al.*, 2006):

1. *Aleatoria (Random)*: la estrategia aleatoria actúa bajo un criterio simple, selecciona la máquina de manera aleatoria.

2. *Mínima carga por procesador* (*Min Load per-proc, MLp*): elige la máquina con la mínima carga por procesador (número de trabajos entre el número de procesadores de la máquina).

$$min\ (n_i/m_i),\ i = 1, 2\ ,\ldots,\ M_m \tag{11}$$

   donde n_i es el número de trabajos asignados a la máquina i y m_i denota el tamaño de la máquina (número de procesadores), M_m denota m máquinas que componen el sistema Grid.

3. *Mínima carga paralela* (*Min Parallel Load, MPL*): selecciona la máquina con la mínima carga paralela por procesador (suma de los tamaños de los trabajos sobre el número de procesadores en la máquina).

$$\min\left(\frac{1}{m_i}\sum_{j=1}^{n} size_j\right), i\ = 1, 2, \ldots, M_m \tag{12}$$

   donde $size_j$ es el número de procesadores solicitados por el trabajo j, m_i denota el número de procesadores en la máquina i.

4. *Menor cota inferior de tiempo de finalización* (*Min Lower Bound, MLB*): elige la máquina con la menor cota inferior de tiempo de terminación de los trabajos. El tiempo real de ejecución de un trabajo no está disponible, por lo tanto el valor empleado es el proporcionado por el usuario.

$$\min\left(\frac{1}{m_i}\sum_{j=1}^{n} v_j\right), i = 1, 2, \ldots, M_m \quad (13)$$

donde $v_j = size_j * p'_j$, p'_j es el tiempo de procesamiento definido por el usuario.

Las estrategias que requieren crear un calendario por anticipado para asignar los trabajos son (Madrigal *et al.*, 2006):

5. *Menor tiempo de finalización* (*Min Completion Time, MCT*): elige la máquina con el menor tiempo de terminación de los trabajos asignados, incluyendo el trabajo por asignar.

$$Min\,(C_{max}^{i}), i = 1, 2, \ldots, M_m \quad (14)$$

donde C_{max}^{i} es la longitud del calendario en la máquina *i*.

6. *Menor tiempo de espera* (*Min Waiting Time*, *MWT*): selecciona la máquina con el menor tiempo de espera promedio de los trabajos asignados, incluyendo el trabajo por asignar.

$$Min\left(\sum_{j=1}^{n}\frac{c_j^i - r_j - p_j}{n_i}\right), i = 1, 2, \ldots, M_m \quad (15)$$

donde c_j^i es el instante de finalización del trabajo *j* en la máquina *i*, r_j es el instante a partir del cual se encuentra disponible el trabajo *j* y p_j es el tiempo de ejecución del trabajo *j*, n_i es el número de trabajos en la máquina *i*.

7. *Menor utilización* (*Min Utilization, MU*): elige la máquina con la utilización mínima.

$$Min\left({R_i}/{C_i * m_i}\right), i = 1, 2, \ldots, M_m \quad (16)$$

donde R_i son los recursos consumidos en la máquina *i*, C_{max}^{i} denota la longitud del calendario en la máquina *i*.

8. *Menor tiempo de inicio* (*Min Start Time, MST*): selecciona la máquina con el menor tiempo de inicio de ejecución para el trabajo que se desea asignar

$$Min(C_j^i - r_j), i = 1, 2, \dots, M_m \quad (17)$$

donde C_j^i es el instante de finalización del trabajo *j* en la máquina *i*, r_j el instante a partir del cual se encuentra disponible el trabajo *j*.

9. *Menor tiempo de permanencia en el sistema* (*Min Turnaround, MTA*): elige la máquina con el menor de tiempo de permanencia promedio en el sistema de los trabajos asignados, incluyendo el trabajo por asignar.

$$\min\left(\frac{1}{n_i}\sum_{j=1}^{n} C_j^i - r_i\right), i = 1, 2, \dots, M_m \quad (18)$$

II.4.2 Estrategias de calendarización

Las estrategias de calendarización local son las técnicas que calendarizan los trabajos en los procesadores de las máquinas para ser ejecutados. Cada estrategia tiene la finalidad de elaborar un calendario de acuerdo al criterio (objetivo) que desea mejorar; el rendimiento general del Grid lo afecta la elección de la estrategia de calendarización local (Shan *et al.*, 2003). Algunas estrategias de calendarización local son:

1. *Primero en llegar-primero ser atendido* (*First-Come-First-Serve, FCFS*): de todas las estrategias, es la más simple por la manera en que trabaja. Los trabajos están ordenados en la cola local de trabajo (lista de trabajos en espera de procesadores libres) con base en su tiempo de llegada. El calendarizador comienza a asignar procesadores a los trabajos de acuerdo al orden en que se encuentran en la cola local. Si no hay suficientes recursos disponibles, el calendarizador espera hasta que el trabajo se pueda asignar y procesar, mientras los demás trabajos están detenidos en la cola de trabajo. La técnica FCFS ofrece varias ventajas, no requiere información sobre el tiempo de procesamiento de los trabajos y es fácil de

implementar; sin embargo, puede producir calendarios con un alto porcentaje de tiempo ocioso de los procesadores.

2. *Primero el más grande/chico* (*Largest/Smallest-Size-First, LSF*): esta estrategia ordena los trabajos de la cola local por tamaños de manera ascendente o descendente, posteriormente los trabajos se calendarizan con *FCFS*. Este procedimiento sólo se puede aplicar cuando un conjunto de trabajos está disponible para ejecutarse inmediatamente (lote de trabajos).

3. *Primero el consumidor de recursos más grande/chico* (*Largest/Smallest Resource Consumed First, L/S-RCF*): esta estrategia ordena los trabajos en la cola local de acuerdo a la cantidad de recursos consumidos de forma decreciente o creciente y posteriormente se calendarizan con *FCFS*. Al igual que *LSF* se aplica solamente cuando está disponible un lote de trabajos.

4. *Backfilling-EASY-FirstFit* (*Backfilling EASY First Fit, BEFF, EASY*: *Extensible Argonne Scheduling System* (Etsion y Tsafrir 2005)): es una mejora del algoritmo *FCFS* y ha mostrado ser una buena opción para incrementar el rendimiento tanto para el usuario como para el sistema (Wang Q. *et al.*, 2006; Ramírez *et al.*, 2007).

   Dicha mejora se logra al permitir que algunos trabajos puedan iniciar su ejecución antes del tiempo que les corresponde, según su posición en la cola de espera. Si el trabajo al comienzo de la cola de trabajos no puede iniciar su ejecución debido a que no existen suficientes recursos, una búsqueda identifica los trabajos disponibles para ejecutar inmediatamente. Antes de realizar la búsqueda de trabajos más pequeños, se calcula el tiempo donde habrá recursos suficientes para el trabajo al inicio de la cola. La única limitante para iniciar otros trabajos es que éste no retarde el inicio de la ejecución del trabajo que se encuentra al inicio de la cola

(Lifka, 1998). Cabe mencionar que este algoritmo requiere del tiempo estimado de procesamiento de los trabajos.

II.5 Criterios de evaluación para Grid

Un conjunto de métricas permite evaluar la eficiencia de los calendarios, las métricas se agrupan de acuerdo a cada uno de los beneficiados en el proceso de calendarización (Ramírez *et al.*, 2007): criterios centrados en el sistema, en el usuario, y en el algoritmo de calendarización.

II.5.1 Criterios centrados en el sistema

Utilización (utilization): se define como la fracción de tiempo en la cual el Grid se utilizó. Se calcula como:

$$U_G = \frac{W_G}{C_G * m_G} \tag{19}$$

donde W_G es la cantidad de trabajo del Grid, C_G es el tiempo de finalización de todos los trabajos en el Grid y m_G es el número total de procesadores en el Grid.

Rendimiento del procesamiento (throughput): es el número de trabajos finalizados por unidad de tiempo en el Grid. Se calcula como:

$$Th = \frac{n}{C_G} \tag{20}$$

donde *n* es el número total de trabajos en el Grid.

Desaceleración acotada promedio (mean bounded slowdown): se define como el tiempo promedio de permanencia de los trabajos en el sistema entre el tiempo de procesamiento acotado. Se calcula como:

$$SD_b = \frac{1}{n}\sum_{j=1}^{n} \frac{c_j - r_j}{\max\{10, p_j\}} \tag{21}$$

donde c_j es el tiempo de finalización y r_j es el tiempo de entrega del trabajo j. Max$\{10, p_j\}$ considera el máximo entre el tiempo de procesamiento del trabajo j definido por p_j y diez.

II.5.2 Criterios centrados en el usuario

Tiempo de permanencia promedio (mean turnaround time): se define como el promedio del tiempo que tardan todos los trabajos desde que entran a la cola local hasta que terminan su ejecución. Se calcula como:

$$TA = \frac{1}{n}\sum_{j=1}^{n}(c_j - r_j) \tag{22}$$

Tiempo de espera promedio (mean waiting time): se define como el tiempo de espera promedio de los trabajos antes de iniciar su ejecución. Se calcula como:

$$t_w = \frac{1}{n}\sum_{j=1}^{n}(t_j - r_j) \tag{23}$$

donde t_j es tiempo de inicio de ejecución del trabajo j.

Coeficiente de respuesta (response ratio): se define como el promedio de los coeficientes de respuesta de todos los trabajos. Se calcula como

$$R = \frac{1}{n}\sum_{j=1}^{n}(t_{w_j} + p_j) / p_j \tag{24}$$

donde p_j es el tiempo de procesamiento y t_{w_j} es el tiempo de espera del trabajo j.

II.5.3 Criterios centrados en el algoritmo de calendarización

Coeficiente de desempeño (competitive ratio): es la medida de rendimiento del Grid y se define como

$$\rho = \frac{C_G}{\max(W_G / m_G, p_G^{max})} \tag{25}$$

donde C_G es el tiempo de finalización de todos los trabajos, y $\max(W_G / m_G, p_G^{max})$ el tiempo mínimo posible para terminar los trabajos, p_G^{max} es el máximo tiempo de ejecución de las *n* trabajos.

El coeficiente de desempeño indica la aproximación (una razón) del desempeño de un algoritmo a una cota mínima de tiempo de finalización, es decir, cuántas veces el calendario propuesto por el algoritmo es peor con relación a una cota mínima. Una vez definidos los conceptos de calendario, las estrategias de asignación, las estrategias de calendarización y las métricas que permiten medir la eficiencia de los calendarios, se puede comprender el problema de calendarización en Grid de dos niveles. A continuación se describen los últimos avances en este campo.

II.6 Últimos avances de calendarización en Grid

Se han desarrollado un conjunto de modelos y técnicas (procedimientos) para mejorar la asignación de los recursos en el Grid; los esfuerzos se han centrado en la primera capa de calendarización donde los trabajos se asignan a los recursos del Grid.

Con el objetivo de mejorar el proceso de calendarización de trabajos, existen algunos modelos que consideran las preferencias de los involucrados (usuarios, administradores y proveedores de recursos). La forma de distinguir las preferencia son: (Kurowski *et al.*, 2006): (1) proporcionar explícitamente las preferencias, es decir, cada uno expresa la importancia de los criterios o la importancia relativa entre los criterios, asignando un valor de importancia a cada criterios o por medio de una jerarquía de orden (ordenar los criterios de menor a mayor importancia); (2) descubrir las preferencia basado en decisiones previas, es decir, tomar en cuenta patrones en el historial (eventos anteriores que sirven como indicio).

Posteriormente, el conjunto de preferencia se redefine (Kurowski *et al.*, 2008); las preferencias de los usuarios se consideran restricciones flexibles (*soft constraints*), éstas se pueden cumplir o no, sin embargo, existen restricciones rígidas (*hard constraints*), éstos

son requisitos que deben cumplirse antes de que el calendarizador Grid comience a asignar recursos.

Actualmente la utilización de heurísticas basadas en procesos naturales ha tomado un gran impulso para solucionar problemas variados en diferentes aéreas de la ciencia, la calendarización en sistemas Grid no ha quedado excluida a estas técnicas. A continuación se presentan trabajos que utilizan diferentes técnicas para afrontar el problema de calendarización en sistemas Grid.

II.6.1 Búsqueda local

Las búsquedas locales están constituidas por un conjunto de técnicas basadas en la búsqueda de soluciones mediante la exploración de vecindarios, se genera una solución inicial y se construye una ruta en el espacio de soluciones durante el proceso de búsqueda. Los métodos de esta familia son: escalando la colina (Hill climbing), recocido simulado (simulated annealing), búsqueda tabú (tabu search), entre otros.

II.6.1.1 Escalando la colina

La heurística de escalando la colina se aplica a un punto a la vez. A partir de un punto, se generan varios estados posibles y se selecciona el mejor de ellos. No se tiene retroceso ni se lleva ningún tipo de registro histórico (aunque éstos y otros aditamentos son susceptibles de ser incorporados). El algoritmo puede quedar atrapado fácilmente en óptimos locales, la búsqueda es determinista.

Martincová y Zábovský (2007) emplean un procedimiento de escalando la colina para problemas de calendarización online y offline; el objetivo es minimizar el tiempo de terminación del calendario y comparar el tiempo de preparación de los calendarios. Wan y Gao (2006) estudian la migración de trabajos entre nodos del Grid. El método de escalando la colina selecciona un nuevo nodo para la ejecución del trabajo, una ruta de migración se construye por si es necesario migrar el trabajo varias veces.

II.6.1.2 Recocido simulado

La heurística de recocido simulado (Kirkpatrick, 1984) se basa en el enfriamiento de los cristales. El algoritmo requiere de una temperatura inicial, una temperatura final y una función de variación de la temperatura. Dicha función es crucial para el buen desempeño del algoritmo y su definición es, por tanto, sumamente importante.

Fidanova (2006) utiliza una implementación de recocido simulado en calendarización online, con el objetivo de minimizar el tiempo de terminación de los calendarios. Abdullah y otros (2008) emplean la teoría de carga divisible y una implementación de recocido simulado para minimizar el tiempo de terminación en sistemas Grid. El esquema considera el tiempo de comunicación (cantidad de información y ancho de banda) y la cantidad de datos necesarios por cada trabajo, los cuales residen en fuentes de datos. Kong y otros (2009) proponen un heurística auto-adaptativa basada en recocido simulado para solucionar el problema de calendarización, el objetivo es minimizar el tiempo de terminación y afrontar el problema con características de dinamismo del Grid.

II.6.1.3 Búsqueda tabú

La búsqueda tabú (Glover, 1989) es realmente una meta-heurística, porque es un procedimiento que debe acoplarse a otra técnica, ya que no funciona por sí sola. La búsqueda tabú usa una “memoria” para guiar la búsqueda, de tal forma que algunas soluciones examinadas recientemente se “memorizan” y se vuelven tabú (prohibidas) al tomar decisiones acerca del siguiente punto de búsqueda.

Se han desarrollado trabajos que emplean búsqueda tabú en modelos offline, donde las características de los trabajos y recursos se conocen a priori. Armenta y Yamashita (2000) proponen minimizar la tardanza promedio mediante la utilización de una búsqueda tabú definida por dos tipos de movimientos; uno es la inserción, transfiere un trabajo de una máquina a otra, el otro es el intercambio, mueve dos trabajos entre las máquinas respectivas.

Baraglia y otros (2005) estudian minimizar el tiempo de ejecución de trabajos paralelos en sistemas Grid utilizando un algoritmo de mapeo basado en grafos estáticos. En general, los algoritmos pueden emplear grafos acíclicos dirigidos para modelar aplicaciones paralelas y grafos no dirigidos para modelar sistemas, se propone la utilización de un algoritmo de mapeo estático llamado mapeo heterogéneo multi-fases (Heterogeneous Multi-phase Mapping) mediante búsqueda tabú.

Klusáček y otros (2007) emplean un modelo de calendarización en línea para minimizar el número de trabajos que no cumplen su tiempo de finalización en un ambiente Grid. Cuando un trabajo nuevo entra al sistema, el calendario actual se modifica a través de simples movimientos de los trabajos entre las máquinas, el procedimiento de búsqueda tabú selecciona el trabajo con el tiempo de retardo mayor y lo inserta en la máquina con el menor número de tardanza.

II.6.2 Estrategias basadas en población

II.6.2.1 Algoritmos genéticos

Los algoritmos genéticos (Holland, 1975) son una meta heurística utilizada para resolver problemas de optimización mediante el proceso evolutivo de los organismos. El proceso evolutivo se basa en la variabilidad genética (teoría Neodarwinista[3]) de las poblaciones mediante mutaciones y recombinación de genes, donde la selecciona natural actúa como base evolutiva.

Gao y otros (2004) proponen dos modelos para predecir el tiempo de ejecución de los trabajos en un Grid. Los algoritmos genéticos se utilizan para minimizar el tiempo promedio de terminación de los trabajos mediante la asignación de los trabajos en cada nodo.

[3] **E**laborada en los años treinta y cuarenta, basándose en la variabilidad genética y en la selección natural de Darwin.

Xhafa y otros (2007) presentan un trabajo de calendarización basado en algoritmos genéticos para la asignación de trabajos en los recursos del sistema Grid. Se hace un extenso estudio sobre el uso de los algoritmos genéticos, para diseñar calendarizadores eficientes cuando se requiere minimizar el tiempo de terminación y la suma de los tiempos de finalización de todos los trabajos. Consideran dos esquemas de codificación e implementan diversos operadores genéticos.

Lorpunmanee y otros (2006) proponen la utilización de un modelo multiobjetivo para minimizar el tiempo de terminación y el tiempo de espera promedio, mediante algoritmos genéticos en ambientes Grid. Ellos presentan un modelo de asignación de trabajos para los sitios del Grid y los beneficios de utilizar el enfoque multiobjetivo.

Finalmente, Grimme y otros (2008) introducen una metodología para aproximarse a las soluciones óptimas en el problema de asignación de recursos en un Grid. Proponen utilizar NSGA-II (*Non-dominated Sorting Genetic Algorithm).* La idea es intercambiar trabajos entre sitios heterogéneos del Grid y para ello consideran el problema de encontrar una fracción óptima de toda la carga de trabajo.

II.6.2.2 Optimización por colonia de hormigas

La optimización por colonia de hormigas (Dorigo *et al.* 1998) se propuso para solucionar problemas de optimización combinatoria. Esta heurística describe el comportamiento de las hormigas para encontrar el camino más corto entre las fuentes de comida y su hormiguero. Las hormigas marcan el camino mediante una feromona, el camino marcado con más concentraciones de feromonas es el que tiene mayor probabilidad de elegirse. Este comportamiento es básico para sistemas cooperativos.

Lorpunmanee y otros (2007) comparan el desempeño del algoritmo de optimización por colonia de hormigas con varias estrategias de calendarización (FCFS, MTEDD y MTERD) para minimizar el tiempo total de tardanza. El trabajo considera la adaptación al dinamismo en sistemas Grid. Mathiyalagan y Suriya (2010) proponen una modificación al

modelo estándar, la mejora de la regla de actualización de la feromona, se hace una comparación con modelos existentes que implementa el mismo algoritmo.

II.6.2.3 Optimización por cúmulo de partículas

Optimización por cúmulo de partículas (Kennedy y Eberhart, 1995) inspirado en el comportamiento social de las bandadas de aves o los bancos de peces. Las partículas se mueven a través del espacio de soluciones con una velocidad inicial, posteriormente se aceleran en la dirección del grupo, propuesta por la mejor posición encontrada por una partícula hasta ese momento. Las partículas cambian de velocidad (aceleración) tomando en cuenta su mejor velocidad personal y la mejor velocidad global encontrada.

Para el problema de calendarización cada partícula representa una posible solución (calendario), la velocidad de las partículas se representa por un vector de números reales, este se puede mapear a números enteros y por medio de una función, representar el número de máquina donde cada trabajo se asigna. Se tienen trabajos con el objetivo de minimizar el tiempo de terminación de los calendarios y maximizar la utilización de los recursos (Zhang *et al.* 2006).

Zhu Meijie y otros (2007) proponen una mejora el algoritmo de PSO basado en vecindarios Cognitivos (NCPSO) y decisiones de partículas (SDPSO). El primer paso es mejorar el algoritmo original PSO con NCPSO y posteriormente mediante SDPSO dando lugar al modelo SDNCPSO. Se compara el desempeño del algoritmo modificado con la versión original de PSO.

Para mejorar la tasa de convergencia, se modifica el parámetro de inercia en el algoritmo de PSO. Debido a la disminución lineal del peso de la inercia la velocidad de convergencia es más rápida (Mathiyalagan, Dhepthie *et al.* 2010), de tal forma que produce un mejor rendimiento. Se han realizado trabajos para mejorar la actualización de la posición de una partícula (Izakian *et al.*, 2009), para todos los trabajos se considera una probabilidad de la ejecución en varias máquinas.

II.6.3 Otras técnicas

II.6.3.1 Lógica difusa

Los sistemas difusos consisten en una variedad de conceptos y técnicas para representar e inferir conocimiento impreciso, incierto o poco fiable. Desde la introducción original de los conjuntos difusos (Zadeh, 1965), el concepto se ha extendido a un marco de trabajo completo de la metodología difusa, la incorporación de aspectos tales como números difusos, la aritmética difusa, las relaciones difusas (Ruspini *et al.*, 1998) y el razonamiento difuso (Zadeh, 1975).

Los modelos de calendarización basados en lógica difusa han llamado la atención de la comunidad de investigadores en calendarización (Sowinski y Hapke, 2000). Actualmente se tienen trabajos que consideran modelos difusos para calendarizadores adaptativos en la selección de recursos (Zhou *et al.*, 2007).

También se emplea la lógica difusa para generar políticas de intercambio de trabajos en sistemas Grid (Fölling *et al.*, 2009), donde un controlador actúa según el estado actual del sistema. Los estados se modelan por medio de conjuntos difusos representados por funciones de pertenencia. Las políticas de decisión se basan en un conjunto de reglas, cada una de las cuales describe el estado del sistema donde se toma la decisión de aceptar o negar un trabajo.

Existen varias metodologías para generar estrategias de calendarización en línea con lógica difusa para trabajos paralelos (Franke *et al.*, 2008). La estrategia de calendarización incluye dos pasos: un criterio para ordenar la cola de espera (número de procesadores solicitados, tiempo de ejecución estimado, tiempo de espera, prioridad del grupo de usuarios) y un algoritmo que utiliza el orden para calendarizar uno o más trabajos (FCFS, EASY, CONS, calendarización voraz). El algoritmo de calendarización se basa en un sistema de reglas; el sistema de reglas clasifica todos los posibles estados de calendarización y asigna una estrategia correspondiente de calendarización

II.6.3.2 Teoría de juegos

La teoría de juegos consiste en estudiar situaciones donde un número de individuos independientes (jugadores) toman decisiones (seleccionar una estrategia) con el objetivo de mejorar (optimizar) su rentabilidad. La teoría de juegos analiza la existencia de una solución estable en la cual los jugadores no son capaces de mejorar su rentabilidad, cambiando sólo su estrategia (Morgenstern y Neumann, 1953). Inicialmente, se utilizó para solucionar problemas en economía, pero debido a su eficiencia, se ha probado en problemas de análisis y diseños de algoritmos en cómputo distribuido y redes.

Existen diversos enfoques para representar el problema de calendarización mediante teoría de juego. Young y otros (2003) proponen modelar cada trabajo como un jugador y los recursos disponibles son las estrategias donde los trabajos se pueden ejecutar. El juego utilizado es un juego estático con la información completa, se tiene conocimiento de los recursos (número de procesadores, velocidades de los procesadores) y los trabajos (tiempos de procesamiento, tiempo de llegada, grado de paralelismo). El juego se soluciona mediante la eliminación de estrategias estrictamente dominadas, las peores soluciones se descartan continuamente.

Kwok y otros (2005) proponen tres escenarios de teoría de juegos (dos para la calendarización y uno para la asignación): la ejecución de trabajos dentro de los sitios, la oferta dentro de los sitios y la oferta entre sitios. En el primer caso, la estrategia de las computadoras dentro de los sitios, cada computadora es egoísta y sólo quiere ejecutar trabajos de usuarios locales sin contribuir a trabajos remotos; el segundo problema es sobre la determinación del anuncio de las capacidades de ejecución de trabajos presentados por el calendarizador global.

El calendarizador Grid puede asignar trabajos utilizando algún algoritmo por lo que es necesario conocer las capacidades de ejecución de los sitios (cada computadora participante por sitio puede hacer una declaración); una notificación al calendarizador local especifica el tiempo necesario para ejecutar el trabajo de cada una de las computadoras que lo componen. El último caso es similar al anterior, sólo que esta vez la consulta se hace del calendarizador global hacia los diferentes sitios.

Finalmente, Rzadca (2008) estudia tres ambientes Grid, un sistema offline con trabajos paralelos en mono procesadores dedicados, un sistema online con cargas divisibles, y finalmente un sistema offline con trabajos paralelos. El modelo presentado es un Grid con organizaciones independientes egoístas que comparten sus recursos, donde solo importa el desempeño de los trabajos producidos por sus miembros. El juego resultante es el conjunto de organizaciones (jugadores) y los calendarios de los recursos locales (las estrategias), la ganancia de un jugador es el desempeño de los trabajos locales.

Es posible argumentar que parte del problema es definir una adecuada representación del problema de calendarización en Grid mediante teoría de juegos. Actualmente, se desarrollan trabajos que consideran otros campos en sistemas Grid, por ejemplo la seguridad (Kolodziej y Xhafa, 2010).

II.6.4 Algoritmos de aproximación

Schwiegelshohn (2008) demostró que no existe un algoritmo de tiempo polinomial que garantice calendarios con un factor de competitividad menor a 2 para el problema $GP_m \mid r_j,\ size_j \mid C_{max}$ y todos los casos específicos a menos que $P = NP$. Por lo tanto, el límite de calendarización en lista de sistemas multiprocesadores de 2 - $1/m$ para el caso fuera de línea, asi como para el caso en línea (Naroska *et al.*, 2002), no aplica a la calendarización en sistemas Grid. Aun más, la calendarización en lista no garantiza un factor de competitividad constante para todos los casos específicos presentados (Tchernykh *et al.*, 2006).

Además, el desempeño del algoritmo de calendarización en lista de Garey y Graham es significativamente peor en sistemas Grid que en sistemas multiprocesadores; Schwiegelshohn (2008) presenta un algoritmo en línea no clarividente que garantiza la generación de calendarios con tiempo de terminación con un cociente de desempeño constante de 5.

Tchernykh (2010) analizó estrategias de asignación de trabajos con admisibilidad adaptativa. El modelo considera las principales propiedades del Grid, para algunos casos,

máquinas con diferentes procesadores y trabajos paralelos no clarividentes. El factor de competitividad varía de 17 a infinito con el cambio del factor de admisibilidad. Se demostró que el algoritmo se beneficia bajo ciertas condiciones y permite una eficiente implementación en sistemas reales. Por otra parte, él presenta un enfoque dinámico y adaptativo para hacer frente a diferentes cargas de trabajo y propiedades del Grid.

Tchernykh y otros (2010) realizan un análisis de la heurística basada en el modelo de balanceo de la carga de Bar-Noy *et al.* (2001). Se proporciona un algoritmo de calendarización de trabajos rígidos, clarividente para el tiempo de terminacíon, y un modelo en línea para sistermas Grid con un factor de competitividad de $2e + 1$.

Pascual y otros (2009) modelan un sistema fuera de línea que consta de N clusters con exactamente m procesadores idénticos cada uno y proponen un algoritmo con una razón de desempeño, en el peor de los casos para el tiempo de terminación, igual a 4. Este problema es una versión del problema MPS (Multiple Strip Packing) donde los trabajos se consideran como rectángulos y se deben asignar sin rotación en procesadores consecutivos (Zhuk, 2007), este problema es NP- difícil.

Un caso más general de problema no clarividente considera una colección de k máquinas paralelas de diferentes tamaños y trabajos que no requieren más recursos de los disponibles en la máquina de mayor tamaño, $size_j \leq m_m$ (Tchernykh *et al.*, 2006, Zhuk *et al.*, 2004). Los autores evalúan el desempeño de varios algoritmos de 2 etapas considerando como objetivo el tiempo de terminación, ellos presentan algoritmos con un factor de competitividad de 10.

Schwiegelshohn (2008) presentó un algoritmo fuera de línea de la versión en línea clarividente que garanatiza un factor de competitividad de 3 para sistemas Grid, en donde se permite migración de trabajos entre máquinas. Bougeret (2010) considera la versión clarividente del mismo problema con trabajos que no requieren más recursos de los disponibles en la máquina más chica. El algoritmo de calendarización propuesto alcanza una razón de 5/2.

Capítulo III

Optimización extrema generalizada

III.1 Introducción

La evolución biológica ha creado fuertes y eficientes redes interdependientes donde los recursos no se desperdician. Los sistemas naturales han desarrollado estructuras complejas que optimizan el uso de los recursos en formas sofisticadas, una forma en la que estos sistemas se desarrollan surge cuando la mayoría de sus elementos ineficientes selectivamente se llevan a la extinción. La evolución, por ejemplo, avanza por selección contra las especies menos adaptadas, en lugar de lidiar con los elementos más jóvenes de las especies más adaptadas a su entorno (Boettcher y Percus, 2000).

Bak y otros (1987) propusieron una teoría para explicar la evolución de sistemas constituidos por elementos que interactúan mucho entre ellos y pueden exhibir un comportamiento característico. Esta teoría describe la organización de sistemas dinámicos en un estado complejo pero con una estructura general, los sistemas son complejos en el sentido en el cual ningún evento de carácter único en tamaño existe, no ocurren solo una vez, y no existe una escala de longitud que controle la evolución temporal de estos sistemas. Aunque la dinámica es compleja, el aspecto simplificado de las propiedades estadísticas se describe por leyes simples.

III.2 Auto organización crítica

La teoría de auto-organización crítica (Self organization critical o SOC) se basa en el comportamiento complejo que pueden desarrollar aquellos sistemas en donde el dinamismo varía abruptamente. Se ha utilizado esta teoría para explicar el comportamiento

de fenómenos naturales de sistemas en diferentes campos de la ciencia, como Geología, Economía y Biología (Bak, 1996). El modelo SOC se propuso inicialmente para explicar el origen del ruido 1 / f en sistemas físicos; la teoría de este modelo explica como interactúan entre sí los elementos de un sistema; los sistemas complejos tienen muchos elementos que evolucionan a un estado crítico, donde un pequeño cambio en alguno de los elementos, crea "avalanchas" que pueden alcanzar a cualquier elemento que forme parte del sistema (Kan Chen *et al.*, 1991).

La distribución de probabilidad de tamaño "s" de estas avalanchas se descride por una ley de potencia en la forma:

$$P(s) \sim s^{-\gamma} \tag{26}$$

Donde γ es un parámetro positivo; es decir, hay mayor probabilidad de que ocurran pequeños aludes, pero pueden ocurrir avalanchas tan grandes como el sistema con una probabilidad despreciable. En los últimos años, la teoría de la auto-organización crítica se ha utilizado ampliamente para explicar el comportamiento de sistemas complejos y fenómenos naturales en diferentes campos del conocimiento (Bak, 1996).

III.3 Modelo de Bak-Sneppen

Utilizando un modelo numérico simplificado, que se basa en la teoría de auto organización critica sin considerar los por menores de la interacción biológica entre las especies (Bak *et al.*, 1987), se pueden representar características del modelo evolutivo denominado *equilibrio puntuado* (Gould y Eldredge, 1993). Este modelo procura explicar mediante evidencias paleontológicas cómo la evolución se guía a través de saltos generacionales intercalados, con largos periodos de tiempo sin actividad evolutiva significativa (estasis). Bak y Sneppen (1993) desarrollaron un modelo simplificado de un ecosistema, donde las especies se posicionan una al lado de otra en una línea con condiciones de contorno periódicas (Figura 8).

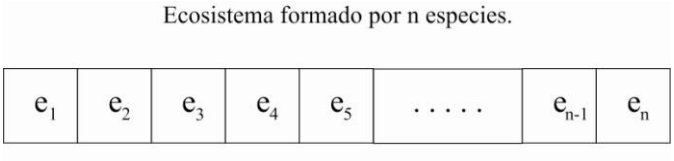


Figura 8. Distribución de especies en el modelo Bak y Sneppen (1993).

La condición de contorno periódicas implica que la especie e_1 está a un lado de la especie e_n. Cada especie tiene asignado un índice de adaptabilidad, de forma aleatoria, con distribución uniforme en el intervalo [0,1]. Las especies menos adaptadas, con el índice más bajo de adaptabilidad, sufren una mutación. El cambio en el índice de adaptabilidad significa que sus vecinos tienen que adaptarse a un competidor local nuevo y por lo tanto, éstos también son obligados a mutar, a pesar de sus altas tasas de adaptabilidad. En las siguientes generaciones, la población evoluciona a un estado crítico, donde todas las especies tienen un índice de adaptación por encima de cierto valor, denominado valor crítico.

A partir de este momento, la dinámica del sistema hace que eventualmente la capacidad de adaptación de un número de especies caiga por debajo del umbral crítico, en forma de "avalanchas" que puedan llegar a todas las especies; las especies con aptitud por debajo del nivel crítico son más activas, es decir, tienen más probabilidad de mutar, el sistema siempre tiende a volver a la situación crítica y por lo tanto pequeñas avalanchas ocurren con mayor frecuencia.

III.4 Optimización extrema

Un método de optimización basado en el modelo de Bak-Sneppen con una búsqueda dinámica que posee características de SOC permite la aparición de soluciones óptimas con rapidez. Cambiar sistemáticamente las especies menos adaptadas de la población permite escapar de mínimos locales mediante aludes. Boettcher y Percus (2000) propusieron una metaheurística para solucionar problemas de optimización combinatoria, el algoritmo propuesto se describe a continuación:

1. Inicializar una configuración C_{actual} de N variables del proyecto x_i (i = 1, N), asigne $C_{mejor} = C_{actual}$.
2. Para la configuración C_{actual} :
   a) Atribuir un índice de adaptabilidad F_i para cada variable x_i.
   b) Buscar *j* tal que $F_j \leq F_i$ para toda *i*, es decir, x_j con el peor índice de adaptabilidad.
   c) Escoger aleatoriamente una nueva configuración C' (en el vecindario de C_{actual}) tal que x_j cambie su estado.
   d) Asigne C_{actual} = C' incondicionalmente.
   e) Si V(C_{actual}) < V(C_{mejor}) entonces $C_{mejor} = C_{actual}$.
3. Repetir el paso 2 la cantidad de veces que se desee.
4. Regrese C_{mejor} y V(C_{mejor}).

donde $V(C_{actual})$ representa el valor de la función objetivo de una configuración C_{actual} y C_{mejor} la mejor configuración encontrada.

El algoritmo de optimización extrema (EO) funciona de la siguiente manera: se genera una solución inicial de forma aleatoria, a cada variable se le asigna un valor de 1 o 0 (1) y se establece como la mejor configuración encontrada (C_{mejor}). Se evalúan las variables de la configuración C_{actual} y reciben un índice de adaptabilidad (2a), se busca la variable con el peor índice de adaptabilidad (2b) y se le aplica una mutación, se cambia el valor asignado de 0 a 1 o viceversa (2c); ésta nueva solución se acepta incondicionalmente (2d). Si el valor de la función objetivo de la configuración actual ($V(C_{actual})$) es menor (problema de minimización) que el valor de la mejor configuración encontrada (V(C_{mejor})), entonces la configuración actual se considera la mejor configuración encontrada (2e). El proceso se repite hasta cumplir con un criterio de paro (3). Al finalizar, el algoritmo mantiene la mejor configuración encontrada (4).

El método de optimización extrema (EO) almacena la solución actual del problema y la mejor solución encontrada hasta el momento. La asignación de adaptabilidad a cada variable de diseño es otra característica que diferencia el modelo EO; sin embargo, “una definición general del índice de aptitud, para las variables individuales, puede resultar ambigua o incluso imposible, para cada problema de optimización un índice de adaptabilidad debe determinarse de forma diferente” (Boettcher y Percus, 2000).

El algoritmo de optimización extrema muestra buen desempeño en problemas donde es posible elegir una nueva configuración entre un conjunto de configuraciones. Sin embargo, en los casos donde solo hay una configuración posible, el algoritmo conduce a una búsqueda determinista y la convergencia a un mínimo local. Para evitar el estancamiento en mínimos locales, el algoritmo se modificó, primero las N especies se ordenadan y posteriormente un índice de adaptación se asigna a cada especie; aquella con la tasa más baja de aptitud recibe el índice 1 (k=1) y la más adaptada el índice N (k= N). En cada iteración del algoritmo, una variable se modifica de acuerdo a la distribución de probabilidad del índice *K*, dada por:

$$P(k) = k^{-\tau} \ , \ 1 \leq k \leq N, \tag{27}$$

donde τ es un parámetro ajustable positivo. Para $\tau \rightarrow 0$, la búsqueda es totalmente aleatoria, todas las variables tienen la misma probabilidad de ser elegidas para mutar, cuanto $\tau \rightarrow \infty$ la búsqueda es completamente determinista y la variable menos adaptada siempre muta. De hecho, para un valor dado de τ, cualquier variable se puede elegir para mutar, las especies con capacidad de adaptación más baja son las más propensas a elegirse. Esta variación de EO se llama τ-EO y es superior a la versión original, incluso en problemas donde EO no dio lugar a mínimos locales (Boettcher y Percus, 2000). La elección de una solución puede llevar inevitablemente a una disminución en el valor de la función objetivo en una generación, pero este mecanismo permite "saltar" fuera de regiones que conducen a mínimos locales.

A pesar de las ventajas descritas anteriormente, existen inconvenientes a la hora de implementar el algoritmo. En ocasiones, definir la aptitud de las variables del ecosistema puede resultar ambiguo e incluso imposible, también puede darse el caso donde las

variables estén fuertemente conectadas y en cada modificación las variables mejor adaptadas se destruyen. En sistemas altamente conectados el desempeño de EO es limitado debido a la evaluación de la aptitud; sin embargo estas desventajas no aplican a varios problemas o simplemente no se pueden superar (Boettcher *et al.*, 2000).

Para mejorar el desempeño de EO, se propusieron varias modificaciones. Barbay y Kenyon (2001) proponen una variación al operador de mutación. En una configuración de n especies, una especie se elige aleatoriamente para mutar (la especie con el menor valor de aptitud tiene mayor probabilidades de ser elegida) junto con sus dos vecinos más cercanos. Este proceso se llama *dinamismo extremo* (Ahmed y Elettreby, 2004) y describe como, en los ecosistemas, varias especies a menudo son forzadas a mutar al mismo tiempo, no se limita a la especie menos adaptada. En general, no existe un número definido de especies a mutar, este número es aleatorio y por lo tanto puede definirse como una generalización para un conjunto de mutaciones dado por la peor especie y sus vecinos más cercanos.

También se tienen trabajos para solucionar problemas multiobjetivo con el algoritmo EO, Chen y otros (2008) proponen el algoritmo de optimización extrema generalizada multi-objetivo basado en población (MOPEO), éste es una variación de EO para resolver problemas de optimización multiobjetivo basado en la dominancia de Pareto. MOPEO cuenta con una población de N elemento (N ecosistemas). Para cada elemento en N, se aplican n mutaciones (una por cada especie del ecosistema); los ecosistemas se ordenan dependiendo de la dominancia sobre los demás elementos, aquellos que no dominen a ningún ecosistema se mutan, finalmente se comparan cuáles de los elementos no son completamente dominados y se agregan al archivo que mantiene la solución, al final se obtiene el frente de Pareto de las soluciones.

Entre los problemas de optimización en donde se ha empleado el algoritmo de EO se pueden mencionar:

(i) MAXSAT (Menaï y Batouche, 2006), PMSAT (El *et al.*, 2006), ISAT (El y Menaï, 2006).

(ii) Grafo bipartito (Boettcher *et al.*, 2000; Boettcher y Percus, 2000).

(iii) TSP (Boettcher y Percus, 2000).

(iv) Spin glasses (Boettcher, 2005).

III.5 Optimización extrema generalizada

El algoritmo de optimización extrema generalizada (GEO) se puede definir como una metaheurística de búsqueda global, basado en el modelo de evolución natural de las especies, utilizado para solucionar problemas de optimización combinatoria. Así como otros métodos, recocido simulado (Kirkpatrick, 1984) y algoritmos genéticos (Holland, 1975), la GEO es un método estocástico aplicable a problemas no convexos o discontinuos.

Con el objetivo de aplicar el método de optimización extrema a una amplia gama de problemas de optimización, se propuso un algoritmo que generaliza la igualdad de oportunidades. El algoritmos de Optimización Extrema Generalizada (GEO - Generalized Extremal Optimization) es básicamente la versión τ - EO, pero a diferencia de ésta y la aplicación canónica, en el método de igualdad de oportunidades, la asignación del índice de adaptabilidad no se hace directamente a las variables de diseños, sino a una "población de especies", representada por un conjunto de variables binarias que codifican las variables de diseño (Figura 9). Cada especie recibe un nivel de adaptabilidad, y finalmente, sigue las normas generales, que son independientes del tipo de problema que se aborda.

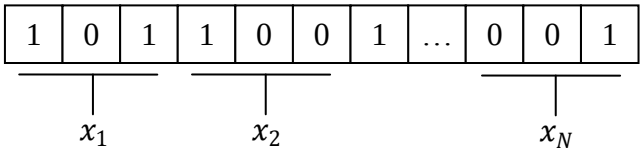


Figura 9. Codificación de N variables para el algoritmo GEO.

El enfoque GEO sigue a τ – EO para definir la mutación de las especies. Un índice de adaptación, proporcional a la ganancia (o pérdida) del valor de la función objetivo, se asigna a cada especie. Después, las especies se ordenan con base en su ganancia (o pérdida)

y se les asigna un índice de adaptación entre 1 y L, 1 para el bit menos adaptado (k=1) y L para el bit mejor adaptado (k=L). Posteriormente, una especie se elige para someterse a una mutación con una probabilidad P(k) $\propto k^{-\tau}$, donde τ es un parámetro real no negativo, este proceso se repite hasta cumplir el criterio de paro.

Basado en el enfoque propuesto por el algoritmo GEO, es indispensable considerar dos aspectos relacionados con las variables del problema. La primera es la representación de las variables del problema. Primero una cadena binaria de longitud L se inicializa de forma aleatoria, hay la misma probabilidad de obtener 1 o 0; esta configuración define una solución candidata. Sin embargo, muchas veces el sistema numérico empleado para representar las variables del problema no es binario sino decimal (o entera). Con el fin de calcular la función objetivo, GEO primero debe convertir una cadena binaria en decimal, es decir, convertir a variables de diseño (valores decimal de una codificación binaria).

La segunda consideración es la definición del número de especies necesarias para representar cada variable; este número de especies depende de los posibles valores que pueden tomar cada una de las variables del problema. Además de estos dos factores, la posibilidad de generar valores no factibles se debe considerar, al mutar una especie el rango de valores permitidos puede sobrepasarse; si este caso se presenta, es posible solucionarlo empleando un condicional que limite el rango de valores.

El parámetro libre τ crea un grado de determinismo de la búsqueda. Existe un valor óptimo de τ para cada problema (en ocasiones una serie de valores), de manera que la eficiencia de la búsqueda global se maximice. Por lo tanto, este valor de τ se puede llamar T*. Para los problemas abordados con GEO, se ha observado que T* está en el rango de 1 a 5 (Sousa *et al.*, 2005).

A continuación se describe el algoritmo GEO:

1. Inicializar aleatoriamente una cadena binaria de longitud n que codifique N variables.
2. Para la configuración C_{actual} de bits, calcular la función objetivo V_{actual} y asigne $C_{mejor} = C_{actual}$ y $V_{mejor} = V_{actual}$.
3. Para cada bit i:

   a) Mutar el bit (de 0 a 1 o viceversa) y calcular el valor de la función objetivo V_i de la cadena con la configuración C_i.
   b) Asignar la aptitud F_i. F_i indica la aptitud relativa de ganancia o perdida que tiene el mutar el bit i.
   c) Regresar el bit a su valor original.
4. Ordenar los n bits de acuerdo a su valor de aptitud, si dos o más elementos tienen el mismo valor de aptitud, el orden se realiza de forma aleatoria, pero siguiendo la misma regla.
5. Elegir un elemento i para mutar de acuerdo a la probabilidad de distribución $P_{ik} = K^{-\tau}$ donde τ es un parámetro ajustable.
6. Asignar $C_{actual} = C_i$ y $V_{actual} = V_i$.
7. Si $F_i < F_{mejor}$ entonces asigna $F_{mejor} = F_i$ y $C_{mejor} = C_i$.
8. Repetir los pasos 3 al 7 hasta que el criterio de paro se cumpla.
9. Regresar F_{mejor} y C_{mejor} .

El algoritmo de optimización extrema generalizada (GEO) funciona de la siguiente manera: se genera una solución inicial de forma aleatoria, a cada variable se le asigna un valor de 1 o 0 (1), ésta configuración actual se establece como la mejor configuración encontrada (2). Para cada bit (especie) de la configuración C_{actual} , se aplica una mutación (i-ésimo bit) y se calcula la función objetivo (V_i) de la configuración C_i (3a), se asigna la aptitud de las variables con la mutación i a la función F_i, esta aptitud indica la ganancia o pérdida con respecto a la configuración C_{actual} (3b). Posteriormente, se regresa el bit a su valor original (3c). Se ordenan los bits de acuerdo a su aptitud, los problemas de minimización ordenan los bits de forma ascendente (4), y se elige un bit con una probalidad que depende de la posición que ocupa en el ordenamiento (5), ésta mutación se mantiene en la configuración actual (6). Si el valor de la función objetivo de la configuración actual (F_i) es menor (problema de minimización) que el valor de la mejor configuración encontrada (F_{mejor}), entonces la configuración actual se considera la mejor configuración encontrada

(7). El proceso se repite hasta cumplir con un criterio de paro (8). Al finalizar, el algoritmo mantiene la mejor configuración encontrada (9).

Entre las modificaciones propuesta al algoritmo GEO se pueden mencionar la llamada GEO_{var} (Mainenti *et al.*, 2008); esta implementación ordena jerárquicamente las especies que componen cada una de las variables de diseño, cada variable de manera independiente, de tal modo que en lugar de mutar una especie dentro del ecosistema, se muta una especie por cada variable del problema.

El uso de variables binarias impone una limitante a los valores de las variables de diseño. Las variables están definidas por una población de especies, cuando la especie más significativa se forza a mutar, el valor de la variable de diseño cambia a otro y es imposible establecer valores intermedios para esta variable de diseño. Esta característica puede limitar el algoritmo en la búsqueda de mejores soluciones (Mainenti *et al.*, 2008).

GEO_{real} es básicamente el algoritmo GEO, sin embargo, las variables se representan por números reales, los valores asignados a las especies son las variables de diseño. Para realizar la mutación de las especies se genera un número con distribución gaussiana, el próximo valor de cada especie se define por la suma del valor actual y un número aleatorio generado (28).

$$x'_i = x_i + N(0,\sigma)x_i \tag{28}$$

La segunda modificación de GEO_{real} considera las N especies, un vector P de valores aleatorios se genera con una distribución gaussiana, para cada variable $N_j(0,\sigma_j)$ donde las desviación estándar σ_j supone que P tiene diferentes valores, uno para cada índice j (j=1, 2,…,P). Este conjunto de valores se utiliza para mutar todas las variables, mediante la ecuación:

$$x'_{ji} = x_i + N_j\left(0,\sigma_j\right)x_i \tag{29}$$

Las desventajas de estas implementaciones son: el incremento del número de parámetros que se deben ajustar y la generación de valores fuera del rango permitido al

momento de asignar un nuevo valor de adaptabilidad a las especies (para solucionar este problema se utiliza una función de penalización).

Entre los problemas de optimización donde se ha empleado el algoritmo de GEO se pueden mencionar:

i. Diseño de tubos de calor (Sousa *et al.*, 2002; Sousa *et al.*, 2003; Sousa *et al.*, 2004; Sousa, Vlassov *et al.*, 2004; Vlassov *et al.*, 2006).
ii. Diseño térmico de satélites (Galski *et al.*, 2007).
iii. Constelaciones de satélites (Galski *et al.*, 2005).
iv. Perfiles planeadores (Sousa *et al.*, 2002; Sousa *et al.*, 2002 y 2003).
v. Pruebas de software (Abreu *et al.*, 2005).
vi. Identificación de las propiedades ópticas de los materiales (Sousa *et al.*, 2005).
vii. Optimización de estructuras (Sousa y Takahashi, 2005).
viii. Calendarización de trabajos en sistemas multiprocesadores (Switalski y Seredynski, 2008, 2009 y 2010).

III.6 GEO para la calendarización de trabajos en sistemas Grid

Durante este capítulo se presentó la estrategia de optimización extrema generaliza desde sus inicios hasta las últimas modificaciones sugeridas para mejorar el desempeño; sin embargo, para emplear esta estrategia en el problema de calendarización en Grid, primero se define la representación empleada. El algoritmo GEO se ha empleado para solucionar problemas de calendarización en sistemas multiprocesadores, mostrando un buen desempeño en comparación con otras estrategias evolutivas empleadas como algoritmos genéticos y recocido simulado (Switalski y Seredynski, 2008,2009 y 2010), con el objetivo de minimizar el tiempo de terminación de los trabajos. La implementación propuesta utiliza una representación binaria, no se utiliza un mecanismo para verificar los valores de las variables porque estos no exceden el rango permitido.

III.6.1 Representación

Para afrontar un problema es preciso definir una representación adecuada de las soluciones, en este caso, los calendarios. Una representación adecuada permite ahorrar trabajo innecesario, simplificar el proceso de búsqueda y concentrarse en la aplicación del algoritmo; estas ventajas se deben a que el proceso de implementación se simplifica, el diseño de los operadores es más sencillo y finalmente se logra un buen desempeño del algoritmo. Tomando en cuenta la representación de los calendarios y el enfoque propuesto por el algoritmo GEO, es posible representar un calendario por medio de un ecosistema, en otras palabras, el ecosistema es una solución al problema de calendarización.

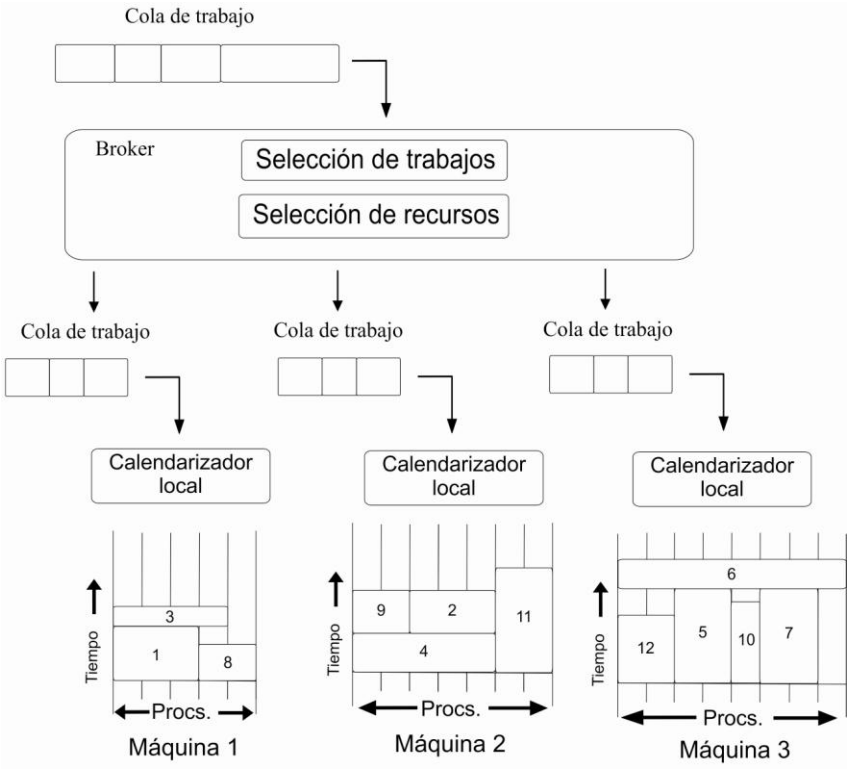
La versión original de GEO propone una representación binaria; sin embargo, el uso de dicha representación tiene varias desventajas (cadenas muy largas, el valor de un especie puede suprimir las contribuciones de aptitud de otras especies en el grupo, puede producir soluciones ilegales, no se pueden alcanzar valores intermedios, etc.). También existen representaciones con números reales, pero de la misma forma se tienen problemas para trabajar con esta representación (definir funciones de mapeo entre los valores reales y la representación de las máquinas, producir soluciones ilegales, etc.). Finalmente se opta por emplear una representación mediante números enteros, esta representación evita los problemas generados por la representación binaria y de números reales.

Los trabajos se representan por medio de las especies que componen el ecosistema del modelo GEO, donde el índice de las especies se asocia al índice de los trabajos. Cada especie es una variable de diseño, por lo tanto a las especies se les puede asignar un valor entero. El intervalo de los posibles valores lo define el número de máquinas o sitios que componen el Grid. De esta forma, el valor asociado a cada especie representa la máquina en la cual se ejecuta el trabajo representado por la especie. Por ejemplo, supongamos que la especie e_j tiene asignado un valor i, esto significa que el trabajo J_j se asigna a la máquina N_i.

La representación propuesta limita la búsqueda de soluciones en la asignación de los trabajos a las máquinas del Grid. Una vez que los trabajos se asignaron a las máquinas (primer nivel del Grid), éstas se envían para calendarizarse en cada una de las máquinas

(segundo nivel del Grid); cuando los trabajos llegan a la cola local de cada sitio, los trabajos se deben asignar a los procesadores por medio de un algoritmo de calendarización; el orden de llegada de los trabajos se considera por medio de alguna técnica de calendarización, por lo tanto, el orden de los trabajos debe tomarse en cuenta.

a)



b)

Tarea	1	2	3	4	5	6	7	8	9	10	11	12
Índice de Orden	1	4	3	1	2	5	4	2	3	3	2	1
Índice de Asignación	1	2	1	2	3	3	3	1	2	3	2	3

Figura 10. Calendarización de trabajos en Grid (a) y la representación empleada en el algoritmo GEO (b).

Si la representación propuesta se emplea, los trabajos representados por las primeras especies inician su ejecución antes que los representados por las últimas especies, lo cual

limita la búsqueda de soluciones. Por lo tanto, no solo es indispensable asignar los trabajos a las máquinas, también se debe considerar el orden en el cual los trabajos llegan a la cola local de cada máquina. Para solucionar este problema, las especies se representan por una tupla, donde el primer elemento define la máquina donde se asigno el trabajo (índice de asignación) y el segundo elemento representa el orden de los trabajos en la cola local de la máquina (índice de orden). La Figura 10 muestra la representación empleada para definir los calendarios, en la parte superior (a) se muestra la calendarización de los trabajos con base en la representación GEO mostrada en la parte inferior (b).

III.6.2 Solución inicial

El algoritmo de GEO trabaja con un solo ecosistema (calendario), aunque hay modelos para trabajar con conjunto de ecosistemas. La solución inicial para el problema de calendarización se puede generar de formas variadas, la representación propuesta no genera ambigüedad en la asignación de las máquinas porque las especies solo tienen asignado un valor (índice de asignación). Para definir el índice de orden se debe tener más cuidado, el proceso no es aleatorio, los trabajos de cada máquina se separan y posteriormente un índice de orden se asigna a cada trabajo. Se proponen tres métodos para generar soluciones iniciales:

DISTRIBUCIÓN ALEATORIA (*Distribución_aleatoria*). Considera el número de trabajos y de máquinas. Para cada trabajo se identifica el tamaño (número de procesadores necesarios para su ejecución), una búsqueda determina las máquinas que cumplen con el tamaño del trabajo y de forma aleatoria se escoge una máquina para procesar el trabajo (en el intervalo de máquinas que pueden procesar el trabajo), todas las máquinas tienen la misma probabilidad de ser elegidas. Durante el proceso de asignación también se asigna el índice de orden. El índice de orden se define como el total de trabajos asignados a la máquina, lo que significa que el índice de orden está dado por el orden en el que se asignó cada uno de los trabajos en cada máquina.

DISTRIBUCIÓN DE LA CARGA (*Distribución_carga*). Considera el número de trabajos, máquinas, procesadores y el tiempo de procesamiento de los trabajos. Inicialmente, se define una cota de procesamiento para cada máquina:

$$P_i = (\sum_{j=1}^{n} p_j \Big/ \mathrm{m_G}) * m_i \qquad (30)$$

donde p_j es el tiempo de procesamiento del trabajo j, $\mathrm{m_G}$ es el número total de procesadores en el Grid y m_i es el número de procesadores en la máquina i. Cada máquina mantiene un registro de la cota de procesamiento y el tiempo de procesamiento de los trabajos asignados a la máquina.

Para cada trabajo, se identifica el tamaño (número de procesadores necesarios para su ejecución); una búsqueda determina las máquinas que cumplen con el tamaño del trabajo, el trabajo se asigna en la primera máquina si ésta cumple con la siguiente condición: Al asignar un trabajo no se sobrepasa la cota de procesamiento establecida para la máquina (P_i); es decir, el tiempo de procesamiento de la carga del sitio más el tiempo de procesamiento del trabajo es menor que la cota establecida. Si la condición no se satisface, el trabajo se asigna en la máquina siguiente (las máquinas están ordenadas por el número de procesadores), el proceso se repite hasta que el trabajo lo acepte alguna máquina.

DISTRIBUCIÓN DEL TRABAJO O AREA (*Distribución_area*). Considera el número de trabajos, máquinas, procesadores, tiempo de procesamiento y tamaño de los trabajos. Inicialmente se define una cota de trabajo[4] (área) para cada máquina:

$$T_i = (\sum_{j=1}^{n} a_j \Big/ \mathrm{m_G}) * m_i \qquad (31)$$

[4] Para no causar ambigüedad con los términos, se utiliza el término área para referirse al trabajo (tiempo de procesamiento por número de procesadores).

donde $a_j = p_j * size_j$ es el área total del trabajo j, m_G es el número total de procesadores en el Grid y m_i es el número de procesadores en la máquina i. Cada máquina mantiene un registro de la cota de área y el área de los trabajos que se han asignado a la máquina.

Posteriormente, para cada trabajo se identifica el tamaño (número de procesadores necesarios para su ejecución); una búsqueda determina las máquinas que cumplen con el tamaño del trabajo; el trabajo se asigna en la primera máquina si esta cumple con la siguiente condición: Al asignar un trabajo no se sobrepasa la cota de área establecida para la máquina (T_i); es decir, el área de la carga del sitio más el área del trabajo es menor que la cota establecida. Si la condición no se satisface, el trabajo se asigna en la máquina siguiente (las máquinas están ordenadas por el número de procesadores), el proceso se repite hasta que el trabajo lo acepte alguna máquina.

En el peor de los casos, para ambas distribuciones, los trabajos no se pueden asignar a ninguna máquina, porque la cota establecida se sobrepasa, entonces el trabajo se asigna a la última máquina (máquina con el mayor número de procesadores). Durante el proceso de asignación, también se designa el índice de orden, se tiene el total de trabajos asignados a cada máquina, el índice de orden está dado por el orden en el que fueron asignados cada uno de los trabajos en cada máquina. Cabe mencionar que la asignación aleatoria genera una solución diferente cada vez que se emplea, mientras que los otros dos procedimientos generan la misma solución, siempre y cuando la carga de trabajo no se altere.

III.6.3 Operadores de mutación

La mutación es un nombre genérico dado a los operadores que generan pequeños cambios en el ecosistema, mediante los cuales se realiza la búsqueda en el espacio de solución. A diferencia del modelo original, donde una mutación representa un cambio en una variable binaria, en este caso las mutaciones se basan en una reasignación del valor de cada especie. Los procedimientos empleados para generar mutaciones en el algoritmo GEO se presentan a continuación:

INTERCAMBIO (*swap*): Se selecciona un trabajo de una máquina fuente y un trabajo de una máquina destino, se verifica que ambas máquinas cumplan con el número de procesadores necesarios para ejecutar los trabajos y se procede a intercambiar los trabajos. La ventaja de esta mutación es que los índices de orden de las demás especies no se modifican.

La Figura 11 muestra el intercambio de los trabajos 2 y 6, primero se tiene un calendario sobre el cual se aplica una mutación de intercambio (a), se eligen dos trabajos para ser intercambiados, posteriormente se intercambian los índices de asignación y orden entre los trabajos (b).

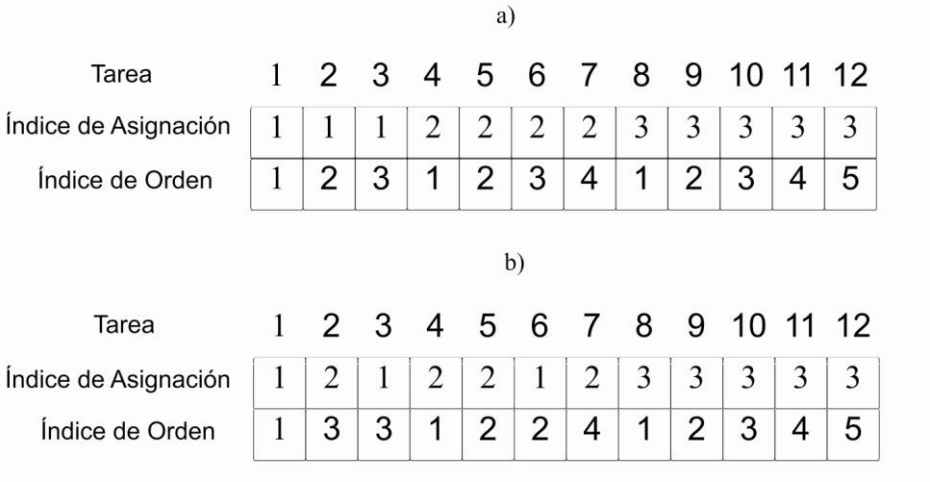


Figura 11. Operador de mutación (intercambio), permite intercambiar dos trabajos entre máquinas.

INSERCIÓN (*insert*): un trabajo de la máquina fuente es insertado en una máquina destino, la máquina destino debe cumplir con el número de procesadores necesarios. La modificación del índice de asignación no provoca ningún problema cuando el trabajo se reasigna a otra máquina, pero esta modificación sí afecta al índice de orden. La alteración en el índice de asignación provoca problemas en los índices de orden en ambas máquinas, la máquina fuente tiene inconsistencia en los índices (falta un índice), mientras que la

máquina destino puede tener un índice repetido o los índices pueden ser inconsistentes (falta de uno o varios índices).

Para solucionar este problema, se propone asignar un índice de orden aleatorio entre uno y el total de trabajos en la máquina destino, esta modificación limita a lidiar con el problema de índices repetidos en la máquina destino. Para solucionar el problema de índice repetido se emplea la siguiente estrategia: Se identifican los trabajos con el índice de orden mayor o igual al índice elegido para el nuevo trabajo y se les asigna un índice mayor en una unidad. De igual forma los índices de los trabajos de la máquina fuente se ajustan, a cada trabajo con índice de calendarización mayor al trabajo que se desea insertar en otra máquina se les asigna un índice menor en una unidad.

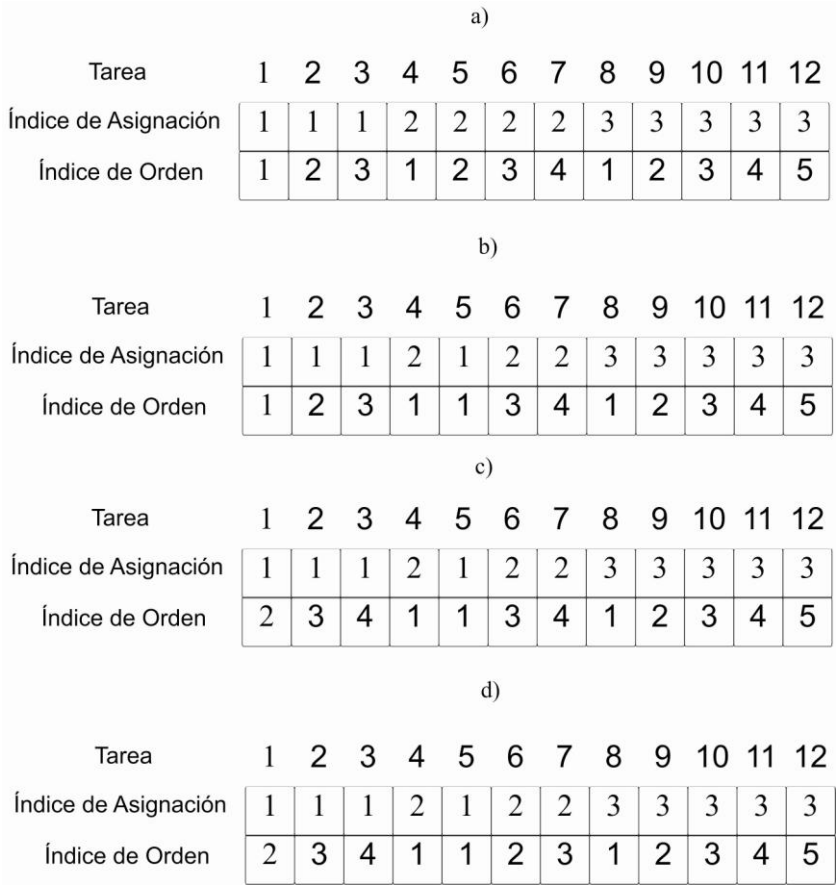


Figura 12. Operador de mutación (inserción), permite insertar un trabajo de una máquina en otra.

La Figura 12 muestra la inserción de la tarea 5 en la máquina 1, primero se tiene un calendario sobre el cual se aplica una mutación de inserción (a), una vez definida la tarea se asignan los índices de asignación y orden (b), se corrigen los índices de orden de la máquina destino (c) y finalmente se corrigen los índices de orden de la máquina fuente (d).

III.6.5 Selección

El proceso de selección lo define la función de probabilidades descrita en el algoritmo GEO, se elige una especie a ser mutada con la probabilidad del orden que ocupa respecto a la ganancia o perdida de su aptitud (26). En cada iteración se almacena la mejor solución encontrada hasta el momento y la solución generada al mutar una especie.

La asignación de la aptitud la define la evaluación del calendario respecto a una métrica (objetivo), para evaluar el desempeño se cuenta con un conjunto de métricas (Sección II.5). Por lo tanto, es indispensable definir un objetivo para guiar la búsqueda.

Capítulo IV

Experimentos y resultados

IV.1 Introducción

La simulación de experimentos se utiliza para evaluar y comparar estrategias de calendarización, las simulaciones se utiliza debido a que son configurables, repetibles y, en general, rápidas. Existen dos limitaciones principales en las metodologías de simulación utilizadas por la investigación en calendarización: (1) no existe un estándar de simulación en la comunidad de investigación, y (2) los modelos tradicionales y las hipótesis acerca de las plataformas de computación ya no son válidos para plataformas modernas debido a los modelos de red simple utilizado en la literatura, la mayoría de la programación no se aplica a las plataformas de computación moderna (Jiang *et al.*, 2007).

En consecuencia, es necesario usar un marco de simulación para llevar a cabo investigaciones sobre la programación de aplicaciones distribuidas. Un marco útil de simulaciones debe tener: (1) buena usabilidad, (2) correr simulaciones rápidamente, (3) construir simulaciones configurables, extensibles y ajustables, y (4) escalabilidad (simulaciones con decenas de miles de recursos y trabajos) (Legrand 2006).

El primer paso para la simulación de experimentos es definir los factores que intervienen en el proceso (carga de trabajo, configuración Grid, ajuste de parámetros, etc.). Es importante considerar estos parámetros porque de ellos depende si el modelo empleado está apegado a la realidad de los calendarizadores comerciales.

IV.2 Carga de trabajo

La carga de trabajo es el conjunto de trabajos utilizados para evaluar los algoritmos de calendarización. En el contexto de calendarización de trabajos, se debe decidir qué carga de trabajo usar y cuáles medidas considerar. Estas decisiones tienen consecuencias sutiles que se pueden pasar por alto fácilmente. La elección de una carga de trabajo afectará casi todas las evaluaciones cuantitativas (Frachtenberg y Feitelson, 2005).

Como regla general, se recomienda utilizar múltiples cargas de trabajo, cada una deber ser lo suficientemente grande para producir resultados medibles. Es importante comprender las características de la carga de trabajo y sus efectos en la evaluación de los resultados, si es posible considerar diferentes modelos de cargas de trabajo y comparar los efectos en la evaluación (Feitelson, 2003).

La carga de trabajo empleada se compone de trabajos con diferentes características (distintos centros de cómputo, zona horaria, tipo de aplicaciones, número de procesadores solicitados por los trabajos, entre otros). El análisis estadístico permite interpretar los resultados de las simulaciones y sustentarlos. Las características de la carga de trabajo afectan el desempeño del Grid, por ejemplo, algunas estrategias de calendarización generan buenos resultados con trabajos grandes y otras con trabajos pequeños. Al conocer las características de la carga de trabajo utilizada se puede afirmar si los resultados de las simulaciones son de calidad (comparados con otros) y apegados a modelos reales.

Los experimentos se basan en una carga de trabajo semi-sintética (denominada sintética porque contiene trabajos de cinco centros de cómputo de alto desempeño). Las cargas pertenecen a los siguientes centros (Feitelson, 2005):

- Cornell Theory Center.
- High Performance Computing Center North.
- Swedish Royal Institute of Technology.
- Los Alamos National Lab.
- The Advanced School for Computing and Imaging (Anoep *et al.*, 2007).

Los registros de los centros de cómputo componen la carga de trabajo, cada uno contiene información sobre los trabajos, la información de cada trabajo se compone de 18 campos; estos campos especifican el número de usuario, tiempo de ejecución, tiempo solicitado, número de procesadores asignados, tiempo de llegada, tipo de trabajos, entre otros. Adicionalmente, se incluye información del centro de cómputo de origen (máquina, número de procesadores, zona horaria, periodo del registro, entre otros datos).

El archivo de la carga de trabajo sigue las convenciones del formato estándar de cargas de trabajo (*Standard Workload Format, SWF*) (Feitelson, 2005), la carga de trabajo contiene 30,000 trabajos (considerando 7 días de trabajos propuestos), con esta cantidad de trabajos se realizan 30 experimentos con lotes de 1,000 trabajos cada uno. Para tener una carga uniforme en los datos es necesario aplicar filtros, los filtros permiten elegir los trabajos que se pueden ejecutar en la configuración Grid propuesta. Para el Grid propuesto se deben filtrar los trabajos que soliciten un número de procesadores mayor a la cantidad de procesadores en la máquina más grande (máquina con el mayor número de procesadores).

Los trabajos provienen de diferentes partes del mundo, con zonas horarios variadas, la diferencia de los horarios genera problemas a la hora de calendarizar los trabajos. Por lo tanto, un filtro se utiliza para ajustar la zona horaria, la zona horaria establecida es GMT-7 por ser la mínima encontrada. También los identificadores de los usuarios se modifican, un usuario con un identificador específico para un centro de cómputo es diferente de otro usuario con el mismo identificador para otro centro de cómputo, el usuario 5 del Cornell Theory Center es diferente al usuario 5 de Los Alamos National Lab.

IV.2.1 Estadísticas de la carga de trabajo

El desempeño de un Grid depende de las características de los trabajos, estrategias y parámetros de los algoritmos. Por ejemplo, algunas estrategias consideran en mejorar un criterio específico y bajo ciertas condiciones, por tal motivo, el análisis estadístico de la carga de trabajo ayuda a sustentar e interpretar los resultados de los experimentos.

Las estadísticas de los trabajos sometidos (hora de llegada, tamaño de los trabajos, recursos consumidos, etc.) contribuyen a decidir la calidad de los resultados, además de reflejar la relación entre los usuarios y los trabajos, el análisis permite identificar problemas con la carga de trabajo y decidir si ésta es adecuada para realizar los experimentos.

El primer parámetro analizado es el número de trabajos sometidos por hora, la Figura 13 muestra la cantidad de trabajos sometidos por hora, los trabajos aumentan a partir de las primeras horas hasta el medio día, posteriormente desciende para mantenerse en promedio por las noches. Los trabajos llegan de diferentes partes del mundo, con la normalización de la zona horario se espera que la llegada de los trabajos sea uniforme durante el día, sin embargo, no sucede así. Este comportamiento de los trabajos puede deberse a los trabajos enviados por un centro de cómputo (los usuarios pertenecientes a una zona horaria envían una mayor cantidad de trabajos).

La Figura 14 muestra el número promedio de trabajos por días de la semana, la cantidad de trabajos suministrados es constante durante la semana, aumentando el día viernes (día 5) y disminuyendo el fin de semana (días 6 y 7).

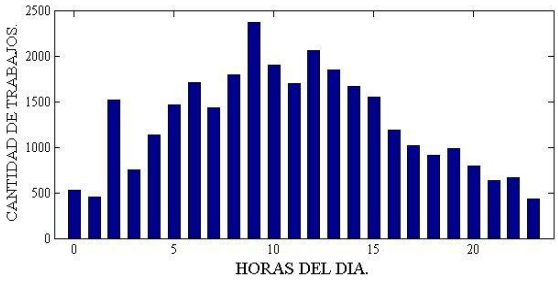


Figura 13. Número promedio de trabajos sometidos por hora.

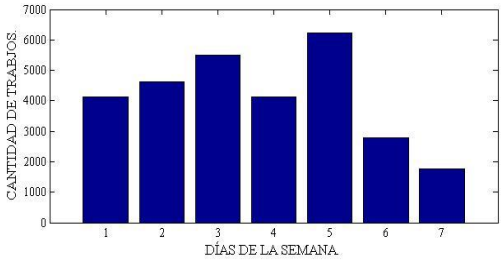


Figura 14. Número promedio de trabajos sometidos por día.

Además de la cantidad de trabajo, es interesante analizar el consumo de recursos por día y por semana. El patrón de los recursos consumidos por hora difiere de la llegada de los trabajos, a partir de la hora 6 la utilización de los recursos aumenta considerablemente hasta las últimas horas del día. El número de trabajos no influye significativamente en el

consumo de recursos (Figura 15), la diferencia entre el número de trabajos y los recursos consumidos no es muy marcada, la cantidad de recursos disponibles es mayor en la última hora y las primeras cinco del día.

La distribución de recursos consumidos en la semana es muy pareja (Figura 16), el domingo desciende un poco pero no de forma significativa. El fin de semana (días 6 y 7) el número de trabajos que llegan al Grid y los recursos empleados discrepan, lo que no sucede durante la semana, los factores que influyen en tal comportamiento son: (i) la cantidad de recursos utilizados por los trabajos, los trabajos sometidos el fin de semana consumen una mayor cantidad de recursos comparados con los trabajos sometidos durante la semana, (ii) el tiempo de utilización de los recursos, el tiempo de procesamiento de los trabajos sometidos el fin de semana es mayor (en promedio) que los trabajos sometidos entre semana, y (iii) los trabajo del viernes influyen significativamente en el consumo de recursos para el fin de semana, los trabajos sometidos el día viernes se ejecutan durante el fin de semana.

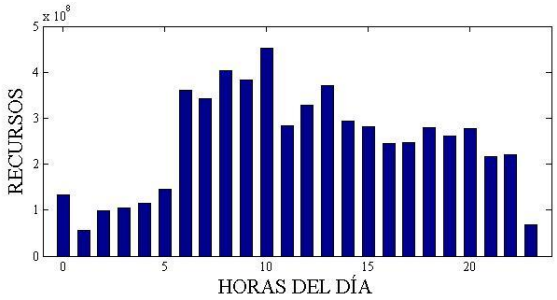


Figura 15. Número promedio de recursos consumidos por hora.

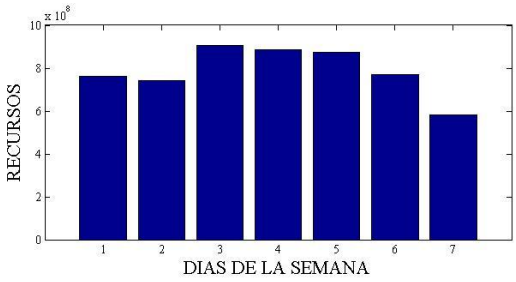


Figura 16. Número promedio de recursos consumidos por día.

La Figura 17 muestra la cantidad de trabajos sometidos por usuarios, los usuarios con identificador 600 someten hasta 5000 trabajos; el resto de los usuarios envían trabajos en menor medida. La Figura 18 muestra el número de trabajos sometidos por usuario con un límite de 1000 trabajos, los usuarios con identificadores mayores a 200 y menores a 500 casi no envían trabajos al Grid.

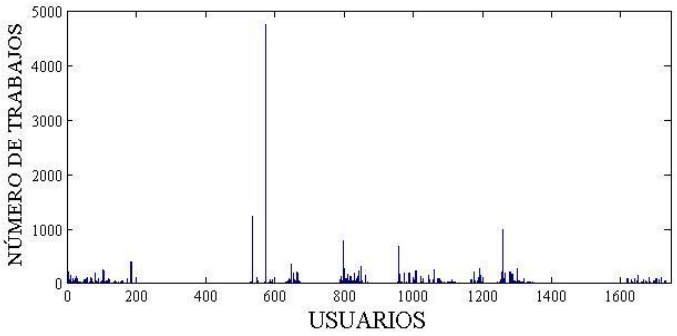


Figura 17. Número de trabajos sometidos por usuario.

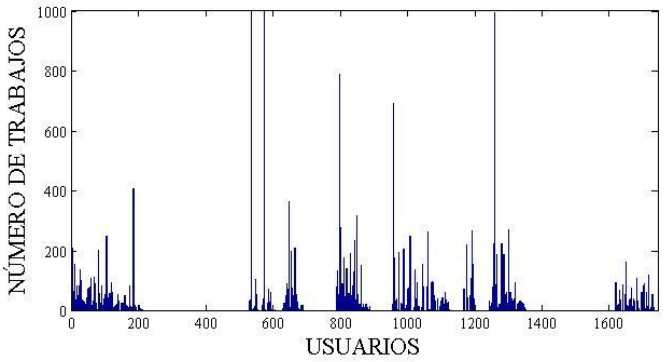


Figura 18. Número de trabajos sometidos por usuario. Usuarios con menos de 1000 trabajos sometidos.

La Figura 19 muestra los trabajos en forma ordenada (los usuarios con mayores trabajos se listan primero); la mayoría de los usuarios someten menos de 100 trabajos, aproximadamente 600 usuarios envían trabajos constantemente al Grid.

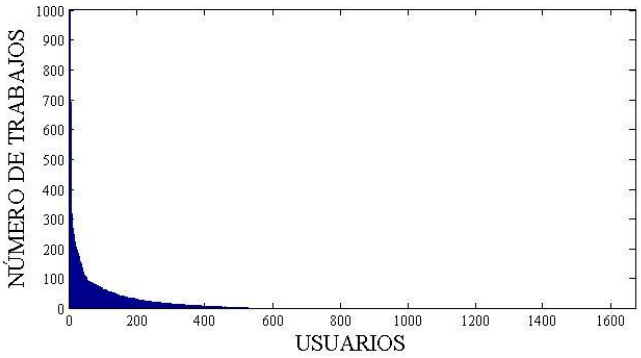


Figura 19. Número de trabajos ordenados sometidos por los usuarios.

Las Figuras 20 y 21 muestran el tamaño de los trabajos sometidos; los trabajos sometidos solicitan en mayor medida procesadores en potencia de dos (2, 4, 8, 16, 32, 64, 128 y 256), el resto de solicitudes es menor. La mayoría de los trabajos son secuenciales porque el número de procesadores más solicitado es 1, pocos usuarios solicitan más de 256 procesadores. El tamaño máximo de los trabajos permitidos lo define la configuración Grid (Tabla I), para este caso es de 340 procesadores.

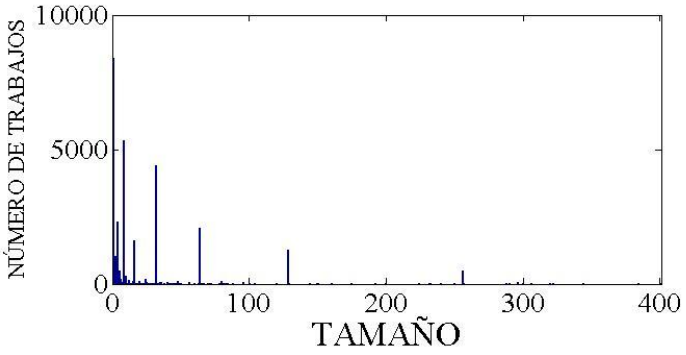


Figura 20. Número de trabajos sometidos por tamaño.

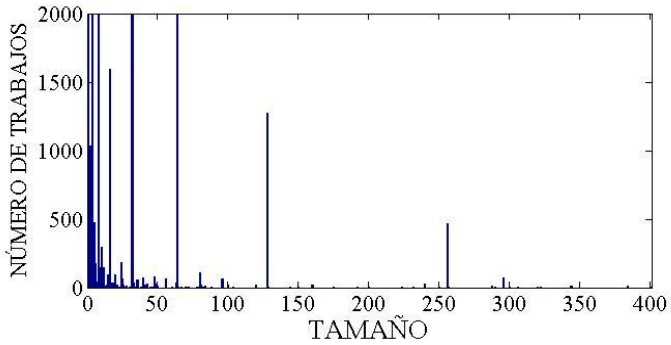


Figura 21. Número de trabajos sometidos por tamaño. Vista detallada de los tamaños con menos de 2000 trabajos.

El tiempo de procesamiento de la mayoría de los trabajos es menor a una hora (Figura 22); al Grid llegan trabajos relativamente pequeños, la vista detallada (Figura 23) permite visualizar un patrón en los tiempos de ejecución de los trabajos, los trabajos se concentran en un intervalo de 20 horas (de 1 hasta 20 horas). Los trabajos que solicitan tiempo de procesamiento mayor a 20 horas son pocos.

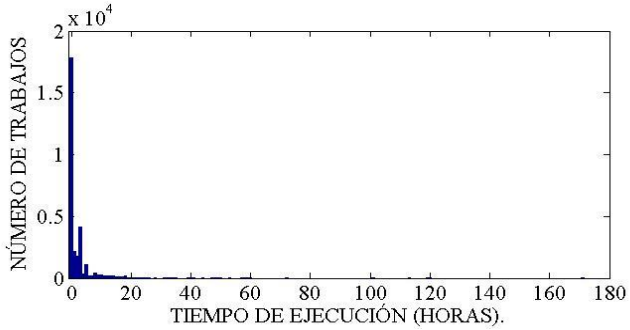


Figura 22. Número de trabajos sometidos por tiempo de ejecución.

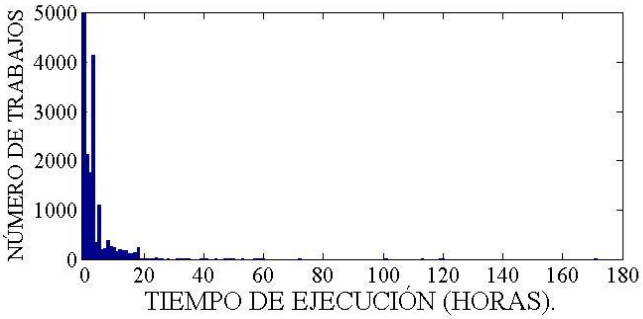


Figura 23. Número de trabajos sometidos por tiempo de ejecución.
Tiempos de ejecución menores a 5000 trabajos.

El tipo de trabajos predominantes son aquellos que cuentan con un tiempo de procesamiento de menor a 20 y un tamaño menor a 50 (Figura 24), además el tiempo de procesamiento entre los trabajos que solicitan 32, 64, 96,128 y 256 procesadores es muy parejo, aunque la cantidad de trabajos que solicitan 96 procesadores es menor que los demás.

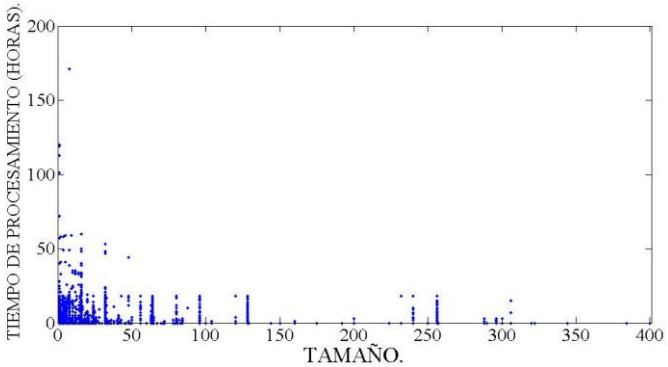


Figura 24. Dispersión de trabajos sometidos por tamaño y tiempo de ejecución.

IV.3 Configuración Grid

El trabajo usa un Grid jerárquico de dos niveles para realizar las simulaciones, por lo tanto, dos estrategias se requieren para calendarizar los trabajos: una estrategia de asignación (primer nivel del Grid) y otra de calendarización (segundo nivel del Grid). Los trabajos llegan a la cola de trabajo del Grid (cola de trabajo global) y se asignan de acuerdo al orden en que van llegando (*FIFO*).

Tabla I. Configuración Grid utilizada en los experimentos.

CONFIGURACIÓN GRID.	
Nodo.	Número de procesadores.
1. Leiden University Pentium; III Linux cluster (DAS2.fs1).	64
2. University of Amsterdam; Pentium-III Linux cluster (DAS2.fs2).	64
3. Delft University of technology. Pentium-III Linux cluster (DAS2.fs3).	64
4. Utrecht University. Pentium-III Linux cluster (DAS.fs4).	64
5. The Swedish Royal Institute of Technology (KTH) IBM SP2.	100
6. Vrije University Amsterdam; Pentium-III Linux cluster(DAS.fs0)	144
7. HPC2N - High Performance Computing Center North, Sweden.	240
8. Cornell Theory Center (CTC). IBM SP2.	430

El Grid se compone de máquinas paralelas; cada máquina tiene una configuración donde se definen las características de la máquina. La configuración especifica el número de máquinas y procesadores; el total de procesadores por máquina puede variar, generalmente en potencias de dos. La configuración empleada para los experimentos se define en la Tabla I. Cada uno de los nodos se representa por una máquina paralela existente, esto permite asegurar que el modelo empleado se puede recrear en la vida real. El total de procesadores en el Grid es de 1170.

IV.4 Configuración de los algoritmos

El desempeño de la heurística GEO depende de los parámetros de configuración, por lo tanto, la definición de éstos resulta de gran importancia. Los parámetros ajustables para la implementación propuesta son dos: la cantidad de mutaciones y la definición del parámetro τ.

El número de mutaciones lo define el modelo original; GEO plantea realizar una mutación a cada una de las especies en el ecosistema por iteración, sin embargo, para el problema de calendarización y su representación significa modificar la asignación de cada tarea y evaluar el resultado. Este esquema consume mucho tiempo (al evaluar cada una de las posibles modificaciones). Por lo tanto, la cantidad de mutaciones se redujo a solo considerar el número de máquinas empleadas, ver Tabla I.

La implementación GEO para el problema de calendarización tiene dos tipos de mutaciones, intercambio e inserción; definir el número de asignaciones es un problema fundamental que se debe considerar. Cuatro modelos se implementaron, el número de mutaciones distingue cada uno de los modelos. El proceso de mutación se realiza de la siguiente manera: para todas las máquinas, un trabajo se elige aleatoriamente y posteriormente es asignado o intercambiado por otro trabajo en diferentes máquinas, el total de mutaciones se define en la Tabla II.

Tabla II. Número de mutaciones por implementación.

Nombre	Número de inserciones	Número de intercambios
GEO_1	1	1
GEO_2	2	2
GEO_3	4	4
GEO_4	8	8

La representación del modelo GEO_4 es la más completa, porque considera aplicar una mutación de intercambio e inserción entre todas las máquinas del Grid; los otros modelos sólo consideran una parte de las máquinas. Los modelos que no consideran el total de máquinas funcionan de forma aleatoria; para todas las máquinas un trabajo es elegido de forma aleatoria, se buscan los sitios donde el trabajo se puede procesar y un sitio se selecciona para realizar la mutación.

La configuración de los experimentos propuestos se describe en la Tabla III, donde el criterio de paro se ha empleado en varios trabajos (Castro, 2010), así como el valor de τ (Switalski y Seredynski, 2008, 2009 y 2010).

Tabla III. Configuración GEO.

Parámetros GEO por experimento.	
Criterio de paro.	10 iteraciones sin mejora.
Número de inserciones.	Ver Tabla II
Número de mutaciones.	Ver Tabla II
Valor de τ.	2.5
Criterio que guie la búsqueda.	Tiempo de espera promedio.
Solución inicial.	*Distribución_area*.

Las estrategias para comparar el desempeño del algoritmo GEO son: MCT, MLB, MLp, MPL, MTA y Random (Sección II.4.1), la Tabla IV presenta el nombre de las estrategias empleadas. Para todas las estrategias de asignación se utiliza la estrategia Backfilling EASY First Fit para calendarizar los trabajos localmente (Sección II.4.2). La ventaja de GEO sobre las estrategias descritas en la Tabla IV es la posibilidad de elegir un objetivo para guiar la búsqueda, además de poder reasignar los trabajos en una máquina diferente.

Tabla IV. Nombre de los algoritmos para comparar con GEO.

Estrategias de asignación.	Estrategias de calendarización.
MCT	BEFF
MLB	
MLp	
MPL	
MTA	
Random	

IV.5 Resultados experimentales.

Para evaluar los resultados entre los modelos propuestos (Sección IV.4) se define una configuración para comparar el desempeño. La Tabla III muestra la configuración empleada; sin embargo, el número de experimentos se redujo a 10 debido al modelo GEO_4; este modelo tarda en promedio 6422 minutos (cerca de 108 horas) en entregar una solución, se puede consultar el Apéndice E para ver las características de las computadoras donde se realizaron las simulaciones.

Tabla V. Evaluación de los modelos propuestos.

	Desaceleración acotada.	Factor de competitividad.	Tiempo de espera promedio.	Tiempo (min)
Solución inicial	1083.24	2.92	93938	1
GEO_1	346.41	2.13	46290	232
GEO_2	173.8	2.01	24432	291
GEO_3	107.9	1.93	20068	471
GEO_4	7.9	1.47	6828	6422

La Tabla V permite apreciar la calidad de la solución encontrada por GEO_4 pero el tiempo empleado es excesivo, el trabajo pretende encontrar una relación entre la calidad de la solución y el tiempo necesario para encontrar tal solución, por lo tanto, el modelo GEO_4 es descartado, el Apéndice C muestra las gráficas de los resultados, promedio y desviación estándar de la Tabla V.

El algoritmo GEO_2 tiene la mejor relación tiempo - calidad de solución (tomando en cuenta las tres métricas) por lo tanto es el algoritmo considerado para la simulación de los experimentos. La simulación final considera el algoritmo GEO_2 para mejorar cada uno de los criterios, GEO para desaceleración acotada (GEO_{SD_b}), tiempo de espera promedio (GEO_{t_w}) y factor de competitividad (GEO_ρ) (Tabla VI).

Tabla VI. Nombre del algoritmo GEO con diferentes criterios.

Estrategias de asignación.	Estrategias de calendarización.	Criterios contemplados.
GEO_{SD_b}	BEFF	Desaceleración acotada.
GEO_ρ		Factor de competitividad.
GEO_{t_w}		Tiempo de espera.

La Tabla VII muestra la configuración de los experimentos finales, donde se comparan las implementaciones de GEO y las estrategias de asignación existentes en la literatura.

Para evaluar el desempeño de GEO como estrategia de asignación se hace una comparación con las estrategias de asignación descritas en la Tabla IV.

Tabla VII. Configuración de los experimentos.

Parámetros de los experimentos.	
Número de experimentos.	30
Trabajos por experimentos.	1,000
Nivel de calendarización.	2
Asignación de trabajos (cola global de espera).	FIFO
Estrategia de asignación	Tablas IV y VI.
Estrategia de calendarización.	Backfilling EASY Fisrt Fit.
Tiempo de ejecución de los trabajos.	Estimación del sistema.

IV.5.1 Metodología de evaluación

En general un algoritmo de calendarización debe mostrar un alto rendimiento, sin embargo, esta condición no siempre se cumple debido en su mayoría a los usuarios. Satisfacer las necesidades de varios usuarios de una manera equitativa resulta complicado. A menudo, los proveedores de recursos y los usuarios tienen diferentes objetivos (minimizar el tiempo de respuesta, minimizar utilización, etc.). La gestión de recursos en sistemas Grid implica múltiples objetivos, por lo tanto, es posible utilizar una metodología para lidiar con este tipo de problemas. Existen diferentes metodologías para problemas multicriterio, sin embargo, en ocasiones resultan inadecuadas para tratar el problema de calendarización. Por ejemplo, se puede aplicar la metodología basada en el óptimo de Pareto, pero es muy difícil encontrar soluciones rápido.

Los problemas multicriterio a menudo se simplifican a problemas de un solo criterio, donde los criterios se pueden tratar de manera independiente o combinados. Para ambos modelos se especifican las preferencias de los objetivo por las partes interesadas, los criterios reciben una importancia explícita o relativa. Debido a la diferente naturaleza de los criterios, la diferencia real puede tener un significado diferente. Una desviación del 10%

para la suma del tiempo de finalización es muy diferente al 10% en el factor de competitividad. Esto puede ser hecho por una definición de los pesos de los criterios o los criterios de clasificación por su importancia.

Con el fin de proporcionar una orientación eficaz en la elección de la mejor estrategia se realizó un análisis de varias métricas de acuerdo a la metodología propuesta por Tsafrir (Tsafrir *et al.*, 2007), un enfoque de análisis multicriterio suponiendo la misma importancia de cada métrica. El objetivo es encontrar una estrategia sólida y con buen desempeño en todos los casos de prueba, con la expectativa de mantenerse en otras condiciones, por ejemplo, con diferentes configuraciones de Grid y cargas de trabajo.

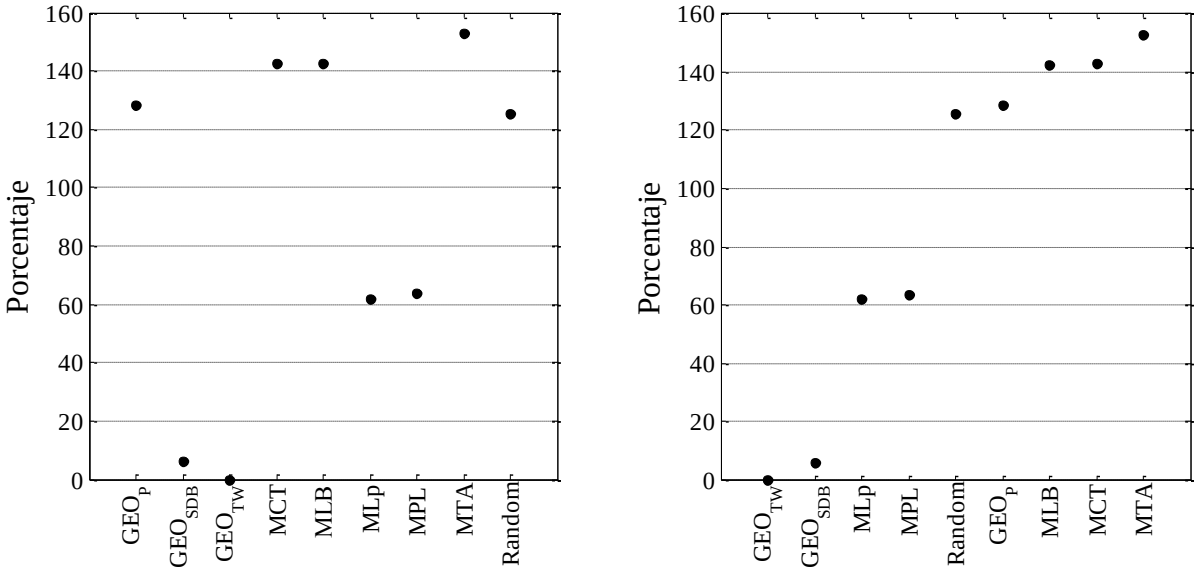
El análisis se lleva a cabo de la siguiente manera. En primer lugar, la degradación del rendimiento de cada estrategia se evalúa en cada una de las tres métricas (factor de competitividad, tiempo de espera y desaceleración acotada). Esto se hace en relación al desempeño de la mejor estrategia para la métrica con la siguiente fórmula: 100* estrategia / mejor_estrategia -100. Cada estrategia se caracteriza por tres números; éstos reflejan la degradación del rendimiento en relación a los casos de prueba. En el segundo paso, se tiene una media de estos tres valores (suponiendo la misma importancia de cada medida). La mejor estrategia, con la degradación de rendimiento más bajo promedio, tiene rango 1, la peor estrategia tiene rango 9.

El objetivo de emplear tres métricas, factor de competitividad, tiempo de espera y desaceleración acotada, se debe a que cada una de ellas representa a uno de los involucrados en el proceso de calendarización (usuarios, administrador del Grid y proveedores de recursos locales). La métrica de desaceleración acotada elimina el énfasis de trabajos muy cortos (tiempos de procesamiento cercanos a cero). El tiempo de espera se utiliza en lugar del tiempo de permanencia en el sistema; la diferencia entre ambas métricas es una constante independiente del calendarizador, en este caso, la constante es el tiempo de procesamiento promedio de todos los trabajos. Finalmente, el factor de competitividad exhibe propiedades del tiempo de terminación del calendario (Schwiegelshohn, 2009). La Tabla VIII muestra los porcentajes de degradación del desempeño considerando las tres principales métricas.

Tabla VIII**.** Porcentaje de degradación del desempeño considerando las tres principales métricas.

Estrategia / Métrica	GEO_{ρ}	GEO_{SD_b}	GEO_{t_w}	MCT	MLB	MLp	MPL	MTA	Random
ρ	5.94	17.2	0	64.45	45.22	16.58	14.92	71.67	69.54
SD_b	337.47	0	74.22	294.46	310	160.57	155.01	300.01	261.07
t_w	128.45	5.94	0	142.76	142.44	61.86	63.69	152.75	125.42
Promedio	157.29	7.71	24.74	167.22	165.89	79.67	77.87	174.81	152.01

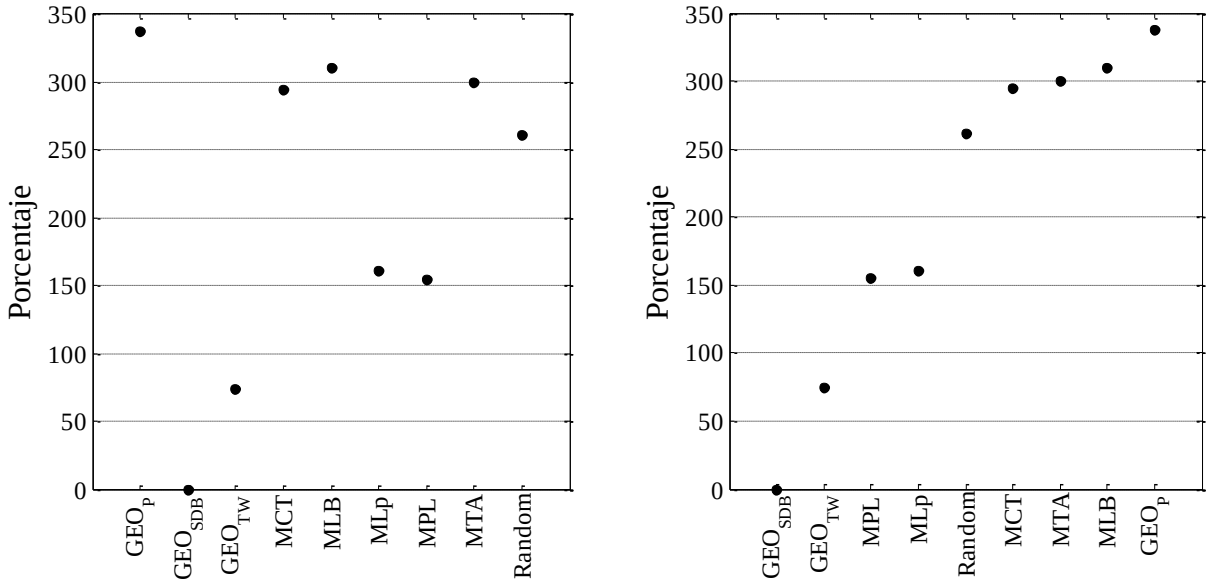
La Figura 25 muestra el desempeño de las estrategias respecto a la métrica de tiempo de espera promedio, las estrategias con mejor desempeño son GEO_{t_w} y GEO_{SD_b}. El algoritmo tomado como referencia es GEO_{t_w}, como es de esperarse, GEO_{t_w} muestra mejor desempeño al optimizar la métrica de tiempo de espera promedio. La peor estrategia para este criterio es MTA, con un calendario 152% peor que el calendario generado por GEO_{t_w}. La diferencia entre GEO_{t_w} y GEO_{SD_b} es de 5.9%.



a) Degradación. b) Rango

Figura 25. Tiempo de espera promedio para 9 estrategias.

La Figura 26 muestra el desempeño de las estrategias respecto a la métrica de desaceleración acotada promedio, las estrategias con mejor porcentaje son GEO_{SD_b} y GEO_{t_w}. El algoritmo tomado como referencia es GEO_{SD_b}; este algoritmo muestra mejor desempeño al optimizar la métrica de desaceleración acotada promedio. La peor estrategia para este criterio es GEO_{ρ}, con un calendario 337% peor que el calendario generado por GEO_{SD_b}. La diferencia de desempeño entre GEO_{SD_b} y GEO_{t_w} es de 74.22%; esta diferencia es más marcada al compararla con las otras dos métricas (ρ y t_w), además esta diferencia afecta a GEO_{t_w} a la hora de aplicar la degradación de las métricas.

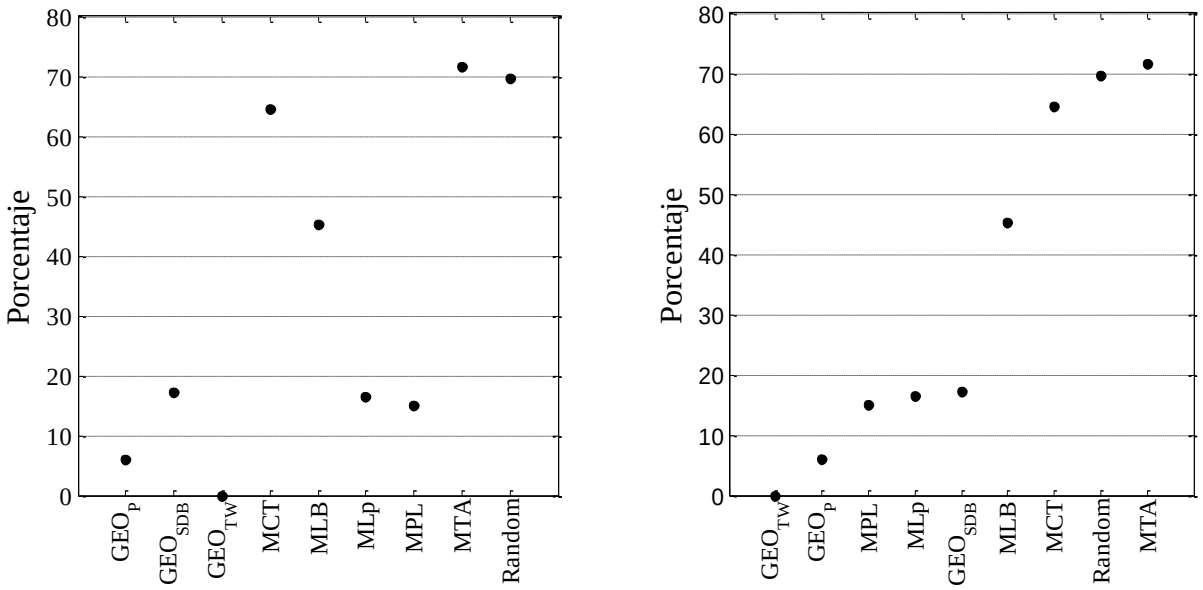


c) Degradación. d) Rango

Figura 26. Desaceleración acotada promedio para 9 estrategias.

La Figura 27 muestra el desempeño de las estrategias respecto a la métrica de factor de competitividad, las estrategias con mejor porcentaje son GEO_{t_w} y GEO_{ρ}, la peor estrategia para este criterio es MTA; sin embargo, se puede apreciar cómo MPL, MLp y GEO_{SD_b} tienen un desempeño muy parejo, alrededor de 1% de diferencia entre cada una. El

algoritmo tomado como referencia es GEO_{SD_b}, se espera que el algoritmo con el mejor desempeño sea GEO_{ρ} por tener como métrica guía al factor de competitividad; sin embargo, la diferencia entre GEO_{SD_b} y GEO_{ρ} es un poco mayor al 5%.

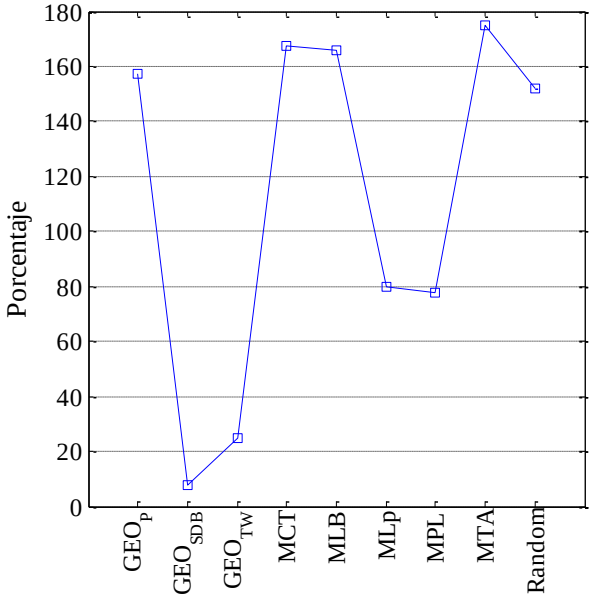


a) Degradación.

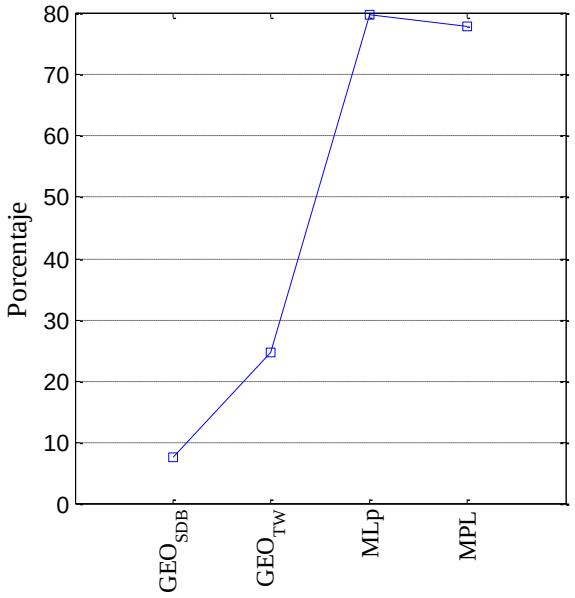
b) Rango

Figura 27. Factor de competitividad promedio para 9 estrategias.

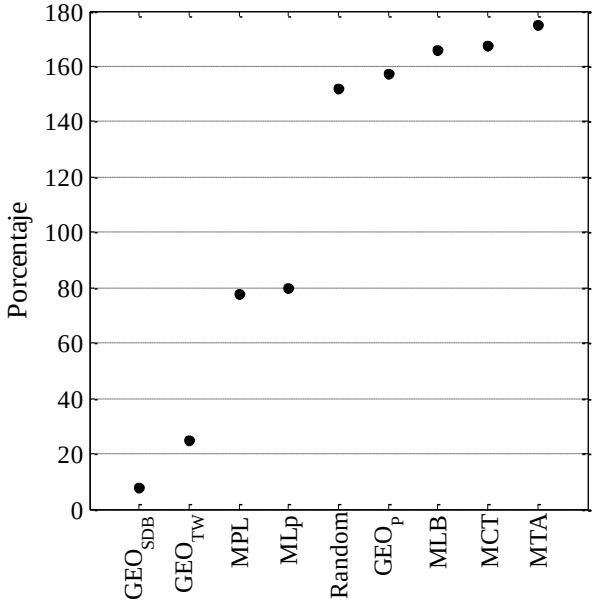
Finalmente, la degradación y el rango para las 9 estrategias se muestran en las Figuras 28a y 28c. Las figuras muestra la mejor estrategia de asignación GEO_{SD_b} y la segunda mejor estrategia es GEO_{t_w}; una comparación de las cuatro mejores estrategias se muestra en las Figuras 28b y 28d; el desempeño de GEO_{ρ} es superado por MPL, MLp y Random.



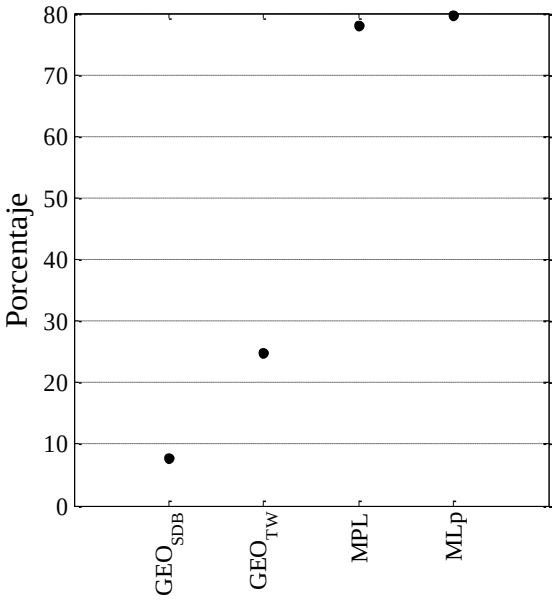
a. Resultados promedio para todas la estrategias.



b. Promedios de las cuatro mejores estrategias.



c. Rango de todas las estrategias.



d. Rango de las cuatro mejores estrategias.

Figura 28. Degradación y rango promedio para 9 estrategias.

IV.6 Análisis de resultados

Las heurísticas se comparan bajo las mismas condiciones, con base en los experimentos y la metodología propuesta para evaluar el desempeño; es posible generar algunas conjeturas sobre el desempeño de las estrategias.

En promedio las implementaciones de GEO, a excepción de GEO_ρ, tienen un buen desempeño, así como MPL y MLp, estas estrategias se posicionan en los primeros lugares en el análisis de degradación. Los datos de la Tabla VIII muestran que la estrategia con mejor desempeño y la cual se toma como referencia para el análisis de degradación es GEO_{SD_b}, la Tabla IX (Apéndice D) muestra la degradación de las 9 estrategias para 7 métricas.

Aunque el valor de GEO_{T_w} se considera como valor base para dos métricas (ρ, t_w) durante el proceso de degradación, la diferencia de rendimiento con GEO_{SD_b} en la métrica de SD_b ocaciona una disminución del rendimiento en el proceso de degradación de la estrategia GEO_{T_w}, si la métrica SD_b no se considera o la diferencia entre ambos valores fuera menor, la estrategia con mejor rendimiento sería GEO_{T_w}.

Un caso particular resulta GEO_ρ, aunque se mantuvo con un desempeñó aceptable (dentro de las métricas comparadas) no pudo mejorar a las estrategias MPL, MLp y Random. El bajo desempeño de GEO_ρ (considerando las demás implementaciones de GEO) se debe a la métrica que trata de mejorar; para calcular el factor de competitividad se emplean el tiempo de terminación del calendario y una cota mínima, el objetivo se reduce a mejorar el tiempo de terminación del calendario; sin embargo, con pequeñas alteraciones del calendario el tiempo de terminación no varía en gran medida; el factor de competitividad tiende a ser más constante y por lo tanto se hace necesario un criterio de terminación con un mayor número de soluciones sin cambio.

La diferencia en el proceso de degradación entre GEO_{T_w} y GEO_{SD_b} es de aproximadamente 17%, tanto GEO_{T_w} como GEO_{SD_b} se puede utilizar para generar calendarios con un rendimiento superior a los demás procedimientos.

Capítulo V

Conclusiones y trabajo futuro

V.1 Resumen

El presente trabajo aborda el problema de calendarización de trabajos en sistemas Grid jerárquicos de dos niveles, se implemento la heurística de optimización extrema con el objetivo de mejorar el proceso de asignación en la primera capa del calendarizador. Los experimentos realizados evaluaron el desempeño del procedimiento comparado con otras estrategias de asignación conocidas, la carga de trabajo consta de 1,000 trabajos para cada experimento, cada configuración consta de 30 experimentos, en total 30,000 trabajos componen la carga de trabajo.

V.2 Conclusiones finales

Para el problema tratado en el presente trabajo se obtienen las siguientes conclusiones:

1 El modelo GEO_2 ofrece en promedio la mejor relación tiempo-calidad de solución de los modelos implementados.

2 La estrategia con mejor desempeño es GEO_{T_w}, y la técnica GEO_{SD_b} es la segunda opción para ser utilizada.

3 MPL, MLp y Random superan el desempeño de la estrategia GEO_{ρ}.

4 Las estrategias MPL y MLp generan calendarios con el menor tiempo de procesamiento y se encuentran entre las cuatro estrategias con mejor desempeño.

Después de implementar, comparar y analizar la estrategia de optimización extrema generalizada aplicada al problema de calendarización de trabajos paralelos en sistemas Grid, se puede concluir que es apropiado aplicar GEO al problema de calendarización planteado, usando al menos dos métricas para guiar la búsqueda, porque GEO supera el desempeño de las estrategias clásicas de calendarización, con ello se cubre el objetivo principal de esta tesis.

V.3 Trabajo futuro

El presente trabajo provee las bases para estudios futuros:

1. Modificar el algoritmo GEO para disminuir el tiempo de generación de los calendarios.
2. Estudiar el número de reasignación de trabajos en una mutación, modificar los operadores de mutación para reasignar un conjunto de trabajos en lugar de uno solo (dinamismo extremo).
3. Desarrollar un modelo multiobjetivo para solucionar el problema de calendarización en sistemas Grid de dos niveles.
4. Desarrollar un modelo hibrido de GEO para funcionar con un conjunto de ecosistemas (varias soluciones al mismo tiempo).
5. Extender el trabajo a problemas de calendarización en línea.
6. Comparar la estrategia GEO con otras estrategias como: algoritmos genéticos, búsqueda tabú, recocido simulado, etc.

Referencias

Abdullah, M. Othman, M. Ibrahim, H. Subramaniam, S., 2008. Simulated annealing algorithm for scheduling divisible load in large scale data Grids**.** En: Proceedings of the International Conference on Computer and Communication Engineering 2008 (ICCCE08). Kuala Lumpur, Malaysia. 13-15 May.

Abreu, B.T., Martins, E. y Sousa, F.L., 2005. Automatic test data generation for path testing using a new stochastic algorithm. En: Simpósio Brasileiro de Engenharia de Software (sbes2005). Uberlandia, Brazil: 247-262. 3-7 Oct.

Aceves Pérez, R., 2003. Estudio de operadores genéticos para un problema de calendarización multi-objetivo. CICESE. Ensenada, Baja California, México.

Ahmed, E. y Elettreby, M., 2004. On multiobjective evolution model. Int. J. Mod. Phys. C 15: 1189–1195.

Anoep, S., Dumitrescu, C., Epema, D., Iosup, A., Jan, M., Li, H. y Wolters, L., 2007. The Grid workload format. Technische Universiteit Delft. http://gwa.ewi.tudelft.nl/TheGridWorkloadFormat_v001.pdf

Armentano, V.A. y Yamashita, D.S., 2000. Tabu search for scheduling on identical parallel machines to minimize mean tardiness. Journal of Intelligent Manufacturing, 11: 453–460.

Attanasio, A., Ghiani, G., Grandinetti, L., Guerriero, E., y Guerriero, F ,2005. Operations research methods for resource management and scheduling in a computational Grid: a survey. Grid Computing: The New Frontier of High Performance Computing. North-Holland: 53-81.

Bak, P., 1996. How nature works: The science of self-organized criticality, 3rd ed. New York, NY, Copernicus: 212 p.

Bak, P. y Sneppen, K., 1993. Punctuated equilibrium and criticality in a simple model of evolution. Physical Review Letters, 71(24): 4083.

Bak, P., Tang, C. y Wiesenfeld, K., 1987. Self-organized criticality: An explanation of the 1/f noise. Physical Review Letters, 59(4): 381.

Bar-Noy, A., Freund, A., 2001. On-line load balancing in a hierarchical server topology. SIAM. Journal of Computing, 31: 527-549.

Baraglia, R., Ferrini, R. y Ritrovato, P., 2005. A static mapping heuristics to map parallel applications to heterogeneous computing systems: Research Articles. Concurr.

Comput.: Pract. Exper., 17(13): 1579-1605.

Barbay, J. y Kenyon, C., 2001. On the discrete Bak-Sneppen model of self-organized criticality. En: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms. Washington, D.C., United States: Society for Industrial and Applied Mathematics: 928-933. 7-9 Ene.

Bernholdt, D., Bharathi, S., Brown, D., Chanchio, K., Chen, M., Chervenak, A. y Cinquini, L., 2007. The earth system Grid: Supporting the next generation of climate modeling research. En: Proceedings of the IEEE, 93(3): 485-495.

Boettcher, S., 2005. Extremal optimization for Sherrington-Kirkpatrick spin glasses. The European Physical Journal, 46(4): 501-505.

Boettcher, S. y Percus, A., 2000. Optimization with extremal dynamics. Phys Rev Lett, 86(23): 5211–5214.

Boettcher, S. y Percus, A., 2000. Nature's way of optimizing. Artif. Intell., 119(2): 275-286.

Boettcher, S., Percus, A. y Grigni, M., 2000. Optimizing through co-evolutionary avalanches. Lecture Notes in Computer Science: 447.

Bougeret, M., Dutot, P.-F., Jansen, K., Otte, C. y Trystram, D., 2010. A fast 5/2 approximation algorithm for hierarchical scheduling. En: 16th International European Conference on Parallel and Distributed Computing (EuroPar'10). Ischia, Italy: 157-167. 31 Ago. – 3 de Sept.

Brucker, P., 2001. Scheduling algorithms, 2nd ed. Springer-Verlag New York, Inc. Secaucus, NJ, USA. 371 p.

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. Softw. Pract. Exper., 35(5): 491-512.

Carlier, J. y Chrétienne, P., 1988. Problem d'ordonnancement : modelisation / compexite / algorithmes. En: french Masson, Paris.

Castro García, Y., 2010. Algoritmos genéticos para un problema de calendarización en un Grid computacional con múltiples criterios usando un método de agregación. CICESE. Ensenada, Baja California, México.

CERN, 2010. Worldwide LHC Computing Grid. Disponible en: http://lcg.web.cern.ch.

Chen, K., Bak, P. y Obukhov, S.P., 1991. Self-organized criticality in a crack-propagation model of earthquakes. Physical Review A, 43(2): 625.

Chen, M., Lu, Y. y Yang, G., 2008. Multiobjective optimization using population-based extremal optimization. Neural Comput. Appl., 17(2): 101-109.

Cook, S., 1971. The complexity of theorem-proving procedures. En: Proceedings of the Third Annual ACM Symposium on Theory of Computing, Association for Computing Machinery. New York, United States: 151 - 158.

CoreGrid, 2010. European research network on foundations, software infrastructures and applications for large scale distributed, Grid and peer-to-peer technologies. Disponible en: http://www.coregrid.net.

Diachin, D., Freitag, L., Heath, D., Herzog, J., Michels, W. y Plassmann, P., 1996. Remote engineering tools for the design of pollution control systems for commercial boilers. International Journal of Supercomputer Applications, 10: 208-218.

Dong, F. y Akl, S.G., 2006. Scheduling algorithms for Grid computing: State of the art and open problems. Technical Report No. 2006-504.

Dorigo, M., Di Caro, G. y Gambardella, L.M., 1998. Ant algorithms for discrete optimization. Artificial Life, 5: 137-172.

Dutot, P., Eyraud, L., Mounie, G., y Trystram, D., 2004. Models for scheduling on large scale platforms: which policy for which application? En: 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, New Mexico: 172.

Ekmecic, I., Tartalja, I. y Milutinovic, V., 1996. A survey of heterogeneous computing: concepts and systems. Proceedings of the IEEE, 84(8): 1127-1144.

El, M. y Menaï, B., 2006. An evolutionary local search method for incremental satisfiability. Springer-Verlag Berlin Heidelberg, 3249: 143-156.

El, M., Menaï, B. y Batouche, M., 2006. A backbone-based co-evolutionary heuristic for partial MAX-SAT. Lecture Notes in Computer Science, 3871: p. 155-166.

Ernst, M., Patrick, F., Papaspyrou, A., Radicke, M., Schley, L., y Yahyapour, R., 2006. A computational and data scheduling architecture for hep applications. En: Proceedings of the Conference on High Energy Physics (CHEP). Feb.

Etsion, Y. y Tsafrir, D., 2005. A short survey of commercial cluster batch schedulers, School of Computer Science and Engineering, the Hebrew University. Jerusalem, Israel. Technical Report 13.

Feitelson, D., 2003. Metric and workload effects on computer systems evaluation. Computer, 36(9): 18-25.

Feitelson, G., 2005. Parallel workloads archive. The Hebrew University, Jerusalem, Israel. http://www.cs.huji.ac.il/labs/parallel/workload.

Fidanova, S., 2006. Simulated annealing for Grid scheduling problem. En: Proceedings of the IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing. IEEE Computer Society: 41-45. 3-6 Oct.

Fölling, A., Grimme, C., Lepping, J. y Papaspyrou, A., 2009. Decentralized Grid scheduling with evolutionary fuzzy systems. En: Job Scheduling Strategies for Parallel Processing: 14th International Workshop (JSSPP 2009) Rome, Italy. Springer-Verlag: 16-36. 29 Mayo.

Foster, I. y Kesselman, C., 2004. The Grid in a nutshell. Grid Resource Management: State of the Art and Future Trends. Kluwer Academic Publishers: 3-13.

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.G., 2005. Pitfalls in parallel job scheduling evaluation. En: Workshop on Job Scheduling Strategies for Parallel Processing: 257-282, 19 Jun.

Franke, C., Hoffmann, F., Lepping, J. y Schwiegelshohn, U., 2008. Development of scheduling strategies with genetic fuzzy systems. Applied Soft Computing, 8(1): 706-721.

Galski, R., Ramos, F., Sousa, F., Preto, A., y Stephany, S. 2004. Implementação paralela do método da otimização extrema generalizada. En: Workshop dos Cursos de Computação Aplicada do Inpe (IV WORCAP). São José dos Campos, SP, Brasil. 20 y 21 Oct.

Galski, R., Sousa, F. y Ramos, F., 2005. Application of a new evolutionary algorithm to the optimum design of a remote sensing satellite constelation. En: International Conference on Inverse Problems in Engineering: Theory and Practice. Cambridge, UK. 11-15 Jul.

Galski, R., Sousa, F., Ramos, F. y Muraoka, I., 2007. Spacecraft thermal design with the generalized extremal optimization algorithm. Inverse Problems in Science and Engineering, 15(1): 61.

Gao, Y., Rong, H. y Huang, J., 2004. Adaptive Grid job scheduling with genetic algorithms. Future Generation Computer Systems, 2.

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

Garey, M. y Johnson, D., 1977. Computers and intractability: A Guide to the theory of NP-Completeness, New York, USA: W. H. Freeman Co.

Garzón, J., Huedo, E., Montero, R., Llorente I. y Chacón, P., 2007. Adaptation of a multi-resolution docking bioinformatics application to the Grid. Journal of Software, 2(2): 1-10.

Glover, F., 1989. Tabu search--Part I. Informs Journal on Computing, 1(3): 190-206.

Gould, S. y Eldredge, N., 1993. Punctuated equilibrium comes of age. Nature, 366(6452): 223-227.

Graham, R., Lawler, E., Lenstra, J. y Kan, A., 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Elsevier: 287-326.

Grimme, C., Lepping, J. y Papaspyrou, A., 2008. Discovering performance bounds for Grid scheduling by using evolutionary multiobjective optimization. En: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation. Atlanta, GA, USA: ACM: 1491-1498. 12 – 16 Jul.

Hamscher, V., Schwiegelshohn, U., Streit, A. y Yahyapour, R., 2000. Evaluation of job-scheduling strategies for Grid computing. En: Proceedings of the First IEEE/ACM International Workshop on Grid Computing. London, UK: 191-202. 17 Dec.

Haynos, M., 2004. Perspectives on Grid: Grid computing next-generation distributed computing. Grid Marketing and Strategy, IBM.

Holland, J., 1975. Adaptation in natural and artificial systems, University of Michigan Press. 211 p.

Izakian, H., Abraham, A. y Snášel, V., 2009. Metaheuristic based scheduling meta-tasks in distributed heterogeneous computing systems. Sensors 2009, 9: 5339-5350.

Jackson, J., 1955. Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California.

Johnson, S., 1953. Optimal two- and three-stage production schedules with setup time included. Naval Research Logistics Quarterly: 61- 67.

Jiang, C., Wang, C., Liu, X. y Zhao, Y., 2007. A survey of job scheduling in Grids. En: Proceedings of the joint 9th Asia-Pacific web and 8th international conference on web-age information management conference on Advances in data and web management. Huang Shan, China: 419-427. 16-18 Jun.

Kapadia, N., Kapadia, N. y Fortes, J.A.B., 1998. On the design of a demand-based network-computing system: the purdue university network-computing hubs. En: Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC'98): 71-80. 31 Jul.

Kennedy, J. y Eberhart, R., 1995. Particle swarm optimization. En: Proceedings of the IEEE International Conference on Neural Networks, 4: 1942-1948. 27 Nov.-1 Dec.

Kesselman, C. y Foster, I., 1998. The Grid: Blueprint for a new computing infrastructure. Publisher Morgan Kaufmann. San Fransisco. 2: 748.

Kirkpatrick, S., 1984. Optimization by simulated annealing: Quantitative studies. Journal of Statistical Physics, 34(5-6): 975-986.

Klusáček, D., Matyska, L. y Rudová, H., 2007. Local search for deadline driven grid scheduling. Third Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2007): 74-81. 26-28 Oct.

Kolodziej, J. y Xhafa, F., 2010. A game-theoretic and hybrid genetic meta-heuristics model for security-assured scheduling of independent jobs in computational Grids. En: International Conference Complex, Intelligent and Software Intensive Systems, Los Alamos, CA, USA: IEEE Computer Society: 93-100. 15-18 Feb.

Kong, X., Chen, X., Zhang, W., Liu, G. y Ji, H., 2009. A dynamic simulated annealing algorithm with self-adaptive technique for grid scheduling. En: 2009 WRI Global Congress on Intelligent Systems (GCIS). Xiamen, China: 129-133. 19-21 May.

Krauter, K., Buyya, R. y Maheswaran, M., 2001. A taxonomy and survey of Grid resource management systems for distributed computing. Software: Practice and Experience, 32(2): 135–164.

Kurowski, K., Nabrzyski, J., Oleksiak, A. y Węglarz, J., 2006. Scheduling jobs on the Grid – Multicriteria approach. Computational Methods in Science and Technology, 12(2): 123-138.

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

Kwok, Y., Song, S. y Hwang, K., 2005. Selfish Grid computing: game-theoretic modeling and NAS performance results. En: Proceedings of the International Symposium on Cluster Computing and the Grid (CCGRID-2005). Los Angeles, CA, USA. 2(2): 1143 – 1150, 9-12 May.

Lee, L., Liang, C. y Chang, H., 2006. An adaptive task scheduling system for Grid computing. En: Six IEEE International Conference on Computer and Information

Technology (CIT'06). Seoul, Korea: 57. 20-22 Sept.

Legrand, A., 2006. The SIMGRID project simulation and deployment of distributed applications. En: 15th IEEE International Conference on High Performance Distributed Computing. París, Francia: 385-386. 10 Jul.

Lenstra, J., Kan, R. y Brucker, P., 1977. Complexity of machine scheduling problems. Annals of Discrete Mathematics, volumen 1: 343-362.

Lenstra, J. y Kan, R., 1978. Complexity of scheduling under precedence constraints. Operations Research, 26(1): 22-35.

Lifka, D., 1998. An extensible job scheduling system for massively parallel processor architectures.Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the Illinois Institute of Technology Chicago.

Lorpunmanee, S., Sap, M., Abdullah, A. y Chompoo, I., 2007. An ant colony optimization for dynamic job scheduling in Grid environment. World Academy of Science, Engineering and Technology, 29.

Lorpunmanee, S., Sap, M., Abdullah, A. y Srinoy, S., 2006. Genetic algorithm in Grid scheduling with multiple objectives. En: Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases. Madrid, Spain: World Scientific and Engineering Academy and Society (WSEAS): 429-435. 15-17 Feb.

Madrigal, D., Galaviz, L. M., Ramírez, J. M., Tchernykh, A., y Verduzco, J. A, 2006. Estrategias de calendarización para un Grid computacional. En: Primer Congreso Internacional de Ciencias Computacionales (CiComp'06), UABC. Ensenada, Baja California, México. 6 – 8 Nov.

Mainenti, I., Sousa, F. y Gadelha, L., 2008. The generalized extremal optimization with real codification. En: International Conference on Engineering Optimization (EngOpt'08). Rio de Janeiro, Brazil. 1-5 Jun.

Martincová, P. y Zábovský, M., 2007. Comparison of simulated Grid scheduling algorithms. En: Systémová Integrace: 69-75.

Mathiyalagan, P., Dhepthie, U. y Sivanandam, S., 2010. Grid scheduling using enhanced PSO algorithm. International Journal on Computer Science and Engineering: 140-145.

Mathiyalagan, P., Suriya, S. y Sivan, S., 2010. Modified ant colony algorithm for Grid scheduling. International Journal on Computer Science and Engineering, 2:132-139.

Menaï, M.E. y Batouche, M., 2006. An effective heuristic algorithm for the maximum satisfiability problem. Applied Intelligence, 24(3): 227-239.

Morgenstern, O. y Neumann, J., 1953. Theory of games and economic behavior. Princeton University Press: 642.

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

Nebro, A., Alba, E. y Luna, F., 2007. Multi-objective optimization using Grid computing. Soft Comput., 11(6): 531-540.

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

Pinedo, M., 2008. Scheduling: Theory, Algorithms, and Systems, 2nd ed. Springer Publishing Company Inc. 678 p.

Potter, C., Brady, R., Moran, P., Gregory, C., Carragher, B., Kisseberth, N., Lydinf, J. y Lindquist, J., 1996. EVAC: A virtual environment for control of remote imaging instrumentation. Computer Graphics and Applications, IEEE, 16(4): 62-66.

Ramírez, J., Rodríguez, A., Tchernykh, A. y Verduzco, J., 2007. Performance of scheduling strategies considering fluctuation of tasks execution time in a computational Grid. En: Conferencia Latinoamericana de Computación de Alto Rendimiento (CLCAR). Santa Marta, Colombia: 273-281. 13 – 18 Ago.

Ranganathan, K. y Foster, I., 2004. Computation scheduling and data replication algorithms for data Grids. En: Grid Resource Management: State of the Art and Future Trends. Kluwer Academic Publishers Norwell, MA, USA: 359-373. 18-22 Oct.

Ruspini, E., Bonissone, P. y Pedrycz, W., 1998. Handbook of fuzzy computation, Taylor y Francis, 1500 p.

Rzadca, K., 2008. Scheduling in multi-organization Grids: measuring the inefficiency of decentralization. En: Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics. Gdansk, Poland: 1048-1058. 9-12 Sept.

Schopf, J.M., 2004. Ten actions when Grid scheduling: the user as a Grid scheduler. En: Grid resource management: state of the art and future trends. Kluwer Academic Publishers Norwell, MA, USA: 15-23. 18-22 Oct.

Schwiegelshohn, U., 2009: An Owner-centric Metric for the Evaluation of Online Job Schedules. En: 4th Multidisciplinary International Conference on Scheduling. Theory and Applications (MISTA 2009). Dublin, Ireland: 557-569. 10-12 Ago.

Schwiegelshohn, U., Tchernykh, A., Yahyapour, R., 2008. Online Scheduling in Grids. En: IEEE International Symposium on Parallel and Distributed Processing 2008 (IPDPS 2008). Miami, FL, USA: 1-10. 14-17 Abr.

Shan, H., Oliker, L. y Biswas, R., 2003. Job superscheduler architecture and performance in computational Grid environments. En: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing: 44. 15-21 Nov.

Smith, W., 1955. Various optimizers for single stage production. Naval Research Logistics Quarterly: 59 - 66.

Sousa, F.L., 2002. Otimização extrema generalizada: um novo algoritmo estocástico para o projeto ótimo. Tese de Doutorado. São José dos Campos: Instituto Nacional de Pesquisas Espaciais (INPE).

Sousa, F. y Ramos, F., 2002. Function optimization using extremal dynamics. En: 4th International Conference on Inverse Problems in Engineering. Rio de Janeiro, Brasil. 26-31 Mayo

Sousa, F., Ramos, F., Galski, R. y Muraoka, I., 2005. Generalized external optimization: a new meta-heuristic inspired by a model of natural evolution. Recent Developments in Biologically Inspired Computing. Hershey, PA, USA: 41-60.

Sousa, F., Ramos, F., Paglione, P. y Girardi, R., 2002. Laminar profile design using extremal dynamics optimization. En: World Congress on Computational Mechanics (WCCM V). Vienna, Austria: 311. 7-12 Jul.

Sousa, F., Ramos, F., Vlassov, V., Paglione, P. y Girardi, R., 2004. Generalized extremal optimization: an overview of a new evolutionary optimum design tool. En: Biannual Brazilian Symposium on Artificial Neural Networks (SBRN'2004). São Luis, Maranhão, Brazil: IEEE Computer Society. 29 Sept.-1 Oct.

Sousa, F., Vlassov, V. y Ramos, F.M., 2004. Generalized extremal optimization: An application in heat pipe design. Applied Mathematical Modelling, 28(10): 911-931.

Sousa, F., Vlassov, V. y Ramos, F.M., 2002. Heat pipe design through generalized extremal optimization. En: Brazilian Congress of Engineering and Thermal Sciences (ENCIT 2002). Caxambu, MG, Brazil. 15-18 Oct.

Sousa, F., Vlassov, V. y Ramos, F.M., 2004. Heat pipe design through generalized extremal optimization. Heat Transfer Engineering, 25(7): 34-45.

Sousa, F., Vlassov, V. y Ramos, F., 2003. Generalized extremal optimization for solving complex optimal design problems. En: Genetic and Evolutionary Computation Conference (GECCO 2003). Chicago, IL, USA: 375-376. 12-16 Jul.

Sowinski, R. y Hapke, M., 2000. Scheduling under fuzziness, Physica-Verlag, 308 p.

Switalski, P. y Seredynski, F., 2008. Multiprocessor task scheduling based on GEO metaheuristic. Intelligent Information Systems: 111–120.

Switalski, P. y Seredynski, F., 2009. Multiprocessor scheduling by generalized extremal optimization. Journal of Scheduling, 13(5): 531-543.

Switalski, P. y Seredynski, F., 2010. Generalized extremal optimization for solving multiprocessor task scheduling problem. Parallel processing and applied mathematics. Lecture notes in computer Science, 6068: 42-51.

Tchernykh, A., Ramírez, J., Avetisyan, A., Kuzjurin, N., Grushin, D. y Zhuk, S., 2006. Two level job-scheduling strategies for a computational Grid. En: 6th International Conference on Parallel Processing and Applied Mathematics PPAM 2005. Springer Berlin / Heidelberg, Poznan, Poland, 3911: 774-781. 11-14 Sept.

Tchernykh, A., Schwiegelshohn, U., Yahyapour, R. y Kuzjurin, N., 2010. Online hierarchical job scheduling on Grids with admissible allocation. Journal of Scheduling. 13(5): 545-552.

Tk̊indt, V. y Billaut, J., 2006. Multicriteria scheduling: Theory, models and algorithms, Springer. 2nd ed., 359 p.

Tsafrir, D., Etsion, Y. y Feitelson, D., 2007. Backfilling using system-generated predictions rather than user runtime estimates. IEEE Transactions on Parallel and Distributed Systems, 18(6): 789-803.

Vlassov, V., Sousa, F. y Takahashi, W., 2006. Comprehensive optimization of a heat pipe radiator assembly filled with ammonia or acetone. International Journal of Heat and Mass Transfer, 49(23-24): p. 4584-4595.

Wang, Q., Gao, Y. y Liu, P., 2006. Hill climbing-based decentralized job scheduling on computational Grids. En: Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06). Hangzhou, China. IEEE Computer Society, 2(1): 705-708. 20-24 Jun.

Wang, Q., Gui, X., Zheng, S. y Liu, Y., 2006. De-centralized job scheduling on computational Grids using distributed backfilling: Research Articles. Concurr. Comput. : Pract. Exper., 18(14): 1829-1838.

Xhafa, F. y Abraham, A., 2010. Computational models and heuristic methods for Grid scheduling problems. Future Generation Computer Systems, 26(4): 608-621.

Xhafa, F., Carretero, J. y Abraham, A., 2007. Genetic algorithm based schedulers for Grid computing systems. International Journal of Innovative Computing, Information and Control (ICIC). 3(5): 1053-1071.

Young, L., Mcgough, S., Newhouse S. y Darlington J., 2003. Scheduling architecture and algorithms within the ICENI Grid middleware. UK E-science: 5-12.

Yu, J. y Buyya, R., 2005. A taxonomy of workflow management systems for Grid computing. Journal of Grid Computing, 3: 3-4.

Zadeh, E., 1965. Fuzzy sets. Information and Control, 8(3): 338-353.

Zadeh, E., 1975. The concept of a linguistic variable and its application to approximate reasoning-I. Information Sciences, 8(3): 199-249.

Zhang, L., Chen, Y. y Yang, B., 2006. Task scheduling based on PSO algorithm in computational Grid. En: Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06). Jinan, China. 3(2): 696-704. 16-18 Oct.

Zhou, J., Yu, K., Chou, C., Yang L. y Luo Z., 2007. A dynamic resource broker and fuzzy logic based scheduling algorithm in Grid environment. En: 8th International Conference on Adaptive and Natural Computing Algorithms, Part I. Warsaw, Poland: 604-613. 11-14 Apr.

Zhu M., Liu H., Sun W. y Zhu T., 2007. Improvement of particle swarm optimization based on neighborhood cognizance and swarm decision. En: Wireless Communications, Networking and Mobile Computing (WiCom 2007): 3669-3672. 25-27 Sept.

Zhuk, S., 2007: Approximate algorithms to pack rectangles into several strips. Discrete Mathematics and Applications 16: 73-85.

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. En: Third International IEEE Conference on Parallel Computing Systems (PCS 2004). Colima, México: 388-392. 5-7 Jul..

Apéndice A

Notación α | β | γ para clasificar los problemas de calendarización.

Los posibles ambientes de las máquinas en el campo α son los siguientes:

Máquina única (1): Solo se tiene una máquina en el sistema. Este es un caso especial, diferente a los demás ambientes de máquinas más complicados.

Máquinas paralelas idénticas (Pm): Se tienen *m* máquinas idénticas en paralelo.

Máquinas paralelas uniformes (Qm): Se tienen *m* máquinas en paralelo, pero las máquinas tienen diferentes velocidades. La máquina i, $1 \leq i \leq m$, tiene una velocidad s_i. El tiempo en completar el trabajo *j* en la máquina *i* (p_{ij}) es igual a p_j / s_i, cuando el trabajo es totalmente procesado en la máquina *i*.

Máquinas paralelas no relacionadas (Rm): Se tienen *m* máquinas en paralelo, pero cada máquina puede procesar los trabajos a diferentes velocidades. La máquina *i* puede procesar el trabajo *j* a una velocidad s_{ij}. El tiempo para completar el trabajo *j* en la máquina *i* (p_{ij}) es igual a p_j / s_{ij}, cuando el trabajo es totalmente procesado en la máquina *i*.

Máquinas dedicadas job shop (Jm): En un job shop con *m* máquinas, cada trabajo tiene su propia ruta predeterminada a seguir, se pueden visitar algunas máquinas más de una vez y pueden no visitar algunas máquinas en absoluto.

Máquinas dedicadas flow shop (Fm): En un flow shop con *m* máquinas, las máquinas están ordenadas y todos los trabajos siguen la misma ruta (de la primera máquina hasta la última máquina).

Máquinas dedicadas open shop (Om): En un open shop con *m* máquinas, cada trabajo necesita ser procesado exactamente una vez en cada una de las máquinas. Pero el orden del procesamiento no importa.

Las características de los trabajos y las limitaciones de los calendarios son especificadas en el campo β, la cual pude contener múltiples entradas, si la entrada no está especificada se considera no permitida.

Interrupciones (pmtn): Los trabajos pueden ser interrumpidos y posteriormente reiniciar su ejecución posiblemente en otra máquina.

Recursos adicionales (res): Además de los procesadores, los trabajos pueden solicitar recursos adicionales durante su ejecución. Los recursos son escasos por lo tanto su disponibilidad es limitada.

Limitaciones de precedencia (prec): Las limitaciones de precedencia especifican las limitaciones de los trabajos en el calendario, en el sentido en el que ciertos trabajos deban ser completados antes que otros trabajos puedan comenzar a ser procesados .La forma más general de limitaciones de precedencia es representada por un grafo acíclico dirigido, donde cada vértice representa un trabajo y cada arista representa una limitación de precedencia.

- Si cada trabajo tiene a lo más un predecesor y un sucesor, las limitaciones pueden representarse por medio de cadenas (chains).
- Si cada trabajo tiene a lo más un sucesor, las limitaciones pueden representarse por medio de arboles (intree).
- Si cada trabajo tiene a lo más un predecesor, las limitaciones pueden representarse por medio de arboles (outtree).

Tiempo de liberación (r_j): El tiempo de liberación r_j del trabajo j es el momento en el cual puede comenzar a ser procesado. Si no está especificado, todos los trabajos pueden iniciar su ejecución desde el comienzo (tiempo cero).

Tiempo de procesamiento (p_i): El tiempo de procesamiento p_i define el tiempo límite del trabajo j en p unidades, $p_i = p$ significa que el tiempo de procesamiento de cada trabajo es p unidades de tiempo. Los tiempos de procesamiento de los trabajos pueden estar acotados por un mínimo y máximo ($p_{min} \leq p_{ij} \leq p_{max}$).

Número de trabajos (nbr): El número de trabajos define el máximo de trabajos permitidos.

Número de tareas de un trabajo (n_j): El número de trabajo define el máximo número de tares que constituyen un trabajo en el caso de job shop, $n_j \leq k$ el número máximo de tareas por trabajo es menor a k.

Fecha limite ($\tilde{d}_j$): La fecha limite $\tilde{d}_j$ del trabajo j representa la fecha límite sin tolerancia que el trabajo debe respetar, entonces, el trabajo debe ser completado para este tiempo.

Sin espera (nwt); La limitación de no espera es solo para flow shop, Los trabajos no tienen permitido esperar entre dos máquinas sucesivas. Después de finalizar su tiempo de procesamiento en una máquina debe ser inicializado inmediatamente en la siguiente máquina.

Apéndice B

Clase de Problemas (complejidad computacional).

A continuación se describen las clases de problemas:

La clase P consiste de aquellos problemas que se pueden resolver en tiempo polinomial. Es decir, son problemas que se pueden resolver en tiempo $O(n^k)$ para alguna constante k y una entrada de tamaño n.

La clase NP consiste de aquellos problemas que son *verificables* en tiempo polinomial. Es decir, si se proporciona un *certificado* (una solución), entonces, es posible verificar la corrección del certificado en tiempo polinomial en función del tamaño de la entrada del problema.

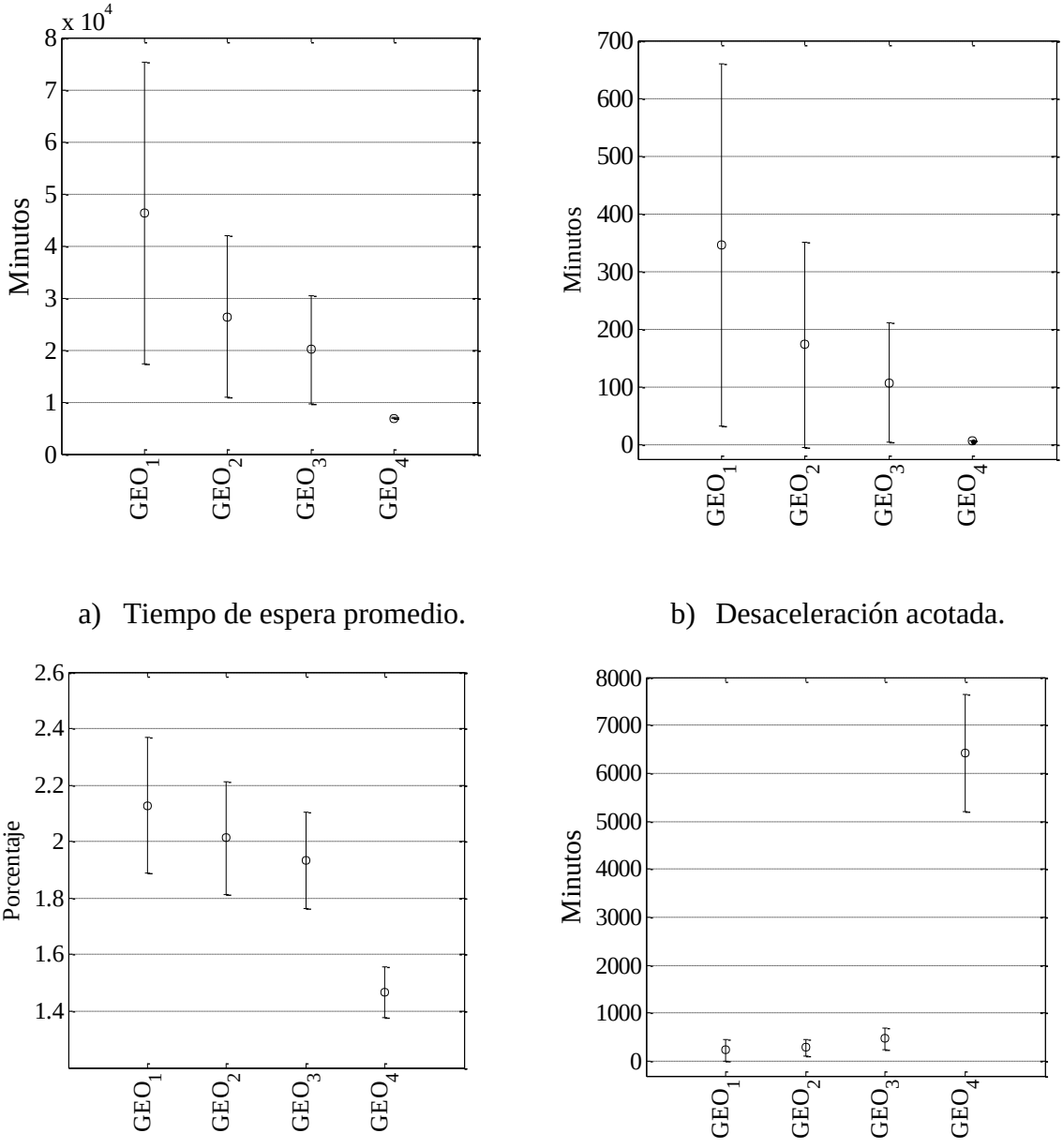
Un problema C se encuentra en la clase $NP - completo$ si: C está en NP y todo problema en NP se pueden reducir a C en tiempo polinomial.

La clase de problemas $NP - dificil$ ($NP - hard$) es el conjunto de problemas de decisión que contiene los problemas H tales que todo problema L en NP se puede transformar de manera polinomial en H.

La relación entre las clases de complejidad P y NP es una pregunta que aún no se ha respondido. En esencia, la pregunta ¿es $P = NP$? significa: ¿si es posible "verificar" rápidamente soluciones positivas a un problema del tipo SI/NO? (donde "rápidamente" significa "en tiempo polinomial"), entonces también ¿se pueden "obtener" las respuestas rápidamente?. Los problemas NP-difícil no son todos *NP*: los problemas *NP-completo*

significa problemas que son *completos* en *NP*, es decir, los más difíciles de resolver en *NP*; *NP-difícil* quiere decir *al menos* tan complejo como *NP* (pero no necesariamente en *NP*).

Apéndice C



a) Tiempo de espera promedio.

b) Desaceleración acotada.

c) Factor de competitibidad.

d) Tiempo.

Figura 29. Promedio y desviación estándar para los 4 modelos GEO propuestos.

Apéndice D

Tabla IX. Degradación del rendimiento para las 9 estrategias y 7 métricas.

Estrategia / Métrica	GEO_{ρ}	GEO_{SD_b}	GEO_{t_w}	MCT	MLB	MLp	MPL	MTA	Random
ρ	5.944	17.201	0	64.448	45.223	16.582	14.918	71.669	69.539
SD	356.905	0	78.994	307.917	321.124	165.4	163.985	311.195	271.587
SD_b	337.471	0	74.223	294.46	309.99	160.572	155.014	300.017	261.068
Th	1.843	13.302	0	40.0815	25.998	12.325	10.05	40.467	38.442
TA	112.67	5.212	0	125.222	124.943	54.255	55.861	133.983	110.012
t_w	128.451	5.942	0	142.761	142.443	61.855	63.685	152.75	125.421
WWT	5.4013	23.779	0	58.854	29.426	31.306	16.221	55.323	45.581
Promedio	135.526	9.348	21.888	147.677	142.735	71.756	68.533	152.2	131.664

Apéndice E

Características de la computadora y el software utilizado en la simulación de los experimentos.

Unidad central de procesamiento:

Procesador: AMD Athlon(tm) 64 X2 Dual Core 5600+ (2.9 GHz.).

Memoria RAM: 2 GB.

Disco duro: 400 GB.

Sistema operativo: Windows 7, 32 bit.

Software:

Matlab: Software especializado en matemáticas que cuenta con su propio lenguaje de programación, el graficador de las estadísticas se implementó en este software.

Java: Lenguaje de programación orientado a objetos, el diseño del algoritmo GEO se implementó en este lenguaje.