INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

ARQUITECTURA MULTIAGENTE PARA EL MONITOREO Y GESTIÓN DE SISTEMAS GRID

TESIS

QUE PARA OBTENER EL GRADO DE
DOCTOR EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA
Héctor Julián Selley Rojas

Directores de Tesis:
Dr. Rolando Menchaca Méndez
Dr. Felipe Rolando Menchaca García

Resumen

El cómputo en grids está llamado a ser uno de los principales paradigmas para el desarrollo de sistemas y el análisis computacional de sistemas complejos debido a las grandes ventajas que ofrecen para el cómputo de alto rendimiento tales como interoperabilidad, expansibilidad y poder de cómputo descentralizado.

Existen supercomputadoras que brindan una gran capacidad de cómputo; sin embargo, su costo no es accesible para la mayoría de las instituciones y empresas. Debido a esto se ha popularizado el uso de grids, las cuales pueden operar con muchos equipos homogéneos o heterogéneos y además ofrecen una capacidad de cómputo de alto rendimiento.

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. 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 de dichas soluciones para escapar de máximos o mínimos locales.

En el trabajo 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 propuesto por medio de una serie de experimentos basados en simulaciones. Los experimentos muestran que el algoritmo propuesto logra en general un desempeño superior con respecto a los algoritmos que componen el estado del arte en algoritmos de planificación en sistemas tipo grid.

Abstract

En este trabajo de tesis se presenta un nuevo algoritmo 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 de dichas soluciones para escapar de máximos o mínimos locales. En este trabajo 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 propuesto por medio de una serie de experimentos basados en simulaciones. Los experimentos muestran que el algoritmo propuesto logra en general un desempeño superior con respecto a los algoritmos que componen el estado del arte en algoritmos de planificación en sistemas tipo grid. Las métricas de desempeño utilizadas son retardo promedio, retardo máximo y utilización de la grid.

Dedicatoria

A mis padres.

Agradecimientos

A mis directores de tesis Dr. Rolando Menchaca Méndez y Dr. Felipe Rolando Menchaca García, que sin su apoyo y guía no habría podido concluir este trabajo de investigación. Gracias a ambos, los admiro mucho.

Al Instituto Politécnico Nacional y al Centro de Investigación en Computación por haberme otorgado el honor de pertenecer a estas instituciones. Al Conacyt por haberme brindado el privilegio de gozar de una beca que sirvió de gran apoyo para la realización de mis estudios tanto de maestría como ahora el doctorado.

A mis padres, que gracias a los valores que me inculcaron desde niño pude concluir este gran reto, el mayor de mi vida. Gracias por todo su apoyo incondicional, son unos padres admirables por su entrega. Los quiero mucho.

A mis sobrinos Mateo, Julián y Victoria. Gracias a ustedes por darme una gran dosis de alegría inesperada y ser la inspiración que necesitaba en momentos poco agraciados. Ustedes son la razón principal por la cual vale la pena mirar adelante.

A mis hermanos Armando, César y Roxana. Sin su ejemplo de vida y apoyo jamás podría haber logrado esto. Los quiero a todos.

A mis amigos de toda la vida Daniel, Víctor, Adrián, Armando, el S y Paco. Sin su apoyo en los momentos difíciles y sin toda la alegría que me brindaron no habría sido capaz de terminar. Gracias en especial a Daniel por ser una gran figura de ejemplo y por persuadirme en aventurarme a este reto. Los quiero.

Índice general

Índice de figuras

Lista de Algoritmos

Capítulo 1

Introducción

1.1. Preámbulo

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 [13], el hardware como servicio (*Hardware as a Service, "HaaS"*)[50] y las infraestructuras como servicio ("*Infrastructure as a Service, IaaS"*) [51]. La figura 1.1 muestra estos paradigmas emergentes. 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 también 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[24]. 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 [25].

El MIT (Instituto de Tecnología de Massachusetts) en el año 2003 señaló a la Grid como una de las 10 tecnologías emergentes que cambiarán el mundo[7]. Desarrollos que afectarán dramáticamente la forma en que vivimos y trabajamos. En la década de los 80's los "protocolos de red" permitieron interconectar dos computadoras, y una vasta red de redes llamada Internet se esparció alrededor del mundo. En los

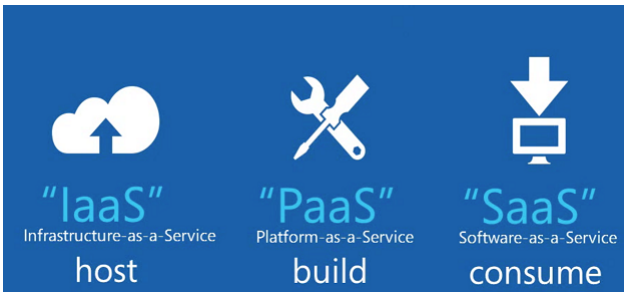

Figura 1.1: Paradigmas Emergentes de Cómputo

90's los "protocolos de transferencia de hipertextos" permitieron unificar cualquier par de documentos, y así surgió una vasta cantidad de bibliotecas, tiendas y demás servicios que juntos componen el "*World Wide Web*" a través de Internet. Hoy en día, los rápidamente emergentes "protocolos grid" permiten enlazar prácticamente cualquier cosa: bases de datos, herramientas de simulación y visualización incluso la misma capacidad de procesamiento de las computadoras en sí.

La promesa de los grid es ofrecer altos recursos de cómputo a un precio menor que una supercomputadora. Una gran virtud de la grid consiste en la inherente capacidad teórica de escalabilidad, dado que puede ser construida por computadoras con la misma cantidad de recursos o incluso distintos. Sin embargo, el que una grid sea altamente heterogénea implica una mayor complejidad técnica en el análisis de la información[12] o desempeño de la misma. Aunque la grid fue inicialmente concebida debido a la necesidad de una alternativa más económica que una supercomputadora, dada la inherente naturaleza escalable de la Grid, su capacidad puede extenderse en función de los recursos compartidos[22]. Sin embargo para lograr plenamente el objetivo de la escalabilidad, es fundamental que los algoritmos encargados de monitorear y administrar los recursos de la grid también lo sean.

Por otro lado, las características que hacen tan atractivos a los sistemas Grid como su capacidad de construirse por medio de componentes heterogéneos, su dinamicidad 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 [19].

Por otro lado, en la actualidad también existe el paradigma *cloud computing* en el cual se ofrecen servicios de computación a través de Internet. Esta es una tecnología naciente que busca ofrecer recursos orientados a servicios en Internet, donde los usuarios pueden acceder a una "nube" de servicios distribuidos sin tener conocimiento de la gestión de los recursos y naturalmente sin poseerlos. Así un usuario puede enviar un trabajo a ejecución en la nube y recibir los resultados sin ser dueño de ella.

Independientemente de si se trabaja en una infraestructura grid o en *cloud* es de gran interés otorgar un buen desempeño. Por lo tanto si implementamos una estrategia eficaz y eficiente para medir el desempeño de una grid, entonces podremos usar esta información para hacer una mejor administración de la grid y así mejorarlo.

1.2. Evaluación del Desempeño

Los usuarios de sistemas computacionales, administradores y diseñadores están interesados en la evaluación del desempeño dado que su meta es obtener o proporcionar el mayor desempeño al menor costo. Esta meta es el resultado de la continua evolución del desempeño y los bajos costos de las computadoras. Conforme la computación evoluciona y dado que la industria se ha vuelto muy competitiva, resulta muy importante asegurar que la alternativa utilizada proporcione la mejor relación costo-desempeño[31].

La evaluación del desempeño es requerida en cada etapa del ciclo de vida de un sistema computacional, incluyendo su diseño, producción, compra/venta, utilización, actualización solo por mencionar algunas. La evaluación del desempeño se requiere cuando un diseñador de sistemas computacionales desea comparar un conjunto de alternativas de diseño y elegir la mejor. También se requiere cuando un administrador de sistemas desea comparar un conjunto de sistemas y debe decidir cual es el más

adecuado para ejecutar un conjunto de tareas específicas. Aún si no hay alternativas, la evaluación del desempeño ayudará a saber cuan bien se están llevando a cabo las tareas y si es necesario hacer algún ajuste en el sistema en cuestión. Desafortunadamente la diversidad de aplicaciones computacionales es muy grande, por lo que no es posible tener un ambiente de evaluación estándar (una aplicación) o bien una técnica estándar que cubra todos los posibles casos. Por esta razón el primer paso consiste en seleccionar la evaluación adecuada del desempeño, la aplicación y las técnicas apropiadas[31].

De acuerdo al autor Raj Jain[31], los siguientes pasos deben llevarse a cabo para poder realizar una correcta evaluación del desempeño.

1. *Seleccionar técnicas apropiadas de evaluación, métricas del desempeño y las cargas de trabajo para el sistema.* Las métricas se refieren al criterio utilizado para determinar el desempeño del sistema.

2. *Realizar correctamente las mediciones del desempeño.* Esto se refiere a que para evaluar el desempeño del sistema se deben utilizar al menos dos herramientas de medición, una para generar carga y otra(s) para monitoreo de los resultados.

3. *Utilizar técnicas estadísticas apropiadas para comparar todas las alternativas.* No utilizar una técnica estadística apropiada puede conducir a conclusiones incorrectas.

4. *Diseñar las mediciones y simulaciones de manera que proporcionen más información con menos esfuerzo.* Dado un número de factores que afectan el desempeño del sistema, resulta útil separar los efectos directos de los factores individuales.

5. *Realizar las simulaciones correctamente.* Al diseñar un modelo de simulación se debe hacer una selección apropiada de lenguaje de simulación, semillas y algoritmos de números aleatorios, longitud de la simulación y análisis de los resultados.

6. *Utilizar modelos de colas para analizar el desempeño de los sistemas.* Los modelos de colas son utilizados comúnmente para modelación analítica de sistemas

computacionales.

Por todo esto y contrario al sentido común, la evaluación del desempeño es un arte[31]. Tal como en el arte, una correcta evaluación no puede realizarse mecánicamente. La evaluación requiere un conocimiento íntimo del sistema en cuestión y una correcta selección de la metodología, cargas de trabajo y herramientas de medición. Por esta razón, es posible que dado un mismo problema dos analistas seleccionen diferentes métricas y metodologías, incluso es posible que dado un conjunto de datos los dos analistas los interpreten de manera distinta. Esto ocurre debido a que cada analista basa su selección y conclusiones en la experiencia que ha adquirido al resolver problemas semejantes.

Existe una serie de pasos comunes para cualquier problema de evaluación del desempeño[31], los cuales son los siguientes:

1. *Metas y Definición del Sistema.* El primer paso en la evaluación del desempeño es definir las metas del estudio y definir lo que constituye el sistema, mediante la delimitación las fronteras del sistema.

2. *Lista de Servicios y Salidas.* Una lista de servicios y salidas resulta útil para la selección de las métricas adecuadas y cargas de trabajo.

3. *Selección de Métricas.* Son los criterios que se utilizan para comparar el desempeño del sistema.

4. *Lista de Parámetros.* Los parámetros son todos aquellos que afectan el desempeño, los cuales pueden dividirse en parámetros del sistema y cargas de trabajo.

5. *Seleccionar Factores a Estudiar.* La lista de parámetros puede ser dividida en dos, aquellos que cambiarán durante la evaluación y aquellos que no. Los parámetros que cambiarán se llaman *factores* y sus valores se llaman *niveles.*

6. *Selección de la técnica de Evaluación.* Las tres técnicas que existen para la evaluación del desempeño son: modelación analítica, simulación y medición del sistema real.

7. *Selección de la Carga de Trabajo*. La carga de trabajo consiste en una lista de peticiones de servicio al sistema.

8. *Diseño de los Experimentos*. La secuencia de los experimentos que ofrezcan mayor información con el menor esfuerzo.

9. Análisis e Interpretación de los Datos. La interpretación de los resultados del análisis es una parte crucial, y debe entenderse que el análisis solo produce resultados y no conclusiones.

10. *Presentar los Resultados*. Es importante que los resultados se presenten de una forma que sea fácil de entender, por lo que es común utilizar gráficas para esto.

En general se puede decir que el desempeño se puede medir mediante una serie de métricas que describen de forma cuantitativa el comportamiento de un sistema. Cabe destacar que las métricas que son importantes para un sistema en particular tienen una relación directa con el servicio que presta o bien el servicio que requiere el usuario final.

En la actualidad existen diversas herramientas que nos ayudan a determinar el desempeño de una computadora, servidor, red, algún hardware o inclusive un equipo de red como un ruteador. Dichas herramientas reciben el nombre de *benchmark*. En computación se entiende el término benchmark como una especificación estándar de un experimento que sirve para caracterizar el desempeño de un sistema de cómputo, o también se dice que es la acción de someter un sistema a una serie de pruebas con la finalidad de obtener resultados no disponibles en sistemas competitivos[31].

Por otro lado, es importante también especificar ciertas metas o expectativas que se esperan alcanzar cuando se definen las métricas que afectan al sistema en cuestión. Estas expectativas servirán como base para estimar un desempeño teórico que servirá como una referencia para los experimentos a realizar. Naturalmente estas dependen de la infraestructura local, orientación de los servicios que se ofrecen y de la capacidad de los equipos con los que se cuenta. Por ejemplo, de acuerdo a CISCO[16] las metas de desempeño mas importantes son:

- Tiempo de Respuesta
- Utilización
- Cantidad de Trabajo que fluye en el sistema (Throughput)
- Capacidad (Tasa máxima de cantidad de trabajo)

Mientras que estas mediciones podrían resultar triviales para una red LAN sencilla, puede resultar muy difícil en una red universitaria o empresarial, y en nuestro caso de interés: la grid. Si se tiene una infraestructura bien planeada, cada una de las metas se pueden definir de una forma que es posible medir. Por ejemplo, el tiempo de respuesta mínimo para una aplicación cualquiera es 500 milisegundos o menos durante las horas pico, esto define la información para identificar la variable, la forma en que se mide y el periodo en el cuál la aplicación será analizada.

En el contexto de esta tesis, es necesario desarrollar técnicas de medición de desempeño que sean capaces de mostrar fotografías instantáneas del estado actual de la grid. Esto debido a que los elementos de planificación basarán sus decisiones en dicha información.

1.3. Tecnologías de Cómputo Distribuido

La computación distribuida, es un modelo para resolver problemas de computación masiva utilizando un gran número de computadoras organizadas en subconjuntos en una infraestructura de telecomunicaciones distribuida.

Un sistema distribuido es un conjunto de computadoras independientes que aparentan ante los usuarios del sistema ser como una única computadora. El usuario accede a los recursos remotos de la misma manera en que accede a recursos locales, o un grupo de computadores que usan un software para conseguir un objetivo en común.[47]

Esta definición tiene dos aspectos. El primero se refiere al hardware: las máquinas son autónomas. El segundo se refiere al software: los usuarios piensan que el sistema

es como una única computadora.

Los sistemas distribuidos deben ser muy confiables, ya que si un componente del sistema falla otro componente deberá ser capaz de reemplazarlo, esto se denomina Tolerancia a Fallos.

Por otro lado el tamaño de un sistema distribuido puede ser muy variado, pueden ser decenas de hosts (red de área local), centenas de hosts (red de área metropolitana), miles y hasta millones de hosts (Internet) lo que se denomina escalabilidad.

Dentro del cómputo distribuido cabe hacer una mención con mayor detalle de lo que son cluster, grid y *cloud computing*.

1.3.1. Cluster

Un cluster es un tipo de sistema distribuido o paralelo que consiste en un conjunto de computadoras interconectadas que trabajan juntas como un único recurso integral de cómputo. [41]

Un nodo puede ser una sola computadora o un sistema de multiprocesamiento (múltiples computadoras, estaciones de trabajo o sistemas de procesamiento paralelo) con memoria, capacidades de Entrada/Salida y Sistema Operativo. Un cluster generalmente se refiere generalmente a dos o más computadoras conectadas entre sí. Los nodos pueden estar físicamente en un mismo gabinete o bien separados y conectados en red. Estos sistemas el usuario los percibe como un sistema único y proveen un mecanismo que otorga una mejor razón costo-eficiencia.

Los nodos del cluster pueden trabajar de forma colectiva como un recurso integral de cómputo o bien como computadoras individuales. El *middleware* es el responsable de ofrecer la ilusión de un sistema unificado y la disponibilidad del conjunto de computadoras independientes interconectadas.

Los cluster ofrecen las siguientes características a un bajo costo:

- Alto Desempeño
- Capacidad de Expansión y Escalabilidad
- Alto Rendimiento
- Alta Disponibilidad

La tecnología cluster permite que las organizaciones incrementen su poder de procesamiento utilizando tecnología estándar que pueden adquirir a un costo relativamente bajo.

1.3.2. Grid

El término para denotar un Grid de computación distribuida y el entorno de almacenamiento fue introducido en 1998 por Ian Foster y Carl Kesselman[23], junto con la definición principal de la misma:

"*Una grid computacional es una infraestructura de software y hardware que proporciona acceso confiable, consistente, adaptable y con bajo costo a capacidades de cómputo de alto rendimiento*"

El cómputo en grid es considerado como la piedra angular de la siguiente generación del cómputo distribuido que coordina los recursos compartidos a gran escala y resolución de problemas en organizaciones virtuales y multi-institucionales. Mediante Internet, la grid permite que la gente coopere y comparta recursos de cómputo a través de las fronteras corporativas, institucionales y geográficas sin perder la autonomía local.[9] La figura 1.2 muestra como una grid puede ser construida a partir de recursos heterogéneos.

El término hace referencia a la red de energía eléctrica: la capacidad de cómputo equivale a una toma de corriente y sin necesidad de instalar y mantener infraestructuras de TI complejas en todos los lugares que necesitan acceso a aplicaciones. Esta gran visión de una infraestructura global de TI virtual en este momento aún no se ha materializado en la forma en que los visionarios tenía en mente pero vemos cada

Figura 1.2: La capacidad de la Grid de construirse por medio de componentes heterogéneos

vez más de sus características en funcionamiento hoy.

Las principales fuerzas motrices, desde una perspectiva de negocio, detrás de la grid son sin duda, la simplificación y optimización de los recursos de TI. La simplificación surge del hecho de que los usuarios de red pueden concentrarse en sus aplicaciones de negocio en lugar de tener que mantener una compleja infraestructura de TI completa, se obtiene optimización de los centros de datos porque no tiene que ser de un tamaño máximo de la carga, pero inteligentemente se puede compartir la carga, esto se denomina como la informática bajo demanda. Además, una infraestructura de TI distribuida a nivel mundial es natural en muchas organizaciones muy grandes como IBM, Shell, Philips, la NASA, donde la necesidad de un cálculo grande por un trabajo puede presentarse en cualquier momento y lugar, y viajar libremente a través del mundo entero.

Uno de los ejemplos más conocidos de la red es el proyecto SETI @ Home, que utiliza las computadoras de escritorio de todo el mundo para ayudar en la búsqueda de inteligencia extraterrestre. Esta aplicación está todavía activa y es un buen ejemplo de una grid computacional. Grandes configuraciones Grid están formados por

las fuerzas combinadas de los centros de datos científicos y se está trabajando para proyectos en la astronomía y la física de alta energía, donde se generan grandes cantidades de datos. El proyecto BEinGRID ha demostrado, con sus 25 experimentos de negocios, que el concepto de grid es comercialmente viable también en menor escala, tanto para la informática y para las tareas de almacenamiento orientado. Algunas tecnologías que forman parte de la visión de grid son el Servicio de Orientación, el software como un servicio (SaaS) y Cloud Computing.

¿Qué es la computación grid y en qué se diferencia de otras formas de computación distribuida?. En este punto en la historia de las TI, hay muchas formas de la informática distribuida que tienen una gran similitud. Las tres características más importantes de la grid, según Ian Foster[24], son las siguientes:

1. Los nodos que componen la grid no se controlan de forma centralizada sino que tienen su propia gestión y propiedad.

2. El uso de normas abiertas, sin derechos de propiedad, las normas para el intercambio de información entre los nodos.

3. Permite la coordinación de diferentes calidades de servicio, de acuerdo a las necesidades de las aplicaciones que hacen uso de la grid.

Por otro lado, existen muchas y diferentes tipos de grid. Una grid representa un patrón arquitectónico que se puede utilizar en diferentes escalas. En la actualidad las Grid se crean a menudo sólo para una o un pequeño número de aplicaciones. No hay nada malo en esto ya que nos permite ganar experiencia en la aplicación del modelo. El hecho de que se tiene Grid, SOA (Arquitectura Orientada a Servicio), SaaS, IaaS (Infraestructura como un Servicio), PaaS (Plataforma como un Servicio) y Cloud Computing demuestra la importancia de los avances en la informática distribuida.

En este trabajo se toma la definición de grid Ian Foster y Carl Kesselman citada en el inicio de esta sección.

1.3.3. Cloud Computing

El paradigma *Cloud Computing* apunta a un futuro en el que no se realiza el procesamiento en computadoras locales, se realiza en los recursos ubicados dentro de las instalaciones de un tercero. Esta idea la tuvo en 1961 un visionario pionero del cómputo John McCarthy, quien predijo que *la computación algún día estaría organizada como un recurso público* y especuló en como ocurriría esto.

Un grupo de expertos Ya hizo un consenso en como definir Cloud[10], sin embargo Ian Foster[26] la define de la siguiente forma:

"Es el paradigma de cómputo distribuido a gran escala que es conducido por las ventajas del costo que se obtienen debido a la expansión, en el cual se tiene un conjunto de recursos abstractos, virtualizados, escalables dinámicamente, con poder de cómputo administrado, para almacenamiento, plataformas y servicios que son entregados a petición de los usuarios externos en Internet."

Existen algunos puntos clave en esta definición, en primer lugar Cloud Computing es un paradigma de cómputo distribuido especializado. Se diferencia de los tradicionales en que son escalables masivamente y pueden ser encapsulados como entidades abstractas que proporcionan diferentes niveles de servicios para los usuarios fuera de la nube. Es conducido por las ventajas del costo que se obtienen debido a la expansión y los servicios pueden ser dinámicamente configurados y entregados según se requieran.

La figura 1.3 muestra como el paradigma *cloud computing* ofrece recursos orientados servicios a través de Internet a diversos dispositivos de cómputo.

Las instituciones gubernamentales, institutos de investigación y los líderes de la industria están adoptando la Cloud para resolver su siempre creciente necesidad de procesamiento y almacenamiento que han surgido desde que apareció Internet.

Existen tres factores principales que contribuyen al interés en Cloud. En primer lugar los recursos de hardware cada vez son más baratos y el incremento en poder de cómputo y capacidad de almacenamiento, aunado a que las arquitecturas de múltiples

Figura 1.3: Cloud computing ofrece recursos orientados a servicios en Internet

núcleos y las supercomputadoras modernas que consisten en miles de núcleos. Luego, el crecimiento exponencial de los datos en los instrumentos y simulaciones científicas, así como en la publicación en Internet y almacenamiento de información. Finalmente la amplia adopción por los servicios de cómputo y aplicaciones Web 2.0.

1.4. Planificación

Por definición, un sistema distribuido consta de varios procesadores, éstos se pueden organizar como colección de estaciones de trabajo personales, una pila pública de recursos o alguna forma híbrida. En todos los casos, se necesita cierto algoritmo para decidir cuál proceso hay que ejecutar y en que recurso.

1.4.1. Modelos de Asignación

Las estrategias de asignación de recursos se pueden dividir en dos amplias categorías[47]. En la primera, que llamaremos **no migratoria**, al crearse un proceso, se toma una decisión acerca de dónde colocarlo. Una vez colocado en un recurso, el proceso permanece ahí hasta que termina. No se puede mover, no importa lo sobrecargado que esté el recurso ni que existan muchas otros recursos inactivos. Por el contrario, con los algoritmos **migratorios**, un proceso se puede trasladar aunque haya iniciado su ejecución. Mientras que las estrategias migratorias permiten un mejor balance de carga, son más complejas y tienen un efecto fundamental en el diseño del sistema. Este tipo de estrategias son las de interés y aplicación en una grid.

1.4.2. Algoritmos de Asignación

Con el paso de los años, se han propuesto un gran número de algoritmos para la asignación de recursos[47]. En esta sección analizaremos algunas de las opciones claves en estos algoritmos y señalaremos los distintos puntos intermedios. Las principales decisiones que deben tomar los diseñadores se pueden resumir en cinco aspectos:

1. Algoritmos deterministas vs. heurísticos.
2. Algoritmos centralizados vs. distribuidos.
3. Algoritmos óptimos vs. subóptimos.
4. Algoritmos locales vs. globales.
5. Algoritmos iniciados por el emisor vs. iniciados por el receptor.

Los algoritmos deterministas son adecuados cuando se sabe de antemano todo acerca del comportamiento de los procesos. En el otro extremo están los sistemas donde la carga es por completo impredecible. Las solicitudes de trabajo dependen de quién esté haciendo y qué, y puede variar de manera drástica cada hora, e incluso cada minuto. La asignación de recursos en tales sistemas no se puede hacer de manera determinista o matemática, sino que por necesidad utiliza técnicas *ad hoc*

llamadas **heurísticas**.

El segundo aspecto del diseño es centralizado vs. distribuido. La recolección de toda la información en un lugar permite tomar una mejor decisión, pero menos robusta y coloca una carga pesada en la máquina central. Son preferibles los algoritmos descentralizados, pero se han propuesto algunos algoritmos centralizados por la carencia de alternativas descentralizadas adecuadas.

El tercer aspecto está relacionado con los dos anteriores: ¿intentamos encontrar la mejor asignación, o sólo una que sea aceptable?. Se pueden obtener las soluciones óptimas tanto en los sistemas centralizados como en los descentralizados, pero por regla son más caros que los subóptimos. Hay que recolectar más información y procesarla un poco más.

El cuarto aspecto se relaciona con lo que se llama a menudo **política de transferencia**. Cuando se está a punto de crear un proceso, hay que tomar una decisión para ver si se ejecuta o no en la máquina que lo genera. Si esa máquina está muy ocupada, hay que transferir a otro lugar al nuevo proceso. La opción en este aspecto consiste en basar o no la decisión de transferencia por completo en la información local.

El último aspecto de la lista trata de la **política de localización**. Una vez que la política de transferencia ha decidido deshacerse de un proceso, la política de localización debe decidir a donde enviarlo. Es claro que esta política no puede ser local, necesita información de la carga en todas partes para tomar una decisión inteligente. Sin embargo, esta información se puede dispersar de dos maneras: en uno de los métodos, los emisores inician el intercambio de información, mientras que en el otro el receptor toma la iniciativa.

1.4.3. Flujos de Trabajo en Grid

Los flujos de trabajo en Grid pueden definirse como un conjunto de tareas atómicas a procesar en diversos recursos con cierto orden bien definido, con la finalidad de completar una meta grande y sofisticada.

En la actualidad, los Grafos Dirigidos sin Ciclos (DAG) han sido utilizados extensamente en la modelación de flujos de trabajo científico, especialmente en aplicaciones grid que requieren una gran cantidad de datos o procesamiento de cómputo[52][42], tales como física de alta energía, geofísica, astronomía, procesamiento de imágenes medicas y bio-informática.

La planificación de flujos de trabajo en Grid determina como serán distribuidas todas las tareas atómicas en un número de limitado de recursos de cómputo distribuido[54]. El problema puede ser descrito como las N tareas que son planificadas en M recursos de cómputo, sujeto a tres condiciones: la restricción de precedencia en la ejecución, solo una tarea puede ser ejecutada en un nodo a la vez y finalmente no existe una preferencia en la ejecución de las tareas. Este problema de planificación se ha mostrado que es NP completo[9].

El problema de la planificación de flujos de trabajo DAG ha sido estudiado extensamente y se han propuesto algunas heurísticas de planificación. Estas heurísticas pueden ser clasificadas[20][29] en algoritmos de planificación de listas[48], algoritmos de planificación de grupos[40] y algoritmos de cluster[37].

La planificación por listas es uno de los algoritmos más usados. En esta planificación se le asigna un peso a cada tarea y arista, de acuerdo a una lista de tareas en las que se asigna una prioridad a cada tarea. Luego, las tareas son seleccionadas en orden de acuerdo a su prioridad, y cada tarea seleccionada es planificada al nodo que minimice la función costo.

Un flujo de trabajo DAG puede ser representado por un grafo dirigido sin ciclos $G = (T, E)$ donde $T = \{t_1, t_2, ..., t_N\}$ es el conjunto de N tareas y E es el conjunto de

aristas indicando la dependencia y precedencia entre las tareas.

En un grafo, una tarea sin precedencia se llama una tarea de entrada, y una tarea sin sucesores es una tarea de salida. Si hay más de una tarea de entrada, puede añadirse una tarea con costo cero y las demás tareas de entrada pueden ser conectadas a esta con aristas que tengan costo cero.

De manera similar, si existen más de una tarea de salida se puede añadir una tarea con costo cero y que las demás tareas de salida se conecten a esta con aristas de costo cero. Esto asegura que haya solo una tarea de entrada y una de salida en un flujo de trabajos DAG.

1.5. Planteamiento del Problema

Para este trabajo se considera una grid heterogénea en el que se tiene control directo sobre la administración y los planificadores de tareas, y los nodos de la grid se encuentran en distintas ubicaciones geográficas, y son conectados a través de Internet. éste tipo de arquitectura posee diversas características que otorgan muchas ventajas, que van desde la ampliación dada la intrínseca capacidad de escalabilidad de la grid, almacenamiento de grandes cantidades de información distribuida en la grid y hasta la ejecución de algún trabajo requerido por un usuario para alguna aplicación particular.

En la infraestructura de la grid se tienen diversos equipos que realizan diferentes tareas de acuerdo a la arquitectura específica. Por otro lado, una de las características más importantes de la grid es que los sitios (llamados organizaciones virtuales) pueden encontrarse en ubicaciones remotas y compartir recursos de cómputo, evitando así que los equipos deban concentrarse en algún punto geográfico estratégico común. Esta cualidad no solo facilita la donación de recursos sino también la utilización, es decir, alguna oficina virtual puede tener acceso al poder de cómputo de la grid, incluso a recursos o sitios que se encuentren del otro lado del mundo, tan solo con una conexión a Internet. La grid de esta forma hereda la gran capacidad

global que posee el Internet mismo, la cual permite el acceso a información desde cualquier punto de la red, con la ligera variante que en este caso no se tiene acceso a información sino a recursos y poder de cómputo.

Ofrecer servicios a través de una grid resulta de vital importancia, esto debido a que se debe garantizar que los trabajos que los usuarios envíen a ejecución serán resueltos en tiempo y forma, y que el resultado debería entregarse en el tiempo que se le ha establecido al usuario. Esto de manera muy burda se podría definirse como el desempeño de la grid, desde el punto de vista del usuario.

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[23], 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-Hard [27], 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 del problema de la mochila (*Knapsack*) [44][53].

1.6. Objetivos

1.6.1. Objetivo General

- Diseñar, implementar y caracterizar una plataforma multiagente de monitoreo de sistemas Grid que soporte un algoritmo de planificación aleatorizado basado en estrategias ρ-aproximadas.

1.6.2. Objetivos Particulares

- Diseñar e implementar una arquitectura de monitoreo distribuido basada en agentes independientes.
- Diseñar y verificar un algoritmo de planificación 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.
- Desarrollar una plataforma experimental que sirva de base para comparar el desempeño de los diferentes algoritmos.
- Desarrollar una serie de experimentos basados en simulaciones para caracterizar tanto el desempeño del algoritmo propuesto como el de una serie de algoritmos que componen el estado del arte en planificación en sistemas Grid.

1.7. Justificación

Dada la imposibilidad práctica para encontrar soluciones óptimas al 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 [34][35]

[28] que garantizan que las soluciones encontradas no pueden estar arbitrariamente lejos del óptimo, y los basados en *heurísticas y metaheurísticas* como la presentada en [43] que está basada en búsqueda local [30], [18] y [55] que utilizan heurísticas basadas en poblaciones y [11] que utiliza metaheurísiticas híbridas.

La principal desventaja de los algoritmos basados en heurísticas y metaheurísiticas es que a pesar de que para algunas instancias pueden encontrar soluciones muy cercanas al óptimo, no proveen ningún tipo de garantía para instancias arbitrarias. 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[17][36].

En esta tesis 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 [38][39]. 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. En este trabajo se muestra que el costo que se paga con respecto al algoritmo de aproximación base es únicamente de orden $O(n)$.

Resolver el problema es importante debido a que cuando un usuario envía un trabajo a ejecución, es deseable anticipar el desempeño que experimenta un sitio en particular, para tomar en cuenta este parámetro como un factor decisivo en la asignación del recurso en cuestión. Si se brinda el poder de cómputo como un servicio al usuario, entonces resulta de vital importancia para el prestador del servicio que la petición sea atendida y resuelta con el mayor desempeño posible.

Por otro lado los servicios grid han cobrado gran importancia y hoy día son una

realidad. El MIT (Instituto de Tecnología de Massachusetts) en el año 2003 señaló a la Grid como una de las 10 tecnologías emergentes que cambiarán el mundo[7], desarrollos que afectarán dramáticamente la forma en que vivimos y trabajamos. Actualmente existen diversas aplicaciones científicas que resultan de gran trascendencia, entre ellas destacan SETI@Home, Einstein@Home y el LHC.

SETI@Home[5] es un experimento científico que utiliza ordenadores conectados a Internet para la búsqueda de inteligencia extraterrestre (en inglés SETI son las siglas de "Search for Extraterrestrial Intelligence"). Se puede participar ejecutando un programa libre que se descarga y analiza datos obtenidos por radio telescopios. Este programa se utiliza como un salva pantallas que aprovecha los tiempos de ocio de la computadora, ya que cuando se activa se comunica con la NASA y recibe trabajos a procesar.

Einstein@home[2] es un proyecto de computación distribuida desarrollado por Bruce Allen y su equipo. El proyecto ha sido diseñado para buscar ondas gravitacionales en los datos recogidos por los observatorios LIGO en Estados Unidos y GEO 600 en Alemania. De acuerdo con la Teoría General de la Relatividad este tipo de radiación debería ser emitido por estrellas extremadamente densas y en rápido movimiento de rotación, generalmente llamadas estrellas compactas, que "arrastran" el espacio-tiempo alrededor de ellas mientras rotan. El programa procesa los datos de los observatorios utilizando Transformadas Rápidas de Fourier o FFT, y los datos se procesan en los tiempos de ocio de las computadoras, tal y como lo hace el SETI@Home.

El LHC[3] (Gran Colisionador de Hadrones) es un acelerador y colisionador de partículas ubicado en la Organización Europea para la Investigación Nuclear (CERN), cerca de Ginebra, en la frontera franco-suiza. Fue diseñado para colisionar haces de hadrones, más exactamente de protones, de hasta 7 TeV de energía, siendo su propósito principal examinar la validez y límites del Modelo Estándar, el cual es actualmente el marco teórico de la física de partículas, del que se conoce su ruptura a niveles de energía altos. Dentro del colisionador dos haces de protones son acelerados

en sentidos opuestos hasta alcanzar el 99,99 % de la velocidad de la luz, y se los hace chocar entre sí produciendo altísimas energías (aunque a escalas subatómicas) que permitirían simular algunos eventos ocurridos inmediatamente después del big bang. Se espera que el proyecto genere 27 Terabytes de datos por día, esta gran cantidad de información a procesar requiere de una capacidad de procesamiento de cómputo sin precedentes, es justo donde la grid puede cubrir tal necesidad.

1.8. Conclusiones

En este capítulo se ha tratado de algunos aspectos teóricos e introductorios en el tema de la presente tesis. En primer lugar se habló de algunas características y beneficios de la Grid, la importancia que tienen en la actualidad y una predicción del futuro que les depara.

Luego se detalló las formas teóricas para la evaluación del desempeño, los pasos y aspectos más importantes que se deben considerar para realizarlo. Posterior a esto se describe la definición de las tecnologías de cómputo distribuido, dentro de las que destacan los cluster, grid y el paradigma cloud computing. Se detallan sus principales características, ventajas y desventajas.

Posteriormente se habla acerca de la planificación, algunos de los modelos y algoritmos de asignación y la definición formal de los flujos de trabajos.

Luego se habla acerca del planteamiento del problema, donde se detalla las características principales del problema y la forma en la que se pretende resolverlo. Esto lleva luego a la definición formal del problema.

Posteriormente se plantean los objetivos generales y particulares que se pretenden alcanzar al realizar esta tesis.

Finalmente en la justificación se describen los aspectos sociales y científicos por

los que es importante resolver este problema. Se habla también de la complejidad del problema, la propuesta de solución y del impacto que tendría el resolverlo.

Capítulo 2

Trabajo Relacionado en Planificación en Sistemas Grid

2.1. Introducción

A pesar de que la planificación de tareas en sistemas paralelos y distribuidos han sido intensamente estudiados, existen nuevos retos en los sistemas grid que aún lo hacen un tema interesante y muchos proyectos de investigación lo abordan[19].

En el presente capítulo se describen algunos de los trabajos relevantes que sirven como antecedente y base para este trabajo. En dichos trabajos se ha estudiado la planificación de tareas en sistemas distribuidos por lo que se describe la técnica utilizada por los autores de ellos haciendo una discusión acerca de sus virtudes y limitantes.

2.2. Algoritmo de aproximación para el problema de planificación generalizado

Los autores Shmoys y Tardos[46] plantean que el problema genérico de planificación puede ser visto como el problema de planificar máquinas paralelas con costos.

Cada trabajo será procesado por exactamente una máquina, el trabajo j a procesar en la máquina i requiere tiempo p_{ij} e incurre en un costo c_{ij}, cada máquina i se encuentra disponible por T_i unidades de tiempo y el objetivo es minimizar el costo total. Los autores proponen un algoritmo que dado un valor C pruebe que no existe un plan que cumpla con esa condición o bien que encuentre el plan que cueste máximo C donde cada máquina i es utilizada máximo $2T_i$ unidades de tiempo.

El resultado principal del artículo consiste en un algoritmo de tiempo polinomial en el que, dados ciertos valores para C y T encuentra un plan cuyo costo es máximo C y *makespan* máximo $2T$, siempre y cuando exista un plan de costo C y *makespan* T. Aunque existen diversos algoritmos que se basan en resolver relaciones lineales de un problema de optimización y luego redondear la solución a valores enteros cercanos, el autor propone una nueva técnica de redondeo.

El algoritmo que propone el autor es el siguiente:
Algoritmo:

1. Formar el grafo $B(x)$ con el costo en sus aristas.

2. Encontrar el costo mínimo entero M que coincida exactamente con todos los nodos en $B(x)$.

3. Para cada arista en M, planificar el trabajo j en la máquina i.

2.3. Algoritmos de planificación multiprocesadores para procesamiento paralelo eficiente

Los autores Kasahara y Narita[32] 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). 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. Esto permite resolver problemas de muy

gran escala que contengan cientos de tareas.

En la propuesta CP/MISF la tarea que tiene le 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 heurísticamente a los nodos generados durante la búsqueda y la segunda parte es la búsqueda de profundidad primero.

El nivel l_i de la tarea i se define como la ruta más larga desde el nodo final hasta la tarea i, esto es:

$$l_i \triangleq \max_k \sum_{j \in \pi_k} t_j$$

donde π_k corresponde a la k-ésima ruta de salida de N_i.

En la propuesta CP/MISF la tarea que tiene le mayor número de tareas sucesivas inmediatas se le asigna la mayor prioridad. El método CP/MISF consiste de los siguientes pasos:

1. Determinar el nivel l_i para cada tarea.

2. Construir la lista de prioridad en orden descendiente de l_i y el número de tareas sucesivas inmediatas.

3. Ejecutar la planificación de la lista de acuerdo en la lista de prioridades.

Para probar la efectividad de la heurística CP/MISF, los autores utilizaron 200 casos aleatorios con el número de tareas en el rango de 10 a 200. Se obtuvieron soluciones óptimas en el 67 % de los casos probados. Las soluciones aproximadas con error menor a 5 % fueron obtenidas para el 87 % de los casos y con error menor a 10 % el 98.5 % de los casos.

El método DF/IHS consiste de dos partes: la parte de pre-procesamiento para la asignación de prioridades heurísticamente a los nodos generados durante la búsqueda

y la segunda parte es la búsqueda de profundidad primero.

Los resultados experimentales demuestran que la heurística encuentra efectivamente un valor 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.

2.4. Comparación de heurísticas de planificación multiprocesador

Los autores Khan, McCreary y Jones[33] cuantifican en su trabajo las diferencias entre diversas heurísticas con respecto a varios criterios y también definen criterios para la clasificación de los DAG.

La clasificación sugerida para los algoritmos de planificación es:

1. *Heurística de ruta crítica.* Para los DAG's con aristas y vértices con peso, el peso de una ruta se determina mediante la suma de los pesos de los vértices y aristas en la ruta dada. Una ruta crítica es la ruta con el mayor peso desde el nodo origen hasta el final. Algunas técnicas tratan de reducir el tamaño de la ruta mas grande en el DAG quitando requisitos de comunicación y fusionando procesos adyacentes.

2. *Heurística de lista de planificación.* Estos algoritmos asignan prioridades a las tareas y las planifican de acuerdo a su prioridad. Extendiendo la heurística de lista de planificación en planificación clásica, estos algoritmos utilizan una heurística *greedy* y se planifican las tareas en cierto orden.

3. *Método de descomposición del grafo.* Basado en la teoría de descomposición de grafos, el método descompone el grafo en una jerarquía de sub-grafos representando la relación de independencia/precedencia en grupos de tareas. Los costos de comunicación y ejecución son aplicados al árbol para determinar el tamaño de la granularidad que resulte en la planificación más eficiente.

Para la clasificación de los grafos utilizan tres características:

1. La *granularidad* se define como el tamaño promedio por unidad secuencial de cómputo en el programa. Esto no considera el peso de las comunicaciones entre las tareas. La granularidad sirve para determinar el paralelismo posible. Cuando su valor es grande el retraso en la comunicación es pequeño y el nivel de paralelización es alto. Si el valor es pequeño los altos costos de comunicación rebasan la reducción de tiempo obtenida por la paralelización.

2. El grado de salida de un nodo es el número de aristas que salen del nodo. El *grado de salida de anclaje* es la moda de los grados de salida de los nodos. Este resulta una buena forma de medir el factor bifurcaciones en un programa.

3. El rango de peso de los nodos de un grafo es el intervalo desde el peso menor de los nodos hasta el mayor.

Los resultados de este trabajo muestran que la ruta crítica y el algoritmo de lista de planificación falla cuando el costo de la comunicación es muy alto con respecto al costo de ejecución.

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.

2.5. Método probabilístico de planificación de tareas para ambientes grid

Los autores Entezari y Movaghar[21] 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. 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 recursos. Esta matriz, llamada matriz de conectividad, muestra la topología del entrono de la grid. Adicionalmente la probabilidad de conexión entre administradores y recursos es representada por una columna de la matriz renglón para cada nodo en la grid.

2.5.1. El método propuesto

Para un entorno de grid se puede obtener una matriz que represente la conexión entre los administradores y recursos. Esta matriz, llamada matriz de conectividad, muestra la topología del entrono de la grid. Adicionalmente la probabilidad de conexión entre administradores y recursos puede ser representado por una columna de la matriz renglón para cada nodo en la grid.

La matriz de probabilidad del ambiente se puede obtener multiplicando la matriz renglón por el renglón correspondiente de la matriz de conectividad.

La matriz P especifica un diagrama de transición de estados (o grafo dirigido finito) donde el estado i del grafo representado por un vértice y las transiciones del estado i al estado j representada por las aristas con su respectiva probabilidad de transición p_{ij}. Basado en una definición una cadena de Markov de tiempo discreto en estado finito gráficamente se puede representar mediante un diagrama de transición de estados. Por lo tanto, la matriz P muestra la matriz de transición de un paso.

2.5.2. Evaluación del Desempeño

En esta sección se compara el desempeño del algoritmo propuesto con otros algoritmos similares con respecto a dos parámetros: tiempo de respuesta promedio y makespan total. Para esto se elijen 5 algoritmos que han sido utilizados como benchmark en diversos trabajos y se describe la comparación de varios parámetros. 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 son los siguientes: balanceo de carga inmediata (OLB), tiempo de ejecución mínimo (MET), tiempo de finalización mínimo (MCT), algoritmo de intercambio (SA) y algoritmo de mejor k-por ciento (kPB).

En los experimentos que realizan en este trabajo utilizan dos entornos diferentes de grid, para ambos casos se plantean las matrices correspondientes y se se calcula la inversa paramétrica de la matriz fundamental N y el problema de optimización NLP obtenido se resuelve en ambos casos utilizando el software LINGO[4]. Los resultados muestran que la planificación obtenida es mejor en todos los casos que los algoritmos benchmark utilizados como referencia, sin embargo solo para procesos Singleton y no consideran el caso de tareas que tengan un DAG de procesos.

2.6. Algoritmo de aproximación para la planificación multiproceso

El autor Van Laarhoven[49] describe un algoritmo de aproximación para el problema de encontrar el mínimo *makespan* de un *job shop*. El algoritmo está basado en aproximación combinatoria simulada, una generalización de una mejora de la aproximación por iteración hacia 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 asintóticamente converge 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 computacionales muestran que el algoritmo puede encontrar makespan más cortos que dos aproximaciones recientes que son más próximas al problema de planificación *job shop*. Esto es, sin embargo, a un costo mayor de tiempos de ejecución. El algoritmo muestra evidencia en los experimentos de funcionar desde

5 hasta 30 tareas que poseen a su vez de tantas operaciones (procesos) como maquinas en los mismos experimentos.

2.7. Conclusiones

En este capítulo se ha hablado acerca de los trabajos y los autores que han explorado el problema de la planificación de tareas en un sistema distribuido. Se han presentado diversas técnicas y estrategias que los autores han planteado para atacar el problema de la planificación obteniendo resultados que han aportado a la búsqueda de la solución de este problema.

Los autores Shmoys y Tardos[46] plantean que el problema de planificación como un problema de planificar máquinas paralelas con costos. Su principal resultado es el planteamiento de un algoritmo de tiempo polinomial.

Los autores Kasahara y Narita[32] proponen un algoritmo de aproximación con el objetivo de resolver problemas que contengan cientos de tareas. Este algoritmo encuentra una buena aproximación al plan óptimo pero con la limitante de que solo puede resolver una tarea compuesta por un DAG de procesos aunque el número de procesos puede ser potencialmente grande.

Los autores Khan, McCreary y Jones[33] definen criterios para la clasificación de los DAG. Los resultados de este trabajo muestran que la ruta crítica y el algoritmo de lista de planificación falla cuando el costo de la comunicación es muy alto con respecto al costo de ejecución y cuando la granularidad es baja en los planes obtenidos el tiempo paralelo ocasionalmente es mayor que el tiempo en serie.

Los autores Entezari y Movaghar[21] presentan un método de planificación probabilística de tareas para minimizar el tiempo de respuesta promedio en un ambiente grid. Sus resultados experimentales mejoran a los algoritmos que utilizan como referencia pero su propuesta solo planifica procesos Singleton y no considera tareas compuestas por un DAG de procesos.

Finalmente el autor Van Laarhoven[49] describe un algoritmo de aproximación para el problema de encontrar el mínimo *makespan* de un *job shop*. Sus resultados muestran evidencia de funcionar pero solo desde 5 hasta 30 tareas que poseen hasta 9 operaciones cada una de ellas.

En los trabajos analizados se describen diversas propuestas para resolver el problema de planificación en sistemas distribuidos. En algunos casos solo se han planificado tareas que consisten de un proceso y en otras DAG's de procesos. Por los resultados y entornos experimentales utilizados por los autores es notorio que el problema de planificar DAG's de procesos no ha sido resuelto, debido a que las tareas que utilizan son grafos con pocos procesos. Es claro que si aumenta la complejidad del grafo que modela una tarea, aumentará también la dificultad en la planificación de esa tarea en un sistema distribuido heterogéneo.

Finalmente entre las estrategias usadas a la fecha, podemos mencionar.

- **Algoritmos de ρ-aproximación**: Se garantiza que las soluciones encontradas no estén arbitrariamente lejos del óptimo.
- **Heurísticas y metaheurísticas**: Encuentran soluciones muy cercanas al óptimo para algunas instancias pero no hay garantía en instancias arbitrarias.

Capítulo 3

Marco de Trabajo Experimental

3.1. Introducción

Según R.E. Shannon[45] "La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias -dentro de los límites impuestos por un cierto criterio o un conjunto de ellos- para el funcionamiento del sistema". La simulación en las ciencias de la computación representa un marco de trabajo controlado donde es posible llevar a cabo la etapa de experimentación de los algoritmos que el método científico requiere.

En este capítulo se habla desde los fundamentos básicos de la simulación, diferentes técnicas y análisis estadísticos que permiten estudiar los resultados de una simulación.

3.2. Fundamentos de la Simulación

Existen modelos que pueden ser resueltos mediante métodos analíticos o numéricos. A pesar de que la clase de modelos que existen es muy amplia, aún hay modelos que no pueden ser resueltos adecuadamente con las técnicas presentadas. Estos modelos, sin embargo, pueden ser analizados mediante la simulación. Con la simulación no existen restricciones fundamentales en lo que los modelos pueden resolver. Sin

embargo existen restricciones practicas debido a que la cantidad de tiempo o memoria de una computadora necesarios para ejecutar una simulación son limitados.

En esta sección se habla un poco acerca de la configuración general de las simulaciones así como de las técnicas estadísticas para el estudio de las simulaciones.

3.2.1. La idea de la simulación

Para comparar los conceptos de simulación, y las técnicas analíticas y numéricas se discute la simulación de la resolución de una integral[14]. Se requiere calcular el área bajo la curva de $f(x) = x^2$, en el intervalo $x \in [0, 1]$. Analíticamente el resultado es el siguiente:

$$A = \int_0^1 x^2 \, dx = (\frac{1}{3}x^3)_0^1 = \frac{1}{3}$$

Sin embargo, si fuera el caso de la integral con $f(x) = x^{sen\, x}$, entonces esta ya no puede ser resuelta de forma analítica aunque si puede ser aproximada numéricamente por el método trapezoidal por ejemplo. Para este método se debe dividir el intervalo en n subintervalos consecutivo, y si se hacen lo suficientemente pequeños se aproximará al valor de A con un buen nivel de precisión. Este es un ejemplo de una técnica de solución numérica.

De igual forma, también se puede obtener una aproximación razonable de A mediante simulación estocástica. Analizando un poco la función en el intervalo dado se puede saber que $0 \leq f(x) \leq 1$. Así, se toman dos muestras aleatorias x_i y y_i de la distribución uniforme en $[0, 1]$, lo que equivale a tomar un valor en el área misma que se desea evaluar. Repitiendo esta operación N veces se puede tener una buena *estimación* a al valor del área deseado. La variable aleatoria $\hat{A}$ es llamada un *estimador*, mientras que $\hat{a}$ es llamado un *estimado*. Además la variable aleatoria $\hat{A}$ debe ser definida de forma que obedezca algunas propiedades, de manera que la *estimación* $\hat{a}$ pueda ser precisa.

- $\hat{A}$ debe ser imparcial, es decir $E[\hat{A}] = a$.

- $\hat{A}$ debe ser consistente, esto es que mientras más muestras se tomen más preciso será.

Cabe remarcar que si existe una solución analítica para algún problema en particular, esta será mucho más precisa que cualquier resultado numérico obtenido mediante una simulación. Los resultados de una simulación solamente dan información acerca de una solución particular de un problema, y no cubren todo el rango de posibles soluciones. Esta situación permite percibir la sensibilidad que tiene la solución en función de uno o más parámetros del modelo.

3.2.2. Clasificación de las Simulaciones

Se pueden clasificar las simulaciones de acuerdo a dos criterios: espacio de estados y evolución en el tiempo[14]. Estos criterios determinan las dos categorías de las simulaciones de eventos: *continuos* y *discretos*.

En la *simulación de eventos continuos* los sistemas son estudiados en la forma en que los estados cambian con respecto al tiempo. Estos sistemas típicamente son procesos físicos que pueden ser descritos mediante sistemas de ecuaciones diferenciales con condiciones de frontera, donde la solución numérica de este sistema es algunas veces llamada una simulación. En los sistemas físicos el tiempo es un parámetro continuo, aunque los sistemas solo se pueden observar en un tiempo finito predefinido.

Por otro lado, en las *simulaciones de eventos discretos* (DES) los cambios de estado ocurren en puntos discretos de tiempo. Nuevamente se puede considerar el tiempo como un parámetro continuo o discreto, dependiendo de la aplicación alguno de ellos será más apropiado que el otro.

La figura 3.1 muestra la clasificación de las simulaciones descritas anteriormente.

$$\text{Simulación} \begin{cases} \text{Eventos Discretos} \begin{cases} \text{Basados en Eventos} \begin{cases} \text{Orientados a Eventos} \\ \text{Orientados a Procesos} \end{cases} \\ \text{Basados en Tiempo} \end{cases} \\ \text{Eventos Continuos} \end{cases}$$

Figura 3.1 Clasificación de las Simulaciones

En un sistema de eventos discretos, los estados cambian con respecto al tiempo. La causa del cambio en la variable se llama *evento*. En las simulaciones discretas se salta de un evento a otro y es justo el orden de los eventos y el tiempo en el que ocurren son lo que resultan de interés debido a que esto describe con exactitud el desempeño del sistema simulado. Haciendo un seguimiento de esos eventos y su tiempo de ocurrencia, es posible realizar mediciones como el tiempo promedio de llegadas o bien el tiempo exacto entre dos pares de eventos específicos.

En la *simulación basada en tiempo*, también llamada *simulación asíncrona* el ciclo de control principal de la simulación controla el progreso del tiempo t en transiciones constantes. Al inicio de ese ciclo de control, el tiempo se incrementa Δt por igual en cada transición donde este incremento es pequeño. Luego se debe observar si ocurre algún evento dentro de ese intervalo de tiempo, y si es ejecutado. Esto quiere decir que el estado cambiará de acuerdo a la ocurrencia de los eventos, antes de que el siguiente ciclo comience.

La simulación basada en tiempo es fácil de implementar. La implementación consiste en métodos numéricos para resolver ecuaciones diferenciales, sin embargo existen algunas desventajas asociadas con este método. Una de ellas consiste en suponer que el orden de ocurrencia de los eventos en los intervalos Δt no es importante y para suponer que los eventos son independientes entre si requiere que los intervalos Δt sean muy pequeños, con la finalidad de minimizar la probabilidad de que exista dependencia mutua entre los eventos. Por esta razón si se toman valores de Δt muy pequeños entonces la simulación se vuelve muy eficiente.

En la *simulación basada en eventos* (también llamada *simulación asíncrona*) ocurre lo contrario a la basada en tiempo. Para este tipo de simulación se tienen transiciones de tiempo variable, de manera que hay exactamente un solo evento en cada transición. De esta forma la simulación es controlada por la ocurrencia de los eventos siguientes. Esto resulta muy eficiente dado que las transiciones son lo suficientemente largas para continuar con la simulación, y a su vez son lo suficientemente cortas como para excluir la posibilidad de tener más de un evento por transición.

Esto soluciona el problema de tener eventos dependientes en una sola transición.

Considérese por ejemplo la llegada de un trabajo a una cola. Este evento ocasiona que el estado del sistema cambie, pero también ocasionará al menos un evento en el futuro. Todos los eventos futuros son generalmente almacenados en una lista de eventos ordenados. El primer evento en la lista es el siguiente a ocurrir el tiempo en que se llevará a cabo. La cola de esta lista contiene los eventos futuros y el orden de ocurrencia. Cuando la simulación se lleva a cabo nuevos eventos pueden ser creados, estos eventos son insertados en la lista de eventos en lugares apropiados. Luego de esto, la nueva cabeza de la lista de eventos es procesada.

La mayoría de las simulaciones de eventos discretos que se hacen en el campo de la evaluación del desempeño en computación y comunicaciones son del tipo basada en eventos. La única limitante en este tipo de simulación es que debe ser posible computar las instancias de tiempo en las que los eventos futuros ocurrirán.

3.2.3. Análisis estadístico de los Resultados de la Simulación

Cuando se ejecuta una simulación se puede poner etiquetas de tiempo a eventos relevantes[14]. Las muestras resultantes pueden ser almacenadas en un archivo bitácora. A pesar de que un archivo bitácora contiene toda la información disponible, generalmente resulta poco práctico dado que resulta muy difícil poder leer o bien interpretar algo a partir de una gran cantidad de datos. Por esta razón se debe buscar alguna técnica estadística que permita que la información sea interpretable por una persona.

Muestreo de Eventos Individuales.

Se pueden clasificar dos tipos de mediciones que pueden ser obtenidas a partir de una simulación:*orientadas a usuario* y *orientadas a sistema*. Las mediciones orientadas a usuario son típicamente obtenidas monitoreando a usuarios específicos del sistema en cuestión. Un ejemplo de medición orientada a usuario es el tiempo de estancia de un trabajo en alguna parte específica del sistema. Cuando el i-ésimo

trabajo llega a esa parte del sistema se toma una muestra de tiempo t_i^a y cuando el trabajo deja esa parte del sistema se toma la muestra t_i^d. La diferencia $t_i = t_i^d - t_i^a$ es el tiempo de estancia del trabajo. El promedio de estos tiempos es una estimación del tiempo promedio de estancia de los trabajos.

Por otro lado, en las mediciones orientadas a sistema no se monitorean los trabajos individualmente, en su lugar se monitorea el estado mismo del sistema. Un ejemplo típico es el caso donde la medida de interés es la probabilidad a largo plazo de que un buffer finito se llene. Cada vez que el estado del modelo del sistema cambie, se verificará si esta condición ocurrió o no. Si la condición es verdadera, se toma una marca de tiempo t_i^f. La siguiente marca de tiempo t_i^n se toma cuando el buffer ya no está completamente ocupado. La diferencia $t_i^n - t_i^f$ es la realización de una variable aleatoria que podría ser llamada periodo de buffer lleno. El promedio de estos tiempos es una estimación de la probabilidad a largo plazo de buffer lleno.

Valores Promedio e Intervalos de Confianza

Supongamos que se está ejecutando una simulación para estimar el promedio de la variable aleatoria X con esperanza desconocida $E[X] = a$. La simulación genera n muestras x_i, $i = 1, 2, ..., n$, donde cada una puede interpretarse como la realización de la variable estocástica X_i. La simulación ha sido construida de forma que las variable estocásticas X_i están todas distribuidas con respecto a la variable aleatoria X. Además, para poder calcular los intervalos de confianza, es necesario que todas las variables X_i sean independientes entre si.

Valor Promedio

Para estimar la esperanza $E[X]$ se debe definir una variable estocástica $\hat{X}$, la cual es un estimador de X. Para todos los valores de $E[\hat{X} = a$, el estimador $\hat{X}$ es *imparcial*. Para todos los valores $Pr\{|\hat{X} - a| < \epsilon\} \to 0$ cuando $n \to \infty$, el estimador $\hat{X}$ es *consistente*. Esta última condición requiere que la varianza sea muy pequeña si el número de muestras es muy grande. Siempre que las observaciones $X_1, X_2, ..., X_n$

sean independientes.

$$\hat{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

$\hat{X}$ es un estimador imparcial y consistente de $E[X]$ dado que $E[\hat{X}] = E[X]$ y $var[\hat{X}] = var[X]/n$, dado que la varianza $var[\hat{X}] \to 0$, cuando $n \to 0$.

Garantía de Independencia

Existen diversos métodos para obtener observaciones lo suficientemente independientes de las simulaciones, que son necesarios para construir los intervalos de confianza. Los tres métodos más usados son los siguientes.

- **Replicas Independientes**. La ejecución de la simulación es replicada n veces, y en cada iteración se tiene una semilla distinta para la generación de un número aleatorio. En la i-ésima ejecución de la simulación se toman las muestras $x_{i,1}, x_{i,2}, ..., x_{i,m}$, y a pesar de que las muestras no son independientes entre sí, los promedios

  $$x_i = (\sum_{j=1}^{m} x_{ij})/m, i = 1, 2, ..., n$$

  son considerados independientes entre sí.

- **Conjunto de promedios**. Este método requiere que una sola ejecución de la simulación, de la cual se toman las muestras $x_1, x_2, ..., x_{n*m}$ y son separadas en n conjuntos de tamaño m. Dentro de cada conjunto las muestras son promediadas de la siguiente forma:

  $$y_i = \frac{1}{m}\sum_{j=1}^{m} x_{(i-1)m+j}$$

  Se asume que las muestras $y_1, y_2, ..., y_n$ son independientes y se utilizan para

calcular el promedio total y los intervalos de confianza.

- **Regenerativo**. En este método se realiza una sola simulación y se dividen las muestras en conjuntos, tal como el anterior. En este caso la división se hace en puntos regenerativos. Los puntos regenerativos se definen de manera tal que el comportamiento antes del punto sea completamente independiente al comportamiento después. La principal ventaja de este método es que las muestras son realmente independientes entre sí.

Intervalos de Confianza

Dado que se asume que las variables X_i son independientes e idénticamente distribuidas, el estimador $\hat{X}$ se aproximará a la distribución Normal con una media a y varianza σ^2/n. Esto implica que la variable aleatoria

$$Z' = \frac{\hat{X} - a}{\sigma/\sqrt{n}}$$

está distribuida de forma Normal $N(0,1)$. Se utiliza la distribución *t-Student* que con grado tres o más, es una forma de campana simétrica muy similar a la Normal. Utilizando una tabla estándar para esta distribución se puede encontrar el valor de $z > 0$ tal que $Pr\{|Z| \leq z\} = \beta$. Usando valores críticos en ambos extremos se puede escribir la siguiente expresión:

$$Pr\{|Z| \leq z\} = Pr\left\{\frac{|\hat{X} - a|}{\sigma/\sqrt{n}} \leq z\right\} = Pr\{|\hat{X} - a| \leq z\sigma/\sqrt{n}\} = \beta$$

en la que la probabilidad del estimador $\hat{X}$ que solo varía $\sigma/\sqrt{n}$ del valor medio a, es β. En otras palabras, la probabilidad de que X se encuentre dentro del *intervalo de confianza* $[a - z\sigma/\sqrt{n}, a + z\sigma/\sqrt{n}]$ es β, la que es llamada nivel de confianza. Para los casos prácticos es recomendable que se tengan al menos 10 grados de libertad, es decir $n > 10$.

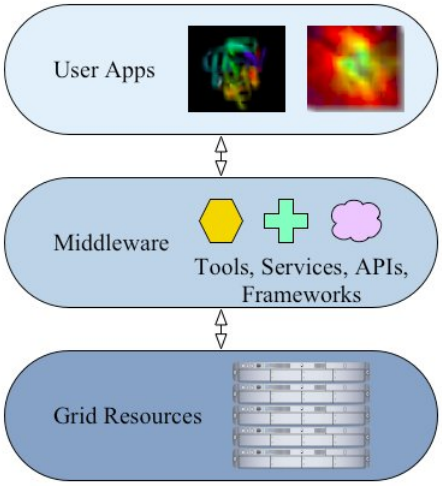

Figura 3.1: El middleware gLite como una capa de servicios

3.3. Entorno Experimental

Un algoritmo de planificación como el que se propone en este trabajo de tesis, cuando se encuentra implementado en una grid real, funciona en el nivel de middleware. Por esta razón es que a continuación se describe el middleware de grid gLite[8] que es utilizado por diversas grids en la actualidad tales como EELA-2[1] y CERN LHC[8][3] entre otras.

3.3.1. gLite

El middleware grid es la capa que permite la agregación, compartimiento y gestión de recursos distribuidos de forma transparente. Esta capa que alcanza los objetivos de un sistema grid: integrar, virtualizar y gestionar los recursos de cómputo y servicios disponibles por las múltiples organizaciones virtuales. En otras palabras el middleware gLite es la capa entre aplicaciones de usuarios y recursos computacionales y de almacenamiento.

La figura 3.1 muestra un esquema en el que se observa como es que la capa middleware gLite se ubica entre el usuario y los recursos de cómputo.

El middleware gLite es ligero y se considera de última generación para la computación en los sistemas distribuidos grid. Esta infraestructura está basada en una arquitectura orientada a servicios (SOA), que son la principal funcionalidad que reciben los usuarios al acceder a ella. La infraestructura nace de los esfuerzos colaborativos de instituciones de investigación académicas e industriales parte del proyecto EGEE (Habilitar Grids para la E-ciencia, proyecto europeo que pretende implantar una grid colaborativa).

Cuando algunos usuarios desean unirse a una Oficina Virtual (VO), deberán a su vez contribuir con recursos y posteriormente se negocian los accesos. De esta forma, todos los servicios adicionales (de personas y del middleware) potencian el grid, por lo que el resultado y meta de un proyecto de esta naturaleza es en términos generales único y de beneficio para todos los participantes: Colaboración.

El middleware grid gLite permite el acceso y uso compartido de los recursos de cómputo:

- Elementos de almacenamiento (SE).
- Elementos de computo (CE).

La figura 3.2 muestra un diagrama de como es que se realiza la interconexión de los recursos de diferentes organizaciones virtuales.

Arquitectura

En la arquitectura gLite se requiere que los equipos de cómputo actúen de diversas formas para poder tener un control e integración con la infraestructura de la grid. Cada uno de esos equipos tiene una tarea especial y a continuación se describen cada una de ellas.

- La **Interfaz de Usuario (UI)** es el punto de entrada al grid, es parte de la estación de trabajo del usuario. Se considera parte del WMS.
- El **Sistema de Administración de Carga de Trabajo (WMS)** es un conjunto de componentes cuyo objetivo es encontrar el recurso que cumple con

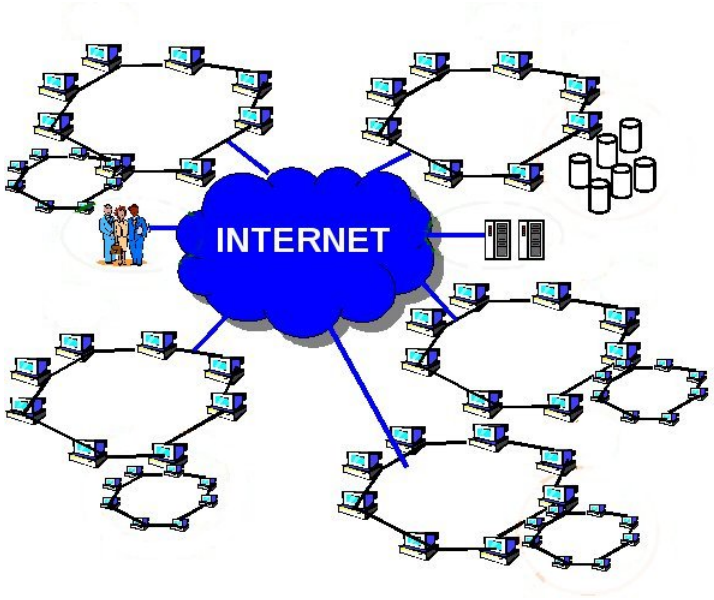

Figura 3.2: Diagrama de Interconexión de VO's

los requerimientos de un trabajo de un usuario entre los CEs disponibles. Esto es, encontrar la máquina donde finalmente el trabajo será ejecutado.

- El **Elemento de Cómputo (CE)** es el punto de entrada al sistema de colas local (PBS,LSF, CONDOR)

- Los **Nodos Trabajadores (WN)** son las máquinas donde los trabajos son realmente ejecutados. Están enlazados con el CE a través del sistema de colas local, al cual los trabajos son enviados.

- El **Sistema de Información y Monitoreo (IS y MON)**, mantienen los datos sobre los recursos disponibles y el estado del sistema.

- El **Servicio de Ingreso y Registro (LB)**, hace seguimiento a los eventos que le ocurren a los trabajos.

- El **Servicio de Administración de las Organizaciones Virtuales (VOMS)**, provee mecanismos de autenticación y autorización en el acceso a los recursos.

- El **Elemento de Almacenamiento (SE)** y **Catálogo de Archivos (LFC)**, permiten gestionar transferencias de archivos grandes o facilitar la disponibilidad y localización de archivos de datos que los trabajos requieren.

- El **Repositorio de Información de la Base de Datos Berkeley (BDII)** colecciona toda la información relacionada a los recursos de cómputo del sitio en el que se encuentra, el BDII es quien sabe con cuantos recursos se cuenta y como están siendo utilizados. Así como cada sitio tiene su BDII existe otro BDII jerárquicamente encima que recauda la información de todos, de manera que éste conoce los recursos de todos los sitios de la grid.

Servicios de gLite

Existen dos formas de acceder a los servicios que ofrece gLite: por línea de comandos o mediante una API generada por los web services. En cualquiera de los casos, una vez que la oficina virtual accede a la grid tiene a su disposición cinco servicios de alto nivel, los cuales son los siguientes:

- **Servicio de Asistencia.** Este servicio contiene los componentes de Servicio de Instrumentación y Control, Servicio de Convenio y el Servicio de Reservación y Asignación de Ancho de Banda. Estos servicios consultan el estado de los recursos, implanta el protocolo de comunicación SLA (Servicio de Nivel de Convenio) y Controla, Balancea y gestiona flujos de red respectivamente.

- **Servicio de Seguridad.** Dentro de estos servicios se encuentra la Autorización, Autenticación, Auditoría y Conectividad Dinámica. Brinda principalmente la capacidad de permitir el acceso a la grid a las organizaciones virtuales que hayan sido autorizadas como participantes de la grid.

- **Servicio de Información y Monitoreo.** Este servicio consiste de Monitoreo e Información, Monitoreo de Trabajos, Monitoreo de Red y el Servicio de Descubrimiento. Estos servicios de información son componentes vitales de bajo nivel de la grid. Estos componentes permiten recibir información acerca de los trabajos en ejecución, los recursos ocupados, el estado de la red y algunas otras cosas más. La información es almacenada jerárquicamente bajo un modelo de

árbol y se puede tener acceso a dicha información en tres niveles: recurso, sitio u oficina virtual.

- **Servicio de Datos.** Esta compuesto de un Catálogo de Metadatos, Catálogo de Archivos y Replicas, Elemento de Almacenamiento y Movimiento de datos. Los servicios requeridos para la gestión de datos son al menos los siguientes: Backend de almacenamiento (controladores y hardware), Interfaz del Storage Resource Manager (Administrador de Recursos de Almacenamiento), Servicio de transferencia de datos (GridFTP), API de E/S para archivos tipo POSIX (gLite-E/S) y servicios auxiliares de contabilización e ingreso.

  El DS (Calendarizador de Datos) hace un seguimiento de las solicitudes de transferencia de usuarios y servicios, mientras que el FTS/FPS (Transferencia de Archivos/Servicio de Ubicación) localiza las réplicas de los datos correspondientes. Esto se realiza mediante una cola de transferencia (una tabla) y un agente de transferencia (la red misma).

- **Servicio de Administración de Trabajos.** Este servicio consiste de los componentes de Contabilización, Gestor de Paquetes, Procedencia de los Trabajos, Administración de la Carga de Trabajo y Elemento de Cómputo.

  Los registros por el Sistema de Contabilización (consultas: usuarios, grupos, recursos) y los servicios de la grid deberán registrarse con un servicio tarificador cuando se requiera contabilización para propósitos de cobranza y determinación de costos. El servicio acumula información sobre el uso de recursos por usuarios o grupos de usuarios (VOs). La captura de información de servicios/recursos del grid requiere sensores (Medición de recursos, capa de abstracción de medición, registros de uso).

  El CE (Elemento de Cómputo) se refiere a un conjunto o cluster de recursos de cómputo (WN) gestionado por un manejador de colas (LRMS), para despachar y ejecutar trabajos que cumplan con los requerimientos del usuario. Este servicio representa el recurso de cómputo que es responsable de la administración de las tareas: envío, control, etc. Existen dos modelos de envío de trabajos (de

acuerdo a solicitudes del usuario o políticas del sitio): PUSH (los trabajos son enviados a la cola del CE) y PULL (los trabajos son traídos a la cola del CE cuando se encuentra vacía). Es responsable además de recolectar información de contabilización.

El WMS (Servicio de Administración de Carga de Trabajo) es el conjunto de componentes del middleware responsables de la distribución y gestión de trabajos en los recursos del grid. Esta compuesto por dos componentes: WM (Administración de Carga de Trabajo): Acepta y satisface requerimientos, el proceso de asignar el mejor recurso disponible es denominado "Matchmaking" y L&B (Ingreso y Registro): Seguimiento de la ejecución de trabajos en términos de eventos: enviado, corriendo o finalizado.

La figura 3.3 muestra un esquema de los servicios que ofrece la grid mediante la infraestructura gLite y muestra además que existen dos formas de acceso a ella.

Ciclo de Vida de un Job

Es interesante observar la forma en la que un trabajo es llevado a cabo ya en la arquitectura de gLite. En la figura 3.4 se observa la secuencia que sigue un trabajo al ingresar a ella. En primer lugar, el usuario ingresa su trabajo en la Interfaz de Usuario (UI) ya sea a través de la línea de comandos del sistema operativo o mediante la API otorgada por el servicio web. Una vez ingresado el trabajo, el Sistema de Administración de Carga de Trabajos (WMS) se encarga de buscar entre los Elementos de Cómputo (CE) que tiene a su alcance al que cumpla con las características adecuadas para llevar el trabajo, donde dichas características son principalmente la disponibilidad y los recursos de cómputo necesarios para la ejecución del trabajo. Una vez que el CE haya sido elegido, este posiciona el trabajo en el sistema de colas local, en el cual los Nodos Trabajadores (WN) finalmente tienen acceso y ejecutan el trabajo cuando les haya sido asignado el momento de la ejecución de la misma.

Nótese que los bloques tienen retroalimentación, mediante la cual comunican el resultado de la realización de su trabajo particular al bloque predecesor, y de

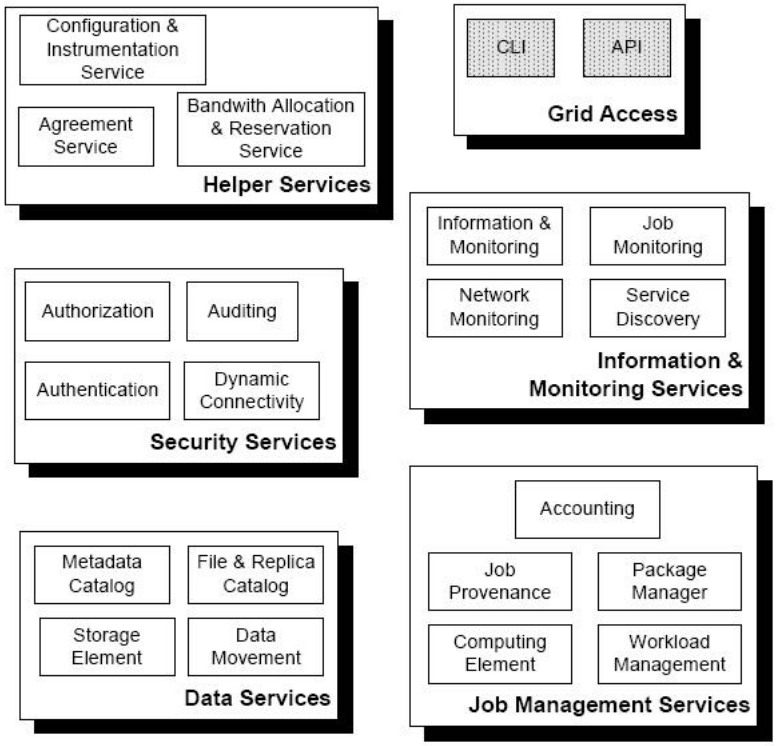

Figura 3.3: La gama de servicios que ofrece gLite

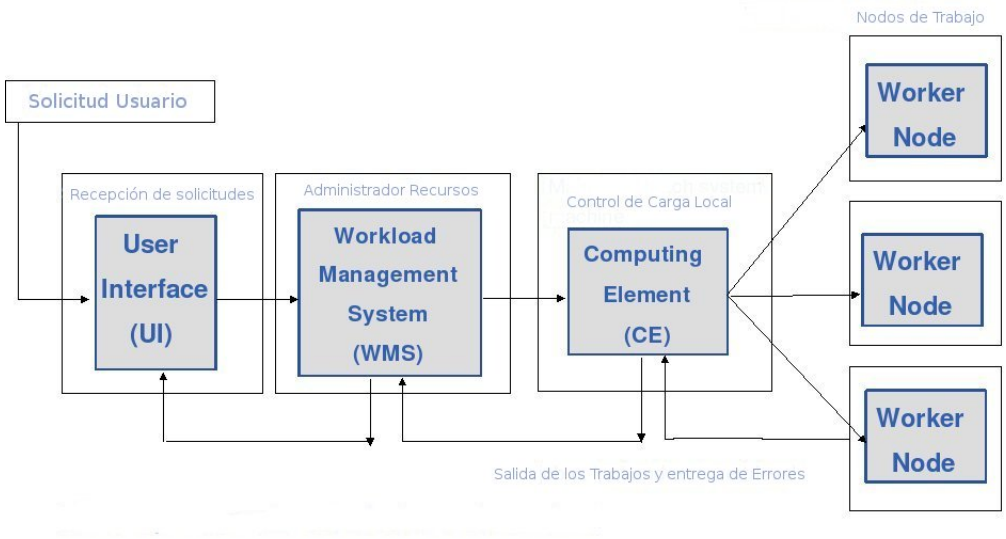

Figura 3.4: Ciclo de vida de un trabajo, desde su ingreso hasta su salida

esa forma es en la que finalmente los datos son almacenados en los Elementos de Almacenamiento (SE) o entregados a la Interfaz de Usuario (UI) según sea el caso que corresponda. Una vez que los resultados hayan sido entregados al usuario mediante la Interfaz de Usuario, el trabajo será entonces eliminado de la cola local de trabajos y de los Elementos de Almacenamiento (SE) que se hayan utilizado.

3.4. Conclusiones

En este capítulo se trato acerca de los fundamentos teóricos de la simulación. Se describen los principales clasificaciones de la simulación, la forma en la que se analizan los resultados de éstas. Estos conceptos son de gran importancia para el desarrollo de este trabajo debido a que se seguirán estas definiciones para la realización de las simulaciones y el análisis de los resultados.

Posteriormente se describe la infraestructura experimental, donde se describe el middleware gLite, su arquitectura, servicios y principales características. Este middleware gLite es importante debido a que es el que se utiliza en la etapa experimental, y particularmente en la grid EELA.

Capítulo 4

El Problema de Planificación en Sistemas Grid

4.1. Introducción

En la presente sección se describe 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 un grafo que modela la topología física de la grid, así como las capacidades de hardware, tanto de los nodos como de los enlaces de comunicación.

De manera análoga se propone modelar la carga de trabajo de una grid como un flujo de tareas con requisitos de procesamiento y memoria. 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).

4.2. Definición formal de Grid

Sea $G = (N, L)$ un grafo que modela la topología de una grid, donde cada uno de los vértices $n \in N$ representa un nodo de la grid y cada una de las aristas $l \in L$ modela un enlace de comunicación entre nodos de la grid.

De manera similar sea $NG = (M, O)$ un grafo que modela la topología de un nodo grid, donde cada uno de los vértices $m \in M$ representa una máquina en el nodo grid y las aristas $o \in O$ modelan un enlace de comunicación entre las maquinas de acuerdo a alguna topología de red. La figura 4.1 muestra como es que una Grid está compuesta de diversos Nodos Grid.

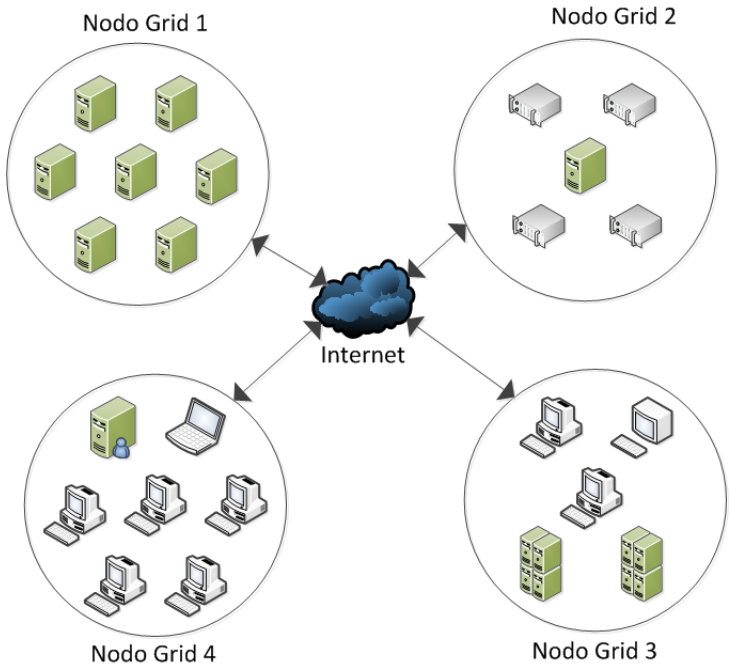

Figura 4.1: Una Grid está compuesta por Nodos Grid

Las capacidades tanto de nodos como de enlaces se especifican por medio de las funciones c, μ, p donde:
$c : A \to Z^+$ es una función que describe la capacidad de los enlaces de comunicación $a \in A$.
$u : M \to Z^+$ y $p : M \to Z^+$ son funciones que describen la capacidad de memoria y procesamiento de una $m \in M$.

4.2.1. Definición Formal de Carga de Trabajo

En la presente sección se define formalmente la carga de trabajo. Tal como se describe a continuación una instancia de una carga de trabajo consiste de los reque-

rimientos de recursos de cómputo y un DAG que modela las dependencias entre los procesos.

Sea $T_i = \{llegada_i, memoria_i, t_{cpu\,i}, DAG_i\}$ una 4-tupla que define los requerimientos estimados por el usuario de la tarea i, donde $llegada_i$ es el tiempo en el que la tarea i fue enviado a la grid G, $memoria_i$ es la cantidad de memoria que requiere la i-ésima tarea para su ejecución, $t_{cpu\,i}$ es la cantidad de tiempo que una tarea requiere para su ejecución y DAG_i es el grafo dirigido sin ciclos que describe la jerarquía de los procesos en T_i.

$DAG_i = (V, A)$, donde V es el conjunto de procesos que componen a la tarea T_i y A es el conjunto de relaciones de precedencia entre procesos que deben ser cumplidas por la tarea.

En la figura 4.2 se muestran tres ejemplos de DAG's típicos que representan la jerarquía ejecución de los procesos dentro de una tarea.

Sea una carga de trabajo (C) el conjunto de trabajos que llegan a la grid para ser planificados:

$$C = \{T_1, T_2, T_3, ..., T_m\}$$

donde m es la cantidad de tareas T_i contenidas en el flujo de trabajo. La figura 4.3 muestra una carga de trabajos que ha sido sometida a la Grid para ser planificada.

4.2.2. Definición Formal de Plan de la Grid

Un problema de planificación consiste en generar un plan S de una grid que se puede definir por medio de los planes individuales S_{n_0} de cada uno de los nodos de trabajo que lo componen.

$$S = \{S_{n_1}, S_{n_2}, S_{n_3}, ..., S_{n_N}\}$$

A su vez los planes individuales S_{n_0} es un conjunto de 2-tupla que especifica

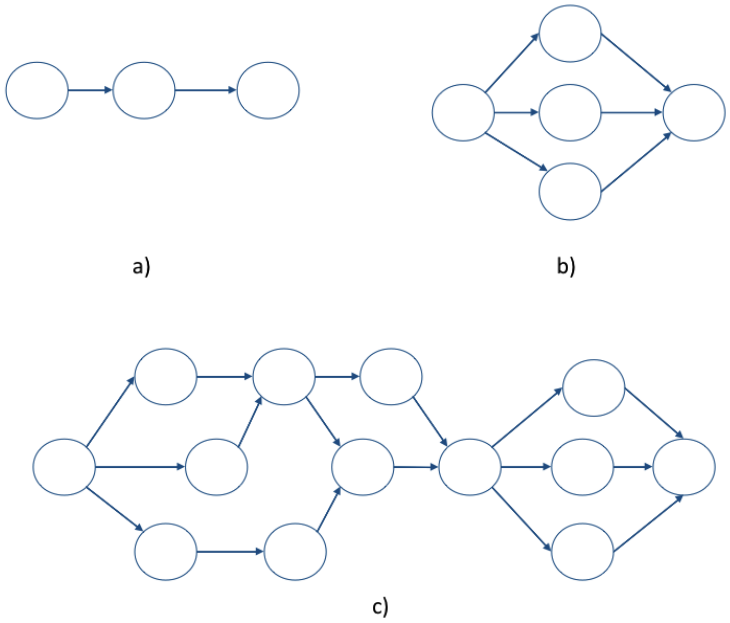

Figura 4.2: Ejemplos de DAG que especifican las dependencias entre los procesos que componen una tarea a) secuencial, b) paralela y c) compleja

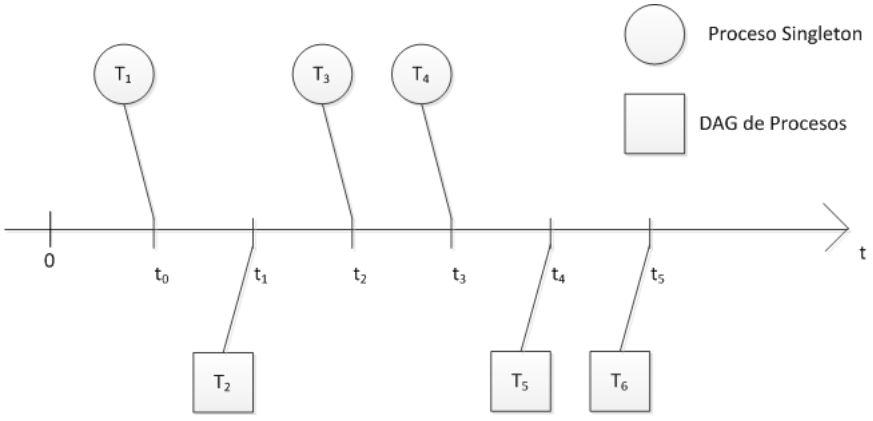

Figura 4.3: Ejemplo de una carga de trabajos sometida a la Grid

la asociación de un proceso y un procesador así como el tiempo de utilización del mismo.

$$S_{n_j} = \{(P_{k,i}, \tau_{k,i,j}), (P_{k',i'}, \tau_{k',i',j'}), ...\}$$

donde:

$P_{k,i}$ es el k-ésimo proceso de la i-ésima tarea que es asignado al procesador n_j en el plan S

$\tau_{k,i,j}$ es el tiempo en que el proceso $P_{k,i}$ comienza a ser ejecutado en el procesador j

Para cada proceso existe un número no negativo t_{k,i,n_j} que es el tiempo de procesamiento del k-ésimo proceso de la i-ésima tarea asignado al procesador n_j. Para cada proceso k de la tarea i en el procesador n_j, sea τ_{k,i,n_j} el tiempo de inicio de k en n_j.

La ecuación 4.1 establece que el tiempo de inicio de ejecución del proceso k de la i-ésima tarea en el procesador n_j debe se mayor a cero.

$$\tau_{k,i,n_j} \geq 0, \ \forall \ k \in V, i \in T, n_j \in J \tag{4.1}$$

En la ecuación 4.2 se establece que el tiempo de inicio τ_{k,i,n_j} del proceso k de la tarea i en el procesador n_j debe ser mayor o igual al tiempo de inicio de su predecesor más el tiempo en que será ejecutado.

$$\tau_{k,i,n_j{}^j_t} \geq \tau_{k,i,n_j{}^j_{t-1}} + t_{k,i,n_j{}^j_{t-1}}, \ \forall \ n_j \in J, \ t = 2, 3, ..., m \tag{4.2}$$

En la ecuación 4.3a se establece que dadas dos tareas i e i', el tiempo de inicio τ_{k,i,n_j} del proceso k de la tarea i en el procesador n_j debe ser mayor o igual al tiempo de inicio τ_{k,i',n_j} mas el tiempo en que será ejecutado. De forma recíproca en la ecuación 4.3b el tiempo de inicio τ_{k,i',n_j} k del proceso k de la tarea i' en el procesador n_j debe ser mayor o igual al tiempo de inicio τ_{k,i,n_j} mas el tiempo en que será ejecutado.

$$\tau_{k,i,n_j} \geq \tau_{k,i',n_j} + t_{k,i',n_j} \tag{4.3a}$$

$$\tau_{k,i',n_j} \geq \tau_{k,i,n_j} + t_{k,i,n_j} \tag{4.3b}$$

$$\forall\, k \in V, i \in T, n_j \in J$$

Las condiciones establecidas por las ecuaciones 4.2, 4.3a y 4.3b garantizan que no habrá traslapes y de esta forma que en el plan S cada procesador solo tendrá asignada la ejecución de un proceso a la vez.

Los eventos antes definidos contienen un conjunto de datos relacionados con el estado de las tareas y los procesos que las componen. De tal forma dada una grid, una carga de trabajo y un plan válido se obtiene un conjunto de eventos a partir de los cuales se puede calcular el valor de las métricas que queremos optimizar.

4.3. El Problema de Planificación en Sistemas Grid

Un problema de planificación consiste en generar un plan S para la grid. Un plan válido es aquel que cumple con las relaciones de precedencia definidas en los DAG's y donde un único proceso es asignado a un procesador a la vez. El problema de planificación como un problema de optimización consiste en encontrar el plan válido que optimiza una de las métricas o funciones objetivo mostradas a continuación.

1. Retardo máximo
2. Retardo promedio
3. Utilización de la grid

A continuación se definen formalmente de cada una de estas métricas, que finalmente es la forma en como se calculan en los planes obtenidos por el algoritmo propuesto.

Sea R el retardo, que es el tiempo que le toma a una tarea ser ejecutada en su totalidad. Dadas las tareas T_i se define el retardo R como:

$$R_{T_{l_0}} = T_{final_{j,k}} - T_{inicial_{j,k}},\ \forall\, l_0 = i \tag{4.4}$$

El retardo máximo (1) R_{max} se define entonces como:

$$R_{max} = max_i\{R_{T_i}\} \tag{4.5}$$

Un plan se considera óptimo cuando el valor del retardo máximo es mínimo.

El retardo promedio (2) $\bar{R}$ se define como:

$$\bar{R} = \frac{1}{|T|}\sum_{l=1}^{i} R_{T_l} \tag{4.6}$$

donde $|T|$ es la cardinalidad del conjunto que contiene las tareas ejecutadas.

En este caso un plan se considera óptimo cuando el valor del retardo promedio es mínimo.

La utilización 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 tiempo que cada procesador se encuentra ocupado ejecutando un proceso y el tiempo en que le toma a la grid atender la tarea completa. Es decir:

Sea $T_{inicial_{i,j,k}}$ el *tiempo de llegada* del k-ésimo proceso de la tarea i-ésima en el j-ésimo procesador. Sea $T_{final_{i,j,k}}$ el *tiempo de terminación* del k-ésimo proceso de la tarea i-ésima en el j-ésimo procesador.

Sea ρ_{l_0} la utilización del i-ésimo procesador.

$$\rho_{l_0} = \frac{\sum_{P_{k,i} \in S_{n_{l_0}}} (T_{final_{P_{k,i}}} - T_{inicial_{P_{k,i}}})}{T_{final} - T_{inicial}} \tag{4.7}$$

donde $T_{inicial}$ es el tiempo de inicialización del primer proceso en el j-ésimo procesador y T_{final} es el tiempo de finalización del último proceso en el j-ésimo procesador.

Sea ρ_T la utilización promedio de la Grid (3)

$$\rho_T = \frac{1}{|J|}\sum_{j=l} \rho_j \tag{4.8}$$

donde $|J|$ es la cardinalidad del conjunto de procesadores en el recurso dado y ρ_T es el promedio de utilización de los procesadores.

Un plan se considera óptimo cuando el valor del uso de la grid es máximo.

4.4. Conclusiones

En este capítulo se definió formalmente el problema de planificación en sistemas grid.

El problema de planificación de tareas en un sistema multiprocesador distribuido es del tipo NP-Hard, por lo que solo es posible aproximarse a la solución óptima.

En este capítulo se definió la carga de trabajo, y el plan de la grid. Un planificador de tareas tiene como objetivo encontrar un plan válido que idealmente sea óptimo con respecto a algún parámetro.

Los parámetros que se ha seleccionado por ser relevantes son las métricas: retardo promedio, retardo máximo y uso de la grid. Las dos primeras métricas son relevantes para el usuario ya que sugieren el tiempo que tarda la grid en resolver una tarea. La tercer métrica es relevante para el administrador ya que indica si cuantos recursos han sido utilizados, ya que idealmente una tarea debería ser distribuida en todos ellos.

Capítulo 5

Diseño del Algoritmo Propuesto

5.1. Introducción

Este capítulo se muestra la especificación del esquema de planificación propuesto que tiene como su elemento principal el algoritmo basado en distribuciones deslizantes.

El esquema de planificación consiste de un sistema de colas para cada nodo grid, en el cuál se gestiona la llegada de tareas que han sido sometidas por el usuario. Las tareas pueden ser recibidas en algún nodo grid si es que el usuario se encuentra en alguna entidad contenida en dicho nodo, y también se podrán recibir tareas que hayan sido enviadas por otro nodo dentro de la grid.

Una vez que una tarea ha ingresado a un nodo, ya sea local o remota, deberá crearse el plan mediante el algoritmo y encolarlo en los recursos para que la tarea sea resuelta.

5.2. Algoritmo a Nivel Grid

A continuación se describe la forma en como una tarea es planificada una vez que ha llegado a la grid.

5.2.1. Función de Planificación a nivel Grid

Consideremos una grid que está compuesta por un conjunto de nodos grid. Cada nodo grid contiene a su vez recursos de cómputo heterogéneos los cuales administra.

Consideremos una carga de trabajo como la que se muestra en la figura 4.3. Como ya se mencionó en el capítulo anterior, una carga de trabajos esta formada por tareas que pueden contener muchos procesos o un solo proceso singleton. En el caso de que la tarea tenga muchos procesos, la relación de interdependencia se modela a través de un DAG.

Las tareas llegan a la cola de algún nodo grid en algún tiempo. Cuando un nodo grid recibe una tarea esta entra en el sistema de colas que se muestra en la figura 5.1, el cual se encuentra en cada uno de los nodos grid. En este sistema de colas se analiza y decide si la tarea será planificada en el nodo grid que recibió la tarea o si será enviada a otro nodo grid.

Como se puede observar el la figura 5.1 la cola local tiene $n-1$ aristas, donde n es el número de nodos grid, y cada arista corresponde a cada uno de los nodos grid vecinos. Cada arista además tiene asociado un peso que corresponde a la probabilidad de que la tarea sea enviada a otro nodo grid o no. Si la tarea es enviada a otro nodo grid mediante la cola remota entonces entra directamente a la cola general, esto para evitar que una tarea quede "flotando.^entre todos los nodos sin ser atendida. En caso de que se decida que la tarea no se enviará a otro nodo, esta es enviada directamente a la cola general local.

El algoritmo 1 se encarga de calcular las probabilidades de cada una de las aristas en cada uno de los sistemas de colas en todos los nodos grid.

La memoria que el usuario solicita en la descripción de la tarea sometida es exactamente la cantidad de información que la tarea genera en la ejecución.

La capacidad estática del Nodo Grid está dada por la suma de todas las maquinas que lo componen, donde la capacidad de cada una de ellas está dada por el resultado

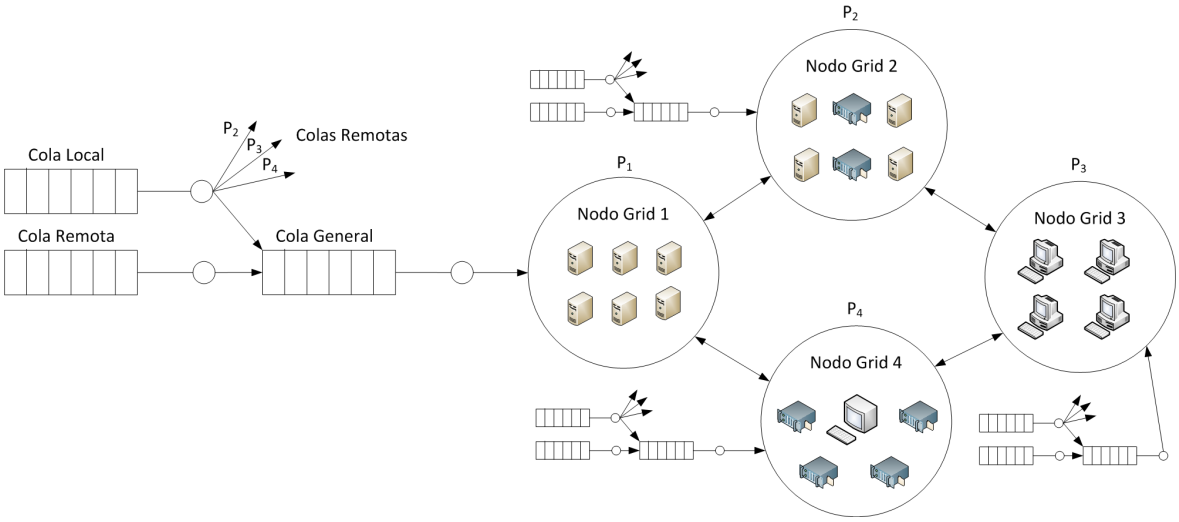

Figura 5.1: Sistema de Colas en los Nodos Grid

Algoritmo 1: Función de Planificación a nivel Grid

2 Extraer tarea T_i de la cola de entrada
4 Planificar(Tarea T_i) **begin**
5 $k \leftarrow 1$
7 **while** $k \leq N$ **do**
8 $t_{transporte_{jk}} \leftarrow$ tiempoRutaMasCorta($Datos_i$)
9 $t_{fin_k} \leftarrow \dfrac{t_{exe_i}}{coefNodoGrid_k} + t_{transporte_{jk}}$
10 $coef_{afin_k} \leftarrow \dfrac{1}{t_{fin_k}}$
11 $coef_{afin_T} \leftarrow coef_{afin_T} + coef_{afin_k}$
12 $k \leftarrow k + 1$
13 $k \leftarrow 1$
15 **while** $k \leq N$ **do**
16 $p_k \leftarrow \dfrac{coef_{afin_k}}{coef_{afin_T}}$
17 $k \leftarrow k + 1$
18 $x \leftarrow$ selecciónAleatoria($p_1, p_2, p_3, ..., p_N$)
19 PosicionarEnCola(T_i,NG_x)
20 **end**

del benchmark de diferentes SpecInt de acuerdo a la marca, modelo y arquitectura de cada una de ellas.

El usuario somete su tarea a ejecución a través de una interfaz de usuario. La interfaz de usuario se encuentra dentro de un nodo grid. Suponemos que el tiempo de ejecución que el usuario solicita se encuentra normalizado de acuerdo a la capacidad de un CPU o recurso Grid (según sea el caso) de referencia. La Grid contiene N Nodos Grid y se sabe que la tarea i llegó al Nodo Grid j, a su vez cada Nodo Grid tiene M número de CPU's.

En la línea 2, el algoritmo 1 extrae la tarea de la cola de entrada y la planifica en la línea 4.

La tarea es ingresada al sistema de colas en el que se decide si será ejecutada en ese nodo Grid o se envía a algún otro.

El ciclo en la línea 7 recorre todos los Nodos Grid y calcula el coeficiente de afinidad para cada uno de ellos. Este coeficiente de afinidad está compuesto por los tiempos de transporte utilizando Dijkstra para encontrar el camino más corto en tiempo. El parámetro $Datos_i$ es la suma de los datos de entrada y una estimación de los datos de salida de la tarea T_i. El tiempo fin combina el tiempo de transporte y de ejecución resultando un total en el que la tarea a planificar le tomaría ser atendida por el Nodo Grid dado.

El coeficiente de afinidad de un Nodo Grid es directamente proporcional a la tendencia de que la tarea sea ejecutada en él. Se utiliza un coeficiente de afinidad acumulado para el calculo de las probabilidades de los Nodos Grid.

En el ciclo del renglón 15 se calculan las probabilidades de los Nodos Grid en función de los coeficientes de afinidad de todos ellos. Con estas probabilidades se selecciona aleatoriamente uno de los nodos grid y finalmente se encola en ese Nodo Grid.

El algoritmo 2 realiza la selección aleatoria del recurso de acuerdo a las probabilidades que recibe como parámetro. Forma intervalos con esas probabilidades y mediante la generación de un número aleatorio determina en cual intervalo pertenece ese número que corresponde a su vez a la selección aleatoria.

Algoritmo 2: Selección Aleatoria

Data: $p_1, p_2, p_3, ..., p_N$: Probabilidad asociada a los CPU's

```
1  begin
2  |  num ← númeroAleatorio()
3  |  if 0 < num < p_1 then
4  |  selección ← 1
5  |  else if p_1 < num < p_1 + p_2 then
6  |  |  selección ← 2
7  |  else if p_1 + p_2 < num < p_1 + p_2 + p_3 then
8  |  |  selección ← 3
9  |  ⋮
10 |  else if p_1 + p_2 + ... + p_{N-1} < num < 1 then
11 |  |  selección ← N
12 |  return selección
13 end
```

5.3. Algoritmo a Nivel Nodo Grid

Una instancia del problema de planificación se especifica por medio de una Grid G y una carga de trabajo C. El objetivo es generar un plan de asignación de procesos a procesadores, tal que se optimicen una o varias funciones objetivo. El Nodo Grid recibe una carga de trabajos en la cola, para cada tarea se obtiene su orden topológico y posteriormente el agente de planificación obtiene un plan (o calendario) para la ejecución de la tarea en los procesadores asociados. La figura 5.2 muestra como se lleva a cabo esta operación en el nodo Grid.

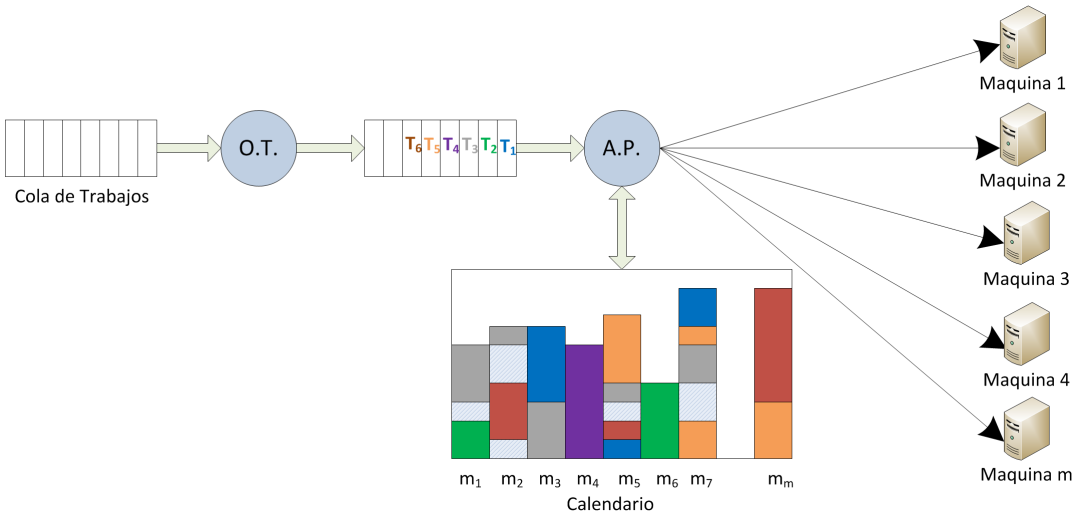

Figura 5.2: Generación de un plan para una carga de trabajos

5.3.1. Algoritmo de la creación de colas

Como ya se definió en la sección anterior cada Nodo Grid tiene un sistema de colas para el flujo de tareas entrantes. Una vez que que una tarea es encolada al Nodo Grid el algoritmo 3 clasifica en tareas que contienen un DAG de procesos (línea 3) o bien del tipo Singleton (línea 6).

Para las m tareas del flujo que contienen un DAG de procesos, existe un orden de precedencia que debe ser considerado para la planificación. Por esta razón se crea una cola para cada una de las tareas DAG en la que los procesos donde se encolan de acuerdo a su orden topológico, lo cuál garantiza que se respetará la precedencia.

En el caso de los procesos Singleton se tiene una cola común para todos ellos y se ordenan de acuerdo a su tiempo de llegada.

Algoritmo 3: 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  |  Cola_i.Crear Cola_i.Encolar(OrdenTopológico(T_i.DAG))
6  else
7  |  ColaSing.Encolar(T_i))
8  Actualizar(NúmeroActualColas)
```

5.3.2. Algoritmo de recorrido de las colas

Ya que han sido creadas las $m + 1$ colas para todo el flujo de tareas entrante al Nodo Grid, el algoritmo 4 muestra como se recorren y planifican todas ellas una a una de acuerdo a su tiempo de llegada. Se toma cada cola y dados los valores establecidos del parámetro $kT \in \{100, 10, 1\}$ se obtienen n planes para cada uno de ellos, donde n es el número de procesos que contiene cada tarea a planificar. De todos los n planes para cada valor de kT, se selecciona el mejor de ellos de acuerdo a alguna de las métricas.

La figura 5.3 muestra gráficamente como es que cada uno de los planes obtenidos tiene diferentes valores para las métricas. El algoritmo selecciona el mejor de ellos con respecto a la métria en cuestión.

La distribución Gibbs-Boltzmann tiene el parámetro kT (línea 19 del algoritmo 5) que permite modificar el comportamiento del algoritmo en la selección de un procesador para un proceso a planificar. Por cada tarea compuesta se obtienen $3n$ planes con $kT = \{100, 10, 1\}$, donde:

- n es el número de procesos en la tarea a planificar.
- $kT = 100$: selección aleatoria.
- $kT = 10$: análisis de soluciones en el vecindario extendido a la ρ-aproximación.
- $kT = 1$: análisis de soluciones en el vecindario cercano a la ρ-aproximación.

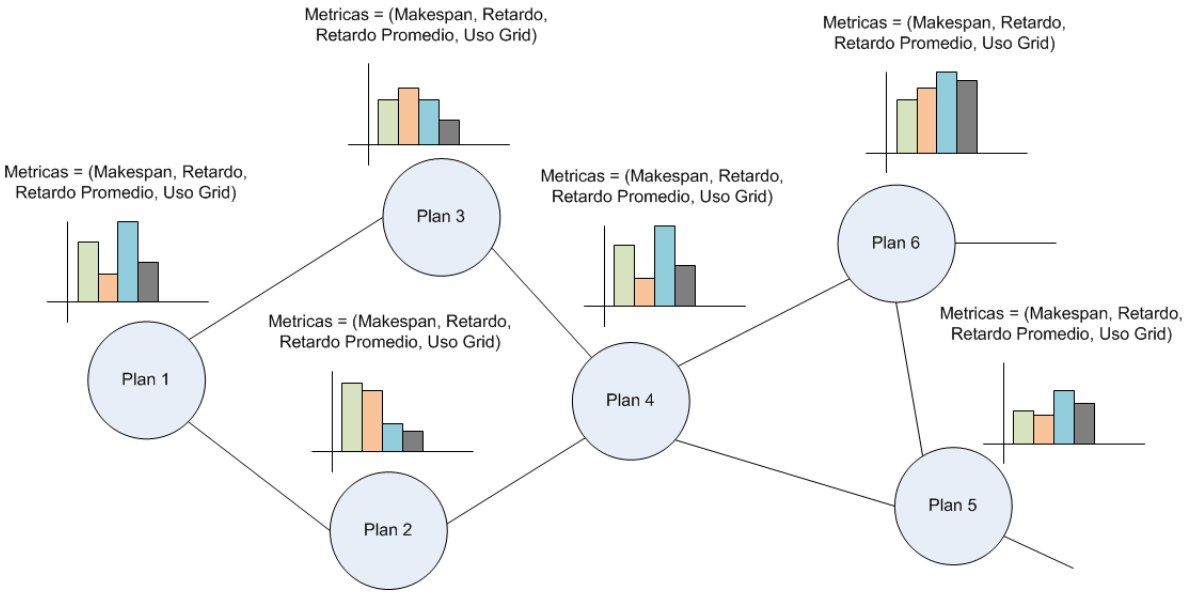

Figura 5.3: Ejemplo de diversos planes con valores de métricas

De los $3n$ planes se selecciona el mejor de ellos con respecto a alguna métrica.

La figura 5.4 muestra como se comporta la función de distribución Gibbs-Boltzmann en la ecuación de la línea 19 del algoritmo 5. Cuando $kT = 1$ los tiempos de finalización menores tendrán una mayor probabilidad y viceversa, mientras que cuando $kT = 100$ todos los procesadores tendrán una misma probabilidad lo que implica que la selección será aleatoria.

De esta forma en el algoritmo de recorrido de las colas (Algoritmo 4) el ciclo de la línea 8 itera en los valores 100, 10 y 1 que se han determinado para kT. En cada una de esas iteraciones se generan n planes (línea 14), donde n es el número de procesos en la tarea que se está planificando. Luego de obtener los n planes se selecciona el mejor de ellos de acuerdo (línea 17) a la métrica que se desee optimizar, con esto se obtiene el mejor plan para cada valor de kT con respecto a alguna métrica. Al finalizar el recorrido de los valores de kT se tienen 3 mejores planes y en la línea 20 se selecciona el mejor de ellos con respecto a la misma métrica.

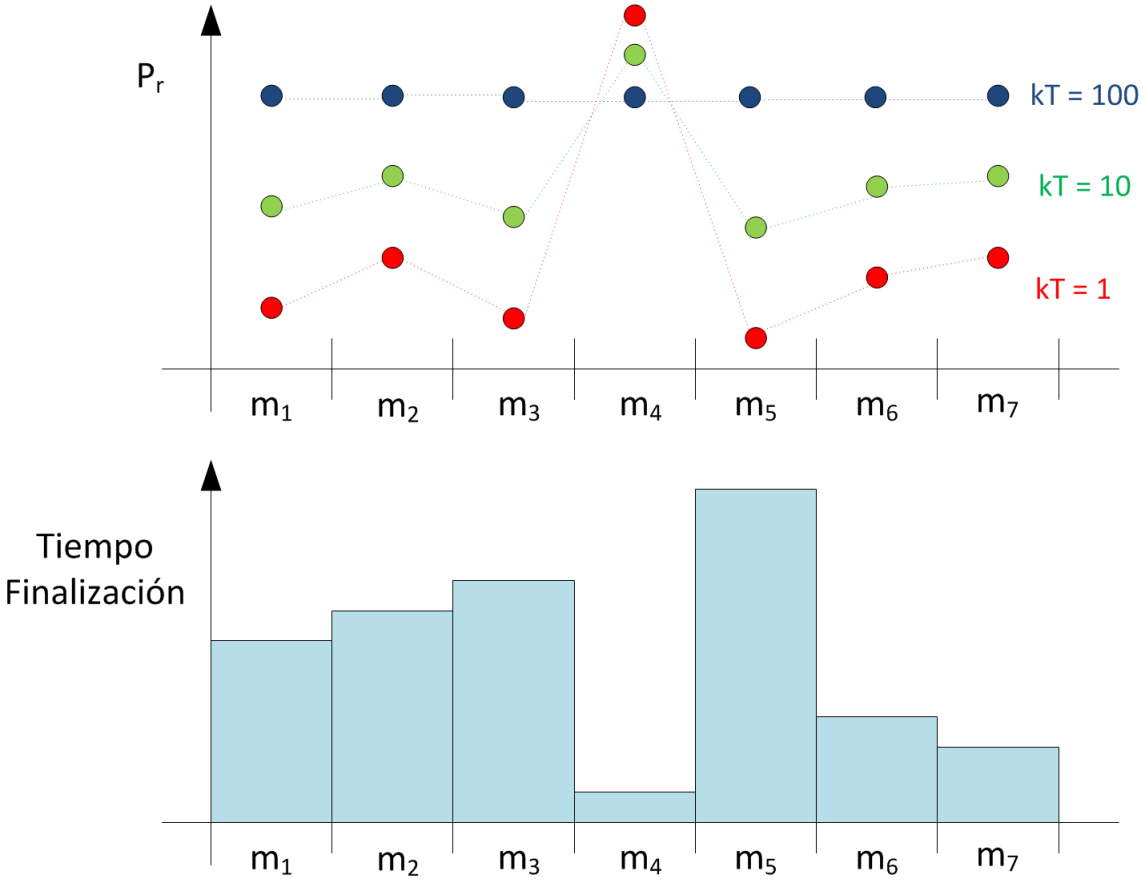

Figura 5.4: Comportamiento de la asignación de probabilidad a los procesadores

Algoritmo 4: Algoritmo de recorrido de las colas

```
1  recorridoColas (m colas DAG, cola Singleton) begin
2      i ← 1; j ← 1; k ← 1; kT ∈ {100, 10, 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                 Plan_k ← Planificar(T_i, kT_j)
15                 k ← k + 1
17             Plan_j ← selMejorPlan(P_1, P_2, ..., P_n, metrica)
18             j ← j + 1
20         Plan_{T_i} ← selMejorPlan(P_1, P_2, P_3, metrica)
21         Encolar(Plan_{T_i})
22         i ← i + 1
23 end
```

5.3.3. Función de Planificación

La función de planificación (algoritmo 5) recibe la tarea T_i a planificar. En el ciclo de la línea 8 se recorren los M 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 y posiblemente indispensables para la ejecución del proceso siguiente, por lo que deberán estar en el procesador donde se planifique si son requeridos.

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[6], 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 5 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 los CPU para finalmente hacer la selección aleatoria (línea 28) y encolar el proceso (línea 30).

Algoritmo 5: 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
                   t_exe_k ← P_i.tiempo_exe / coefCPU_k
15
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
               p_k ← coef_afin_k / coef_afin_T
25
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
32 end
```

5.3.4. Función que busca espacio de tiempo para un proceso

El algoritmo 6 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, transporte y el plan del CPU. En primer lugar se debe determinar si el proceso pertenece a un DAG o si es un Singleton (línea 3). En caso de que se tratara de un proceso perteneciente a un DAG, se debe saber el tiempo en que terminarán su ejecución los predecesores ya que este proceso debe comenzar después. De esta forma es que se elije como mínimo el tiempo mayor de finalización de los predecesores, lo que garantiza que este comenzará después de que hayan terminado todos ellos (línea 6). Con esto se puede buscar 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 y el mínimo.

En caso de que se tratase de un Singleton, se busca también el espacio de tiempo pero solo considerando el tiempo de ejecución y el tiempo mínimo, que para este caso es el tiempo de llegada del proceso. El tiempo de transporte solo aplica para un DAG.

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

Algoritmo 6: 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
11 end
```

5.4. Conclusiones

En este capítulo se definió la propuesta para resolver el problema de planificación de tareas en un sistema multiproceso distribuido tipo grid. La propuesta consiste de dos niveles: nivel grid y nivel nodo grid.

El algoritmo a nivel grid contiene un sistema de colas que permite migrar tareas si es que el nodo se encuentra muy ocupado y así distribuir la carga en la grid.

El algoritmo a nivel nodo grid contiene una llamada función deslizante que permite que mediante una selección aleatoria se pueda aproximar a una solución óptima. El algoritmo es capaz de explorar el vecindario de posibilidades para poder escapar de alguna región óptima local.

El algoritmo de distribución deslizante tiene las siguientes propiedades:

- Combina las ventajas de los algoritmos de aproximación deterministas y los algoritmos aleatorios tipo Montecarlo.
- Con complejidad polinomial que entrega soluciones aproximadas al óptimo con alta probabilidad.
- Analiza el vecindario extendido de las soluciones para escapar de máximos y mínimos locales.
- El costo adicional por analizar el vecindario extendido es Θ(n).

Capítulo 6

Resultados Experimentales

6.1. Introducción

En esta sección se describe el entorno experimental utilizado para caracterizar de forma cuantitativa el desempeño del algoritmo propuesto con respecto a las métricas: retardo promedio, retardo máximo y uso de la grid.

En primer lugar se describen los algoritmos que se utilizarán como referencia para comparar el desempeño en los experimentos diseñados. Estos algoritmos deben ser del mismo tipo para que la comparación sea válida.

Posteriormente se describe el entorno experimental para finalmente hacer un análisis estadístico mediante intervalos de confianza que finalmente se presentan en gráficas para comparar el desempeño del algoritmo propuesto y los benchmark.

6.1.1. Evaluación del Desempeño

Para comparar el desempeño del algoritmo propuesto con otros algoritmos similares con respecto a los parámetros: retardo promedio, retardo máximo y uso de la grid. Para esto se elijen 3 algoritmos que han sido utilizados como *benchmark* en diversos trabajos[21][15] y se describe la comparación de varios parámetros. 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 los siguientes: balanceo de carga inmediata (OLB), tiempo de ejecución mínimo (MET) y tiempo de finalización mínimo (MCT).

- *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 el 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.

Luego de introducir los algoritmos *benchmark* se describen los parámetros de comparación: retardo promedio, retardo máximo y uso de la grid.

Para calcular los dos primeros parámetros se obtiene en primer lugar el retardo de cada tarea. El retardo en cada tarea consiste en la diferencia del último proceso que terminó su ejecución menos el tiempo en el que inició el primero. Una vez que se tienen esos retardos, se puede calcular el promedio y obtener el máximo de ellos. Estos parámetros, o funciones objetivo, se optimizan con el valor mínimo en ellas.

El segundo parámetro de comparación, uso de la grid, se calcula como el promedio del uso de todos los procesadores en la grid. El promedio de uso de los procesadores se obtiene mediante el cociente del tiempo de uso del procesador y el tiempo total en el que las tareas fueron ejecutadas. Este parámetro, o función objetivo, se optimiza con un valor máximo en ella.

Utilizando estos parámetros, se puede comparar el desempeño del algoritmo propuesto con los algoritmos similares. Los primeros dos parámetros comparan el tiempo en que una tarea es resuelta, siendo métricas que le conciernen al usuario final, mientras que la comparación del tercer parámetro compara la utilización de los recursos en la grid, que es una métrica importante para el administrador del sistema.

6.1.2. Entorno Experimental

Para comparar el desempeño del método con los algoritmos *benchmark* se analizará el impacto del tiempo entre llegadas para cada uno de ellos. Para comparar el desempeño de los algoritmos se generan cargas de trabajo con algunos parámetros aleatorios. Estas cargas son planificadas por los algoritmos *benchmark* y el propuesto. Una vez generados los planes por todos los algoritmos se calculan los valores de las métricas antes mencionadas con los que se podrá comparar finalmente el desempeño entre ellos.

Dados los valores 2, 20, 100 y 200 segundos del *tiempo entre llegadas* se generan 10 cargas de trabajo aleatorias para cada uno de ellos. Cada carga de trabajo consiste de 120 tareas que tienen algunos parámetros aleatorios exponencialmente distribuidos. Cada tarea en la carga de trabajo puede consistir de un DAG de procesos o bien un solo proceso singleton. Los parámetros aleatorios definen las características que tendrá el DAG de procesos o el proceso singleton que componen a cada tarea en la carga de trabajo. Dichos parámetros son las siguientes: tiempo entre llegadas, duración de cada proceso, procesadores requeridos, probabilidad de que una tarea sea un DAG, ancho y largo del DAG y el grado de cada proceso en caso de que sea DAG.

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. Los *procesadores requeridos* son los ne-

cesarios por un proceso para que pueda ser ejecutado. La *probabilidad de que una tarea sea un DAG* determina cuan probable una tarea será un DAG o un proceso singleton. El *ancho*, *largo* y *grado* son las características del grafo en caso de que la tarea sea un DAG.

Los parámetros aleatorios para la generación de las cargas de trabajo se describen en la siguiente tabla:

Parámetro	Valor Promedio
Tiempo entre llegadas	2, 20, 100, 200 s
Duración proceso	20 s
Procesadores Requeridos	3
Probabilidad DAG	0.75
Ancho DAG	4
Largo DAG	3
Grado Nodo DAG	2

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 este genera los planes de todos los recursos. El nodo grid contiene un total de 204 CPU's heterogéneos, donde la distribución de las capacidades de todos ellos están descritas en la siguiente tabla clasificados por su coeficiente de capacidad.

Número de CPU's	Coeficiente de Capacidad
34	0.6
34	0.8
34	1.0
34	1.2
34	1.5
34	2.0

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.

6.1.3. Resultados

A continuación se presentan los resultados de las métricas obtenidos por los algoritmos benchmark y el propuesto. Estos resultados se presentan en gráficas en un plano xy, en las cuales en el eje de las abscisas se encuentran los tiempos entre llegadas 2, 20, 100 y 200 y en las ordenadas se encuentra el retardo máximo, retardo promedio o uso de la grid. Los puntos en las gráficas se presentan como un intervalo de confianza definidos por una media y una probabilidad.

La Figura 6.1 compara el algoritmo propuesto con los benchmark de acuerdo a la primer métrica, el retardo máximo. Tal como se puede ver el algoritmo propuesto se encuentra por debajo de los benchmark OLB y MET, y muy cercano al MCT. También se observa que el algoritmo propuesto obtiene un retardo máximo menor para tiempos entre llegadas menores. Un plan se aproximará al óptimo mientras menor sea el retardo promedio. Esta es una métrica que le concierne al usuario ya que sugiere el tiempo que más ha tardado la grid en resolver una tarea.

La siguiente tabla muestra los resultados numéricos del retardo máximo promedio obtenido por los algoritmos para los valores establecidos del tiempo entre llegadas..

Algoritmo / Tiempo entre llegadas	2s	20s	100s	200s
MCT [s]	248.3162	210.8845	205.6858	193.7832
MET [s]	727.8582	701.3704	682.5721	627.9376
OLB [s]	734.9316	688.7309	677.7813	649.0786
Función Deslizante [s]	250.5778	229.7346	219.9843	204.3487

Resultados del Retardo Máximo para los tiempos entre llegadas

La Figura 6.2 compara el algoritmo propuesto con los benchmark con respecto al retardo promedio. Se observa en que nuevamente el algoritmo propuesto se encuentra

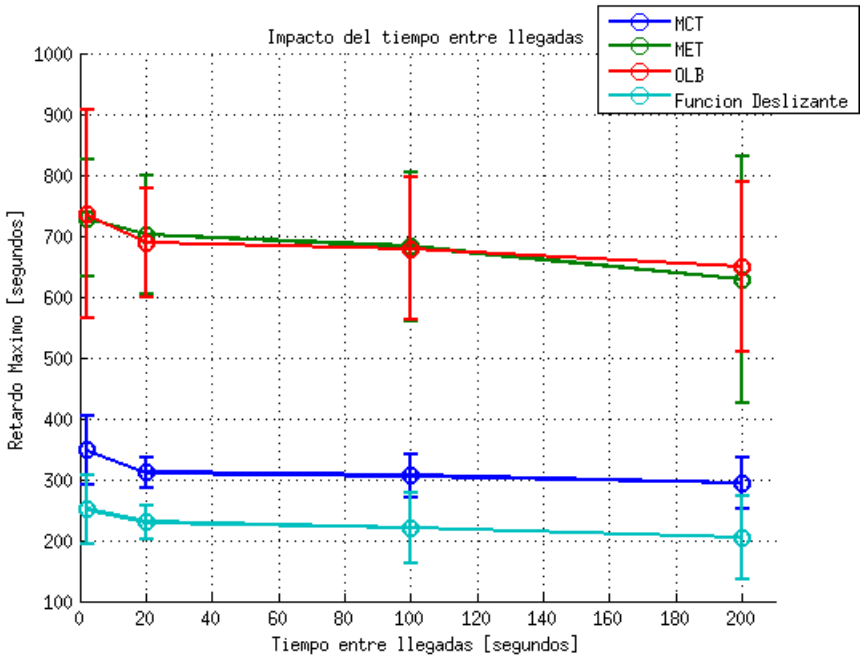

Figura 6.1: Impacto del tiempo entre llegadas para el retardo máximo en los algoritmos

por debajo de los benchmark OLB y MET, y cercano al MCT. Nuevamente el menor retardo promedio se obtiene para el menor tiempo entre llegadas. Un plan se aproximará al óptimo mientras sea menor el retardo promedio. Esta métrica le concierne al usuario final ya que sugiere el tiempo en que una tarea puede ser planificada y resuelta.

La siguiente tabla muestra los resultados numéricos del retardo promedio medio obtenido por los algoritmos para los valores establecidos del tiempo entre llegadas..

Algoritmo / Tiempo entre llegadas	2s	20s	100s	200s
MCT [s]	46.9029	45.5330	45.8320	40.8597
MET [s]	98.2673	96.2218	94.8847	93.7584
OLB [s]	192.9315	154.2988	145.1901	124.3763
Función Deslizante [s]	47.3855	46.5043	45.9899	42.9516

Resultados del Retardo Promedio para los tiempos entre llegadas

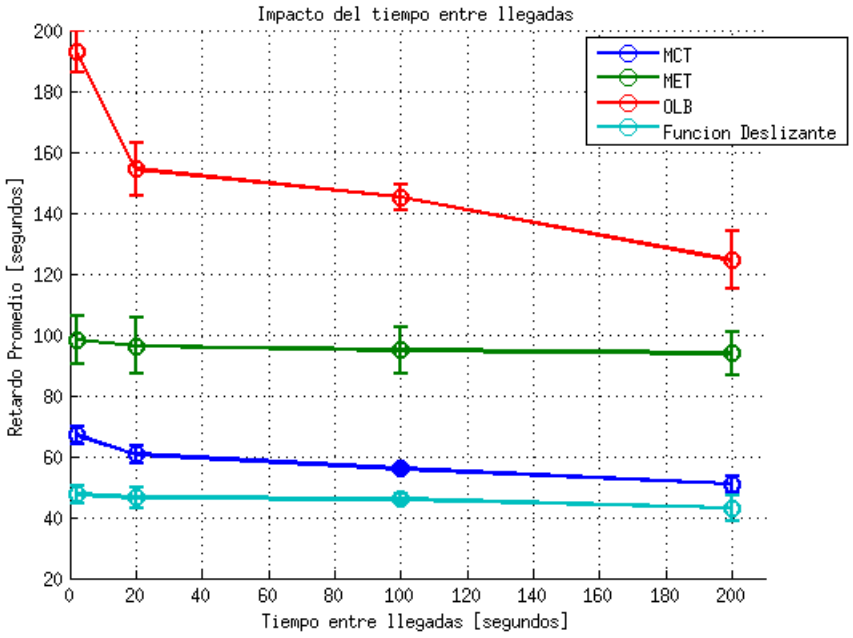

Figura 6.2: Impacto del tiempo entre llegadas para el retardo promedio en los algoritmos

Debe tenerse en cuenta que el retardo es inversamente proporcional al tiempo entre llegadas, es decir, a menor tiempo entre llegadas mayor retardo y viceversa.

En la Figura 6.3 se compara el algoritmo con los benchmark de acuerdo al uso de la grid. En esta gráfica se observa que el algoritmo propuesto se encuentra por encima de los benchmark. Esto demuestra que el algoritmo utiliza de mejor forma los recursos de la grid, es decir, los tiempos de ocio de los procesadores mientras se ejecuta una tarea son menores en todos los casos.

Finalmente la siguiente tabla muestra los resultados numéricos del uso fe Grid medio obtenido por los algoritmos para los valores establecidos del tiempo entre llegadas.

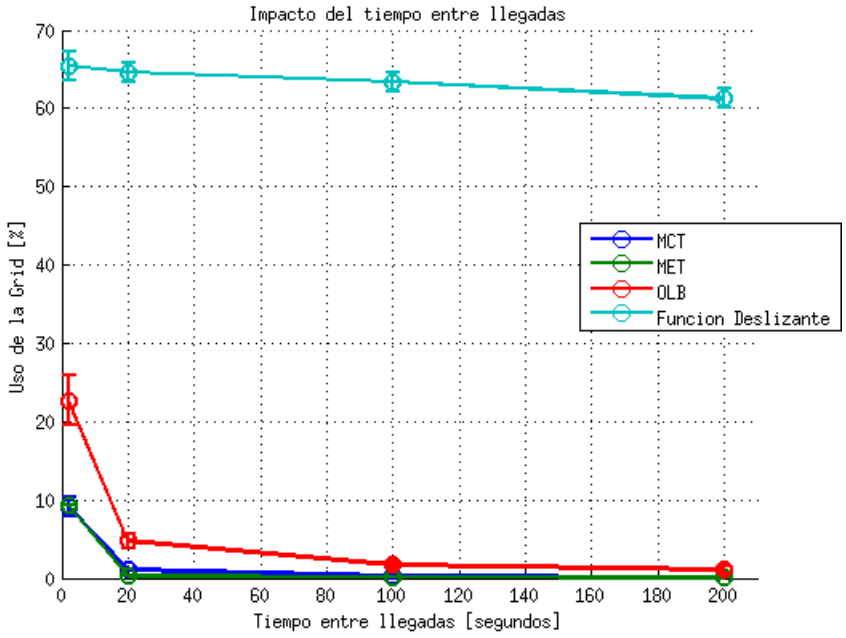

Figura 6.3: Impacto del tiempo entre llegadas para el uso de la grid en los algoritmos

Algoritmo / Tiempo entre llegadas	2s	20s	100s	200s
MCT [s]	9.1703	1.0295	0.2195	0.1035
MET [s]	0.2105	0.2443	0.1896	0.0987
OLB [s]	22.6480	4.7311	1.7146	1.0284
Función Deslizante [s]	65.4241	64.6089	63.3885	61.2322

Resultados del Uso de Grid para los tiempos entre llegadas

6.2. Conclusiones

Los resultados presentados en las gráficas muestran que el algoritmo propuesto se comporta mejor en casi todos los casos con respecto a los competidores.

Una de las razones por las que el algoritmo propuesto es una aportación sobre los competidores es porque considera el caso de que las tareas a planificar son DAG's, ya que en un caso realista las tareas están compuestas por procesos que pueden ser

ejecutados en paralelo. Por esta razón en la Figura 6.3 se observa que el algoritmo propuesto se encuentra muy por encima de los benchmark dado que es capaz de utilizar en mejor medida todos los recursos de la Grid.

Cabe mencionar que los algoritmos benchmark originalmente solo planifican tareas con un solo proceso y para el desarrollo de estos experimentos se tuvo que hacer una adaptación de ellos para que puedan planificar tareas que consistan de un DAG de procesos.

Capítulo 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.

Se propuso un arquitectura distribuida basada en agentes independientes para el monitoreo y la gestión de recursos de un sistema Grid. La naturaleza distribuida de la arquitectura la hace flexible y escalable.

Se propuso una nueva técnica, llamada de distribuciones deslizantes, para el diseño de algoritmos de optimización combinatoria. Por medio de la parametrización de la distribución de Gibbs-Boltzmann, el algoritmo combina las propiedades de los algoritmos aleatorizados tipo Montecarlo con los de ρ-aproximación.

El problema de planificación de tareas sin relación de precedencia en múltiples procesadores es NP-Hard, por lo que solo se podrá realizar una aproximación al óptimo. Los resultados de los experimentos demuestran que el algoritmo aleatorizado propuesto se aproxima al óptimo utilizando una función deslizante.

La distribución se "desliza"para que el algoritmo cambie su comportamiento de netamente aleatorio a buscar soluciones ρ-aproximadas. La técnica propuesta es general y se puede aplicar a todo problema para el cual se conozcan algoritmos ρ-

aproximados.

Se desarrolló un entorno de experimentación basado en un simulador de eventos discretos para caracterizar el desempeño de algoritmos de planificación para sistemas Grid.

El entorno cuenta con un generador de cargas sintéticas que permite generar cargas de trabajos a partir del muestreo de diferentes distribuciones de probabilidad. El entorno permite especificar las propiedades de los elementos de procesamiento.

Se desarrolló un generador aleatorio de grafos acíclicos dirigidos que modelan las relaciones de precedencia de los trabajos compuestos.

Los resultados experimentales muestran que la distribución deslizante le permite al algoritmo propuesto escapar de mínimos locales donde el algoritmo ρ-aproximado queda atrapado. El costo de utilizar las distribuciones deslizantes es $\Theta(n)$.

Las métricas que se consideran para analizar las posibles soluciones del problema de planificación le conciernen al usuario final y al administrador de la grid porque sugieren información acerca del desempeño de la grid. El retardo promedio y máximo son métricas que reflejan el desempeño de la grid con respecto al usuario final, dado que le indican el tiempo que puede ser resuelta una tarea sometida. El uso de la Grid es una tarea que refleja el desempeño de la grid con respecto al administrador ya que indica el nivel de uso de los recursos computacionales con los que cuenta la grid.

7.1. Trabajos a Futuro

- Utilizar la metaheurística propuesta para desarrollar otros algoritmos para el problema de planificación en sistemas Grid tomando como base otros algoritmos ρ-aproximados como el de Lam y Sethi.
- Considerar otro tipo de funciones objetivo como el consumo de energía y la calidad de servicio.

- Profundizar en la caracterización formal de las propiedades de los algoritmos basados en las distribuciones deslizantes.
- Desarrollar nuevas funcionalidades dentro del simulador propuesto.
- Considerar errores en la estimación de los tiempos de atención a los procesos.
- Desarrollar modelos más realistas del comportamiento de los subsistemas de almacenamiento y comunicaciones.

Bibliografía

[1] Eela-2: E-science grid facility for europe and latin america. URL http://www.eu-eela.eu/.

[2] Einstein@home. URL http://einstein.phys.uwm.edu/.

[3] The large hadron collider. URL http://lhc.web.cern.ch/lhc/.

[4] Lingo section [online]. URL http://www.lindo.com/.

[5] Search for extraterrestrial intelligence. URL http://setiathome.ssl.berkeley.edu/.

[6] The standard performance evaluation corporation (spec). URL http://www.spec.org/.

[7] Mit technology review. 2003. URL http://www.technologyreview.com/.

[8] Egee middleware architecture and planning (release 1). 2004. URL https://edms.cern.ch/document/476451/.

[9] *DAGMap: Efficient Scheduling for DAG Grid Workflow Job*. 9th. Grid Computing Conference, IEEE, 2008.

[10] Twenty experts define cloud computing. 2008. URL http://cloudcomputing.sys-con.com/node/612375/print.

[11] A. Abraham, R. Buyya, B. Nath, et al. Nature's heuristics for scheduling jobs on computational grids. En *The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000)*, págs. 45–52. 2000.

[12] D. Alexandre, N. Piotr, A. Retico, y D. Vicinanza. Global grid monitoring: The egee/wlcg case. *ACM*, 2007.

[13] M. Ambrust, A. Fox, G. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[14] Haverkort R. Boudewijn. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons Ltd., 1998. ISBN 0-470-84192-3.

[15] T. D. Braunt et al. 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*, (61):810–837, 2001. URL `http://docs.lib.purdue.edu/ecetr/19`.

[16] Inc. Cisco Systems. Performance management.

[17] X. Deng y Y. Zhang. Minimizing mean response time in batch processing system. *The 5th Annual International Conference on Computing and Combinatorics*, págs. 231–240, 1999.

[18] V. Di Martino y M. Mililotti. Sub optimal scheduling in a grid using genetic algorithms. *Parallel Computing*, 30(5):553–565, 2004.

[19] F. Dong y S. G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Inf. Téc. 504, Queen's University, 2006.

[20] F. Dong y S. G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Inf. Téc. 504, Queens University Kingston, 2006.

[21] R. Entezari y A. Movaghar. A probabilistic task scheduling method for grid environments. *Future Generation Computer Systems*, (28):513–524, 2012. URL `www.elsevier.com/locate/fgcs`.

[22] I. Foster y C. Kesselman. The anatomy of the grid: Enabling scalable virtual organizations. .

[23] I. Foster y C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, . ISBN ISBN 1-55860-933-4.

[24] I. Foster y C. Kesselman. What is the grid? a three point checklist. *Argonne National Laboratory*, 2002.

[25] I. Foster, C. Kesselman, et al. *The grid 2: Blueprint for a new computing infrastructure*. Morgan Kaufmann, 2003.

[26] I. Foster, Z. Yong, R. Ioan, y L. Shiyong. Cloud computing and grid computing 360-degree compared.

[27] M. R. Gary y D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. 1979.

[28] L. A. Goldberg, M. Paterson, A. Srinivasan, y E. Sweedyk. Better approximation guarantees for job-shop scheduling. En *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, págs. 599–608. Society for Industrial and Applied Mathematics, 1997.

[29] R. Hall, A. L. Rosenberg, y A. Venkataramani. A comparison of dag-scheduling strategies for internet-based computing. En *in Proceedings of Workshop on Resource Management*, Symposium of Parallel and Distributed Proccesing. 1996.

[30] H. H. Hoos y T. Stützle. *Stochastic local search: Foundations & applications*. Morgan Kaufmann, 2004.

[31] Rag Jain. *Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation and Modeling*. Wiley Computer Publishing. John Wiley & Sons, Inc., 1991. ISBN 0471503363.

[32] H. Kasahara y S. Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers*, 33(11):1023–1029, 1994.

[33] A. A. Khan, McCreary C. L., y M. S. Jones. A comparison of multiprocessor scheduling heuristics. En *Proceedings of the International Conference on Parallel Processing*, International Conference on Parallel Processing. AL 36849, 1994.

[34] J. K. Lenstra y A. H. G. R. Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.

[35] J. K. Lenstra, D. B. Shmoys, y É Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1):259–271, 1990.

[36] H. Li y R. Buyya. Model-based simulation and performance evaluation of grid scheduling strategies. *Future Generation Computer Systems*, 25:460–465, 2009.

[37] J. C. Liou y M. A. Palis. An efficient task clustering heuristic for scheduling dag's on multi-processors. En *in Proceedings of Workshop on Resource Management*, Symposium of Parallel and Distributed Processing. 1996.

[38] M. Mitzenmacher y E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge, 2005. ISBN 0-521-83540-2.

[39] R. Motwani y P. Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[40] N. Muthuvelu, J. Liu, N. L. Soe, S. R. Venugopal, A. Sulistio, y R. Buyya. A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. En *in Proceedings of the 3rd. Australasian Workshop on Grid Computing and e-Research (AusGrid'05)*. Australia, 2005.

[41] Buyya Rajkumar. *High Performance Cluster Computing*, tomo 1 de *Volume 1: Architecture and Systems*. Prentice Hall PTR, 1999. ISBN 0-13-013784-7.

[42] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, y M. Samidi. Scheduling data intensive workflows onto storaged-constrained distributed resources. En *in Procceedings of the 7th IEEE Symposium on Cluster Computing nad the Grid (CCGrid'07)*. IEEE, 2007.

[43] G. Ritchie y J. Levine. 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*, 2003.

[44] R. Sahu y A. K. Chaturvedi. Many-objective comparison of twelve grid scheduling heuristics. *International Journal of Computer Applications*, 13(6), 2011.

[45] R. Shannon y J. D. Johannes. Systems simulation: the art and science. *IEEE Transactions on Systems, Man and Cybernetics*, (6):723–724.

[46] D. B. Shmoys y É Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, (62):461–474, 1993. North-Holland.

[47] A. Tanenbaum. *Sistemas Operativos Distribuidos*. Prentice Hall Hispanoamericana S.A., 1 ed$^{\underline{\text{ón}}}$.

[48] H. Topcuoglu, S. Hariri, y M. Wu. Performance effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[49] P. J. M. van Laarhoven, H. L. Aarts, y J. Karel. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992. URL `http://www.jstor.org/stable/171189`.

[50] L. Wang, J. Tao, M. Kunze, A.C. Castellanos, D. Kramer, y W. Karl. Scientific cloud computing: Early definition and experience. En *HPCC'08. 10th IEEE International Conference on High Performance Computing and Communications*, págs. 825–830. IEEE, 2008.

[51] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, y C. Fu. Cloud computing: a perspective study. *New Generation Computing*, 28(2):137–146, 2010.

[52] M. Wieczorek, R. Prodan, y T. Fahringer. Scheduling of scientific workflows in the askalon grid enviroment. *in ACM SIGMOD Record*, 34(3):56–62, 2005.

[53] T. Yang. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE*, 5(9):951, 1994.

[54] S. Y. You, H. Y. Kim, D. H. Hwang, y S. C. Kim. Task scheduling algorithm in grid considering heterogeneous enviroment. En *in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '04)*, págs. 240–245. 2004.

[55] A. Y. Zomaya y Y. H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *Parallel and Distributed Systems, IEEE Transactions on*, 12(9):899–911, 2001.