ISSN 2007-9737

Algoritmo aleatorizado basado en distribuciones deslizantes para el problema de planificación en sistemas Grid

Hector J. Selley-Rojas[1,2], Jesus Garcia-Diaz[1], Manuel A. Soto-Ramos[1], Felipe R. Menchaca-Garcia[3], and Rolando Menchaca-Mendez[1]

[1] Instituto Politécnico Nacional, Centro de Investigación en Computación, México DF, México

[2] Universidad Anáhuac Norte, Facultad de Ingeniería, México DF, México

[3] Instituto Politécnico Nacional, Escuela Superior de Ingeniería Mecánica y Eléctrica, México DF, México

{hector.selley, jesgadiaz, yakovichs, fmenchac}@gmail.com, rmen@cic.ipn.mx

Resumen. En este artículo se presenta un algoritmo aleatorizado para el problema de planificación de tareas compuestas por procesos con restricciones de precedencia en ambientes distribuidos tipo Grid. El algoritmo aleatorizado propuesto está basado en una nueva técnica que hemos denominado como de *distribuciones deslizantes*, la cual busca combinar las ventajas de los algoritmos de aproximación deterministas y de los algoritmos aleatorizados tipo Montecarlo. El objetivo es proveer un algoritmo que con alta probabilidad entregue soluciones ρ-aproximadas, pero que al mismo tiempo tenga la capacidad de analizar el vecindario extendido de dichas soluciones para escapar de máximos o mínimos locales. En el artículo se demuestra que el algoritmo propuesto es correcto y se caracteriza de manera formal su complejidad temporal. Así mismo, se evalúa el desempeño del algoritmo por medio de una serie de experimentos basados en simulaciones. Los experimentos muestran que el algoritmo propuesto logra en general un desempeño superior al de los algoritmos que componen el estado del arte en planificación en sistemas Grid. Las métricas de desempeño utilizadas son retardo promedio, retardo máximo y utilización de la Grid.

Palabras clave. Optimización combinatoria, algoritmo aleatorio, sistemas Grid, planificación de tareas.

Randomized Algorithm based on Sliding Distributions for the Scheduling Problem in Grid Systems

Abstract. In this paper we present a randomized algorithm for the online version of the Job Shop problem where jobs are composed of processes with precedence constraints and processors are organized in a Grid topology. The proposed randomized algorithm is based on a new technique that we have denominated as *sliding distributions*, which aims at combining the advantages of the deterministic approximation algorithms with those of the Montecarlo randomized algorithms. The objective is to provide an algorithm that delivers ρ-approximated solutions with high probability, but at the same time, is able to investigate an extended neighborhood of such solutions so that it can escape from local extrema. We formally characterize the temporal complexity of the proposed algorithm and show that it is correct. We also evaluate the performance of the proposed algorithm by means of a series of simulation-based experiments. The results show that the proposed algorithm outperforms the traditional state of the art algorithms for scheduling in Grid systems. The performance metrics are average delay, maximum delay, and Grid utilization.

Keywords. Combinatorial optimization, randomized algorithm, Grid systems, scheduling.

1. Introducción

En la actualidad, los sistemas *Grid* han alcanzado una gran relevancia científica y tecnológica debido a las ventajas que ofrecen para soportar tanto al cómputo de alto desempeño, como a los paradigmas emergentes como el cómputo en la nube [4], el hardware como servicio (*Hardware as a Service, "HaaS"*) [32] y las infraestructuras como

servicio ("*Infrastructure as a Service, IaaS"*) [33]. A diferencia de las supercomputadoras que suelen ser altamente costosas, los sistemas *Grid* pueden construirse por medio de un conjunto dinámico y heterogéneo de sistemas de cómputo conectados también por una red heterogénea. Así, los sistemas Grid se han consolidado como una alternativa escalable y efectiva para proveer cómputo intensivo o de alto rendimiento a bajo costo [12]. Más aún, los sistemas Grid se han comenzado a establecer como el medio para proporcionar poder de cómputo y almacenamiento de forma flexible y transparente y con acceso casi ubicuo [13].

Por otro lado, las características que hacen tan atractivos a los sistemas Grid como su capacidad de construirse por medio de componentes heterogéneos, su dinámicidad y flexibilidad, su naturaleza inherentemente distribuida, así como su habilidad para ejecutar tareas altamente heterogéneas, también traen consigo una conjunto de retos y problemas abiertos [9]. Entre los componentes más relevantes de un sistema Grid, y del cual depende fuertemente su escalabilidad y rendimiento, está el servicio de planificación distribuida de tareas que es el encargado de asignar recursos, tanto de cómputo como de comunicaciones a las tareas que son sometidas a la Grid para su ejecución. La planificación de tareas es uno de los problemas más trabajados y bien conocidos en ambientes de cómputo distribuido como las Grid [11], sin embargo, y a pesar de todo el esfuerzo invertido en su solución, sigue siendo un problema de investigación abierto. En términos más concretos, el problema de planificación de tareas en una Grid consiste en asignar un conjunto de tareas, que pueden estar compuestas por procesos con relaciones de precedencia, a un conjunto de CPUs heterogéneos para su ejecución. Para dicha asignación, el planificador debe tomar en cuenta no sólo el tiempo de ejecución, sino el costo de comunicación que implicará transportar la tarea y sus datos a los CPUs asignados, así como llevar de vuelta los datos resultantes a un sitio designado. Es importante mencionar que el problema básico de planificación en múltiples procesadores, donde la carga de trabajo es conocida de antemano, las tareas no tienen relaciones de precedencia, los procesadores son idénticos y no hay costo de comunicación, es NP-Difícil (NP-Hard) [14], y por lo tanto, no puede ser resuelto de manera exacta a menos que la clase de problemas P sea igual a la clase NP. Debido a que el problema de planificación de trabajos con precedencias en un sistema Grid es una generalización del problema básico de planificación en múltiples procesadores, es fácil demostrar que también pertenece a la clase NP-Hard. Dicha demostración se ha realizado por medio de una reducción polinomial del problema de la mochila (*Knapsack*) [28, 30].

Dada la imposibilidad práctica para encontrar soluciones óptimas a este problema de planificación, se han propuesto un número considerable de algoritmos cuyo objetivo es encontrar soluciones relativamente buenas, en tiempo bajo. Esas propuestas se pueden clasificar en *algoritmos de aproximación* como los presentados en [20, 21], [16] que garantizan que las soluciones encontradas no pueden estar arbitrariamente lejos del óptimo; y los basados en *heurísticas y metaheurísticas* como la presentada en [27] que utiliza búsqueda local [17, 8, 36] que utilizan heurísticas basadas en poblaciones y [3] que utiliza metaheurísiticas híbridas. Otras propuestas son los algoritmos que utilizan inteligencia artificial para la planificación como [15, 34, 26] o incluso algoritmos que utilizan redes neuronales [34] y algoritmos genéticos [35]. La principal desventaja de los algoritmos basados en heurísticas y metaheurísiticas como algoritmos genéticos o búsqueda local es que a pesar de que para algunas instancias pueden encontrar soluciones muy cercanas al óptimo, no proveen ningún tipo de garantía teórica para instancias arbitrarias. Más aún, la convergencia de estos algoritmos suele ser lenta, lo cuál no es una alternativa válida en el contexto de los sistemas Grid debido a que los planes deben ser generados en tiempo real. En todos los casos anteriores, la calidad de las soluciones se caracteriza por medio de métricas de desempeño que sirven como función objetivo para el problema de optimización. En el caso de los sistemas Grid, las principales métricas de desempeño como el tiempo de respuesta promedio, total o máximo, están orientadas a medir la calidad en la experiencia del usuario [7, 22].

En este artículo presentamos un nuevo algoritmo basado en una técnica que hemos llamado

de *distribuciones deslizantes*, la cual busca combinar las ventajas de los algoritmos de aproximación deterministas y de los algoritmos aleatorizados tipo Montecarlo [24, 25]. El resultado, es un algoritmo aleatorizado con complejidad polinomial que entrega soluciones ρ-aproximadas con alta probabilidad, y que a diferencia de los algoritmos determinísticos, es capaz de analizar un vecindario extendido de dichas soluciones para escapar de máximos o mínimos locales mejorando así la calidad de la solución encontrada. Como se muestra en la Sección 5, el costo que se paga con respecto al algoritmo de aproximación base es únicamente de orden $O(n)$, mientras que como se muestra en la Sección 6, los beneficios en el desempeño son considerables.

El resto del artículo está organizado de la siguiente forma. En la Sección 2 se presenta una muestra de la basta literatura que aborda el problema de planificación en múltiples procesadores. En la Sección 3 se presenta el planteamiento formal del problema de planificación en sistemas Grid como un problema de optimización combinatorio. En la Sección 4 se muestra la especificación del esquema de planificación propuesto que tiene como elemento principal al algoritmo basado en distribuciones deslizantes. La Sección 5 contiene una serie de teoremas que establecen que el algoritmo propuesto es correcto y que caracterizan de manera formal su complejidad temporal. En la Sección 6 se describe el entorno experimental utilizado para caracterizar de forma cuantitativa el desempeño del algoritmo propuesto. Los resultados experimentales presentados en esta sección muestran que el algoritmo basado en distribuciones deslizantes tiene un desempeño similar o superior a los algoritmos que conforman el estado del arte en en términos del retardo promedio y el retardo máximo. Finalmente, la Sección 7 muestra una breve discusión acerca de los resultados obtenidos, así como las conclusiones a las que se llegó como resultado del presente trabajo.

2. Trabajo relacionado

A pesar de que la planificación de tareas en sistemas paralelos y distribuidos ha sido intensamente estudiada, aún existen un número importante de problemas abiertos y nuevos retos. Lo anterior es de esperarse, ya que el problema básico de planificación multiprocesador en el que todos los procesadores son idénticos, el costo de asignar cualquier trabajo a cualquier procesador es el mismo, los trabajos son independientes unos de otros y toda la carga de trabajo se conoce de antemano pertenece a la clase NP-Difícil [14]. Por otro lado, para modelar de manera realista a los sistemas altamente heterogéneos de propósito general, como lo son los sistemas Grid [9], es necesario relajar muchas de las suposiciones anteriores e introducir una nueva serie de costos y restricciones, tanto sobre los procesos que componen a los trabajos a ser ejecutados como a la Grid misma. A continuación se describe una muestra que consideramos representativa de los trabajos relacionados con la propuesta presentada en este artículo. Como se verá, los diferentes trabajos proponen soluciones a diferentes variantes del problema básico de planificación multiprocesador.

A pesar de que la planificación de tareas en sistemas paralelos y distribuidos ha sido intensamente estudiada, aún existen un número importante de problemas abiertos y nuevos retos. Lo anterior es de esperarse, ya que el problema básico de planificación multiprocesador en el que todos los procesadores son idénticos, el costo de asignar cualquier trabajo a cualquier procesador es el mismo, los trabajos son independientes unos de otros y toda la carga de trabajo se conoce de antemano pertenece a la clase NP-Difícil [14]. Por otro lado, para modelar de manera realista a los sistemas altamente heterogéneos de propósito general, como lo son los sistemas Grid [9], es necesario relajar muchas de las suposiciones anteriores e introducir una nueva serie de costos y restricciones, tanto sobre los procesos que componen a los trabajos a ser ejecutados como a la Grid misma. A continuación se describe una muestra que consideramos representativa de los trabajos relacionados con la propuesta presentada en este artículo. Como se verá, los diferentes trabajos proponen soluciones a diferentes variantes del problema básico de planificación multiprocesador.

En [29], Shmoys y Tardos proponen una 2-aproximación para el problema de asignación generalizado (*generalized assignment problem*) que

consiste en asignar tareas a máquinas denotadas, tal que ejecutar la tarea i en la máquina j requiere de p_{ij} unidades de tiempo con un costo de c_{ij} unidades y donde cada máquina i está disponible T_i unidades de tiempo. En este problema el objetivo es minimizar el costo total C que se incurre al planificar la carga de trabajo. El algoritmo propuesto por Shmoys y Tardos garantiza que para una instancia arbitraria del problema y un costo total propuesto C, el algoritmo determina que no existe un plan factible o entrega un plan de costo mínimo que cumple con el presupuesto C y donde cada máquina i es utilizada máximo $2T_i$ unidades de tiempo. El algoritmo propuesto construye un grafo bipartita que codifica la solución por medio de una relajación de programación lineal. En [18], Kasahara y Narita proponen una heurística llamada CP/MISF (Ruta Crítica/ Primero los Sucesores Más Inmediatos) y un algoritmo de aproximación/optimización llamado DF/IHS (Profundidad Primero/ Búsqueda heurística implícita) para el problema de planificación de tareas parcialmente ordenadas en sistemas multiprocesador. DF/IHS es un método de planificación que puede reducir notoriamente la complejidad de espacio y el tiempo promedio de cómputo mediante la combinación del método *branch-bound* con CP/MISF. Lo anterior permite resolver problemas de muy gran escala que contengan cientos de tareas. En la propuesta CP/MISF la tarea que tiene el mayor número de tareas sucesivas inmediatas se le asigna la mayor prioridad. El método DF/IHS consiste de dos partes. La parte de pre-procesamiento para la asignación de prioridades a los nodos generados durante la búsqueda y la segunda parte es la búsqueda de primero por profundidad (DFS). Los resultados experimentales muestran que la heurística encuentra valores cercanos al óptimo pero con la limitante de solo planificar una sola tarea modelada por un DAG de procesos, aunque el número de procesos que puede ser potencialmente muy grande.

Khan, McCreary y Jones [19] realizan un análisis experimental del desempeño de una serie de heurísticas determinísticas de planificación de tareas con restricciones de precedencia en sistemas multiprocesador con respecto a criterios como el porcentaje de instancias para las cuales el tiempo de ejecución es $o(n)$, el aumento en la velocidad de la ejecución de las tareas que se define como $speedup = SerialTime/ParellelTime$ y la eficiencia que se define como $Efficiency = Speedup/NumberOfProcessors$. Los grafos acíclicos dirigidos (DAG) que codifican las relaciones de precedencia entre los procesos que componen a una tarea y que fueron usados para los experimentos se generaron de manera aleatoria . La clasificación sugerida para los algoritmos de planificación es: *heurística de ruta crítica*, *heurística de lista de planificación* y *método de descomposición del grafo*. Para el caso de la clasificación de los grafos utilizan tres características: *granularidad*, *grado de salida* y el *rango de peso de los nodos*. Los resultados de este trabajo muestran que la ruta crítica y el algoritmo de lista de planificación fallan cuando el costo de la comunicación es alto con respecto al costo de ejecución. También se encontró que cuando la granularidad es baja, en los planes obtenidos el tiempo paralelo ocasionalmente es mayor que el tiempo en serie. La esencia de este problema surge del punto de inicio de los algoritmos. Cada nodo es inicialmente posicionado en un procesador, desde este punto de inicio los algoritmos usan una heurística para encontrar una asociación de nodos con procesadores.

En [10], Entezari y Movaghar presentan un método de planificación probabilística de tareas para minimizar el tiempo de respuesta promedio en un ambiente Grid. Mediante la utilización de una cadena de Markov en tiempo discreto (DTMC) que representa el proceso de planificación de tareas, se define un problema de programación no lineal (NLP por sus siglas en inglés). Luego de resolver el problema de programación no lineal se obtienen las probabilidades de conexión y así se puede obtener la mejor ruta de planificación dentro del ambiente y a su vez obtener el tiempo de respuesta promedio mínimo de una tarea en particular. Dado un entorno de grid, los autores obtienen una matriz que representa la conexión entre los administradores y los recursos de cómputo. Esta matriz, llamada matriz de conectividad, muestra la topología del entrono de la Grid. En los experimentos que se realizan en este trabajo utilizan dos entornos grid diferentes, para ambos casos se plantean las matrices correspondientes y se se calcula la inversa paramétrica de la matriz

ISSN 2007-9737

fundamental N y el problema de optimización NLP obtenido se resuelve en ambos casos utilizando el software LINGO [1]. Los resultados muestran que la planificación obtenida es mejor en todos los casos que los algoritmos *benchmark* utilizados como referencia, sin embargo, ésto es cierto sólo para tareas *Singleton* y por lo tanto que no consideran el caso en que las tareas contengan varios procesos con restricciones de precedencia.

Van Laarhoven [31] y otros describen un algoritmo de aproximación para el problema de encontrar el plan con tiempo de terminación total (*makespan*) mínimo de un taller de trabajos *job shop*. El algoritmo está basado en "enfriamiento simulado" (*simulated annealing*), que consiste en una generalización del proceso de mejora de la aproximación por iteración empleado en problemas de optimización combinatoria. La generalización involucra la aceptación de transiciones de costo incremental con una probabilidad no nula para evitar que el problema quede atrapado en un mínimo local. Se demuestra que el algoritmo converge asintóticamente en probabilidad a una solución mínima global, a pesar del hecho de que las cadenas de Markov generadas por el algoritmo son generalmente irreducibles. Los experimentos muestran que el algoritmo puede encontrar planes con tiempo de terminación total (makespan) menores que algoritmos de aproximación específicamente diseñados para el problema de planificación en talleres de trabajos (*job shop*). Lo anterior, sin embargo, a un costo mayor en términos del tiempo de ejecución.

En [5] Blythe y otros analizan una heurística para la planificación de flujos de trabajo que consiste en dos programas, el generador de un flujo de trabajo abstracto (AWG) y uno concreto (CWG). AWG selecciona y configura componentes de aplicación de acuerdo a las capacidades especificadas y si es posible que puedan terminar la tarea deseada. Por otro lado, CWG selecciona recursos específicos, archivos y trabajos adicionales necesarios para que el flujo de trabajo pueda ser ejecutado en el entorno Grid. Los autores desarrollaron un sistema de planificación con Inteligencia Artificial que opera de la siguiente manera. En primer lugar el planificador permite que el usuario ingrese las características de su trabajo en forma de metadatos; en segundo lugar el planificador representa las restricciones del flujo de trabajo como dependencias de datos en el programa y limitantes en los recursos; en tercer lugar el planificador crea diversos planes alternativos y devuelve el mejor de ellos de acuerdo a algún criterio de calidad específico. La calidad de los planes generados se obtiene con respecto a alguno de los siguientes parámetros. Tiempo esperado de ejecución, la probabilidad de ejecución satisfactoria, la utilización de los recursos o datos que son costosos o restringidos para el usuario. Esta técnica, sin embargo, utiliza solamente un planificador centralizado dejando abierto e inconcluso un planificador distribuido en la Grid.

Xian-He y Ming [34] presentan un sistema de evaluación de desempeño y planificación de tareas para la solución de aplicaciones de gran escala en un ambiente de red compartido, llamado GHS. El sistema está compuesto de cinco subsistemas, Evaluación de desempeño, medición del desempeño, asignación de tareas, planificación de tareas y administración de la ejecución. En cuanto a la planificación de tareas, este sistema utiliza un algoritmo de particionamiento de tareas con respecto al tiempo medio para distribuir la carga de trabajo de manera que la diferencia del promedio y el tiempo de ejecución de cada subtarea sea mínimo. Además, implementan un algoritmo min-min para agrupar las subtareas de una meta-tarea y generar una ruta hacia el recurso para cada conjunto de subtareas de acuerdo a la predicción del tiempo de ejecución de cada conjunto. La evaluación del desempeño la realizan mediante una predicción. Utilizan redes neuronales artificiales para modelar el tráfico en la red, y mediante el aprendizaje en línea, este modelo puede considerar cambios en el entorno y adaptarse a ellos. Este modelo predice el desempeño de la red en función del ancho de banda disponible y la latencia. Para la planificación de tareas, proponen un algoritmo de planificación heurístico para encontrar una solución cercana al óptimo con un costo razonable. Este sistema utiliza técnicas de inteligencia artificial para la predicción del desempeño y posteriormente hacer una planificación de las tareas en la Grid en función de dicha predicción, más no utiliza la inteligencia artificial la planificación misma.

Guangwen y otros autores [35] publican un algoritmo genético cuántico para la planificación de tareas en un sistema Grid. Los algoritmos genéticos tradicionales tienen desventajas tales como requerir una gran cantidad de procesamiento, pérdida de los mejores cromosomas y maduración temprana que no permiten obtener la solución óptima. Los autores proponen un algoritmo basado en teoría de cómputo cuántico y algoritmos genéticos. El algoritmo genético cuántico (QGA) presentado, es un algoritmo basado en teoría cuántica tal como superposición cuántica y entrelazado cuántico. El sistema propuesto consiste de un módulo de tareas, módulo de análisis de tareas, módulo de administración de la planificación y módulo de administración de recursos. El módulo de análisis de tareas recauda información de las tareas como el tiempo de terminación de la tarea, dependencias de las tareas y retraso de la red. El módulo de administración de la planificación selecciona una estrategia apropiada de planificación y asigna las tareas a los recursos. QGA utiliza *q-bits* para cromosomas código e introduce entrelazado cuántico para la codificación genética obteniendo la evolución de los cromosomas con una compuerta giratoria cuántica. Los experimentos comparan los algoritmos Min-Min basado en *QoS*, un algoritmo genético tradicional y QGA. Los resultados muestran que el desempeño total de la planificación basada en QGA es superior a los competidores. Sin embargo, aunque el *time span* en un Algoritmo Genético es superior a otros algoritmos heurísticos cuando la tarea está en una escala pequeña, si la escala se amplía, entonces el algoritmo genético obtiene un mínimo local muy rápido y converge lentamente, el mejor cromosoma se pierde fácilmente ocasionando una convergencia inmadura.

Prado y otros autores [26] presentan un metaplanificador basado en reglas difusas para cómputo en Grid. Presentan resultados comparativos entre dos estrategias de aprendizaje no genéticas derivadas de algoritmos bio-inspirados, que son Evolución Diferencial (DE) y Optimización de Enjambre de Partículas (PSO), para la evolución del meta-planificador Basado en Reglas Difusas (FRBS). Los FRBS son sistemas expertos utilizados para resolver la problemática de la planificación debido a su capacidad de tratar con información incierta y simular el razonamiento humano. Sin embargo, dada la alta dependencia del desempeño de FRBS con la calidad de sus bases de conocimiento, el problema de aprendizaje en planificadores basados en reglas difusas es un problema crítico. La estructura del meta-planificador difuso funciona de la siguiente forma. El esquema FRBS examina el estado de los recursos disponibles proporcionada por los planificadores locales y almacena esta información en variables. Luego, el metaplanificador inicia un proceso de traducción de los estados difusos de los recursos en índices de selección, que representan la conveniencia para el planificador. La evolución diferencial para la adquisición del conocimiento en los meta-planificadores basado en reglas difusas opera tal como los algoritmos evolutivos, esto es inicialización, mutación, cruce y selección. El proceso de mutación, cruce y selección es iterado hasta que las condiciones de paro se cumplen, siendo el número de generaciones la condición de paro. KASIA es la adaptación de PSO para el aprendizaje de las reglas base en FRBS. De esta manera, cada partícula representa una regla para la población llamada enjambre cuya meta es alcanzar la ubicación óptima para las partículas. En otras palabras, el conjunto de reglas proveen el mejor desempeño en términos de métricas seleccionadas. Los resultados de los experimentos muestran el comportamiento de la convergencia de las diferentes estrategias durante el proceso de aprendizaje. Aunque en la etapa inicial KASIA no muestra una diferencia relevante con respecto a sus competidores, posteriormente se observa una convergencia más rápida, que significa un menor esfuerzo de cómputo. A pesar de que KASIA muestra una mejora con respecto a sus competidores, aún son necesarias al menos 100 iteraciones para el aprendizaje.

3. El problema de planificación en sistemas Grid

En la presente sección se plantea de manera formal el problema de planificación de tareas en un sistema Grid. Como se muestra a continuación, una instancia de este problema consta de las especificaciones de las capacidades y topología de la

ISSN 2007-9737

Grid, así como de las tareas que componen la carga de trabajo que debe ser atendida. Los nodos del grafo modelan a los elementos de procesamiento con que cuenta la Grid, y las aristas a la infraestructura de comunicaciones que interconecta a los elementos de procesamiento. Adicionalmente, existen funciones que mapean tanto a nodos como aristas con sus respectivas capacidades. Por su parte, las cargas de trabajo se modelan como un flujo de tareas que a lo largo del tiempo son sometidas a la Grid para ser atendidas. Cada una de estas tareas puede estar compuesta por un único proceso o por un conjunto de procesos cuyas dependencias funcionales son modeladas por medio de un grafo acíclico dirigido (DAG por sus siglas en inglés). Adicionalmente, existen funciones que mapean a las tareas con sus requerimientos específicos de procesamiento, memoria y comunicación.

3.1. Definición de una Grid

Una *Grid* G se modela por medio de un grafo no dirigido $G = (N, L)$, donde cada uno de los vértices $n \in N$ representa un nodo de la Grid y cada una de las aristas $l \in L$ modelan un enlace de comunicación entre nodos de la Grid. A su vez, los nodos Grid están definidos por medio de un grafo $n = (M_n, O_n)$ que modela la topología interna de un nodo Grid, donde cada uno de los vértices $m \in M_n$ representa una máquina en el nodo Grid n y las aristas $o \in O_n$ representan un enlace de comunicación entre las máquinas que componen al nodo Grid n.

Ahora para modelar las capacidades de los diferentes elementos de la Grid se utilizan las funciones c, c_{N_G}, μ y ϱ donde $c : L \rightarrow \mathbb{N}$ describe la capacidad de cada uno de los enlaces de comunicación entre nodos Grid, es decir una función que para cada arista asigna el ancho de banda disponible en el enlace de comunicación representado por dicha arista. $c_{N_G} : O_n \rightarrow \mathbb{N}$ describe la capacidad de los enlaces de comunicación que conectan a los procesadores que componen a un nodo Grid n y $\mu : M_n \rightarrow \mathbb{N}$, $\varrho : M_n \rightarrow \mathbb{N}$ describen respectivamente la capacidad de memoria y de procesamiento de un nodo $m \in M_n$.

3.2. Definición de carga de trabajo

Sea $T_i = \{llegada_i, DAG_i\}$ una dupla que define los recursos estimados por el usuario para la tarea i, donde $llegada_i$ es el tiempo en el que la tarea i es sometida a la Grid G, $DAG_i = (V_i, A_i)$ es un grafo acíclico dirigido que describe las relaciones de precedencia entre los procesos que componen a la tarea T_i, donde V_i es el conjunto de procesos que componen a la tarea T_i y $(u, v) \in A_i$ es un conjunto de relaciones de precedencia entre procesos que indican que la tarea $v \in V_i$ necesita que la tarea $u \in V_i$ finalice para poder ser ejecutada. Adicionalmente, la función $\delta : A \rightarrow \mathbb{N}$ asocia a cada arista dirigida $(u, v) \in A_i$ con una cantidad de datos que el proceso u debe transmitir al proceso v antes de que este último pueda ser ejecutado. A su vez, cada uno de los procesos $p \in V_i$ tiene asociada una cantidad de unidades de memoria ($memoria_p$) y de *tiempo de procesamiento normalizado* (t_{cpu_p}) que el proceso p requiere para su ejecución. El *tiempo de procesamiento normalizado* está definido como el tiempo que le toma a un procesador de referencia ejecutar un proceso dado.

A partir de los conceptos antes expuestos, podemos definir a una *carga de trabajo* $\mathbb{L} = \{T_1, T_2, T_3, ..., T_m\}$ como a un conjunto de $|\mathbb{L}|$ trabajos T_i que son sometidos a una Grid para ser ejecutados.

3.3. Definición de plan de ejecución para una Grid

Dada una Grid $G = (N, L)$ y una carga de trabajo $\mathbb{L}$, un plan de ejecución S es una función $S : \bigcup_{\forall T_i \in \mathbb{L}} V_i \rightarrow \bigcup_{\forall n \in N} M_n \times \mathbb{N}$ que asigna los procesos que componen a la carga de trabajo, a los procesadores (o máquinas) que componen la Grid y les asigna un tiempo en el que comenzarán a ser ejecutados. Para especificar los tiempos de inicio de ejecución, asumimos que se utilizan unidades discretas de tiempo que pueden estar en función de la longitud del ciclo de máquina. Así, los planes pueden ser vistos como un conjunto de 3-tuplas de la forma $\{p, m, t^p_{ini}\}$ que indican que el proceso p comienza su ejecución en el tiempo t^p_{ini} en la máquina m. El plan individual de una máquina m denotado por S_m está compuesto por todas

aquellas 3-tuplas en las que participa la máquina m. Es decir, el plan S_m especifica el tiempo de inicio de todos los procesos que fueron asignados a la máquina m.

Para que un plan S sea *válido*, se deben cumplir las dos condiciones que se enumeran a continuación.

1. Todos los procesos que componen a la carga de trabajo deben ser asignados a un único procesador y los intervalos definidos por los tiempos de inicio y fin de ejecución de los procesos no pueden estar traslapados.
2. El plan debe respetar las restricciones de precedencia definidas por los grafos acíclicos dirigidos que componen a las tareas, es decir que para todo par de procesos u, v, tal que $(u, v) \in DAG_i$ para alguna tarea T_i, entonces el tiempo en que se planifica el proceso v (t^v_{ini}) debe ser mayor al tiempo en el que el proceso u termina de ser ejecutado (t^u_{fin}). Más aún, debe ser cierto que $t^v_{ini} \geq t^u_{fin} + \delta(u, v)/BW_{camino}$, donde BW_{camino} denota la velocidad de transmisión disponible en el camino que conecta a los procesadores donde u y v son ejecutados.

3.4. El problema de planificación en sistemas Grid

Una vez que hemos definido formalmente todas las entidades involucradas, podemos formular al problema de planificación en sistemas Grid como un problema de optimización. El problema de planificación en sistemas grid consiste en, dada una Grid G y una carga de trabajo $\mathbb{L}$, generar un plan válido S que optimice alguna de las métricas o funciones objetivo que se enuncian a continuación.

1. Retardo máximo. Es el retardo máximo experimentado por alguna de las tareas en la carga de trabajo. El retardo se mide como la diferencia entre el tiempo en que la tarea fue sometida a la Grid y el tiempo que concluyó su ejecución el último de los procesos que la componen.
2. Retardo promedio. Es el retardo promedio experimentado por todas las tareas de la carga de trabajo.
3. Utilización de la Grid. Es el porcentaje de tiempo que los procesadores que componen a la Grid se mantuvieron ocupados.

A continuación se definen formalmente de cada una de estas métricas en términos de los planes S generados para atender una carga de trabajo $\mathbb{L}$ en una Grid G.

El retardo R_{T_i} experimentado por la tarea $T_i = \{llegada_i, DAG_i\}$ dado un plan válido S se define como $R_{T_i} = T_{i_{fin}} - T_{i_{ini}}$ donde $T_{i_{fin}} = \max_{p \in DAG_i} \{t^p_{ini} + t_{cpu_p} \times \varrho(m)\}$ y $T_{i_{ini}} = \min_{p \in DAG_i} \{t^p_{ini}\}$ con $\{p, m, t^p_{ini}\} \in S$.

El retardo máximo R_{max} se calcula por medio de la Ecuación 1 y el retardo promedio $\bar{R}$ por medio de la Ecuación 2.

$$R_{max} = max_{T_i \in \mathbb{L}}\{R_{T_i}\} \tag{1}$$

$$\bar{R} = \frac{1}{|\mathbb{L}|} \sum_{T_i \in \mathbb{L}} R_{T_i} \tag{2}$$

Finalmente, la utilización global de la grid se encuentra en función de la utilización individual de cada uno de los procesadores que la componen. Para calcular la utilización de cada uno de los procesadores es necesario calcular el porcentaje del tiempo que cada procesador se encuentra ocupado ejecutando un proceso.

Sean $T_{inicial}$ y T_{final} el tiempo en que inicia y termina la simulación respectivamente. La utilización de la máquina $m_i \in M_n$, denotado por ρ_{m_i}, se calcula por medio de la Ecuación 3 y la utilización de la grid, denotado por $\bar{\rho}$ se calcula por medio de la Ecuación 4.

$$\rho_{m_i} = \frac{\sum_{\{p,m,t^p_{ini}\} \in S : m = m_i} \left[t^p_{ini} + t_{cpu_p} \times \varrho(m)\right]}{T_{final} - T_{inicial}} \tag{3}$$

$$\bar{\rho} = \frac{1}{|M_n|} \sum_{m_i \in M_n} \rho_{m_i} \tag{4}$$

ISSN 2007-9737

4. Algoritmo de distribución deslizante

En la presente sección se muestra la especificación del esquema de planificación propuesto que utiliza la distribución de Gibbs-Boltzmann [23] para *deslizar* su modo de operación desde puramente voraz (*pure greedy*) hasta puramente aleatorio. En el modo voraz, se parametriza la distribución de Gibbs-Boltzmann de la Ecuación 5 para que el algoritmo tome decisiones en base a la estrategia voraz con una muy alta probabilidad, mientras que en el modo aleatorio, se parametriza la distribución para que el algoritmo pueda tomar cualquier decisión con la misma probabilidad. En otras palabras, definimos una distribución que se desliza desde el punto en que es altamente probable tomar una decisión voraz hasta el punto en que todas las decisiones son igualmente probables. A continuación detallaremos el funcionamiento del algoritmo propuesto.

En primera instancia, cada Nodo Grid tiene un sistema de colas para atender el flujo de tareas entrantes y la forma cómo las tareas son admitidas en las colas es descrito por el Algoritmo 1. Una vez que una tarea T_i es aceptada en el Nodo Grid se clasifica ya sea en tarea que contiene un grafo de procesos (DAG_i, línea 3) o bien en tarea Singleton (línea 6). Para cada tarea que contenga un grafo de procesos se crea una cola individual que almacenara a los procesos de la tarea de acuerdo a su orden topológico. En el caso de los procesos Singleton se tiene una cola común para todos ellos.

Algorithm 1: Algoritmo de la creación de las colas

input : T_i: Extraer Tarea de cola
1 $ColaSing.Crear\ ObtenerSiguienteTarea()$
3 **if** $T_i.type = DAG$ **then**
4 $\quad Cola_i.Crear$
$\quad Cola_i.Encolar(OrdenTopológico(T_i.DAG))$
6 **else**
7 $\quad ColaSing.Encolar(T_i))$
8 $Actualizar(NúmeroActualColas)$

En un instante dado, el sistema puede tener m colas no vacías que contienen a los procesos restantes de las tareas que han sido sometidas al Nodo Grid. En este caso, el Algoritmo 2 recorre las colas y planifica los procesos de acuerdo a su tiempo de llegada. Por cada tarea compuesta se obtienen $3n$ planes con $kT = \{10, 1, 0.1\}$ donde n es el número de procesos en la tarea a planificar, kT es el parámetro de la distribución Gibbs-Boltzmann [23] (Ecuación 5) que permite modificar el comportamiento del algoritmo de selección de procesador destino. Cuando $kT = 10$ la selección de los procesadores es aleatoria, cuando $kT = 1$ se analizan las soluciones en el vecindario extendido de una solución ρ-aproximada y cuando $kT = 0.1$ se analizan las soluciones en el vecindario cercano a una solución ρ-aproximada. De las opciones anteriores, se selecciona el mejor de los $3n$ planes con respecto a alguna de las métricas: retardo máximo, retardo promedio y utilización de la Grid (línea 20 del Algoritmo 2).

El algoritmo de recorrido de las colas (Algoritmo 2) obtiene los n planes para cada valor de kT (línea 14), $3n$ en total, y selecciona el mejor de ellos con respecto a alguna métrica (líneas 17 y 20).

La función de planificación (Algoritmo 3) recibe la tarea T_i a planificar. En el ciclo de la línea 8 se recorren los $|M_n|$ CPU's que contiene el Nodo Grid y calcula para cada uno de ellos el tiempo de inicio de la ejecución y el coeficiente de afinidad. El ciclo de la línea 11 recorre para cada uno de los CPU's todos los demás y junta los *datos* que algún proceso predecesor hubiese dejado como resultado, calculando el tiempo que le toma transportarlos al CPU en cuestión. Estos *datos* son el resultado de la ejecución de los procesos predecesores que son posiblemente indispensables para la ejecución del proceso siguiente, por lo que deberán estar en el procesador donde se planifique si son requeridos.

La probabilidad de asignar el proceso j al procesador μ es igual a:

$$Pr_{j\to\mu} = \frac{e^{-t_{fin_{j\to\mu}}/kT}}{\sum_{\forall m\in M} e^{-t_{fin_{j\to m}}/kT}} \tag{5}$$

Esta probabilidad considera:

ISSN 2007-9737

Algorithm 2: Algoritmo de recorrido de las colas

```
1  recorridoColas (m colas DAG, cola Singleton) begin
2      i ← 1; j ← 1; k ← 1; kT ∈ {10, 1, 0.1};
3      while i ≤ (m + 1) do
4          T_i ← extraerCola()
5          n ← numeroProcesos(T_i)
6          j ← 1
8          while j ≤ |kT| do
9              k ← 1
11             while k ≤ n do
12                 /* n planes ∀ kT                    */
14                 S_k ← Planificar(T_i, kT_j)
15                 k ← k + 1
17             S_j ← selMejorPlan(S_1, S_2, ..., S_n, metrica)
18             j ← j + 1
20         S_{T_i} ← selMejorPlan(S_1, S_2, S_3, metrica)
21         Encolar(S_{T_i})
22         i ← i + 1
```

- La carga asignada a los procesadores.
- Las relaciones de precedencia entre los procesos.
- El costo de red de mover los datos hacia el procesador destino.

Dado que en el nodo Grid los recursos son heterogéneos, las capacidades de ellos pueden ser distintas. Por esta razón la capacidad de cada recurso es normalizada con respecto a uno que se toma como referencia para toda la Grid. En primer lugar se obtiene la capacidad de todos los recursos mediante el *benchmark* SPECint [2], luego se realiza el cociente de la capacidad de algún recurso entre el que se usa como referencia, con lo que se obtiene el coeficiente de capacidad. Entonces el tiempo de ejecución de un proceso en un CPU se debe calcular como el cociente del tiempo normalizado solicitado para ese proceso y el coeficiente de capacidad del CPU dado.

El tiempo de espera indica el momento en el que el proceso podrá iniciar la ejecución en el CPU. Por último el tiempo de fin indica el tiempo que le toma al proceso ser ejecutado en el procesador considerando todos los tiempos anteriores.

El coeficiente de afinidad que se calcula en la línea 19 del algoritmo 3 se obtiene mediante la distribución Gibbs-Boltzmann para un valor dado de kT.

Una vez que se tienen todos los tiempos y coeficientes de afinidad, en el ciclo de la línea 24 calcula las probabilidades de asignar al proceso actual cada uno de los los CPUs con los que cuenta la Grid para finalmente hacer la selección aleatoria (línea 28) y encolar el proceso (línea 30).

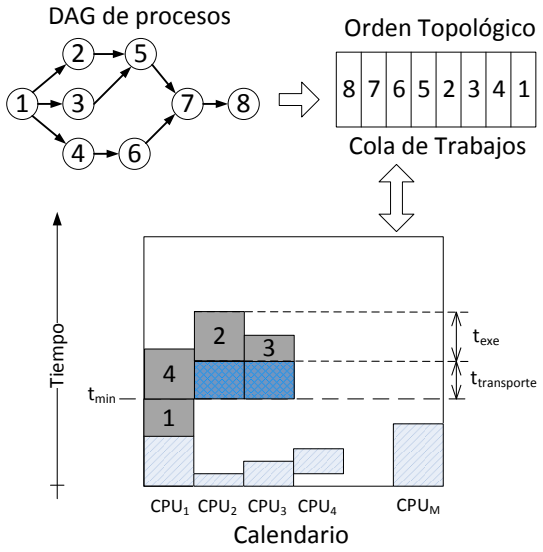

Fig. 1. Planificación de un DAG de procesos

El Algoritmo 4 encuentra el tiempo en que un CPU puede ejecutar un proceso de acuerdo a su disponibilidad. Para esto toma en cuenta el tiempo de ejecución del proceso, el tiempo que toma transportar los datos del proceso a través de la red (tiempo de transporte), así como la carga actual del CPU. En primer lugar se debe determinar si el proceso pertenece a un grafo de procesos (DAG_i) o si es un Singleton (línea 3). En caso de que se tratara de un proceso perteneciente a un grafo de procesos, se debe determinar el tiempo en que terminarán su ejecución todos sus procesos predecesores (línea 6). A este instante, lo denominaremos

Algorithm 3: Función de planificación

```
1  Planificar(Tarea T_0, Coeficiente kT) begin
2      i ← 1
3      n ← numeroProcesos(T_0)
4      while i ≤ n do
5          P_i ← extraerProcesoSiguiente(T_0)
6          k ← 1
8          while k ≤ M do
9              l ← 0; t_transporte ← 0; coef_afin_T ← 0
11             while l ≤ M do
12                 if CPU_l.Datos(P_i) then
13                     Datos_l ← CPU_l.Datos(P_i)
                       t_transporte ← t_transporte +
                       tRutaMasCorta(Datos_l, l, k)
14                 l ← l + 1
15             t_exe_k ← P_i.tiempo_exe / coefCPU_k
16             t_espera_k ←
               dispCPU(t_exe_k, t_transporte, CPU_k, P_i)
17             t_fin_k ← t_exe_k + t_transporte + t_espera_k
19             coef_afin_k ← e^(−t_fin_k/kT)
20             coef_afin_T ← coef_afin_T + coef_afin_k
21             k ← k + 1
22         k ← 1
24         while k ≤ M do
25             p_k ← coef_afin_k / coef_afin_T
26             k ← k + 1
28         x ← seleccionAleatoria
           (p_1, p_2, p_3, ..., p_M)
30         posicionarEnCola (P_i, CPU_x)
31         i ← i + 1
```

por tiempo de inicio mínimo (t_{min}). En base a esta información, se busca en el plan del CPU (línea 7) un espacio de tiempo (tiempo de espera) para el proceso en cuestión considerando el tiempo de ejecución, el de transporte en red y el tiempo en que terminan sus procesos predecesores.

En la Figura 1 se muestra un ejemplo de la forma en que se determina el tiempo mínimo y el tiempo de transporte. En este ejemplo, dado que los procesos 2, 3 y 4 dependen del proceso 1 su tiempo mínimo se establece como el momento en el que éste último proceso termina su ejecución.

Al planificar los procesos subsecuentes 2, 3 y 4 se debe considerar como necesario el resultado de la ejecución del predecesor y el tiempo que tomará llevarlo al proceso donde el subsecuente será ejecutado, éste lo denominaremos como tiempo de transporte. En consecuencia si el proceso siguiente se ejecuta en el mismo procesador que su predecesor, el tiempo de transporte será igual a cero. El tiempo de ejecución se determina de acuerdo a las capacidades de cada procesador dado que se trata de un entorno heterogéneo.

En caso de que se tratase de un Singleton, se busca también el espacio de tiempo, pero sólo considerando el tiempo de ejecución y el tiempo de inicio mínimo, que para este caso es el tiempo de llegada del proceso. El tiempo de transporte sólo aplica para procesos de una tarea compuesta por un grafo de procesos DAG_i.

Cabe mencionar que el tiempo de inicio mínimo debe ser necesariamente mayor al tiempo en que la tarea llegó a la Grid, dado que un proceso no puede ser planificado antes de su llegada a la grid.

Algorithm 4: Función que busca un espacio de tiempo para el proceso dado

```
1  dispCPU (t_exe_k, t_transporte, CPU_k, P_i) begin
3      if J_i.type = DAG then
4          t_espera ← 0
6          t_min ← máx_{1<i<M} {tFinalización(P_i.predecesores)}
7          t_espera ←
           CPU_k.tiempo_disponible(t_exe, t_transporte, t_min)
8      else
9          t_espera ←
           CPU_k.tiempo_disponible(t_exe, 0, t_min)
10     return t_espera
```

5. Análisis

Esta sección contiene una serie de teoremas que establecen que el algoritmo propuesto es correcto y que caracterizan de manera formal su

complejidad temporal. En concreto, se demuestra que los planes generados respetan las relaciones de precedencia de todos los procesos. De igual manera, se presentan una serie de teoremas que caracterizan de manera aproximada la probabilidad de que las soluciones entregadas sean una ρ-aproximación.

Teorema 1. *Todo plan S para la grid G generado por el algoritmo propuesto, respeta las relaciones de precedencia de todo proceso contenido en la carga de trabajo $\mathbb{L}$.*

Demostración. Sea $DAG_i = (V_i, A_i)$ un grafo acíclico dirigido cuyos nodos representan a los procesos de una tarea arbitraría y sea ϑ cualquier orden topológico de DAG_i. Demostraremos por inducción sobre el número de procesos en ϑ que todas las restricciones de precedencia de un proceso $j \in V_i$ son respetadas por los planes generados por los Algoritmos 3 y 4.

Caso base. El primer proceso que aparece en el orden topológico ϑ, denotado por u_0, es planificado respetando sus restricciones de precedencia debido a que $(x, u_0) \notin A_i$ para cualquier $x \in V_i$. En caso contrario ϑ no sería un orden topológico de DAG_i.

Hipótesis de inducción (HI): Suponemos que los primeros k procesos en ϑ son planificados por los Algoritmos 3 y 4 de forma tal que todas sus relaciones de precedencia son respetadas. Por demostrar que, el $k+1$-ésimo proceso también es planificado de forma tal que todas sus relaciones de precedencia son respetadas. Sea u $k+1$-ésimo proceso en ϑ. Debido a que el Algoritmo 3 considera a los procesos en base al orden topológico ϑ, todo proceso v tal que $(v, u) \in A_i$ ya ha sido planificado en el momento que la tarea u está siendo considerada. En particular el proceso v_{ultimo} cuyo tiempo de finalización es máximo entre los procesos v tal que $(v, u) \in A_i$. Ahora, el Algoritmo 4 asigna un tiempo de inicio a la tarea u que es mayor al tiempo de finalización de v_{ultimo} y por lo tanto al tiempo de finalización de todo proceso v tal que $(v, u) \in A_i$. En consecuencia, todos las relaciones de precedencia del proceso u son respetadas. Aplicando la HI podemos concluir que los $k+1$ primeros procesos en ϑ son planificados respetando todas sus restricciones de precedencia. □

Teorema 2. *La complejidad del Algoritmo 3 es $O(n(mn + ma \log m))$ donde n es el número máximo de procesos por tarea y $m = |M_n|$ es el número de procesadores con que cuenta la grid G y $a = |O_n|$ es el número de enlaces de comunicación en la grid G.*

Demostración. El Algoritmo 4 itera m veces revisando los planes de los m procesadores. Revisar el plan de un procesador para encontrar el primer hueco que pueda contener al proceso actual toma $O(n)$. Por lo tanto, la complejidad del Algoritmo 4 es $O(mn)$. Ahora, el Algoritmo 3 itera sobre los n procesos que componen a la tarea y por cada proceso se itera sobre los m procesadores que componen la Grid. Por cada una de estas iteraciones se llama una vez al Algoritmo 4 y m veces al algoritmo de Dijkstra cuya complejidad es $O(a \log m)$. Como $O(mn) + O(ma \log m) = O(mn + ma \log m)$, entonces la complejidad final del Algoritmo 3 es de $O(n(mn + ma \log m))$. □

En los siguientes teoremas se muestra que el algoritmo propuesto provee una 2-aproximación con alta probabilidad. Para hacer las demostraciones tratables, los teoremas y lemas siguientes asumen que los procesos no tienen restricciones de precedencia.

Lema 1. *En un nodo Grid conformado por m procesadores, el tiempo de finalización óptimo (T^*) de un trabajo J compuesto de n procesos está acotado por $T^* \geq \frac{1}{m} \sum_{i \in T} t_i$*

Demostración. Dado que el tiempo de procesamiento total es $\sum_{i \in J} t_i$, al menos una máquina debe ejecutar una fracción de al menos $\frac{1}{m}$-ésimo del trabajo total. □

Lema 2. *En un nodo Grid conformado por m procesadores, el tiempo de finalización óptimo es al menos $T^* \geq \max_{i \in J} t_i$.*

Demostración. El proceso más grande debe ser ejecutado en alguna máquina. □

Teorema 3. *Sea un conjunto de procesos $i \in J$. Entonces, un algoritmo voraz que selecciona el procesador que produce el menor tiempo de finalización para cada proceso, provee una $\rho = 2$ aproximación al problema de planificación de retardo mínimo.*

Demostración. Consideremos la carga L_B de la máquina B que termina más tarde en ejecutar sus procesos asignados, y sea j el último proceso que se le asignó. Cuando j le fue asignado a B, B tenía el menor tiempo de terminación (en otro caso el algoritmo voraz no la hubiera seleccionado). Por lo tanto $L_B - t_j \leq L_k$ para todo $1 \leq k \leq m$ y $L_B - t_j \leq \frac{1}{m}\sum_{i \in T} t_i$. Ahora por el Lema 1 podemos concluir que $L_B - t_j \leq T^*$. De igual manera, por el Lema 2 sabemos que $t_j \leq T^*$ y por lo tanto podemos concluir que $L_B \leq 2T^*$. □

Teorema 4. *Sea un conjunto de procesos $i \in J$ y sea G un nodo Grid conformado por m procesadores. Entonces, el algoritmo aleatorizado basado en distribuciones deslizantes que toma la secuencia de procesos definida en ϑ falla en encontrar una $\rho = 2$ aproximación al problema de planificación de retardo mínimo con probabilidad $\left[1 - \prod_1^{|\vartheta|}(e^{-t_{fin_{min_i}}}/\aleph_i)\right]^n$ donde los $\aleph_i$ son las constantes de normalización calculadas en cada iteración del algoritmo.*

Demostración. Por el Teorema 3 sabemos que una solución 2-aproximada se obtiene al tomar procesos en la secuencia dada por ϑ y asignarlos a la máquina donde su tiempo de terminación sea menor.

Ahora, sea ε_i el evento de asignar el i-ésimo proceso de la secuencia ϑ a la máquina donde terminará primero.

Entonces la probabilidad de que el algoritmo encuentre una solución 2-aproximada es $\geq Pr[\bigcap_1^{|\vartheta|} \varepsilon_i] = Pr[\varepsilon_1] \times Pr[\varepsilon_2|\varepsilon_1] \times \ldots \times Pr[\varepsilon_{|\vartheta|-2}|\varepsilon_1 \cap \varepsilon_2 \ldots \varepsilon_{|\vartheta|-3}]$.
Ahora, dada la distribución impuesta sobre nuestro algoritmo, tenemos que la probabilidad de encontrar una 2-aproximación en una iteración completa es $Pr[\bigcap_1^{|\vartheta|} \varepsilon_i] = \prod_1^{|\vartheta|}(e^{-t_{fin_{min_i}}}/\aleph_i)$.

Por lo tanto, la probabilidad de no encontrar una 2-aproximación después de n intentos es $\left[1 - \prod_1^{|\vartheta|}(e^{-t_{fin_{min_i}}}/\aleph_i)\right]^n$. □

6. Resultados experimentales

En esta sección se describen una serie de experimentos basados en simulaciones, mediante los cuales se caracteriza de forma cuantitativa el desempeño del algoritmo propuesto, así como el de algunos de los algoritmos más representativos del estado del arte en el contexto de planificación en sistemas Grid. Como métricas de desempeño utilizamos el retardo promedio, el retardo máximo y el uso de la grid. Estas métricas fueron definidas a detalle en la Sección 3.4. En todas las simulaciones se consideran tanto los costos de comunicación como los costos de procesamiento. Los costos de comunicación dependen de la cantidad de datos que transfieren los procesos con precedencia y de la capacidad de la red de interconexión de los procesadores de la Grid.

Los algoritmos elegidos para nuestro análisis de desempeño han sido usados en diversos trabajos como [10, 6] y se han consolidado como un estándar *de facto* para análisis comparativos en el desempeño de algoritmos de planificación para sistemas grid. Dado que el método propuesto planifica una tarea a un recurso inmediatamente después de haber llegado a los administradores, el método propuesto cae en el esquema de planificación de modo inmediato. Los algoritmos de planificación de modo inmediato seleccionados son balanceo de carga inmediata (OLB), tiempo de ejecución mínimo (MET) y tiempo de finalización mínimo (MCT). A continuación se describe brevemente el funcionamiento de cada uno de ellos.

- *OLB*. Este algoritmo asigna a cada tarea al recurso que esté mas pronto disponible sin consideración alguna del tiempo de ejecución de la tarea en el recurso. Si dos o más recursos están disponibles se selecciona uno arbitrariamente. La filosofía del algoritmo es mantener los recursos tan ocupados como sea posible. Una ventaja del algoritmo es su simplicidad

pero debido a que no considera el tiempo de ejecución de la tarea el plan obtenido no es óptimo.

- *MET*. Este algoritmo asigna a cada tarea al recurso de forma que resulte en el menor tiempo de ejecución sin importar la disponibilidad de la máquina. Cuando una tarea llega, todos los recursos son examinados para determinar aquel que entregue el tiempo de ejecución menor para dicha tarea. La filosofía detrás del algoritmo es asignar a cada tarea el mejor recurso existente para su ejecución, pero asociar una tarea a un recurso sin considerar su disponibilidad ocasiona un desbalance en la carga en recursos grid.
- *MCT*. Este algoritmo asigna a cada tarea al recurso que proporcione el tiempo de finalización más próximo. Cuando una tarea llega a la grid todos los recursos disponibles son analizados para determinar cual proporciona el menor tiempo de finalización. En este algoritmo una tarea puede ser asignada a un recurso que no tenga el menor tiempo de ejecución. La filosofía detrás de este algoritmo es combinar las ventajas de *OLB* y *MET* evitando sus desventajas.

Los algoritmos anteriormente mencionados fueron modificados para cumplir con las relaciones de precedencia de las tareas compuestas por más de un proceso.

Es importante mencionar que en este análisis no se incluyeron propuestas basadas en técnicas como algoritmos genéticos o aquellas basadas en aprendizaje, debido a que no caen dentro de la clase de algoritmos de planificación de modo inmediato.

6.1. Entorno experimental

Para comparar el desempeño del algoritmo propuesto con el de los algoritmos del estado del arte, se desarrolló una plataforma experimental que consiste de un generador de cargas de trabajo y una Grid. Como se definió en la Sección 3.2, las cargas de trabajo $\mathbb{L} = \{T_1, T_2, T_3, ..., T_m\}$ consisten de una secuencia de tareas que son sometidas a la Grid en diferentes tiempos. En nuestro caso, suponemos que las llegadas siguen un proceso de Poisson con parámetro $1/\lambda$ que define el tiempo promedio entre llegadas. A su vez, con una probabilidad dada que hemos denominado *probabilidad DAG*, cada tarea puede estar compuesta por un grafo de procesos o por un proceso individual. Cuando una tarea contiene un grafo de precedencia procesos (un DAG), nuestro sistema genera un grafo dirigido sin ciclos aleatorio muestreando distribuciones exponenciales para definir diferentes parámetros del grafo como son el *Ancho del DAG* (capas del DAG), el *Largo del DAG* (número de nodos del recorrido mayor sin considerar el nodo icial) y el *Grado Nodo DAG* (número de incidencias en cada nodo). En la Figura 2 se muestra un ejemplo de un DAG de tareas. De manera similar, los requerimientos de memoria y de procesamiento de cada proceso individual son obtenidos mediante el muestreo de una distribución exponencial con parámetros igual a *Memoria* y a *Duración de proceso* y *Procesadores Requeridos* respectivamente. El *tiempo entre llegadas* es la diferencia de tiempo que hay entre una tarea sometida a la grid y la siguiente. La *duración de cada proceso* es el tiempo normalizado que requiere un proceso para ser ejecutado. La *memoria requerida por cada proceso* es la cantidad de kbytes que necesita un proceso para su ejecución o bien la cantidad de kbytes que un proceso genera en su ejecución. Los valores utilizados en los experimentos son descritos en la Tabla 1.

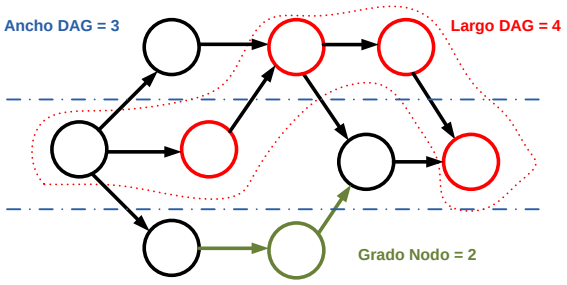

Fig. 2. Ejemplo de un DAG de tareas

Para los resultados mostrados a continuación, dados los valores 2, 20, 100 y 200 segundos del

ISSN 2007-9737

tiempo entre llegadas se generan 10 cargas de trabajo aleatorias para cada uno de ellos. Cada carga de trabajo consiste de 1000 tareas.

Tabla 1. Parámetros utilizados para generar las cargas de trabajo sometidas a la Grid

Parámetro	Valor Promedio
Tiempo entre llegadas	2, 20, 100, 200 s
Duración proceso	20, 40, 80, 160 s
Memoria	1000 Kbytes
Procesadores Requeridos	3
Probabilidad DAG	0.25, 0.5, 0.75, 1
Ancho DAG	4
Largo DAG	3
Grado Nodo DAG	3

Para nuestros experimentos se considera un nodo Grid que contiene un administrador de los recursos que se encarga de realizar el plan y asignar trabajo a todos los procesadores. El administrador conoce las capacidades y los planes de trabajo de todos los recursos en el nodo Grid. Todas las tareas que los usuarios someten a la Grid llegan al administrador. En los experimentos, las cargas de trabajo llegan al administrador y éste genera los planes de todos los recursos. El nodo Grid contiene un total de 475 CPU's heterogéneos, donde la distribución de las capacidades de todos ellos están descritas en la Tabla 2 y se muestran en términos de su coeficiente de capacidad.

Tabla 2. Parámetros de capacidad de procesamiento de la Grid de prueba

Número de CPU's	Coeficiente de Capacidad
80	0.6
80	0.8
80	1.0
80	1.2
80	1.5
75	2.0

Finalmente, suponemos que las computadoras que contienen los CPU's dentro del nodo Grid están conectadas en topología estrella debido a que nos interesa analizar el comportamiento de los algoritmos para el caso de tareas que requieren capacidad de procesamiento.

6.2. Resultados

Se realizan tres experimentos en los que se analiza el impacto en el desempeño de los diferentes algoritmos del tiempo entre llegadas, la duración de los procesos y la probabilidad de que una tarea sea DAG. Se presentan los resultados promedio con intervalos de confianza con un nivel de confianza de 0.95. Para cada punto en las gráficas se usaron 10 ejecuciones independientes con 10 semillas diferentes.

6.2.1. Impacto del tiempo entre llegadas

El primer experimento consiste en el análisis del impacto del tiempo promedio entre llegadas. Para este experimento se generaron 10 cargas aleatorias para cada uno de los valores del tiempo entre llegadas, los cuales fueron de 2, 20, 100 y 200 segundos. En todos los casos la duración promedio de los procesos fue 20 segundos y la probabilidad DAG fue 0.75. Los resultados se presentan en las Figuras 3, 4 y 5.

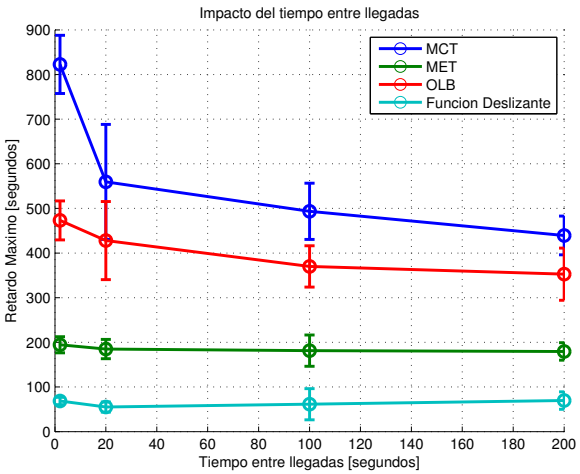

Fig. 3. Impacto del tiempo entre llegadas para retardo máximo

En la Figura 3 se puede observar que el algoritmo propuesto entrega retardos máximos considerablemente más pequeños que los entregados por

ISSN 2007-9737

OLB, MET, y MCT para todos los valores de tiempo promedio entre llegada.

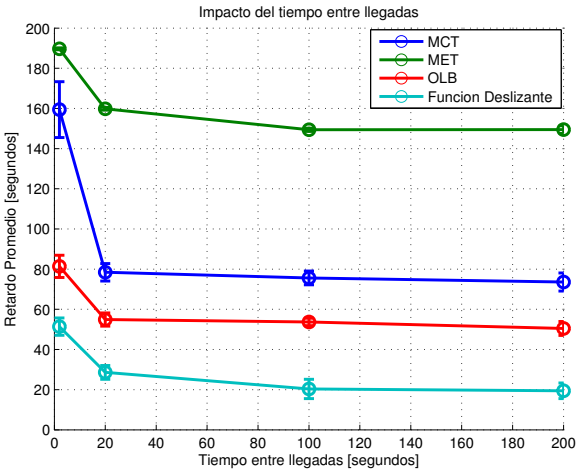

Fig. 4. Impacto del tiempo entre llegadas para retardo promedio

La Figura 4 muestra una situación similar para el caso del retardo promedio. El algoritmo propuesto entrega consistentemente retardos promedios más bajos que los otros algoritmos. Ambos comportamientos se deben a la capacidad del algoritmo propuesto de analizar un espacio de soluciones extendido. Más aún, este análisis se realiza de manera más minuciosa en una región del espacio de soluciones donde es altamente probable encontrar soluciones que no pueden ser arbitrariamente malas como lo podrían ser los mínimos locales en los que quedan atrapados los otros algoritmos.

Para completar este experimento, la Figura 5 muestra el porcentaje de utilización de la Grid alcanzado por los diferentes algoritmos. Como puede observarse en la figura, el algoritmo propuesto aprovecha de mejor manera los recursos de la Grid ya que mantiene ocupados a los procesadores que la componen la mayor parte del tiempo. Este resultado es consistente con los dos anteriores debido a que esta mejor utilización de los recursos le permiten al algoritmo propuesto entregar retardos promedios y máximos más bajos que sus competidores.

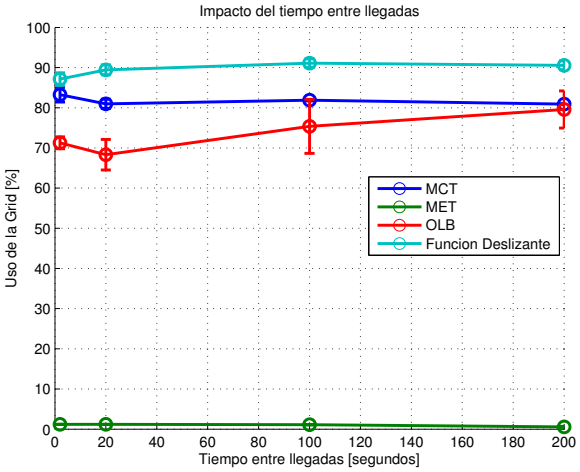

Fig. 5. Impacto del tiempo entre llegadas para uso de la grid

6.2.2. Impacto de la duración de los procesos

En el segundo experimento se analiza el impacto de la duración promedio de los procesos en el desempeño de la Grid. En este caso, mientras mayor sea el promedio de la duración del tiempo de atención, mayor será la carga de trabajo a la que es sometida la Grid. Como en el caso anterior, se generan 10 cargas aleatorias para cada valor de duración de los procesos, que son 20, 40, 80 y 160 segundos. En todos los casos el tiempo promedio entre llegadas fue 2 segundos y la probabilidad DAG fue 0.75. Los resultados se presentan en las Figuras 6, 7 y 8.

En la Figura 6 podemos observar que el retardo máximo experimentado por las tareas bajo los planes de los cuatro algoritmos se incrementa conforme se incrementa la duración promedio de los procesos individuales. Este comportamiento es el esperado debido a que mientras más largo sea el tiempo promedio de los procesos, mayor será el tiempo que le tome a la Grid completarlos.

De la Figura 6 también podemos observar que el algoritmo que presenta el mejor desempeño es el basado en distribuciones deslizantes, mientras que el peor algoritmo es MET. Lo anterior se debe a que MET tiende a concentrar la carga en los procesadores más veloces sin importar que tan saturados se encuentren sus planes de ejecución.

ISSN 2007-9737

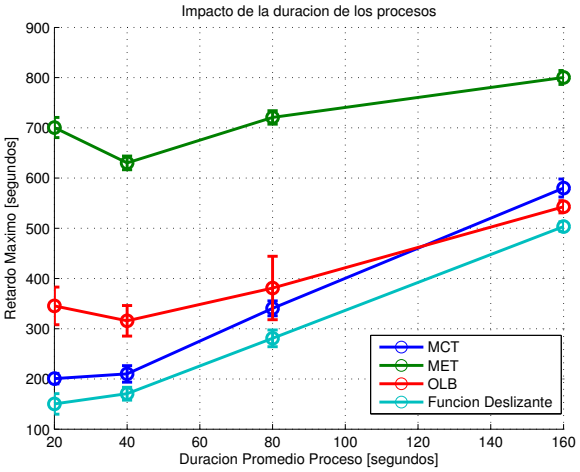

Fig. 6. Impacto duración de procesos para retardo máximo

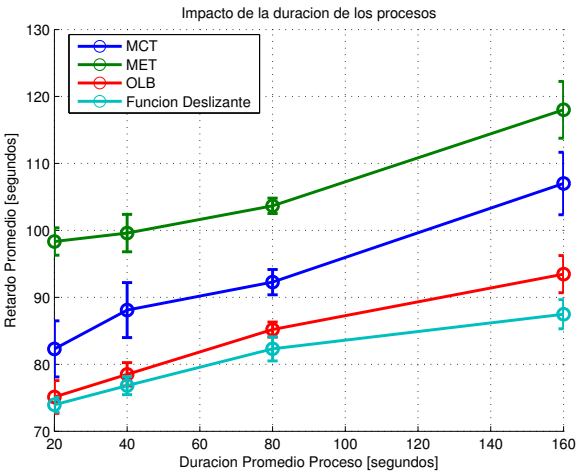

Fig. 7. Impacto duración de procesos para retardo promedio

La Figura 7 nos muestra un panorama similar para el retardo promedio en que las tareas son atendidas. Al igual que en el caso anterior, el algoritmo propuesto, basado en distribuciones deslizantes provee el mejor desempeño para todos los valores del promedio de duración de los procesos. Lo anterior confirma que la capacidad del algoritmo de trabajar en diferentes modos de operación le permite entregar buenas soluciones en una amplia gama de condiciones.

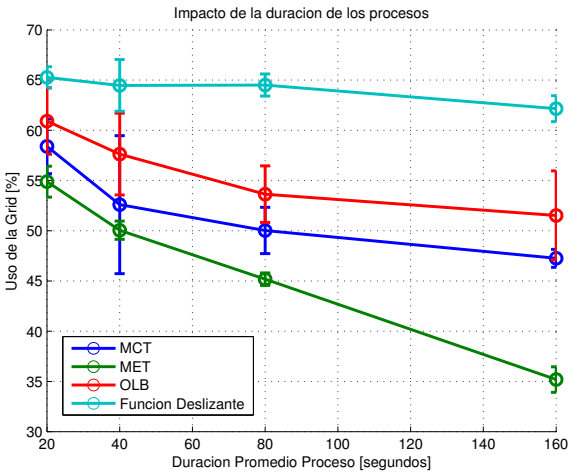

Fig. 8. Impacto de duración de procesos para uso de grid

Finalmente, la Figura 8 muestra la utilización de la Grid alcanzada por los diferentes algoritmos. A partir de la figura podemos observar que mientras más grande es el tiempo promedio que se necesita para atender a un proceso, también es más complicado encontrar un hueco en el plan de ejecución de un procesador que lo pueda contener. Lo anterior aumenta la fragmentación interna de los planes de ejecución, lo que tiende a reducir la utilización de los procesadores y por lo tanto de la Grid. Este efecto se acentúa para en el algoritmo MET ya que tiende utilizar a los procesadores más rápidos. Por su parte, el algoritmo propuesto logra mantener una utilización aceptable que siempre se mantiene por arriba del 60 %.

6.2.3. Impacto de la probabilidad de que las tareas estén compuestas por un grafo de procesos

En el tercer experimento se analiza el impacto de la probabilidad de que las tareas estén compuestas por un conjunto de procesos con relaciones de precedencia. Como en los casos anteriores, se generan 10 cargas aleatorias para cada valor de probabilidad, que son 0.25, 0.5, 0.75 y 1.0. Estas probabilidades son un indicador del

ISSN 2007-9737

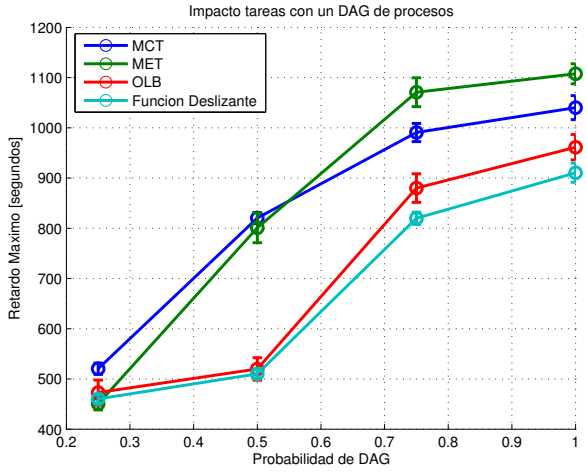

Fig. 9. Impacto probabilidad tareas DAG para retardo máximo

porcentaje de tareas que están compuestas por múltiples procesos, siendo el caso extremo cuando la probabilidad es uno en el que todas las tareas están compuestas por múltiples procesos interrelacionados. Para estos experimentos los valores del tiempo promedio entre llegadas y la duración promedio de los procesos es de 2 y 20 respectivamente. Los resultados se presentan en las Figuras 9, 10 y 11.

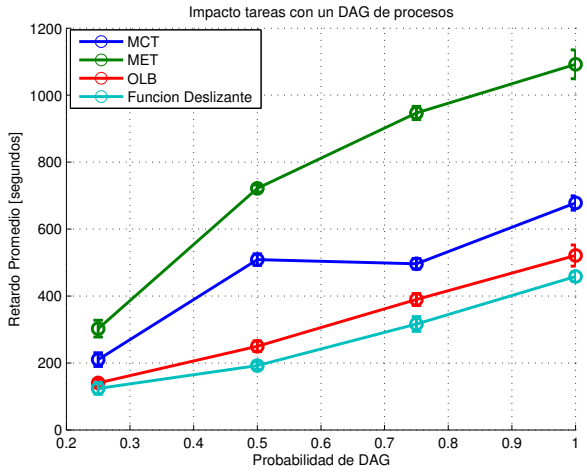

Fig. 10. Impacto probabilidad tareas DAG para retardo promedio

Como puede observarse en las tres figuras, conforme crece la proporción de tareas que están compuestas por conjuntos de procesos con relaciones de precedencia, aumenta también la dificultad de encontrar planes eficientes. Esto se debe principalmente a dos razones. La primera es que debido a que el número de tareas es fijo, mientras más tareas estén compuestas por conjuntos de procesos, la carga total a la que es sometida la Grid aumenta. La segunda es que las relaciones de precedencia actúan como restricciones extra en el problema de optimización, y por lo tanto tienden a reducir el espacio de soluciones viables. En términos concretos, estas restricciones imponen que un proceso sea planificado forzosamente después del tiempo de terminación de cualquier otro proceso que lo preceda.

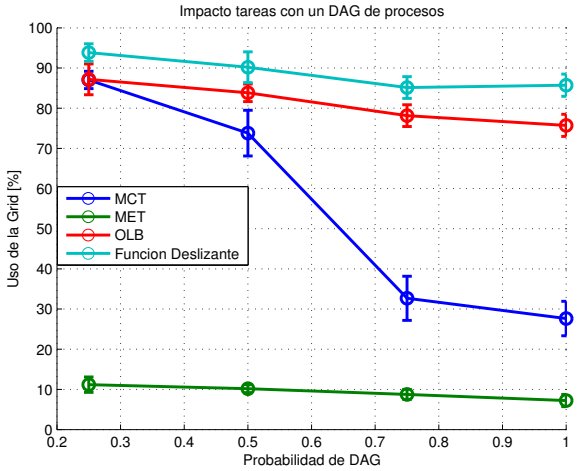

Fig. 11. Impacto probabilidad tareas DAG para uso de grid

Así, las Figuras 9 y 10 nos muestran que tanto el retardo máximo como el retardo promedio experimentado por las tareas se incrementa conforme aumenta la probabilidad de tener tareas compuestas. Sin embargo, en todos los casos el algoritmo propuesto entrega un desempeño superior al de OLB, MET, y MCT. De igual forma, el algoritmo propuesto logra la utilización de la Grid más alta en todos los casos (ver Figura 11). Lo anterior se debe nuevamente a que el algoritmo hace una búsqueda más detallada de la región del espa-

ISSN 2007-9737

cio de soluciones donde se encuentran soluciones que no pueden ser arbitrariamente malas. Un análisis más detallado del comportamiento de los algoritmos revela que el algoritmo propuesto reduce considerablemente la fragmentación interna de los planes individuales de los procesadores y por lo tanto reduce el tiempo que los procesadores permanecen inactivos.

6.2.4. Análisis de la probabilidad de no encontrar una ρ-aproximación

Para concluir la caracterización experimental del comportamiento del algoritmo propuesto, en la Figura12 se presentan los resultados de un experimento en el que calculamos la probabilidad de que en una de las $3n$ ejecuciones del algoritmo propuesto no se encuentre una ρ-aproximación. A este evento de "no encontrar una ρ-aproximación" lo hemos denominado como $error$. En la figura se muestran los valores promedio de esta probabilidad de $error$ para diferentes parámetros de la distribución de Gibbs-Boltzmann, en particular, se muestran las probabilidades de $error$ cuando $kT = \{10, 1, 0.1\}$.

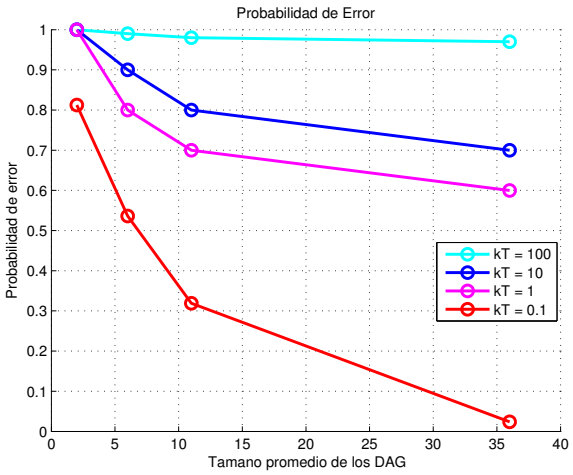

Fig. 12. Probabilidad de error de los algoritmos

Esta probabilidad de $error$ es una medida del porcentaje de soluciones que tienen una estructura similar a las soluciones generados por el algoritmo puramente voraz. Así, una probabilidad de $error$ alta indica que la mayoría de las soluciones encontradas por el algoritmo propuesto serán diferentes a las soluciones puramente voraces y cuando es baja que la mayoría de las soluciones serán parecidas a las soluciones voraces. De la Figura 12, podemos observar que para valores kT pequeños, el algoritmo se comporta como un algoritmo voraz, mientras que para valores altos el algoritmo se comporta de manera aleatoria. Estos resultados muestran gráficamente cómo los diferentes parámetros de la distribución de Gibbs-Boltzmann provocan un $deslizamiento$ en el comportamiento del algoritmo propuesto, de puramente aleatorio a puramente voraz.

7. Conclusiones

En este trabajo se presenta un nuevo algoritmo para el problema de planificación de tareas compuestas por procesos con restricciones de precedencia en ambientes distribuidos tipo Grid. Este problema es una generalización del problema de planificación multiprocesador, el cuál es bien sabido que pertenece a la clase de problemas NP-Difícil (*NP-Hard*).

Se propuso una nueva técnica para el diseño de algoritmos de optimización combinatoria qur por medio de la parametrización de la distribución de Gibbs-Boltzmann combina las propiedades de los algoritmos aleatorizados tipo Montecarlo con las propiedades de los algoritmos de ρ-aproximación. En el esquema propuesto, el algoritmo de optimización toma decisiones en base al muestreo de la distribución de Gibbs-Boltzmann la cual se "desliza" para que el algoritmo cambie su comportamiento de puramente aleatorio a completamente (*voraz*). De esta forma, el algoritmo es capaz de entregar soluciones ρ-aproximadas como el algoritmo netamente voraz, pero al mismo tiempo puede analizar un vecindario ampliado para escapar de máximos o mínimos locales. El costo de utilizar las distribuciones deslizantes es Θ(n) debido a que para planificar cada tarea se crean $3n$ planes, n totalmente aleatorios pero que respetan las restricciones de precedencia, n utilizando una distribución de probabilidad guiada por el algoritmo voraz, y n que con muy alta probabilidad generan una

solución tipo *voraz*. Adicionalmente, se presentaron una serie de teoremas que demuestran que el algoritmo propuesto es correcto y que caracterizan su complejidad temporal.

Finalmente, una serie de resultados experimentales muestran que el algoritmo propuesto basado en distribuciones deslizantes supera ampliamente el desempeño de una serie de algoritmos del estado del arte en algoritmos de planificación de modo inmediato. Las métricas utilizadas fueron retardo promedio, retardo máximo y utilización global de la Grid.

Agradecimientos

Este artículo está basado en trabajo realizado gracias al financiamiento del Instituto Politécnico Nacional, el Instituto México Estados Unidos de la Universidad de California (UC MEXUS) y el Consejo Nacional de Ciencia y Tecnología de México (CONACyT).

Referencias

1. Lingo section [online], http://www.lindo.com/.
2. The standard performance evaluation corporation (SPEC) [online], http://www.spec.org/.
3. **Abraham, A., Buyya, R., & Nath, B.** (**2000**). Nature's heuristics for scheduling jobs on computational grids. *The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000)*, pp. 45–52.
4. **Ambrust, M., Fox, A., Griffith, G., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M.** (**2010**). A view of cloud computing. *Communications of the ACM*, Vol. 53, No. 4, pp. 50–58.
5. **Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Agarwal, A., Metha, G., & Vahi, K.** (**2003**). The role of planning in grid computing. *ICAPS-03 Proceedings*, American Asociation for Artificial Intelligence, pp. 154–163.
6. **Braunt, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. A., Theys, M. D., Yao, B., Hensgen, D., & Freund, R. F.** (**2001**). A comparison study of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, Vol. 61, pp. 810–837.
7. **Deng, X. & Zhang, Y.** (**1999**). Minimizing mean response time in batch processing system. *The 5th Annual International Conference on Computing and Combinatorics*, pp. 231–240.
8. **Di Martino, V. & Mililotti, M.** (**2004**). Sub optimal scheduling in a grid using genetic algorithms. *Parallel Computing*, Vol. 30, No. 5, pp. 553–565.
9. **Dong, F. & Akl, S. G.** (**2006**). Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 504, Queen's University.
10. **Entezari, R. & Movaghar, A.** (**2012**). A probabilistic task scheduling method for grid environments. *Future Generation Computer Systems*, Vol. 28, pp. 513–524.
11. **Foster, I. & Kesselman, C.** *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.
12. **Foster, I. & Kesselman, C.** (**2002**). *What is the Grid? A Three Point Checklist*. Argonne National Laboratory, University of Chicago.
13. **Foster, I. & Kesselman, C.** (**2003**). *The grid 2: Blueprint for a new computing infrastructure*. Morgan Kaufmann.
14. **Gary, M. R. & Johnson, D. S.** (**1979**). *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York.
15. **Gil, Y., Deelman, E., Blythe, J., Kesselman, C., & Tangmunarunkit, H.** (**2004**). Artificial intelligence and grids: Workflow planning and beyond. *E-Science*, pp. 27–33.

16. **Goldberg, L. A., Paterson, M., Srinivasan, A., & Sweedyk, E.** (**1997**). Better approximation guarantees for job-shop scheduling. *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, pp. 599–608.

17. **Hoos, H. H. & Stützle, T.** (**2004**). *Stochastic local search: Foundations & applications*. Morgan Kaufmann.

18. **Kasahara, H. & Narita, S.** (**1994**). Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers*, Vol. 33, No. 11, pp. 1023–1029.

19. **Khan, A. A., L., M. C., & Jones, M. S.** (**1994**). A comparison of multiprocessor scheduling heuristics. *Proceedings of the International Conference on Parallel Processing*.

20. **Lenstra, J. K. & Kan, A. H. G. R.** (**1978**). Complexity of scheduling under precedence constraints. *Operations Research*, Vol. 26, No. 1, pp. 22–35.

21. **Lenstra, J. K., Shmoys, D. B., & Tardos, É.** (**1990**). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, Vol. 46, No. 1, pp. 259–271.

22. **Li, H. & Buyya, R.** (**2009**). Model-based simulation and performance evaluation of grid scheduling strategies. *Future Generation Computer Systems*, Vol. 25, pp. 460–465.

23. **Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E.** (**1953**). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, Vol. 21, No. 6, pp. 1087–1092.

24. **Mitzenmacher, M. & Upfal, E.** (**2005**). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge.

25. **Motwani, R. & Raghavan, P.** (**1995**). *Randomized algorithms*. Cambridge University Press.

26. **Prado, R. P., García-Galán, S., & Muñoz Expósito, J. E.** (**2011**). KASIA approach vs. differential evolution in fuzzy rule-based metaschedulers for grid computing. IEEE.

27. **Ritchie, G. & Levine, J.** (**2003**). *A fast, effective local search for scheduling independent jobs in heterogeneous computing environments*. Technical report, Centre for Intelligent Systems and their Applications, University of Edinburgh.

28. **Sahu, R. & Chaturvedi, A. K.** (**2011**). Many-objective comparison of twelve grid scheduling heuristics. *International Journal of Computer Applications*, Vol. 13, No. 6.

29. **Shmoys, D. B. & Tardos, É.** (**1993**). An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, Vol. 62, pp. 461–474.

30. **Tao, Y. & Gerasoulis, A.** (**1994**). DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 9, pp. 951–967.

31. **van Laarhoven, P. J. M., Aarts, H. L., & Karel, J.** (**1992**). Job shop scheduling by simulated annealing. *Operations Research*, Vol. 40, No. 1, pp. 113–125.

32. **Wang, L., Tao, J., Kunze, M., Castellanos, A., Kramer, D., & Karl, W.** (**2008**). Scientific cloud computing: Early definition and experience. *HPCC'08, 10th IEEE International Conference on High Performance Computing and Communications*, IEEE, pp. 825–830.

33. **Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., & Fu, C.** (**2010**). Cloud computing: a perspective study. *New Generation Computing*, Vol. 28, No. 2, pp. 137–146.

34. **Xian-He, S. & Ming, W.** (**2005**). GHS: A performance system of grid computing. *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, IEEE.

35. Zan, M., Guanwen, W., Yunhui, H., & Hongwei, L. (**2010**). Quantum genetic algorithm for scheduling jobs on computational grids. *2010 International Conference on Measuring Technology and Mecatronics Automation*, IEEE.

36. Zomaya, A. Y. & Teh, Y. H. (**2001**). Observations on using genetic algorithms for dynamic load-balancing. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 9, pp. 899–911.

Article received on 28/04/2014, accepted on 09/10/2014. Corresponding author is Hector J. Selley-Rojas.