

TESIS DEFENDIDA POR

Vanessa Miranda López

Y APROBADA POR EL SIGUIENTE COMITÉ:

Dr. Andrei Tchernykh

Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez

Miembro del Comité

Dr. Edgar Leonel Chávez González

Miembro del Comité

Dr. Raúl Rangel Rojo

Miembro del Comité

Dr. Hugo Homero Hidalgo Silva

*Coordinador del Programa de
Posgrado en Ciencias de la
Computación*

Dr. David Hilario Covarrubias Rosales

Director de Estudios de Posgrado

29 de Noviembre de 2010

**CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR
DE ENSENADA**



**PROGRAMA DE POSGRADO EN CIENCIAS
DE LA COMPUTACIÓN**

**ESTUDIO TEÓRICO Y EXPERIMENTAL DE CALENDARIZACIÓN DE
TRABAJOS EN LÍNEA CON ESTRATEGIAS “TIEMPO DE TERMINACIÓN
MÍNIMO, COTA INFERIOR MÍNIMA Y CARGA MÍNIMA (BAR-NOY)”
EN UN GRID COMPUTACIONAL DE DOS NIVELES.**

TESIS

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

MAESTRO EN CIENCIAS

Presenta:

Vanessa Miranda López

Ensenada, Baja California, México. Noviembre de 2010

RESUMEN de la tesis de **Vanessa Miranda López**, presentada como requisito parcial para obtención del grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, Baja California. Noviembre de 2010.

Estudio teórico y experimental de calendarización de trabajos en línea con estrategias “Tiempo de terminación mínimo, Cota inferior mínima y Carga mínima (Bar-Noy)” en un Grid computacional de dos niveles

Resumen aprobado por:

Dr. Andrei Tchernykh
Director de Tesis

En el presente trabajo se propone un análisis teórico y experimental para el problema de calendarización de trabajos paralelos en un Grid computacional. Se considera un modelo de Grid de dos niveles. En el primer nivel los trabajos son asignados a una máquina adecuada, en el segundo nivel los trabajos son calendarizados de manera independiente en cada máquina. Se hace uso del esquema de asignación de trabajos conocido como *estrategia de admisibilidad*.

Se analiza teóricamente la combinación de las estrategias de asignación *Tiempo de terminación mínimo (MCT)* y *Cota inferior mínima (MLB)* con el mejor algoritmo de calendarización local conocido. Asimismo, se lleva a cabo un análisis experimental con las estrategias MCT, MLB y una nueva estrategia de asignación denominada *carga mínima (Bar-Noy) (BN)* para determinar y comparar su desempeño promedio en diferentes casos de prueba.

Palabras clave: Grid computacional, calendarización en línea, asignación de trabajos

ABSTRACT of the thesis presented by **Vanessa Miranda López**, as a partial requirement to obtain the **MASTER IN SCIENCE** degree in **COMPUTER SCIENCE**. Ensenada, Baja California. Noviembre de 2010.

**Theoretical and experimental study of online job scheduling
with strategies "Min Completion Time, Min Lower Bound and
Min Load (Bar-Noy)" on a hierarchical Grid**

Abstract approved by:

Dr. Andrei Tchernykh
Thesis Director

In this work, we present a theoretical and experimental analysis of the parallel job scheduling problem on a Grid. We consider a Grid scheduling model with two stages. At the first stage, jobs are allotted to a suitable machine. In the second stage, local scheduling is independently applied to each machine. We use the admissible job allocation strategy.

We analyze theoretically the combination of the allocation strategies *Min Completion Time (MCT)* and *Min Lower Bound (MLB)* with the best local scheduling algorithm known in the literature. We also present an experimental analysis of the strategies *MCT*, *MLB* and a new allocation strategy named *Min Load (Bar-Noy) (BN)* to determinate and compare their performance on different test cases.

Keywords: Grid computing, job allocation, online scheduling

DEDICATORIAS

A mis papás: *Orestes y María de los Ángeles*

A mis hermanos: *Lorenia, Francisco Javier y Carlos Andrés*

A mi primer sobrinita: *Mia Gisselle*

En especial a *Fermín* por apoyarme durante
estos dos años incondicionalmente

AGRADECIMIENTOS

A mi asesor *Dr. Andrei Tchernyk* por su paciencia
y valiosa ayuda en este tiempo

A mi comité de tesis:
Dr. Carlos Alberto Brizuela Rodríguez,
Dr. Edgar Leonel Chávez González
Dr. Raúl Rangel Rojo

A mis compañeros que me ayudaron en las desveladas
y en los momentos de desesperación:
Ivetth Corona, Daniel Hernández, Raúl Loredó, Daniel Fajardo,
Ismael Espinosa, César Carranza, Jorge Mario Cortés, Ariel Quezada,
Juan Manuel Ramírez, Adán Carbajal y Jessica Beltrán

Al Centro de Investigación Científica y de Educación Superior de Ensenada

Al Consejo Nacional de Ciencia y Tecnología
por su apoyo económico el cual hizo posible que pudiera realizar este sueño.

CONTENIDO

RESUMEN.....	i
ABSTRACT.....	ii
DEDICATORIAS.....	iii
AGRADECIMIENTOS.....	iv
CONTENIDO	v
Lista de Figuras	vii
Lista de Tablas	viii
I. Introducción.....	1
I.1 Antecedentes y motivación.....	1
I.2 Trabajos previos.....	3
I.3 Importancia de la investigación	7
I.4 Objetivo general.....	7
I.4.1 Objetivos específicos.....	7
I.5 Organización del documento	8
II. Marco Teórico	9
II.1 Balanceo de la carga	9
II.2 Calendarización	10
II.3 Calendarización en un Grid computacional	11
II.3.1 Problemas dentro de un ambiente Grid	13
II.4 Estructuras de un Grid computacional	14
II.4.1 Centralizada	16
II.4.2 Descentralizada	16
II.4.3 Jerárquica.....	18
III. Definición del problema.....	19
III.1 Formulación del problema	19
III.3 Estrategias de asignación	22
III.4 Algoritmos de calendarización local	27
III.4.1 Calendarización en lista	29
III.5 Métricas de desempeño.....	31
III.6 Estrategia de admisibilidad	34
III.7 Estrategia de balanceo de la carga de Bar-Noy.....	37
III.7.1 Estrategia de asignación “Carga mínima (Bar-Noy)”.....	42

III. Análisis teórico	47
IV.1 Análisis teórico de la estrategia $MLB_a + PS$	48
IV.1.1 Modelo fuera de línea	48
IV.1.2 Modelo en línea	52
IV.2 Análisis teórico de la estrategia $MCT_a + PS$	54
IV.2.1 Modelo fuera de línea	54
IV.2.2. Modelo en línea	57
IV.3 Análisis de resultados estrategias $MLB_a + PS$ y $MCT_a + PS$	57
V. Análisis experimental	62
V.1 Configuración del Grid	62
V.2 Configuración de la carga de trabajo	63
V.3 Análisis de resultados	64
V.3.1 Desempeño de las estrategias de asignación	65
V.3.2 Degradación del desempeño de las estrategias de asignación.....	67
V.3.3 Degradación promedio para Grid 1 y Grid 2.....	71
VI. Conclusiones	74
VI.1 Resumen.....	74
VI.2 Conclusiones	74
VI.3 Trabajo a futuro.....	76
Referencias	77
APÉNDICE A.....	82
APÉNDICE B.....	84

Lista de Figuras

Figura	Página
Figura 1. Evolución en la tecnología del Grid computacional (Moallem, 2009).....	11
Figura 2. Modelo de un Grid (Rajasekaran y Reif, 2008).....	15
Figura 3. Calendarización centralizada (Hamscher <i>et al.</i> , 2000).....	16
Figura 4. Calendarización descentralizada con comunicación directa (Hamscher <i>et al.</i> , 2000).....	17
Figura 5. Calendarización descentralizada con pila de trabajos (Hamscher <i>et al.</i> , 2000).....	18
Figura 6. Estructura jerárquica de un Grid computacional (Hamscher <i>et al.</i> , 2000).....	18
Figura 7. Configuración de Grid computacional de dos niveles (Kurowski <i>et al.</i> , 2008).....	21
Figura 8. Ejemplo de un calendario basado en lista.....	35
Figura 9. Concepto del modelo de admisibilidad.....	36
Figura 10. Clasificación de nueva estrategia de asignación.....	43
Figura 11. Asignación admisible con factor a	49
Figura 12. Comportamiento del algoritmo en línea.....	53
Figura 13. Factor de admisibilidad ($a \leq mf, mmf0, m$).....	58
Figura 14. Factor de competitividad ($a > mf, mmf0, m$).....	59
Figura 15. Factor de competitividad ($a=1/2$) para $(MLB_a, MCT_a) + PS$	59
Figura 16. Factor de competitividad.....	65
Figura 17. Slowdown promedio limitado.....	66
Figura 18. Tiempo de espera promedio.....	66
Figura 19. Degradación del factor de competitividad.....	68
Figura 20. Degradación de Slowdown promedio limitado.....	69
Figura 21. Degradación del Tiempo de espera promedio.....	70
Figura 22. Degradación promedio de todas las estrategias de asignación del Grid 1 y Grid 2.....	71
Figura 23. Clasificación de la degradación promedio para todos los casos de estudio.....	71
Figura 24. Posibilidades para la problemática si $\mathcal{P} \neq NP$ (Sipser, 2005).....	83

Lista de Tablas

Tabla	Página
Tabla I. Lista de estrategias de asignación	4
Tabla II. Lista de algoritmos de calendarización local.....	5
Tabla III. Configuración Grid1	62
Tabla IV. Configuración Grid2.....	63
Tabla V. Porcentajes de la degradación del desempeño de las cinco mejores estrategias para el Grid 1 y el Grid 2	72
Tabla VI. Valores teóricos y experimentales para <i>MCT</i> , <i>MLB</i> y <i>BN</i>	72

Capítulo I

I. Introducción

I.1 Antecedentes y motivación

En 1998 se introdujo el término Grid para denotar una infraestructura de cómputo distribuido con el objetivo de solucionar problemas dentro de ciertas organizaciones virtuales¹. Tales organizaciones (libres de unirse o dejar el Grid) varían en su propósito individual, alcance, tamaño, estructura y sociología (Kesselman y Foster, 1998).

Los recursos disponibles, la carga del sistema y el poder de cómputo difieren entre los sistemas Grids debido a la heterogeneidad, naturaleza dinámica y autonomía de los mismos (Sgall, 1998; Zhuk *et al.*, 2004). Por ello, dentro de la calendarización en Grid es necesario un proceso eficiente y automático para la asignación de trabajos computacionales a los recursos disponibles.

En Garey y Johnson, (1979) se demuestra que el problema de calendarizar trabajos en un conjunto de recursos heterogéneos, con el objetivo de minimizar el tiempo de terminación del calendario pertenece a los problemas NP-Complejos² (tales como donde los trabajos requieren una unidad de tiempo y dos procesadores para su ejecución, ver (Ullman, 1975; Blazewicz *et al.*, 2007)). Así como este ejemplo, se ha comprobado que muchos de los problemas de calendarización pertenecen a la clase de problemas NP-Difíciles³ (como el problema de calendarización en dos máquinas paralelas con el objetivo de minimizar el tiempo de terminación del calendario (problema *Partición*), ver sección III.4.1) por lo cual,

¹ Una organización es una entidad administrativa que agrupa usuarios y recursos computacionales.

² Un problema \mathcal{P} pertenece a NP-Complejos si y solo si \mathcal{P} pertenece a NP y cualquier otro problema $\mathcal{P}' \in NP$ se puede reducir polinomialmente a \mathcal{P} , ver Apéndice A.

³ Un problema \mathcal{P} pertenece a NP-Difícil si cualquier problema \mathcal{P}' se reduce polinomialmente a él, pero \mathcal{P} no necesariamente pertenece a NP, ver Apéndice A.

la mayoría de las investigaciones se han enfocado al diseño de algoritmos de aproximación⁴.

A pesar de la gran cantidad de algoritmos desarrollados no existe ningún algoritmo adecuado para todos los escenarios Grid. Por lo tanto, una alternativa es seleccionar el mejor algoritmo que se adecue al Grid, de acuerdo a la heterogeneidad entre las tareas, máquinas y conectividad de la red. Por otro lado, las estrategias y algoritmos de calendarización tradicionales no se pueden aplicar directamente para la calendarización de trabajos dentro de un sistema Grid, debido a las diferencias en las características principales de estos sistemas. Para mayor información sobre estas características ver (Schwiegelshohn *et al.*, 2008; Kwok *et al.*, 2006; Berman, 1999).

La demanda en el servicio de un Grid computacional continúa creciendo enormemente, derivando en la búsqueda de una calendarización cada vez más efectiva. En años recientes, el desempeño de los recursos y la infraestructura del Grid se han mejorado contribuyendo a la implementación de nuevas soluciones. En la actualidad, existen varios sistemas de calendarización tales como: Condor (University of Wisconsin Madison, 1988), Legion (University of Virginia, 1993), LSF (Platform Computing, Inc., 2008), Gridbus (University of Melbourne, 2004), Globus (University of Chicago 2002; University of Chicago 1997); sin embargo la desventaja de estos sistemas es que se basan en la adaptación de modelos de calendarización tradicionales. Por ello, el problema de buscar un mejor desempeño de calendarización en Grids, sigue siendo un reto para los investigadores.

La carga dinámica y la disponibilidad de los recursos requieren mecanismos que continuamente monitoreen su estatus. De igual forma, el mapeo de los trabajos en un Grid computacional debe ser adaptivo, escalable y tolerante a fallas debido a la naturaleza dinámica de estos ambientes.

⁴ De acuerdo a Vazirani, (2001) un algoritmo de aproximación produce una solución (que es posible calcular teóricamente) factible muy "cercana" al óptimo.

También se debe considerar que la arquitectura de un Grid se puede dividir en varios niveles. En un Grid de un nivel las máquinas se comunican entre sí o por medio de un calendarizador central siendo esta una limitación para un Grid de gran escala. En cambio en un Grid de dos niveles, las tareas llegan a un calendarizador el cual bajo alguna estrategia de asignación dada selecciona una máquina adecuada. Posteriormente, cada máquina aplica un calendarizador local para calendarizar las tareas que le fueron asignadas en el nivel anterior (para más detalles ver sección II.4.).

Por esta razón, el presente trabajo provee de nuevos resultados teóricos y experimentales dentro de la calendarización en línea en un Grid computacional de dos niveles, con la finalidad de encontrar nuevas cotas para el peor caso que permitan un crecimiento y funcionamiento eficiente del sistema.

A continuación, se describe brevemente el trabajo previo en el cual se sustenta gran parte del documento, su importancia y objetivos. Finalmente, se da una descripción general del documento.

I.2 Trabajos previos

En esta sección se presentan algunos resultados previos de trabajos relacionados al tema de investigación.

Dentro de la calendarización de trabajos en un Grid computacional de dos niveles, se han realizado algunos estudios teóricos que se han enfocado al análisis de diversas estrategias de asignación combinándolas con diferentes algoritmos de calendarización local para observar su desempeño.

Multiple Parallel Scheduling (MPS) denota el problema de calendarizar trabajos dentro de un conjunto de máquinas paralelas en un Grid. *MPS* se compone por dos etapas,

en la primera etapa se utiliza una estrategia de asignación (MPS_Alloc) dada. En la segunda etapa se aplica un algoritmo de calendarización local (PS): $MPS = MPS_Alloc + PS$, para más detalles referirse a la sección III.2.

A continuación se enlistan algunas de las estrategias y algoritmos de calendarización conocidos dentro de la literatura. Para ver más detalle de cada estrategia de asignación y algoritmo de calendarización referirse a las secciones III.3 y III.4, respectivamente:

Tabla I. Lista de estrategias de asignación

Abreviación	Estrategia de asignación
<i>Rand</i>	Random
<i>MLp</i>	Carga mínima por procesador
<i>MPL</i>	Carga mínima paralela
<i>Max_AR</i>	Máximo número de recursos disponibles
<i>LBa_Sl</i>	Balanceo de la carga mínima por tamaño
<i>LBal_T</i>	Balanceo de la carga mínima por tiempo
<i>LBal_W</i>	Balanceo de la carga mínima por trabajo
<i>MLB</i>	Cota inferior mínima del tiempo de finalización
<i>MCT</i>	Tiempo de terminación mínimo
<i>MWT</i>	Menor tiempo de espera
<i>MWWT_T</i>	Menor tiempo de espera ponderado por tiempo
<i>MWWT_S</i>	Menor tiempo de espera ponderado por tamaño
<i>MWWT_W</i>	Menor tiempo de espera ponderado por trabajo
<i>MU</i>	Utilización mínima
<i>MST</i>	Menor tiempo de inicio
<i>MTA</i>	Menor tiempo de permanencia en el sistema
<i>MWTA</i>	Menor tiempo de permanencia en el sistema ponderado

Tabla II. Lista de algoritmos de calendarización local

Abreviación	Estrategia de asignación
<i>BL</i>	Bottom-Left
<i>BLD</i>	Bottom-Left Decreasing
<i>LSF</i>	Primero el tamaño más grande
<i>SFJ</i>	Primero el tamaño más pequeño
<i>FCFS</i>	Primero en llegar-primero-primero en ser atendido
<i>BF</i>	Backfilling
List	Calendarización en lista

Zhuk *et al.*, (2004) se enfocan en un estudio comparativo de diversas estrategias en un modelo fuera de línea donde se conoce todo sobre el sistema, así como de las tareas. Las estrategias analizadas se componen de dos partes, en la primera parte se utiliza la estrategia *MCT* para la asignación de las tareas a las máquinas disponibles dentro del sistema. En la segunda parte se utilizan las heurísticas *BL*⁵ y su variante *BLD* como algoritmos de calendarización local y realizan la ejecución de las tareas en cada máquina de forma independiente.

Demuestran que las estrategias *MCT + BL* y *MCT + BLD* no pueden garantizar un factor de competitividad⁶ constante. Asimismo, introdujeron un nuevo enfoque denominado “estrategia de admisibilidad” (ver III.5). Aplicando esta nueva estrategia demuestran que *MCT_a + BLD* puede garantizar un factor de competitividad constante de 10.

Posteriormente, Tchernykh *et al.*, (2006) retomaron el estudio anterior y lo ampliaron al enfocarse a diferentes estrategias de asignación, utilizando como algoritmo de calendarización local *LSF*, el cual ejecuta las tareas de forma decreciente de acuerdo a su grado de paralelismo.

⁵ En la heurística *BL* los trabajos se colocan lo más abajo que se pueda dentro del calendario y después lo más a la izquierda posible. En *BLD* los trabajos son ordenados de forma decreciente de acuerdo a su grado de paralelismo y se calendarizan igual que en *BL*.

⁶ El factor de competitividad de la estrategia *A* está definido como $\rho_A \leq \sup \frac{c_{\max}(A)}{c_{\max}}$ para todas las entradas del problema

En este estudio, demuestran que las estrategias $(ML, MPL, MCT, MLB) + LSF$ no pueden garantizar un factor de competitividad constante. En cambio, aplicando admisibilidad en $(MCT, MLB)_a + LSF$ logran un factor con valor de 10.

Los estudios anteriores están enfocados a un modelo fuera de línea. Por otro lado, Tchernykh *et al.*, (2008) estudiaron el caso de un modelo en línea donde el tiempo de ejecución de las tareas y su tiempo de procesamiento se desconocen de antemano. En este estudio utilizan como algoritmo de calendarización local (PS), el algoritmo presentado por (Naroska y Schwiegelshohn, 2002) cuyo factor de competitividad es $\rho = 2 - \frac{1}{m}$, (donde m denota la cantidad de máquinas dentro del sistema) siendo para nuestro conocimiento el mejor algoritmo de calendarización local en un modelo no clarividente.

Dentro de su trabajo Tchernykh *et al.*, (2008) generalizaron la relación de admisibilidad propuesta por Zhuk *et al.*, (2004) al introducir un nuevo parámetro, llamado *factor de admisibilidad* ($0 < a \leq 1$). Este parámetro define la tasa de admisibilidad para la asignación de una tarea dada tomando en cuenta la carga de trabajo. Con esta modificación demuestran que la estrategia $MLB_a + PS$, bajo ciertas limitaciones en la carga tiene como factor de competitividad $\rho \leq 5 - \frac{1}{m}$ para un valor de $a = 1/2$ (definición original de admisibilidad).

En un estudio posterior, Tchernykh *et al.*, (2010) proponen otro resultado para la estrategia $MLB_a + PS$ con el mismo modelo de calendarización bajo cualquier carga (caso general). El resultado propuesto es un factor de competitividad igual a 17 para un valor de $a = 1/2$.

I.3 Importancia de la investigación

En contraste con la gran cantidad de resultados experimentales, parte del presente trabajo se enfoca a un análisis teórico dentro de la calendarización de trabajos en un Grid computacional de dos niveles.

El análisis está encaminado para obtener una cota del factor de competitividad para el peor caso de las estrategias propuestas y determinar qué tan alejados se encuentran los resultados generados con respecto a los resultados óptimos. La contribución principal de la investigación, consiste en la aportación de resultados teóricos para el problema de calendarización de trabajos paralelos en un sistema Grid de dos niveles, aplicando las estrategias de asignación *MCT* y *MLB* con el mejor algoritmo de calendarización local conocido, utilizando la estrategia de asignación admisible dentro de un modelo en línea.

I.4 Objetivo general

Analizar teóricamente la calendarización de trabajos paralelos en línea con las estrategias de asignación *MCT* y *MLB* en un Grid computacional de dos niveles con asignación admisible para la obtención de una cota del factor de competitividad en el peor caso.

Implementar y analizar experimentalmente el desempeño de la nueva estrategia de asignación basada en la estrategia de Bar-Noy *et al.*, (2001) y comparar su desempeño con estrategias de asignación ya existentes.

I.4.1 Objetivos específicos

1. Definir el problema de calendarización en un Grid computacional de dos niveles en un modelo en línea bajo el esquema de asignación admisible.

2. Analizar de forma teórica las estrategias de asignación MCT y MLB para calendarización en Grid de dos niveles utilizando como base la estrategia de asignación admisible y obtener resultados para el peor caso.
3. Realizar comparaciones teóricas de los resultados obtenidos con respecto a resultados de estrategias ya existentes.
4. Diseñar y analizar la estrategia de asignación *Carga mínima (Bar-Noy)* basada en el trabajo de Bar-Noy *et al.*, (2001) (diseñada para el balanceo de la carga en una red de servidores jerárquicos) aplicada a un modelo de calendarización en un Grid.
5. Implementar las estrategias dentro del simulador Teikoku y observar su desempeño comparado con respecto a otras estrategias de asignación ya conocidas.

I.5 Organización del documento

En el Capítulo II se introducen los conceptos dentro del problema de calendarización y se explican las diferentes estructuras que existen de un Grid computacional. En el Capítulo III se define el problema de calendarización en el que se enfoca la investigación, también se define las estrategias de asignación y algoritmos de calendarización que existen dentro de la literatura. En este capítulo también se explica a detalle la estrategia propuesta por Bar-Noy *et al.*, (2001) y la estrategia de asignación para un modelo de calendarización que se derivó de esta estrategia. En el Capítulo IV se expone el análisis teórico de las estrategias (MCT_a , MLB_a) + PS junto con los resultados relevantes. En el Capítulo V se muestran los resultados obtenidos de las simulaciones realizadas a las estrategias de asignación diseñadas. Asimismo, se compara su desempeño con la evaluación de las estrategias de asignación MCT , MLB , MLp y MPL comúnmente utilizadas en la literatura. Finalmente, en el Capítulo VI se exponen las conclusiones finales de este trabajo de investigación así como el trabajo a futuro.

Capítulo II

II. Marco Teórico

En este capítulo se presenta el marco teórico de la tesis, se introducen los conceptos de calendarización de trabajos paralelos en un Grid computacional, tipos de estrategias de asignación y algoritmos de calendarización con los que se trabaja, así como un marco conceptual de las diferentes arquitecturas de sistemas Grid.

II.1 Balanceo de la carga

El problema del balanceo de la carga se puede definir de la siguiente manera: Se tienen m máquinas paralelas y un número de tareas independientes; las tareas llegan durante tiempos arbitrarios. Cada tarea debe ser asignada de forma inmediata a exactamente una máquina, incrementando la carga de esta máquina. La meta del balanceo de la carga es que cada máquina ejecute una parte equitativa de la carga de trabajo total (Azar, 1998).

Un ejemplo simple dentro del balanceo de la carga en un ambiente en línea es el siguiente: considerando el caso que cada máquina representa un canal de comunicación con un ancho de banda limitado. El problema es asignar cada petición que llega solicitando un ancho de banda específico a uno de los canales. Al realizar la asignación se incrementa la carga en el canal, es decir, se incrementa el porcentaje de uso del ancho de banda. La carga es incrementada por el tiempo de duración asociado a la petición.

La métrica de medición del desempeño que generalmente se utiliza es minimizar la carga máxima. Esta métrica se enfoca en la peor máquina y es equivalente a minimizar el tiempo de terminación del calendario para problemas de calendarización.

II.2 Calendarización

La calendarización de tareas ha sido uno de los problemas más difíciles tanto en computación paralela como en la computación distribuida. Es conocido que en general el problema de calendarización es NP-Completo. Dentro de la literatura se define *calendarización* como la acción de asignar un conjunto de tareas a un conjunto de recursos disponibles en tiempo y espacio, minimizando el tiempo de terminación de las tareas (Jiang *et al.*, 2007).

El problema de calendarización se puede caracterizar por medio de tres conjuntos: un conjunto de n tareas $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$, un conjunto de m procesadores o máquinas $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$ y un conjunto de s recursos adicionales $\mathcal{R} = \{R_1, R_2, \dots, R_s\}$, (Blazewicz *et al.*, 2007).

Como los problemas de calendarización varían en función de las características de las tareas, procesadores (máquinas) y criterios de optimización que los componen Graham *et al.*, (1979) proponen la notación de tres campos, que permite caracterizar un problema de calendarización de manera fácil y sencilla. La notación se compone de tres campos $\alpha|\beta|\gamma$. El primer campo (α) describe las características de los procesadores, el segundo campo (β) denota las características de las tareas y los recursos. Finalmente, el tercer campo (γ) especifica el criterio de optimización para el problema, referirse al Apéndice B para una definición completa de la notación de tres campos.

En términos generales, la calendarización supone un conjunto de recursos y un conjunto de usuarios que son atendidos por los recursos de acuerdo a ciertas políticas. Considerando la naturaleza y las limitaciones tanto por parte de los usuarios como de los recursos, el problema es encontrar una política eficaz que permita el acceso y el uso de estos recursos por parte de varios usuarios; tratando de optimizar algún criterio de desempeño dado tal como la longitud del calendario (El-Rewini *et al.*, 1994).

De igual forma, la calendarización en un sistema Grid consiste en someter las tareas dentro del sistema y asignar estas tareas a los recursos disponibles optimizando algún criterio.

II.3 Calendarización en un Grid computacional

La transición de las metacomputadoras (organización de grandes redes de computadoras antes de la aparición del Grid) al Grid computacional tomó lugar a mediados de 1990 con la introducción del middleware⁷, el cual fue diseñado como una infraestructura para el soporte del procesamiento en línea y de aplicaciones de alto desempeño. La evolución de la tecnología del Grid se muestra en la Figura 1. Los primeros experimentos del Grid funcionaban bajo herramientas o middleware especializados que estaban enfocados en la comunicación de paso de mensajes entre los nodos de la red.

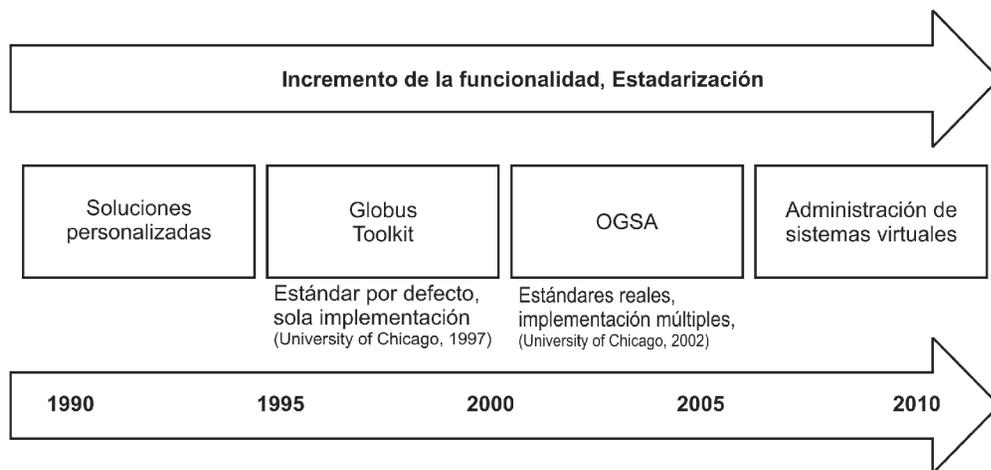


Figura 1. Evolución en la tecnología del Grid computacional (Moallem, 2009)

Dentro de un ambiente Grid existen diferentes mecanismos con diferentes metas cada uno, por ello Berman (1999) señala que la calendarización es “*fundamental*” para alcanzar un buen desempeño del sistema. Algunos de estos mecanismos son: el

⁷ *Middleware es un software de computadora que conecta componentes de software o aplicaciones para que puedan intercambiar información entre éstas, ver (Talia et al., 2008)*

calendarizador de trabajos, el calendarizador de recursos, entre otros. Por lo tanto, un calendarizador de alto desempeño es una parte importante y crucial dentro de ambientes Grid. Berman (1999) define a un calendarizador de alto desempeño como un software el cual utiliza modelos de calendarización para predecir el desempeño, determinar las actividades de calendarización en base a estos modelos e implementar el calendario resultante.

A pesar que el problema de calendarización ha sido extensamente estudiado para problemas de calendarización de trabajos en máquinas paralelas, muchos de los resultados obtenidos no pueden ser aplicados a un ambiente Grid dado que este difiere de la calendarización tradicional por sus características únicas (Zhu, 2003).

En un principio se intentó utilizar las diversas estrategias para ambientes con máquinas de múltiples procesadores simétricos (*SMP*), máquinas con procesadores paralelos masivos (*MPP*) y cluster de estaciones de trabajo (*COW*) e incorporarlas en ambientes Grid, dado que en estos ambientes se requiere una coordinación cuidadosa de recursos, comunicaciones y trabajos para un buen desempeño. Sin embargo, estos modelos de calendarización generalmente producen en la práctica calendarios de poca calidad en ambientes Grid (Rajasekaran and Reif, 2008).

Esto se presenta dado que en la calendarización tradicional se busca una alta utilización del recurso, pero en un Grid computacional se involucra una mayor cantidad de recursos cada uno con sus propias políticas de calendarización. Por lo que se requiere de un sistema calendarizador dinámico y eficiente que se encuentre por encima de cada calendarizador local para alcanzar un desempeño eficiente y una buena utilización del sistema.

Las principales diferencias entre la calendarización en un ambiente Grid y la tradicional se originan debido a ciertos problemas intrínsecos del Grid que surgen al tratar de alcanzar una alta utilización del sistema.

II.3.1 Problemas dentro de un ambiente Grid

A pesar de los esfuerzos para buscar una buena utilización del Grid en los últimos años, existen ciertos problemas propios del Grid que todavía no se han resuelto. Estos problemas se exponen a continuación:

1. *Calendarizadores con múltiples niveles* (Tchernykh *et al.*, 2006): Una de las posibles arquitecturas de un Grid involucra múltiples niveles de calendarización. En el nivel más alto, los calendarizadores del Grid cuentan con una visión más amplia del sistema, sin embargo no se involucran dentro de los recursos donde se ejecutarán las tareas. En el nivel más bajo, los calendarizadores locales administran los recursos donde se ejecutarán las tareas de forma independiente a niveles superiores.
2. *Falta de control sobre los recursos* (Rajasekaran y Reif, 2008; Kurowski *et al.*, 2006): Uno de los mayores retos dentro de la calendarización en un Grid se debe a la falta de control total de los recursos del sistema. Los recursos dentro de un Grid se encuentran dispersos a través de múltiples dominios administrativos que comúnmente poseen sus propias políticas y reglas, limitando al calendarizador del Grid. Es posible que ciertos dominios estén más dedicados a los trabajos locales que a trabajos de otros dominios, por lo tanto, los recursos podrían no ser accesibles de forma equitativa para todos los usuarios. Debido a este factor, existe la posibilidad que no se produzca el mejor calendario posible incluso si los recursos para un trabajo dado se encuentren disponibles.
3. *Naturaleza dinámica de los recursos* (Rajasekaran y Reif, 2008): Uno de los mayores desafíos dentro de la calendarización en un Grid computacional, es diseñar un calendarizador que sea capaz de lograr un desempeño eficiente (para la comunidad de usuarios y las organizaciones involucradas) ante las variaciones (sin previo aviso) en la disponibilidad (los recursos pueden agregarse o eliminarse del Grid) y capacidad de los recursos. Estas variaciones se pueden presentar dado que

los recursos son utilizados tanto por los usuarios del Grid como por usuarios locales.

4. *Diversidad de aplicaciones* (Rajasekaran y Reif, 2008): Las aplicaciones pueden variar de disciplina o campos de estudio como por ejemplo bio-informática, física de la energía, astronomía, meteorología y economía (EC-cofundado GridTalk project, 2008). Esto implica la implementación de diferentes modelos para cada aplicación.
5. *Conflicto de objetivos* (Kurowski *et al.*, 2006): A menudo las organizaciones que proveen los recursos dentro del Grid y los usuarios tiene diferentes metas a alcanzar. Desde optimizar el desempeño de una aplicación hasta obtener el mejor rendimiento o minimizar el tiempo de respuesta del sistema. Pascual *et al.*, (2007) estudiaron este tipo de conflictos y bajo algunas suposiciones en un modelo fuera de línea demuestran que siempre es posible mejorar el desempeño de todo el sistema sin perjudicar los objetivos individuales que persiguen las organizaciones involucradas.

II.4 Estructuras de un Grid computacional

La estructura de un calendarizador se ve reflejado en la arquitectura del Grid computacional (Rajasekaran y Reif 2008; Foster y Kesselman 2004; Ernemann *et al.*, 2002). Esta estructura depende de la cantidad de recursos donde asignarán las tareas y sus respectivos dominios. Un Grid consiste de un número de sitios u organizaciones virtuales, donde cada una cuenta con un conjunto de m recursos computacionales que participan dentro del Grid, ver Figura 2.

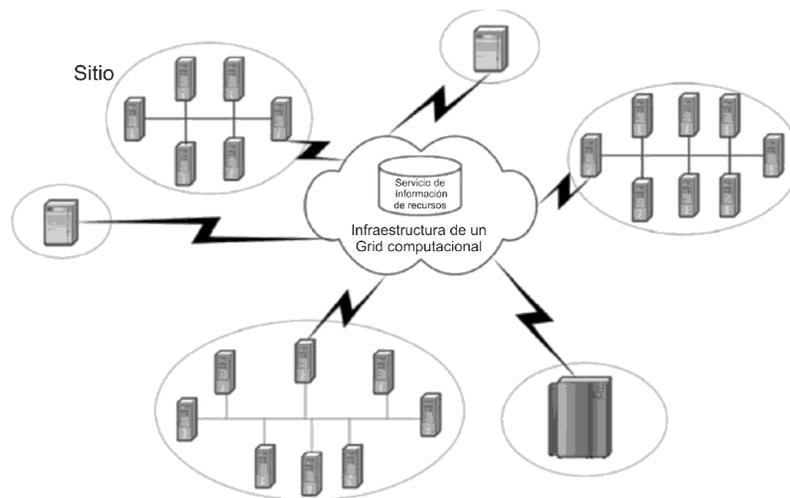


Figura 2. Modelo de un Grid (Rajasekaran y Reif, 2008)

Los recursos en ambientes de cómputo tradicionales tales como supercomputadoras, clusters, etc. son en general estáticos. Sin embargo, en un Grid computacional esta característica es totalmente diferente.

El calendarizador en un Grid debe tomar en cuenta las siguientes características fundamentales:

1. La *escalabilidad* es un aspecto crucial dado que un Grid puede crecer o disminuir de tamaño.
2. Un calendarizador del Grid debe realizar el mapeo de recursos de acuerdo a las limitantes de la calidad de servicio de las aplicaciones.
3. Dentro de un Grid las fallas se encuentran sin excepción. Por lo tanto se debe considerar un enfoque tolerante a fallas.

En términos generales, un calendarizador del Grid puede ser clasificado en tres categorías: centralizado, descentralizado y jerárquico, (Rajasekaran y Reif 2008; Hamscher *et al.*, 2000).

II.4.1 Centralizada

En esta estructura se tiene un calendarizador central el cual contiene toda la información del estado del sistema. Esto resulta en calendarios con un buen desempeño en comparación a las otras dos categorías. En este ambiente centralizado cuando una tarea no puede ser ejecutada inmediatamente se almacena en una cola de trabajo central hasta que algún recurso pueda ejecutarla, ver Figura 3. En este tipo de estructuras escalar el sistema representa una de sus mayores desventajas. Además, está propensa a la posibilidad de fallas.

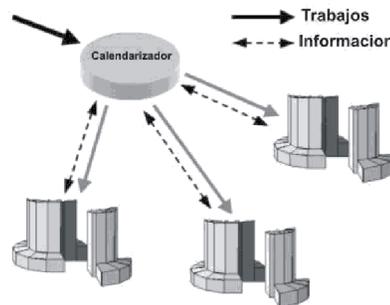


Figura 3. Calendarización centralizada (Hamscher *et al*, 2000)

II.4.2 Descentralizada

En esta estructura a diferencia de la anterior no se cuenta con un calendarizador central, en cambio los calendarizadores son distribuidos e interactúan entre sí. Asimismo, la información sobre el estado actual del sistema no se encuentra localizada en un solo punto. Esto implica que un calendarizador central produce generalmente calendarios sub-óptimos dado que se deben sincronizar las tareas para garantizar una ejecución simultánea de cada programa paralelo. Sus mayores ventajas es que permite la autonomía en cada calendarizador permitiendo que los calendarios locales puedan especializarse en las necesidades de cada recurso, también es tolerante a fallas y es fácil de escalar.

Dentro de esta estructura se tienen dos arquitecturas dependiendo del manejo de las tareas que ingresan al sistema: comunicación directa y comunicación a través de una pila de trabajos.

II.4.2.1 Comunicación directa.

Se permite el envío y la recepción de tareas entre los calendarizadores. Cuando una tarea no puede ser ejecutada inmediatamente en una máquina, se envía a otro recurso que se encuentre disponible para su ejecución, ver Figura 4. Sin embargo, se tiene que considerar el tiempo originado por dicha migración al tiempo de procesamiento de la tarea.

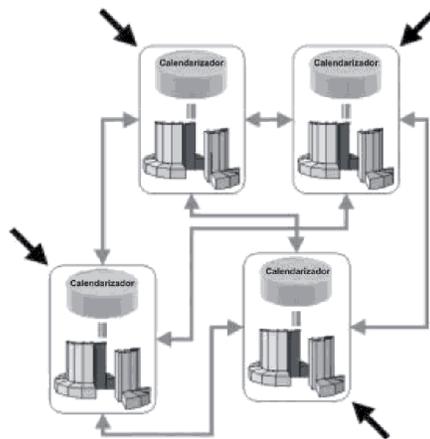


Figura 4. Calendarización descentralizada con comunicación directa (Hamscher *et al.*, 2000)

II.4.2.2 Comunicación a través de una pila central de trabajos.

En esta estructura cuando una tarea no puede ser iniciada inmediatamente en lugar de buscar un recurso disponible se manda a una pila de trabajos. En esta situación los calendarizadores locales pueden mandar o tomar tareas de la pila en cuanto tengan espacio disponible, ver Figura 5.

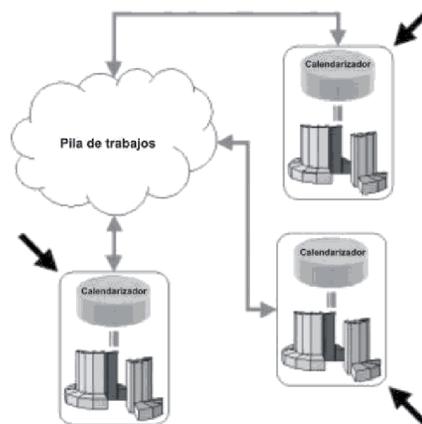


Figura 5. Calendarización descentralizada con pila de trabajos (Hamscher *et al.*, 2000)

II.4.3 Jerárquica

Una estructura jerárquica se puede considerar como un modelo híbrido. En esta estructura se tiene un calendarizador central en el que ingresan las tareas, mientras cada máquina utiliza de forma independiente un calendarizador local, ver Figura 6. Hamscher *et al.*, (2000) consideran esta estructura como parte de una estructura centralizada, dado que las tareas son ingresadas al calendarizador central antes de ser mandadas a cada máquina del sistema. La ventaja principal, es que pueden utilizarse diferentes políticas para calendarizar las tareas de manera local o global.

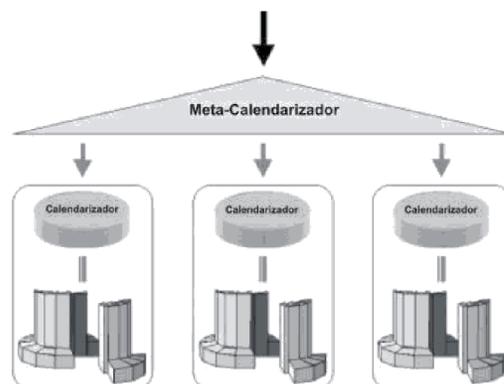


Figura 6. Estructura jerárquica de un Grid computacional (Hamscher *et al.*, 2000)

Capítulo III

III. Definición del problema

En este capítulo se define formalmente el problema de calendarizar trabajos paralelos en un Grid jerárquico de dos niveles. Se presenta la definición del enfoque de admisibilidad. Por último, se presenta el modelo de calendarización para un Grid de dos niveles utilizando admisibilidad y las estrategias utilizadas dentro de la investigación.

III.1 Formulación del problema

Una representación precisa de un Grid computacional para cada problema en particular, es una tarea muy compleja debido a las características del mismo Grid. El modelo de calendarización en el cual se basa la investigación, está compuesto por las siguientes características.

La investigación se enfoca en el problema de calendarización de trabajos (tareas) paralelos con el objetivo de minimizar el tiempo de terminación del calendario. Dentro del modelo se consideran m máquinas paralelas $\{N_1, N_2, \dots, N_m\}$, donde el tamaño de cada máquina N_i se compone por m_i procesadores idénticos⁸. Sin pérdida de generalidad, se supone que las máquinas se encuentran ordenadas de forma creciente con respecto a su tamaño ($m_1 \leq m_2 \leq \dots \leq m_m$).

⁸ *Procesadores idénticos: tienen el mismo tiempo de procesamiento para la ejecución de las tareas. Procesadores uniformes: Los procesadores difieren en sus velocidades, donde la velocidad b_i de cada procesador es constante y no depende de las tareas. Procesadores no relacionados: Las velocidades de procesamiento en cada procesador dependen de la tarea que se esté ejecutando (Blazewicz et al., 2007).*

A su vez se tiene un conjunto de n tareas rígidas $\{J_1, J_2, \dots, J_n\}$. Estas tareas necesitan un número fijo de procesadores para su ejecución y dicha cantidad no puede ser modificada hasta terminar la ejecución de la tarea. Cada tarea J_j está caracterizada por $(r_j, size_j, p_j, p'_j)$ donde:

- $r_j \geq 0$, denota el instante en el que una tarea J_j está disponible para su ejecución.
- $1 \leq size_j \leq m_m$, denota el grado de paralelismo de la tarea J_j
- p_j , denota el tiempo de procesamiento real de la tarea J_j .
- p'_j denota la estimación (por parte del usuario) del tiempo de procesamiento.

Se trabaja en un modelo de espacio compartido⁹, el cual define que una tarea J_j debe ser ejecutada sin interrupciones en $size_j$ procesadores de alguna máquina.

Se trabaja en un modelo en línea, donde el tiempo de llegada de las tareas al sistema es desconocido. También se considera un esquema de calendarización no clarividente en el cual, el tiempo de procesamiento de las tareas se desconoce de antemano y sólo se conoce hasta terminar la ejecución de la tarea.

Se define p_{max} como el tiempo máximo de procesamiento de todas las tareas ($\max_{1,n}\{p_j\}$). Asimismo, se define a $w_j = size_j \cdot p_j$ como el trabajo asociado a J_j conocido como su área o su consumo de recursos dentro del calendario. Finalmente, $g(J_j) = N_i$ denota que la tarea J_j es asignada a la máquina N_i . Donde n_i denota el número de tareas asignadas a la máquina N_i .

⁹ En este método se le asigna a un trabajo acceso exclusivo a un conjunto de recursos para su ejecución. En otras palabras, el número de procesadores para ejecutar un trabajo es igual al solicitado por el trabajo. Por lo tanto, un calendarizador necesita seleccionar un trabajo para su ejecución en un orden apropiado para poder ejecutar varios trabajos simultáneamente de forma eficiente.

III.2 Calendarización en un Grid jerárquico de dos niveles

La estructura del Grid que se considera es jerárquica en dos niveles. Para este tipo de estructura (durante la investigación) se aplica la estrategia *MPS* (*Multiple Parallel Scheduling*) la cual denota el problema de calendarizar trabajos dentro de un conjunto de máquinas paralelas en un Grid (GP_m) expresado como $GP_m|r_j, size_j|C_{max}$ (bajo la notación de tres campos (Graham *et al.*, 1979)). Por otro lado, para describir el problema de calendarizar trabajos dentro de una máquina paralela *PS* (*Parallel Scheduling*) se expresa como $P_m|r_j, size_j|C_{max}$.

La estrategia *MPS* se compone de dos etapas: $MPS = MPS_Alloc + PS$ (ver

Figura 7). En la primera etapa, se selecciona una máquina adecuada para cada tarea utilizando un criterio de selección dado y la estrategia *MPS_Alloc* dada. En la segunda etapa, se aplica un algoritmo de calendarización local *PS* para calendarizar las tareas que fueron asignadas durante la etapa anterior. Se puede observar, que el factor de competitividad del algoritmo *MPS* está limitado inferiormente por el factor de competitividad del mejor algoritmo *PS*.

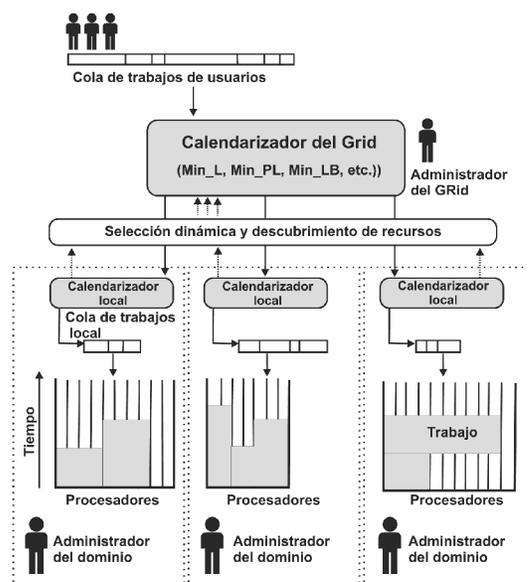


Figura 7. Configuración de Grid computacional de dos niveles (Kurowski *et al.*, 2008).

III.3 Estrategias de asignación

A continuación se presentan las estrategias de asignación. La aplicación de cada una de ellas depende de la cantidad de información que se conoce del sistema y de las características de las tareas.

Para algunas de ellas es necesaria la creación de un pre-calendario para efectuar la asignación de las tareas. En este caso, el calendarizador del Grid manda a cada una de las máquinas disponibles la tarea a calendarizar. Cada máquina crea un pre-calendario utilizando las tareas ya asignadas anteriormente y envía esta información al calendarizador; este último, bajo la estrategia que esté utilizando tomará una decisión adecuada.

A continuación se enlistan las diferentes estrategias de asignación de la tarea a una máquina adecuada dependiendo del grado de información que se tiene sobre el sistema (Ramírez *et al.*, 2010). En cada nivel se conoce el número y tamaño de las máquinas que componen el sistema:

Nivel 1: Una vez que la tarea es sometida se conoce la cantidad de procesadores que requiere para su ejecución. No se tiene información del tiempo de procesamiento de las tareas por lo tanto, se considera una calendarización no clarividente. Pero se puede utilizar la información de las tareas que ya han sido asignadas anteriormente.

1. *Random*: Esta estrategia selecciona de forma aleatoria una máquina del sistema. Es una de las estrategias más simples.
2. *Carga mínima por procesador (MLp)*: Selecciona la máquina con la menor carga por procesador

$$\min_{1 \leq i \leq m} \left\{ \frac{n_i}{m_i} \right\} \quad (1)$$

3. *Máximo número de recursos disponibles (Max_AR)*: Selecciona la máquina con la mayor cantidad de recursos (procesadores) disponibles.

$$\max_{1 \leq i \leq m} \left\{ \frac{\text{avail}_i}{m_i} \right\} \quad (2)$$

4. *Carga mínima paralela (MPL)*: Selecciona la máquina con la menor carga paralela por procesador.

$$\min_{1 \leq i \leq m} \left\{ \sum_{g(J_k)=M_i} \frac{\text{size}_k}{m_i} \right\} \quad (3)$$

5. *Balanceo de la carga mínima (Tamaño) (LBal_S)*: Selecciona la máquina que genera la menor desviación estándar en la distribución de la carga de cada máquina del Grid.

$$\min_{q=1..m} \sqrt{\frac{1}{m} \sum_{i=1}^m (PL_i^q - \overline{PL})^2} \quad (4)$$

donde $PL_{i=1..m}^q = \frac{1}{m_i} \sum_{g_k=i} (\text{size}_k + \text{size}_j^q)$ y size_j^q es el tamaño de la tarea j asignada a la máquina q .

6. *Balanceo de la carga mínima (Tiempo) (LBal_T)*: Selecciona la máquina que genera la menor desviación estándar en la distribución de la carga de cada máquina del Grid.

$$\min_{q=1..m} \sqrt{\frac{1}{m} \sum_{i=1}^m (T_i^q - \overline{T})^2} \quad (5)$$

donde $T_{i=1..m}^q = \frac{1}{m_i} \sum_{g_k=i} (p_k + p_j^q)$ y p_j^q es el tiempo de la tarea j asignada a la máquina q .

7. *Balaceo de la carga mínima (Trabajo) (LBal_W)*: Selecciona la máquina que genera la menor desviación estándar en la distribución de la carga de cada máquina del Grid.

$$\min_{q=1..m} \sqrt{\frac{1}{m} \sum_{i=1}^m (W_i^q - \bar{W})^2} \quad (6)$$

donde $W_{i=1..m}^q = \frac{1}{m_i} \sum_{g_k=i} (w_k + w_j^q)$ y w_j^q es el trabajo de la tarea j asignada a la máquina q .

Nivel 2: En este nivel se conoce la información del nivel 1 además de conocer el tiempo de procesamiento estimado (p'_j) de cada tarea.

1. *Cota inferior mínima del tiempo de finalización (MLB)*: Esta estrategia selecciona la máquina con la menor cota inferior del tiempo de terminación del calendario.

Como el tiempo de ejecución de las tareas no se encuentra disponible de antemano (se trabaja bajo un modelo no-clarividente) se puede utilizar un estimado de este tiempo. Este estimado puede calcularse mediante información histórica o tomar el valor proporcionado por el usuario al momento de someter el trabajo al sistema, ver (Tsafirir *et al.*, 2005).

$$\min_{1 \leq i \leq m} \left\{ \sum_{g(J_k)=M_i} \frac{\text{size}_k \cdot p_k}{m_i} \right\} \quad (7)$$

Nivel 3: Junto con la información de los niveles anteriores se tiene conocimiento de un pre-calendario de cada máquina del sistema. Las estrategias en este nivel requieren de un mayor esfuerzo computacional debido a que realizan el calendario en cada una de las máquinas disponibles para la tarea que se quiere asignar.

1. *Tiempo de terminación mínimo (MCT)*: Esta estrategia selecciona la máquina donde el tiempo de terminación de la tarea J_j sea menor.

$$\min_{1 \leq i \leq m} \{C_{max}^i\} \quad (8)$$

donde, $C_{max}^i = \max_{g(J_k)=M_i} (C_k^i)$ denota el tiempo máximo de terminación de las tareas en la máquina M_i y C_k^i denota el tiempo de terminación de la tarea J_k en la máquina M_i .

2. *Menor tiempo de espera (MWT)*: Selecciona la máquina con el promedio menor de tiempo de espera de las tareas asignadas.

$$\min_{1 \leq i \leq m} \left\{ \sum_{g(J_k)=M_i} \frac{C_k^i - r_k - p_k}{n_i} \right\} \quad (9)$$

3. *Menor tiempo de espera ponderado (Tiempo) (MWWT_T)*: Se selecciona la máquina que tiene el menor promedio del producto del tamaño con el tiempo de espera de cada una de las tareas asignadas.

$$\min_{1 \leq i \leq m} \left\{ \sum_{g(J_k)=M_i} \frac{(C_k^i - r_k - p_k) \cdot p_j}{n_i} \right\} \quad (10)$$

4. *Menor tiempo de espera ponderado (Tamaño) (MWWT_S)*: Se selecciona la máquina que tiene el menor promedio del producto del tamaño con el tiempo de espera de cada una de las tareas asignadas.

$$\min_{1 \leq i \leq m} \left\{ \sum_{g(J_k)=M_i} \frac{(C_k^i - r_k - p_k) \cdot size_j}{n_i} \right\} \quad (11)$$

5. *Menor tiempo de espera ponderado (Trabajo) (MWWT_W)*: Se selecciona la máquina con el menor promedio de tiempo de permanencia de la tarea asignada. En esta ocasión se pondera de acuerdo al trabajo o al espacio que consume la tarea (trabajo) dentro del calendario.

$$\min_{1 \leq i \leq m} \left\{ \sum_{g(J_k)=M_i} \frac{(C_k^i - r_k) \cdot (size_k \cdot p_k)}{n_i} \right\} \quad (12)$$

6. *Utilización mínima (MU)*: Esta estrategia selecciona la máquina con la menor utilización de sus procesadores.

$$\min_{1 \leq i \leq m} \left\{ \frac{W_{total}^i}{C_{max}^i \cdot m_i} \right\} \quad (13)$$

donde, $W_{total}^i = \sum_{g(J_k)=M_i} size_k \cdot p_k$ es el trabajo realizado por la máquina M_i .

7. *Menor tiempo de inicio (MST)*: Selecciona la máquina con el menor tiempo de inicio de ejecución de la tarea que se asignará.

$$\min_{1 \leq i \leq m} \{C_j^i - r_j\} \quad (14)$$

8. *Menor tiempo de permanencia en el sistema (MTA)*: Se selecciona la máquina con el menor promedio de tiempo de permanencia de las tareas asignadas a las máquinas.

$$\min_{1 \leq i \leq m} \left\{ \sum_{g(J_k)=M_i} \frac{(C_k^i - r_k)}{n_i} \right\} \quad (15)$$

9. *Menor tiempo de permanencia ponderado en el sistema (MWTa)*: Esta estrategia selecciona la máquina que tiene el menor promedio de tiempo de permanencia de la tarea asignada ponderado con el tamaño de la misma.

$$\min_{1 \leq i \leq m} \left\{ \sum_{g(J_k)=M_i} \frac{(C_k^i - r_k) \cdot size_k}{n_i} \right\} \quad (16)$$

En el presente trabajo se utiliza en el análisis teórico las estrategias *MLB* y *MCT* con el mejor algoritmo de calendarización local del que se tiene conocimiento. A continuación se mencionan algunos algoritmos de calendarización más utilizados dentro de la literatura.

III.4 Algoritmos de calendarización local

Trabajos como (Feitelson *et al.*, 2005; Albers, 2003; Iovanella, 2002; Braun *et al.*, 2001; Sgall, 1998; Karger *et al.*, 1997) presentan varios algoritmos de calendarización dependiendo del problema de calendarización en el que se enfocan. Muchas de las heurísticas son bastantes simples como el tradicional *FCFS* y otras involucran técnicas mas sofisticadas como algoritmos genéticos.

Uno de los algoritmos más básicos que se conoce es *primero en llegar-primero en ser atendido (FCFS)* ver (Schwiegelshohn y Yahyapour, 1998). En este algoritmo las tareas son ejecutadas de acuerdo al orden en que se someten al sistema. Cuando no existe suficiente espacio para procesar la tarea que se encuentra al principio de la cola, se debe esperar hasta que alguna tarea que esté ejecutándose termine y libere el espacio suficiente.

Una de las mayores desventajas de *FCFS* es que produce una muy baja utilización del sistema. Se han desarrollado varias estrategias que se han enfocado en solucionar esta desventaja como el caso de *FPFS*. Esta estrategia busca dentro de la cola alguna tarea que pueda ser colocada dentro del espacio disponible (Aida *et al.*, 1998).

Por otro lado, otra estrategia propuesta para mejorar la utilización del sistema es *Backfilling* (Lifka, 1995). *Backfilling* es una optimización que trata de mantener un balance entre utilización y el orden propuesto por *FCFS*. Además, se ha demostrado que puede mejorar el rendimiento del sistema visto desde el usuario como del sistema mismo (Ramírez *et al.*, 2007; Wang *et al.*, 2004).

De forma parecida a *FPFS*, *Backfilling* localiza “huecos” y selecciona las tareas dentro de la cola que puedan “rellenar” estos espacios vacíos que de otra forma permanecerían ociosos, siempre y cuando se respete el orden de la cola impidiendo que algunas tareas queden sin ejecutarse.

Existen dos variantes de esta estrategia:

1. *Backfilling agresivo EASY* (Extensible Argonne Scheduling sYstem) (Etsion y Tsafrir, 2005; Srinivasan *et al.*, 2002): En esta estrategia se asigna una reservación a la primer tarea que se encuentra en la cola. Las reservaciones se realizan cuando no se tienen suficientes procesadores para la ejecución de la tarea. Se estima el instante de tiempo cuando la cantidad de procesadores disponibles sea suficiente y se reservan para la tarea. Otros trabajos que puedan adelantarse no podrán violar esta reservación por lo tanto, deberán de terminar antes del tiempo de la reservación o utilizar únicamente los procesadores que no sean requeridos por la primer tarea.
2. *Backfilling conservativo* (Srinivasan *et al.*, 2002): En esta variante, a cada tarea se le asigna una reservación al momento de ingresar al sistema. Nuevamente, se pueden adelantar tareas pequeñas siempre y cuando no retrasen la ejecución de otras tareas que se encuentren primero en la cola.

Ciertos algoritmos de calendarización se pueden clasificar como calendarización en base a prioridades. En este tipo de calendarización a cada tarea se le asigna cierta prioridad (o peso). Posteriormente de acuerdo a esta asignación se van ejecutando las tareas de mayor prioridad a una menor o viceversa dependiendo del algoritmo que se esté utilizando, ver (Aida, 2000; Li y Cheng, 1991). Algunos de estos algoritmos son:

1. *Primero el tamaño más grande (LSF)*: ordena las tareas de forma decreciente, por lo que el calendarizador asignaría primero la tarea de mayor tamaño a los recursos disponibles.

2. *Primero el tamaño más pequeño (SFJ)*: ordena las tareas de forma creciente, por lo que en este caso primero se asignaría la tarea de menor tamaño a los recursos disponibles.
3. *Primero el consumidor de recursos más grande/chico (L/S-RFC)*: las tareas dentro de la cola se ordenan de acuerdo a la cantidad de recursos consumidos ya sea de forma creciente o decreciente. Una vez ordenadas se utiliza *FCFS* para su ejecución.

Para la etapa de calendarización local se pueden utilizar algoritmos simples como *FCFS* o las diferentes variaciones de Backfilling. Sin embargo, estos algoritmos a pesar que se desempeñan razonablemente bien en la práctica, no pueden proveer de un factor de competitividad constante.

En la siguiente sección se explica más a detalle en qué consisten estos tipos de algoritmos y algunos de los resultados más importantes que se han propuesto.

III.4.1 Calendarización en lista

Los algoritmos de calendarización basados en listas han sido estudiados por décadas (Graham, 1966). Aún hoy en día se pueden utilizar algunos de los resultados presentados incluso en ambientes de calendarización como en un sistema Grid.

Un algoritmo de calendarización en lista se puede ver de la siguiente manera: suponiendo que se tiene una lista de tareas ordenada $\mathcal{J} = \{J_1, J_2, J_3, \dots, J_n\}$. En cualquier momento cuando una máquina termina de ejecutar alguna tarea instantáneamente escanea la lista \mathcal{J} desde el inicio hasta encontrar una tarea J_j sin calendarizar. Si se tiene la cantidad suficiente de procesadores para la ejecución de la tarea J_j (donde $size_j \leq m_j$) se ejecuta inmediatamente; en caso contrario, se busca otra tarea dentro de la lista que pueda iniciarse en los procesadores disponibles en este momento. En caso que no se encuentre alguna tarea

lista para ejecutarse, los procesadores quedarán ociosos hasta que alguna tarea pueda iniciarse (Graham, 1966).

Se ha demostrado que en calendarizaciones basadas en lista bajo ciertas variaciones en algunos parámetros se puede ocasionar un aumento en la longitud del calendario resultante (Graham, 1969; Graham, 1966).

Estas posibles variaciones son:

1. Cambiar el orden de las tareas en la lista
2. Relajar la relación de precedencia (orden-parcial $<$)¹⁰
3. Cambiar el número de procesadores de n a n' donde $n < n'$

Por otro lado, unos de los resultados más simples sobre el desempeño en el peor caso de un calendario en lista (factor de competitividad) en un modelo fuera de línea fue propuesto por Graham, (1966). Los autores demuestran que el mejor límite superior está dado por la expresión:

$$\frac{C_{max}(L)}{C_{max}^*} \leq 2 - \frac{1}{m}.$$

donde $C_{max}(L)$ ¹¹ denota la longitud del calendario generado por un algoritmo basado en lista. C_{max}^* es la longitud del calendario óptimo y m denota el número de procesadores paralelos.

En dos resultados posteriores, se demuestra que se puede obtener este factor de competitividad utilizando trabajos paralelos dentro de un modelo de calendarización fuera de línea, ver (Garey y Graham, 1975). Asimismo, se demuestra que este factor también se

¹⁰ Un conjunto parcialmente ordenado (poset, por sus siglas en inglés) consiste en un conjunto con una relación binaria que indica que, para ciertos pares de elementos del conjunto, uno de los elementos precede a otro. Estas relaciones se llaman órdenes parciales para reflejar el hecho de que no todos los pares de elementos de un poset necesitan estar relacionados (Blazewicz et al., 2007).

¹¹ El factor $\frac{C_{max}(L)}{C_{max}^*}$ es el máximo valor del cociente de un calendario y representa la capacidad de un algoritmo para generar una solución con el menor costo posible.

puede obtener en modelos de máquinas paralelas en situaciones donde se desconozca el tiempo de procesamiento de las tareas ver (Naroska y Schwiegelshohn, 2002).

El problema de calendarización en lista en una máquina paralela con el objetivo de minimizar la longitud del calendario, se clasifica dentro de los problemas NP-Difícil (Graham *et al.*, 1979).

Schwiegelshohn *et al.*, (2008) demuestran que no existe un algoritmo de tiempo polinomial que garantice un calendario con un factor $\frac{C_{max}(L)}{C_{max}^*} < 2$ para el problema de calendarizar trabajos paralelos en múltiples máquinas ($GP_m | size_j | C_{max}$) a menos que $P=NP$ de acuerdo a la teoría de la complejidad computacional (ver Apéndice A). Un ejemplo sencillo es realizar un mapeo del problema de calendarización en dos máquinas paralelas al problema de *Partición* que se ha demostrado ser NP-Difícil (Blazewicz *et al.* 2007; Karp, 1972).

Con el resultado anterior Schwiegelshohn *et al.*, (2008) demuestran que un calendario basado en lista no puede ser aplicado en un ambiente Grid. Así mismo, los algoritmos basados en lista no pueden garantizar un factor constante ocasionado por el problema de balanceo de la carga ver (Bar-Noy *et al.*, 2001).

III.5 Métricas de desempeño

Existen varios criterios de optimización o métricas de desempeño que sirven para evaluar la calidad del calendario resultante. Para entendimiento de las métricas se define la siguiente notación:

C_i = tiempo de terminación de la tarea i

F_i = tiempo de respuesta de la tarea i , que está definido como $F_i = C_i - r_i$

w_i = trabajo (peso) de la tarea i

r_i = tiempo de llegada de la tarea i

p_i = tiempo de procesamiento de la tarea i

tw_i = tiempo de espera de la tarea i , el cual está dado por $tw_i = C_i - p_i - r_i$

A continuación se muestran algunas de las métricas comúnmente utilizadas en la literatura, ver (Feitelson *et al.*, 1997; Feitelson, 2001; Blazewicz *et al.*, 2007; Oyetunji, 2009):

1. *Máximo tiempo de terminación* (makespan): También se le conoce como longitud del calendario. Es el tiempo de terminación de la última tarea. Este criterio es utilizado típicamente para medir el nivel de utilización de las máquinas. Se puede enfocar como un problema donde se desea minimizar el máximo de una cantidad (MIN-MAX).

$$C_{max} = \max C_i \quad (17)$$

2. *Tiempo de terminación total* (SCT): Es la suma de los tiempos de terminación de todas las tareas. Un objetivo muy común es el de minimizar el tiempo de terminación total. Este problema se puede referir como minimizar la suma de una cantidad (MIN-SUM)

$$SCT = \sum_{i=1}^n C_i \quad (18)$$

3. *Tiempo de terminación total promedio* (mSCT): El promedio de los tiempos de terminación provee del tiempo promedio que toma para terminar la ejecución de una tarea.

$$mSCT = \frac{1}{n} \sum_{i=1}^n C_i \quad (19)$$

4. *Tiempo de terminación total ponderado* (SWCT): Es la suma de todos los tiempos de terminación multiplicada por el peso relativo (donde $w_i = size_i \cdot p_i$) de las tareas.

$$SWCT = \sum_{i=1}^n (C_i \cdot w_i) \quad (20)$$

5. *Tiempo de respuesta total (FT)*: El flujo de tiempo de una tarea J_i es el tiempo que esta tarea dura dentro del sistema. Es el intervalo de tiempo desde que la tarea ingresa al sistema y el tiempo que dura en ejecución. Comprende la suma del flujo de tiempo de todas las tareas.

$$FT = \sum_{i=1}^n (C_i - r_i) \quad (21)$$

6. *Tiempo de respuesta total ponderado (WFT)*: Representa la suma de todos los flujos de tiempo multiplicada por el peso (donde $w_i = size_i \cdot p_i$) de las tareas.

$$WFT = \sum_{i=1}^n [(C_i - r_i) \cdot w_i] \quad (22)$$

7. *Slowdown (Atraso) (SD)*: Como el tiempo de ejecución de una tarea dada puede considerarse como un límite inferior para el *tiempo de respuesta*. Dado que los tiempos de ejecución de las tareas pueden variar enormemente, también lo hace el *tiempo de respuesta*.

De esta forma se puede considerar como una mejor métrica *Slowdown* la cual normaliza el tiempo de respuesta con respecto al tiempo de ejecución de las tareas.

$$SD = \sum_{i=1}^n \left(\frac{tw_i + p_i}{p_i} \right) \quad (23)$$

8. *Slowdown promedio (mSD)*: *Slowdown* implica que las tareas son atrasadas en proporción a sus tamaño, es decir que tareas pequeñas se atrasan menos que tareas más grandes.

$$mSD = \frac{1}{n} \sum_{i=1}^n \left(\frac{tw_i + p_i}{p_i} \right) \quad (24)$$

9. *Slowdown promedio limitado* (SD_b): Esta variante de la métrica anterior, se propone para eliminar el énfasis en tareas muy pequeñas (tareas con tiempo de ejecución muy cercano a cero) dado que se tiene el tiempo de ejecución en el denominador. Un límite común son 10 segundos.

$$SD_b = \frac{1}{n} \sum_{i=1}^n \left(\frac{tw_i + p_i}{\max\{10, p_i\}} \right) \quad (25)$$

10. *Tiempo de espera promedio* (t_w): El tiempo de espera promedio, denota el tiempo en que la tarea i tarda en empezar su ejecución.

$$t_w = \frac{1}{n} \sum_{i=1}^n (C_i - p_i - r_i) \quad (26)$$

11. *Tiempo de espera promedio ponderado (trabajo)* (Wt_w): Es la suma del tiempo de espera de todas las tareas por el peso (donde $w_i = size_i \cdot p_i$) de las tareas.

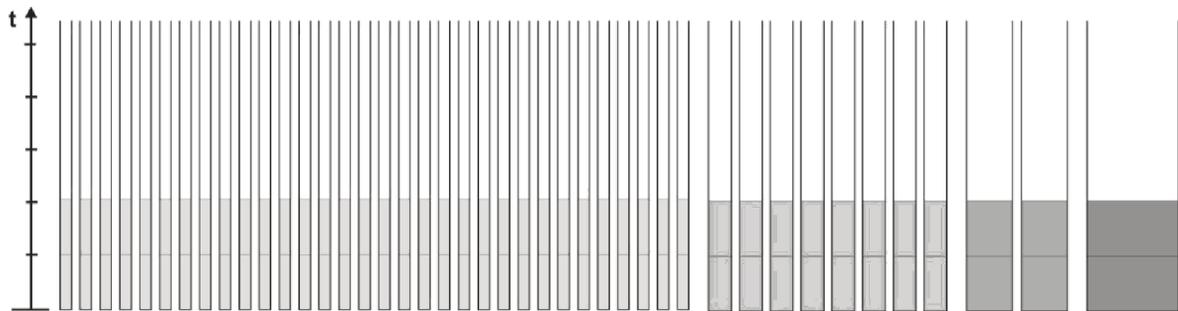
$$Wt_w = \frac{1}{n} \sum_{i=1}^n [(C_i - p_i - r_i) \cdot w_i] \quad (27)$$

Se puede observar que en un modelo fuera de línea (donde $r_i = 0$) la fórmula 21 también representaría el tiempo de terminación total (fórmula 18), asimismo la fórmula 22 daría el tiempo de terminación total ponderado al igual que la fórmula 20.

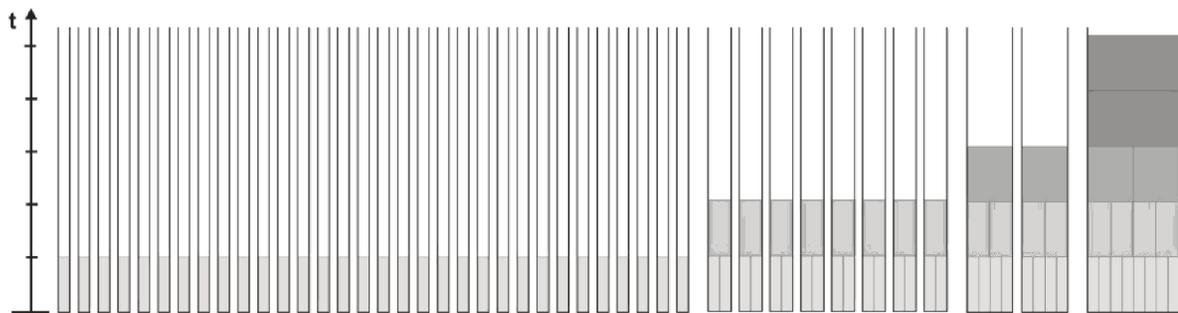
III.6 Estrategia de admisibilidad

Se ha demostrado que la combinación de varias estrategias de asignación (ML , MPL , MLp , MLB , MCT) con $FCFS$ no pueden garantizar un factor de competitividad constante (Tchernykh *et al.*, 2006). Asimismo, se ha demostrado que la combinación de las estrategias MLB y MCT (que toman en cuenta el tiempo de procesamiento de los trabajos) con un algoritmo PS (algoritmo en lista) tampoco pueden garantizar un factor de competitividad constante.

Un ejemplo en el cual no se puede garantizar un factor de competitividad constante para las estrategias $(MLB, MCT) + PS$ (en el cual se tienen grupos de máquinas y de trabajos donde la agrupación de los trabajos es de acuerdo a su tamaño y la cantidad de grupos es igual a las agrupaciones de las máquinas) se muestra en la Figura 8. En la Figura 8(a) se presenta el calendario óptimo con un tiempo de terminación de 2. En cambio, si los trabajos son sometidos de manera creciente de acuerdo a sus tamaños considerando que el tiempo de procesamiento de los trabajos es unitario, las estrategias $(MLB, MCT) + PS$ producirían un calendario como el que se muestra en la Figura 8(b). El tiempo de terminación resultante es igual al número de grupos de trabajos más un último trabajo ver Figura 8 (b).



(a) Calendario óptimo



(b) Calendario resultante para la entrada definida

Figura 8. Ejemplo de un calendario basado en lista

Como se puede apreciar uno de los problemas que surgen en calendarios basados en lista (con las tareas dentro de la lista ordenadas de forma creciente) es la ocupación de tareas de poco grado de paralelismo en máquinas con un gran número de procesadores retrasando la ejecución de tareas con un mayor grado de paralelismo.

Para solucionar este problema se propuso un nuevo enfoque conocido como “*estrategia de admisibilidad*”. Esta estrategia excluye máquinas de gran tamaño dentro del conjunto de máquinas disponibles para tareas de poco grado de paralelismo (Zhuk *et al.*, 2004).

Suponiendo que se indexan las máquinas en un orden creciente con respecto a sus tamaños (cantidad de procesadores, $m_1 \leq m_2 \leq \dots \leq m_m$). A su vez, se define a $f_j = first(J_j)$ como el índice i más pequeño de la máquina donde puede ser asignada la tarea J_j , tal que $m_i \geq size_j$. El conjunto de máquinas disponibles para la tarea J_j corresponde al conjunto de índices de máquina $\{f_j, f_j + 1, \dots, m\}$ denotado por $M_{disponible}(J_j)$, ver Figura 9.

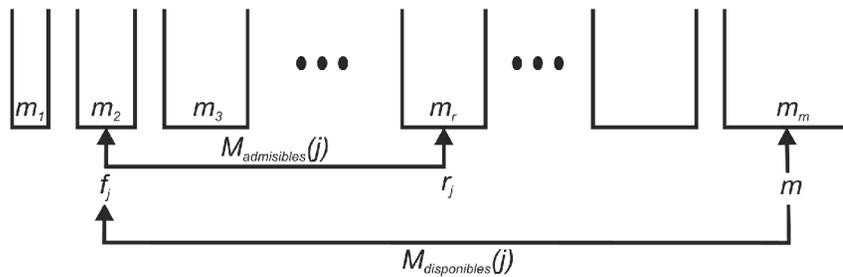


Figura 9. Concepto del modelo de admisibilidad

El conjunto total de máquinas M_{total} está representado por el conjunto $\{1, \dots, m\}$. Como un algoritmo tal vez no tiene acceso a todas las máquinas disponibles para la asignación de una tarea dada. Entonces, se define al conjunto de máquinas admisibles $M_{admisible}(J_j)$ que es un subconjunto del conjunto $M_{disponible}(J_j)$.

En la definición original de admisibilidad, $r_j = last(J_j)$ denota al índice menor de máquina tal que $m_{f,r} \geq \frac{1}{2} \cdot m_{f,m}$. Siendo $m_{f,r} = \sum_{i=f_j}^{r_j} m_i$ la cantidad total de procesadores entre las máquinas m_f a m_r . En la Figura 9, se puede ver representado al conjunto de máquinas $M_{admisible}(J_j)$ por el conjunto de índices $\{f_j, \dots, r_j\}$.

Como se mencionó en el apartado de trabajos previos, Tchernykh *et al.*, (2008) generalizaron la definición de admisibilidad con la introducción del *factor de admisibilidad*. Así, se tiene que el índice r_j está definido por $m_{f,r} \geq a \cdot m_{f,m}$. Donde la selección de $a = 1/2$ produce la definición original (50% del conjunto de máquinas). La selección de $a = 0$ define a la primera máquina disponible y un valor de $a = 1$ selecciona todo el conjunto de máquinas disponibles.

III.7 Estrategia de balanceo de la carga de Bar-Noy

El trabajo de Bar-Noy *et al.*, (2001) se enfoca al problema de balanceo de la carga dentro de un modelo en línea con servidores jerárquicos. El objetivo principal es asignar las tareas a los servidores disponibles para minimizar el *costo* de la asignación, definido como la carga máxima en el servidor. El *costo* de una asignación dada se define como $COSTO = \max_s \{l_s\}$, donde l_s es la carga en el servidor s . En su modelo los servidores se encuentran ordenados bajo cierta prioridad $(1, 2, \dots, m)$ de acuerdo a la capacidad de los servidores. Los servidores más a la izquierda son de mayor prioridad que los servidores más a la derecha.

La secuencia de entrada se compone de un número de tareas las cuales tienen asociado un peso (w_j) y la solicitud de un servidor específico llamado “ s ”. Una tarea J_j que solicite el servidor s también puede ser asignada a cualquier servidor de mayor prioridad (un servidor a la izquierda de s); estos servidores se conocen como *servidores elegibles* para la tarea J_j . Una vez asignada la tarea a un servidor ésta contribuye con cierto peso (w_j) incrementando la carga del servidor.

Los dos modelos discretos que presentan Bar-Noy *et al.*, (2001) en su trabajo son: *modelo integral* (con peso unitario o con peso arbitrario), cada tarea debe ser asignada en su totalidad a un solo servidor. En el *modelo fraccional* el peso de una tarea puede ser dividido

entre varios servidores elegibles. Los autores proporcionan un algoritmo para un modelo fraccional con un valor para el factor de competitividad de e y un modelo integral con peso unitario y con peso arbitrario con un factor de competitividad de e y $(e + 1)$, respectivamente.

El análisis para estos modelos es el siguiente: se tiene una secuencia de entrada y un servidor s , se denota por W_s al peso total de las tareas que solicitan servidores a la izquierda de s , además se define $\mu_s = W_s/|S|$, donde $|S|$ denota la cardinalidad del conjunto de servidores $\{1, \dots, s\}$. Entonces se tiene que $H = \max_s \{\mu_s\}$ como un límite inferior para OPT. Asimismo, el peso máximo (w_{max}) de una tarea (de la secuencia de entrada) también define un límite inferior para OPT en un modelo integral. Cuando la carga de un servidor dado es al menos H se dice que el servidor se encuentra *saturado* (límite superior para OPT).

Para el modelo integral, se considera un algoritmo que asigna cada tarea a su servidor elegible no saturado que se encuentre más a la izquierda. Si siempre se puede encontrar un servidor elegible no saturado, entonces $COSTO < H + w_{max}$. Para demostrar esto, se supone que cuando una tarea con peso w llega, todos sus servidores elegibles están saturados. Si todos los servidores que se encuentran a la izquierda de s están saturados, esto implica que las tareas asignadas a la izquierda de s debieron haber solicitado servidores a la izquierda de s . Dado que su peso total es al menos $(s \cdot H)$, se tiene que $W_s \geq s \cdot H + w > s \cdot H$, lo cual es una contradicción.

Para el modelo fraccional, se modifica el algoritmo anterior de la siguiente manera: cuando una tarea de peso w llega ésta se asigna de la siguiente forma: sea s el servidor elegible no saturado más a la izquierda y sea $\delta = H - l_s$, donde l_s es la carga actual en s . Si $\delta \geq w$, se asigna la tarea en su totalidad. De lo contrario, se divide la tarea y se asignan δ unidades de su peso a s y el restante se trata recursivamente como si fuera una tarea nueva a asignar. Esta versión del algoritmo alcanza un $COSTO \leq H$.

Para esta última versión del algoritmo (para el modelo fraccional) se utilizan *algoritmos sin memoria*. Estos algoritmos asignan cada tarea de manera independiente de tareas anteriores. Las asignaciones realizadas por un algoritmo sin memoria son independientes del número de servidores o del orden de las tareas. Bar-Noy *et al.*, (2001) utilizan un tipo específico de estos algoritmos conocido como *algoritmos uniformes*, los cuales son instancias de un algoritmo genérico (Algoritmo 1). Cada instancia se caracteriza por una función $u: \mathcal{N} \rightarrow (0,1]$ satisfaciendo que $u(1) = 1$.

Algoritmo 1. Uniforme-Genérico

Algoritmo Uniforme-Genérico

Cuando llega una tarea de peso w solicitando un servidor s , hacer:

1. $r \leftarrow w; i \leftarrow s$.
 2. Mientras $r > 0$:
 3. Asignar $a = \min\{w \cdot u(i), r\}$ unidades de peso al servidor i .
 4. $r \leftarrow r - a$.
 5. $i \leftarrow i - 1$.
-

El algoritmo inicia con el servidor solicitado y avanza hacia la izquierda hasta que la tarea haya sido asignada en su totalidad. La fracción de peso asignada a cada servidor i es $u(i)$ a menos que $w \cdot u(i)$ sea mayor que el restante de la tarea cuando i es alcanzado. La condición $u(1) = 1$ garantiza que siempre se asigna la tarea en su totalidad.

Una característica de este algoritmo se da cuando se tiene una tarea de peso w solicitando un servidor a la izquierda de s . Si este servidor se encuentra cerca de s , la tarea dejará $w \cdot u(s)$ unidades de peso en s a pesar que no sea el servidor solicitado. En algún momento (punto p_s), la contribución de peso en s empieza a disminuir conforme la solicitud de los servidores se aleje con respecto a s . Finalmente, en algún momento (punto p'_s) la contribución de peso en s será nula.

$$p_s = \max\{s' \mid \sum_{i=s}^{s'} u(i) \leq 1\}, \quad p'_s = \max\{s' \mid \sum_{i=s+1}^{s'} u(i) < 1\}$$

Se observa que la carga en s debido a los trabajos que solicitan servidores dentro del rango s, \dots, p_s es simplemente $u(s)$ veces el peso total de estos trabajos. Lo que lleva al siguiente lema.

Lema 1: Sea A un algoritmo sin memoria uniforme. Se tiene el siguiente problema: Dado $K > 0$ y algún servidor s , encontrar una secuencia de entrada I que maximice la carga en s en las asignaciones de A sujeto a que $OPT = K$, se resuelve con $I = \langle w_1, w_2, \dots \rangle$ donde:

$$w_i = \begin{cases} 0, & 1 \leq i < p_s \\ p_s K, & i = p_s \\ K, & p_s < i \leq p'_s \\ 0, & i > p'_s \text{ (if } p'_s < \infty) \end{cases}$$

la carga resultante en s (l_s) satisface que $p_s \cdot Ku(s) \leq l_s \leq p'_s Ku(s)$.

Corolario 1: Sea A un algoritmo sin memoria uniforme, y sea C_A el factor de competitividad de A . Entonces $\sup_i \{p_i u(i)\} \leq C_A \leq \sup_i \{p'_i u(i)\}$.

Posteriormente, Bar-Noy *et al.*, (2001) cambian la perspectiva del problema de un modelo discreto a un modelo al que denominan *semicontinuo*. Este modelo se explica por medio de una analogía. Considerando el fondo de un recipiente lleno de cierto líquido no uniforme el cual está aplicando ciertos grados de presión en diferentes puntos. Las fuerzas actuantes en cualquier punto es cero, pero cualquier región (área) sufre un fuerza total igual a la integral de la presión sobre la región. Los servidores son continuos siendo análogos al fondo del recipiente y las tareas son análogas a la cantidad de fluido que está siendo agregado.

El intervalo de servidores está dado por $[0, \infty)$. Una tarea j tiene un peso w_j y solicita el punto $s_j > 0$ dentro del intervalo de servidores. La asignación de la tarea j está especificada por una función integrable $g_j: [0, \infty) \rightarrow [0, \infty)$ satisfaciendo:

1. $\int_0^{s_j} g_j(x) dx = w_j$;
2. $x > s_j \Rightarrow g_j(x) = 0$;

3. g_j es continuo desde la derecha en cualquier punto.

Una asignación completa se define por $g = \sum_j g_j$. La carga l_l (donde $l = (x, x + \Delta), \Delta > 0$) está definido como $l_l = \frac{1}{\Delta} \int_x^{x+\Delta} g(z) dz$. La carga en un punto x es $l_x = \lim_{\Delta \rightarrow 0} \{l_{(x, x+\Delta)}\} = g(x)$. El costo de la asignación es $COSTO = \sup_x \{l_x\}$. Sea $W(x)$ el peso total de las solicitudes realizadas a la izquierda de x (incluyendo x) y $H = \sup_{x>0} \{W(x)/x\}$. Por lo tanto, $OPT = H$.

Para este modelo se adaptaron los algoritmos sin memoria uniformes caracterizados por la función $u: (0, \infty) \rightarrow (0, \infty)$. Para un punto $x > 0$ dado, sea $q(x)$ el punto que satisface la ecuación $\int_{q(x)}^x u(z) dz = 1$. La asignación de la tarea j se define como:

$$g_j(x) = \begin{cases} w_j u(x), & q(s_j) \leq x < s_j \\ 0 & \text{de otra forma} \end{cases}$$

Para un punto $x > 0$ se define $p(x)$ como el punto tal que $\int_x^{p(x)} u(z) dz = 1$. En este modelo a diferencia del modelo discreto tiene la propiedad de juntar los puntos p_s y p'_s .

A continuación se presenta el lema para el peor caso del modelo semicontinuo:

Lema 2: Sea A un algoritmo sin memoria definido por $u(x)$. Se tiene el siguiente problema: Dado $K > 0$ y algún punto $s > 0$ del intervalo de servidores, encontrar una secuencia de entrada que maximice la carga total en s de las asignaciones de A , sujeto a $OPT = H$. Se resuelve por una simple tarea de peso $p(s)K$ solicitando el punto $p(s)$ así, la carga en s resulta en $p(s)Ku(s)$.

Finalmente, el factor de competitividad de A es $\sup_x \{p(x)u(x)\}$.

En el apartado 2.4 de (Bar-Noy *et al.*, 2001) los autores demuestran que el algoritmo Harmónico para el modelo semicontinuo tiene un factor de competitividad igual a e . A partir de este algoritmo se obtienen dos versiones para el modelo fraccional e integral, obteniendo dos algoritmos que también son e -competitivos.

Para convertir del modelo semicontinuo al modelo fraccional, se define una secuencia de entrada I para el modelo fraccional. Se trata a cada servidor como un punto en $(0, \infty)$ esto es, una solicitud al servidor s se ve como una solicitud al punto s por lo tanto, la entrada I representaría una secuencia de entrada para el modelo semicontinuo. Sea A un algoritmo en línea c -competitivo para el modelo semicontinuo. Se define un algoritmo B para el modelo fraccional. Cuando una tarea llega, B asigna $\int_{i-1}^i g_j(x) dx$ unidades de peso al servidor i , para toda i . Donde g_j es la función de asignación generada por A para la tarea.

Finalmente, para el modelo integral se modifica el algoritmo anterior de la siguiente manera. Sea A un algoritmo para el modelo fraccional. Se define un algoritmo B para el modelo integral de la siguiente manera: Conforme van llegando las tareas, B va monitoreando las asignaciones que pudiera realizar A . En esta transformación se dice que un servidor está *sobrecargado* si su carga de acuerdo a las asignaciones de B excede su carga según las asignaciones realizadas por A . En este caso, cuando una tarea llega, B la asigna al servidor elegible más a la izquierda que no esté sobrecargado.

III.7.1 Estrategia de asignación “Carga mínima (Bar-Noy)”

Partiendo de la estrategia de asignación propuesta por Bar-Noy *et al.*, (2001) (ver apartado III.6), Tchernykh *et al.*, (2010) diseñaron una estrategia de asignación para un modelo de calendarización de trabajos paralelos dentro de un ambiente Grid. A partir de esta estrategia se diseñaron dos versiones más las cuales difieren en la función $u(i)$ (esta función determina la fracción de peso que le corresponde a la máquina i) que se esté utilizando. En la Figura 10 se muestra la clasificación de las seis versiones implementadas:

La primer clasificación de los algoritmos depende de cómo se considere la carga dentro del sistema: En la primer categoría se consideran todas las tareas que se han

asignado desde un inicio. En cambio, en la segunda categoría se hace referencia a la carga instantánea es decir, se consideran únicamente las tareas que se encuentran en ejecución o en lista de espera en cada máquina. Cada una de estas categorías se dividen en tres subcategorías las cuales dependen de la función $u(i)$ que se implemente en la parte fraccional del algoritmo.

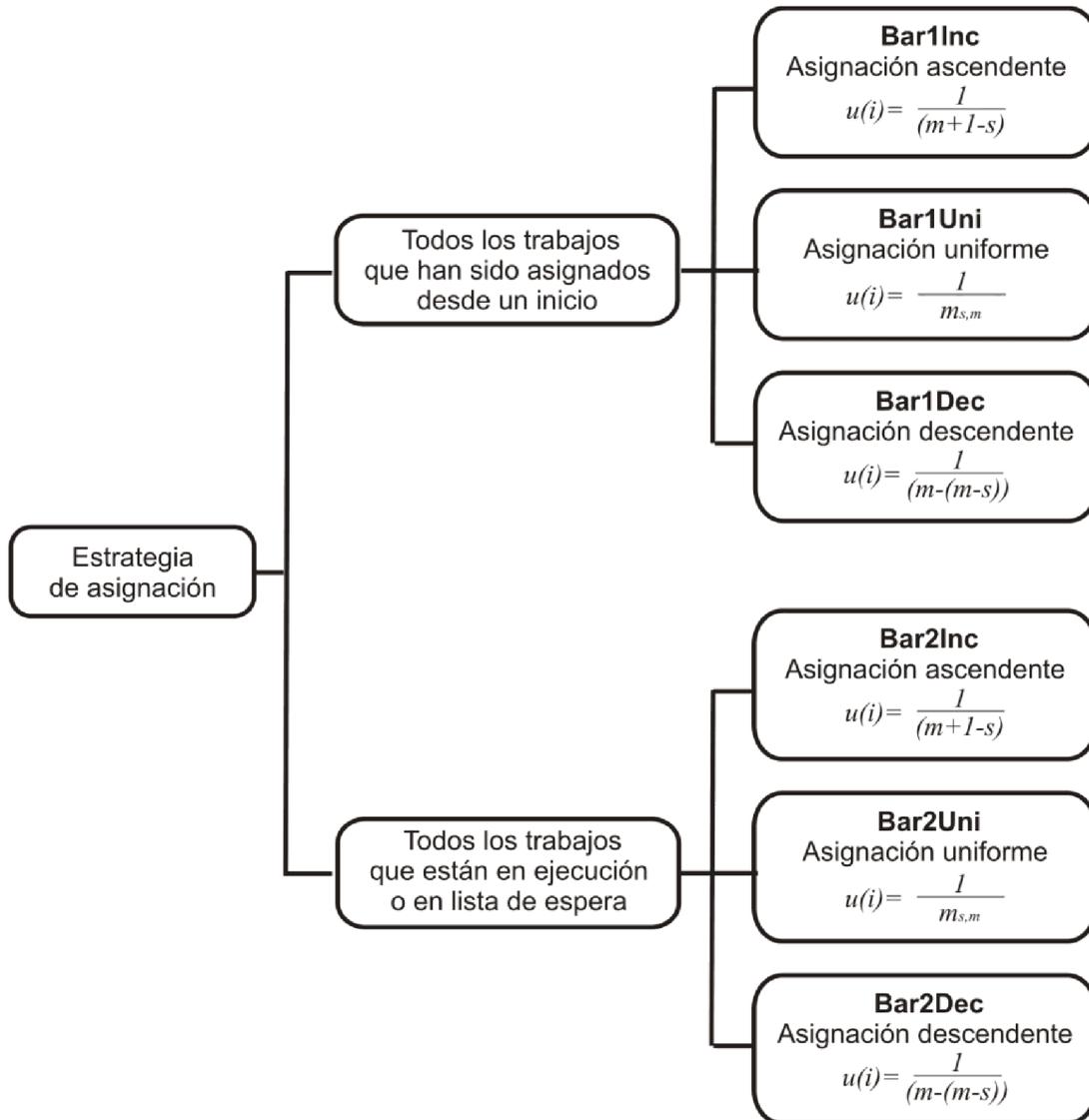


Figura 10. Clasificación de nueva estrategia de asignación

Enseguida se explica con más detalle cada una de las funciones $u(i)$ que fueron implementadas:

1. *Asignación ascendente*: En esta primera versión, la asignación del peso de la tarea en la parte fraccional del algoritmo se calcula mediante la función $u(i) = \frac{1}{m+1-s}$ donde m denota la máquina de mayor tamaño del conjunto de máquinas y s denota la máquina que se está evaluando. La asignación del peso es de forma ascendente conforme se avanza hacia a la derecha dentro del conjunto de máquinas hasta asignar en su totalidad a la tarea.
2. *Asignación uniforme*: La asignación del peso de la tarea en la parte fraccional se calcula mediante la función $u(i) = \frac{1}{m_{s,m}}$ donde $m_{s,m}$ denota la cantidad de procesadores que se tiene desde la primer máquina que puede ejecutar la tarea hasta la máquina de mayor tamaño. La asignación del peso es uniforme en cada una de las máquinas que se necesiten para asignar la tarea completamente.
3. *Asignación descendente*: En esta última versión, la asignación del peso de la tarea se calcula mediante la función $u(i) = \frac{1}{m-(m-s)}$ donde m denota la máquina de mayor tamaño y s denota la máquina que se está evaluando. La asignación del peso es de manera decreciente mientras se avanza hacia la derecha hasta que la tarea sea asignada.

A continuación se describe el algoritmo general diseñado para la nueva estrategia de asignación. El algoritmo se compone de dos partes principales, primero se aplica un modelo fraccional (el peso de la tarea a asignar se puede distribuir entre varias máquinas elegibles) y enseguida se aplica un modelo integral (la tarea debe ser asignada a una sola máquina elegible).

Algoritmo 2. Algoritmo basado en la estrategia “carga mínima (Bar-Noy)”

Algoritmo Genérico

Cuando llega una tarea de peso w_j solicitando un máquina s , hacer:

1. Para 1 a m hacer:
 2. Seleccionar primer máquina donde $m_i \geq size_j$
 3. Mientras $r > 0$:
 4. Calcular $a = \min\{w \cdot u(i), r\}$ unidades de peso para la máquina i .
 5. $r \leftarrow r - a$.
 6. $i \leftarrow i - 1$.
 7. Sumar en *loadFraccMachines* a/m_i a la máquina i
 8. Para 1 a m hacer:
 9. Seleccionar primer máquina donde $m_i \geq size_j$
 10. Calcular w_j/m_i
 11. Si $loadIntMachines(i) \leq loadFraccMachines(i)$ (para el valor de la máquina i)
 12. Asignar la tarea a máquina i
-

Cuando llega una tarea nueva J_j , el algoritmo en un inicio selecciona la primer máquina donde se pueda ejecutar dicha tarea ($m_i \geq size_j$) del conjunto de máquinas disponibles. Se calcula el peso de la tarea J_j el cual está definido como $w_j = size_j \cdot p_j$.

Para el manejo de los valores de referencia se utilizan dos arreglos, *loadFraccMachines* y *loadIntMachines* para la primer y segunda parte del algoritmo respectivamente. En *loadFraccMachines* se almacena el tiempo requerido (el tiempo óptimo) para la ejecución de la tarea J_j dependiendo de la función $u(i)$ y la máquina en la que se esté evaluando. En *loadIntMachines* se almacena el tiempo necesario para la ejecución de la tarea J_j según la máquina donde se asigne la tarea J_j .

En la primer parte del algoritmo se calcula la fracción de peso $fracc = w_j \cdot u(i)$ que se va a asignar comenzado por la primer máquina donde se puede ejecutar la tarea. Posteriormente se calcula el tiempo correspondiente dividiendo la fracción de peso entre el número de procesadores de la máquina i y se almacena este dato en el arreglo

$loadFraccMachines(i)$. A w_j se le resta la fracción de peso ya asignada y se avanza hacia la derecha hasta que la tarea haya sido asignada en su totalidad.

En la segunda parte del algoritmo se implementa el modelo integral con el cual se realiza la asignación de la tarea J_j a una sola máquina. Nuevamente se inicia desde la primera máquina que puede ejecutar la tarea y se compara el valor almacenado en el arreglo $loadIntMachines(i)$ con el valor almacenado en el arreglo $loadFraccMachines(i)$.

En las comparaciones no se considera el peso de la tarea que se va a asignar para el modelo integral. Si el valor en $loadIntMachines(i)$ es menor o igual al valor almacenado en $loadFraccMachines(i)$ la tarea se asigna a esta máquina. En caso contrario se comparan los valores de la siguiente máquina (avanzando hacia la derecha) y así sucesivamente hasta encontrar una máquina donde se puede asignar la tarea en su totalidad.

Para la evaluación correcta del desempeño del algoritmo es necesario conocer cómo se debe configurar el entorno de simulación. De igual forma, se debe configurar correctamente el entorno de pruebas para producir resultados reproducibles y comparables. Para este fin a continuación se describen las dos configuraciones de Grid utilizadas en las simulaciones.

Capítulo IV

III. Análisis teórico

En este capítulo se hace el análisis teórico de las estrategias propuestas para la obtención del factor de competitividad en el peor caso. Para llevar a cabo los análisis, se consideró el modelo de calendarización descrito en la sección III.1. Primero, se evalúan las estrategias en un modelo fuera de línea y partiendo de éste se obtiene el desempeño de las estrategias para un modelo en línea.

Se ha descrito anteriormente que las estrategias $(ML_a, MPL_a) + FCFS$ no pueden garantizar una aproximación constante para un valor de $a = 1/2$ a menos que $P=NP$ (Tchernykh *et al.*, 2006). Por ende, estos algoritmos no pueden garantizar un calendario cuyo tiempo de terminación se encuentre dentro de un factor constante con respecto al calendario óptimo. Este resultado también se mantiene para un valor de a arbitrario y utilizando el mejor algoritmo PS conocido.

Para solucionar esta situación, se considera la selección de una máquina ideal (del conjunto de máquinas admisibles $M_{admisible}(J_j)$ para la tarea J_j) para la ejecución de la tarea J_j y determinar el factor de competitividad de los algoritmos $(MLB_a, MCT_a) + PS$.

Todas las estrategias son analizadas de acuerdo a su factor de competitividad bajo la optimización del tiempo de terminación del calendario. En este caso sean C_{max}^* y $C_{max}(A)$ el tiempo de terminación óptimo del calendario y el tiempo de terminación determinado por la estrategia A, respectivamente. El factor de competitividad de la estrategia A está definido como $\rho_A \leq \sup \frac{C_{max}(A)}{C_{max}^*}$ para todas las entradas del problema. Por simplicidad en la

notación para el resto del trabajo se omite la estrategia A de $C_{max}(A)$ utilizando únicamente C_{max} .

IV.1 Análisis teórico de la estrategia $MLB_a + PS$

IV.1.1 Modelo fuera de línea

Para el análisis en esta sección se supone que la máquina k determina el tiempo de terminación del Grid (C_{max}), por lo tanto se define que $C_k = C_{max}$ (C_k denota el tiempo de terminación del calendario local de la máquina k).

También, se supone que la última tarea en ser asignada a la máquina k es la tarea J_d . Se hace la suposición que esta tarea define el tiempo de terminación de la máquina k de lo contrario, se podría eliminar y no se afectaría (reduciría) el factor de competitividad resultante.

En la Figura 11 se puede observar la asignación de la tarea J_d en su conjunto de máquinas admisibles ($M_{admisibles}(J_d)$), el cual está constituido por las máquinas $f = f_d, \dots, r = r_d$. Como se está haciendo uso de la estrategia de asignación MLB y se determinó que la tarea J_d fue asignada a la máquina k , esto implica que la cota inferior (lower bound) de la máquina k es menor o igual que la cota de cualquier otra máquina dentro del conjunto $M_{admisibles}(J_d)$, por lo que se garantiza que $\frac{w_k}{m_k} \leq \frac{w_i}{m_i}$ para todo $i = f, \dots, r$.

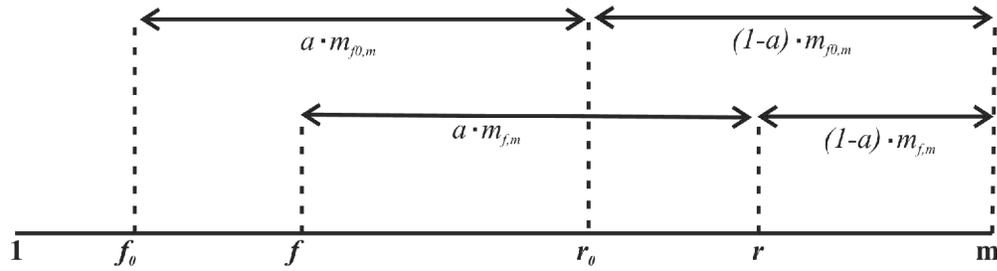


Figura 11. Asignación admisible con factor a

Se ha demostrado que un algoritmo de calendarización en lista L no puede garantizar un factor de competitividad $\frac{C_{\max}(L)}{C_{\max}^*} < 2$, (Schwiegelshohn *et al.*, 2008). Por esta razón, se considera que el tiempo de terminación de la máquina k está determinado por la relación:

$$C_k \leq 2 \cdot \frac{W_k}{m_k} \quad (28)$$

donde W_k representa el trabajo total de la máquina k y m_k es su cantidad de procesadores. En esta expresión se considera que $\frac{W_k}{m_k}$ determina el tiempo de terminación óptimo de la máquina k , (C_{\max}^*) ¹²

Para calcular el tiempo de terminación óptimo del Grid (C_{opt}), primero se determina el trabajo total $W_{f,r}$ en el intervalo de máquinas N_f, \dots, N_r , el cual se determina de la siguiente manera:

$$W_{f,r} = \sum_{i=f}^r W_i = \sum_{i=f}^r \frac{W_i}{m_i} \cdot m_i \geq \sum_{i=f}^r \frac{W_k}{m_k} \cdot m_i = \frac{W_k}{m_k} \sum_{i=f}^r m_i = \frac{W_k}{m_k} \cdot m_{f,r} \quad (29)$$

Después, se supone una tarea J_b cuyo grado de paralelismo es el menor de todas las tareas asignadas en el intervalo de máquinas N_f, \dots, N_r . Para denotar la primera máquina

¹² C_{\max}^* puede determinarse por el máximo del tiempo de procesamiento de un trabajo (p_{\max}) o por el cociente de la suma de todos los tiempos de procesamiento de los trabajos con el número de procesadores ($\sum p_j/m$) (Ramírez *et al.*, 2007)

del conjunto $M_{admisibles}(J_b)$ para el trabajo J_b se utiliza $f_0 = f_b$. Esto implica que las tareas asignadas en el intervalo admisible para J_d no pueden ser asignadas a una máquina con un índice menor que f_0 . Además, como J_b es ejecutada en una de las máquinas del conjunto N_f, \dots, N_r , se tiene que la última máquina de su conjunto admisible es mayor o igual que la primera máquina para J_d , entonces $r_b \geq f$ como se puede observar en la Figura 11. Con esto se puede determinar el tiempo de terminación óptimo como:

$$C_{opt} \geq \frac{W_{f,r}}{m_{f_0,m}} \quad (30)$$

Efectuando una sustitución de $W_{f,r}$ en la expresión anterior, se tiene que el tiempo de terminación óptimo del Grid finalmente está definido por:

$$C_{opt} \geq \frac{W_k}{m_k} \cdot \frac{m_{f,r}}{m_{f_0,m}} \quad (31)$$

En la estrategia *MLB* no se contempla dentro de W_k el trabajo adicional por parte de J_d . Como la longitud mínima del calendario se puede estimar por el tiempo de procesamiento máximo (p_{max}) de todas las tareas. Entonces, en el peor caso considerando que el tiempo de procesamiento de J_d es el máximo en comparación a todas las tareas asignadas a la máquina k , se obtiene que $p_d \leq C_{opt}$.

En un principio se supuso que $C_{max} = C_k$, considerando el resultado anterior ésta igualdad cambiaría a $C_{max} \leq C_k + C_{opt}$ puesto que ahora se considera el tiempo de procesamiento de la tarea J_d . Determinando el desempeño en términos del factor de competitividad queda expresada $\rho \leq \frac{C_k + C_{opt}}{C_{opt}} = 1 + \frac{C_k}{C_{opt}}$. En esta expresión se sustituye C_k por la fórmula 28 quedando:

$$\rho \leq 1 + \frac{2 \cdot \frac{W_k}{m_k}}{C_{opt}} \quad (32)$$

Como se trabaja bajo la estrategia de admisibilidad, a continuación se definen los dos posibles casos que se pueden obtener para el factor de competitividad dependiendo del valor que tome el factor de admisibilidad.

$$\text{Caso 1. } a \leq \frac{m_{f,m}}{m_{f_0,m}}$$

Se toma la definición de admisibilidad $m_{f,r} \geq a \cdot m_{f,m}$. Resultando en $m_{f,m} \leq \frac{m_{f,r}}{a}$ y $m_{f_0,m} \leq \frac{m_{f,r}}{a^2}$.

Sustituyendo la expresión que se acaba de obtener en la fórmula 31 se tiene $C_{opt} \geq \frac{W_k}{m_k} \cdot \frac{m_{f,r}}{m_{f_0,m}} \geq \frac{W_k}{m_k} \cdot a^2$. Finalmente, para obtener el límite del factor de competitividad correspondiente se toma la fórmula 32 y se substituye el nuevo valor de C_{opt} , lo que da como resultado la siguiente relación:

$$\rho \leq 1 + \frac{2 \cdot \frac{W_k}{m_k}}{\frac{W_k}{m_k} \cdot a^2} \leq 1 + \frac{2}{a^2}$$

$$\text{Caso 2. } a > \frac{m_{f,m}}{m_{f_0,m}}$$

Como se puede observar en la Figura 11, se tiene que $r_b \geq f$ esto implica que $m_{f_0,m} \leq m_{f,m} + a \cdot m_{f_0,m}$. De esta fórmula se despeja $m_{f_0,m}$, dando $m_{f_0,m} \leq \frac{m_{f,m}}{1-a}$.

Se sustituye este despeje junto con la fórmula de admisibilidad dentro de la fórmula 31 para obtener que, $C_{opt} \geq \frac{W_k}{m_k} \cdot \frac{m_{f,r}}{m_{f_0,m}} \geq \frac{W_k}{m_k} \cdot \frac{a \cdot m_{f,m}}{\frac{m_{f,m}}{1-a}} = \frac{W_k}{m_k} \cdot a(1-a)$. Al igual que en el caso 1, este último resultado se sustituye en la fórmula 32 para la obtención de la relación que define el límite del valor del factor de competitividad.

$$\rho \leq 1 + \frac{2 \cdot \frac{W_k}{m_k}}{\frac{W_k}{m_k} \cdot a(1-a)} \leq 1 + \frac{2}{a(1-a)}$$

Para cualquiera de los dos casos con un valor del factor de admisibilidad de $a = 1/2$ se obtiene un valor para el factor de competitividad de $\rho \leq 9$.

Con el resultado anterior se deriva el siguiente teorema:

Teorema 1: Calendarización de trabajos rígidos en un modelo fuera de línea en un Grid computacional con procesadores idénticos utilizando el esquema $MLB_a + PS$ con un intervalo de asignación admisible de $0 < a \leq 1$ tiene como factor de competitividad

$$\rho \leq \begin{cases} 1 + \frac{2}{a^2} & \text{para } a \leq \frac{m_{f,m}}{m_{f_0,m}} \\ 1 + \frac{2}{a(1-a)} & \text{para } a > \frac{m_{f,m}}{m_{f_0,m}} \end{cases}$$

donde $1 \leq f_0 \leq f \leq m$ son parámetros que dependen de la configuración de las máquinas y la carga de trabajo.

IV.1.2 Modelo en línea

A continuación se obtiene el comportamiento de la estrategia $MLB_a + PS$ en un modelo en línea, complementando los resultados que se obtuvieron en el apartado anterior.

Para este análisis se supone que en un tiempo r se somete la última tarea en el calendario. En este momento ya existen tareas ejecutándose, por lo que los procesadores pueden estar ocupados. Sin embargo, este tiempo está limitado al tiempo de procesamiento máximo de cualquiera de las tareas que se están ejecutando. Inmediatamente después de este lapso de tiempo (cuando los procesadores queden disponibles) es posible aplicar una calendarización fuera de línea, como se puede observar en la Figura 12.

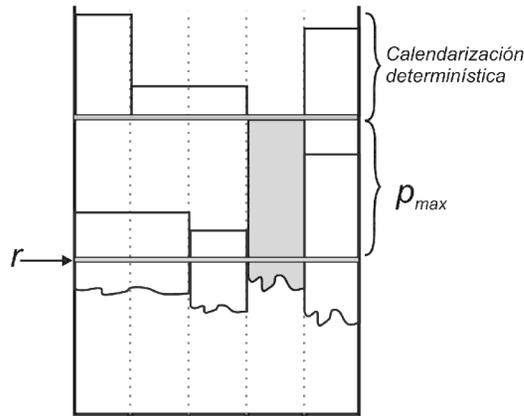


Figura 12. Comportamiento del algoritmo en línea

Por este motivo, un algoritmo para un modelo en línea determina un calendario cuyo tiempo de terminación está dado por:

$$r + p_{max} + C_{max} \quad (33)$$

Si se supone que el tiempo r es una cota inferior para C_{max}^* entonces, $C_{max}^* \geq r$. También, como ya se ha mencionado p_{max} es a su vez una cota inferior de C_{max}^* ($C_{max}^* \geq p_{max}$). Finalmente, el último término de la fórmula 33 está definido por el tiempo de terminación por un algoritmo para un modelo fuera de línea. Estas desigualdades se pueden ver reflejadas en la relación para la cota que define el valor del factor de competitividad resultante.

Teorema 2: Calendarización de trabajos rígidos en un modelo en línea en un Grid computacional con procesadores idénticos utilizando el esquema $MLB_a + PS$ con un intervalo de asignación admisible de $0 < a \leq 1$ tiene como factor de competitividad la siguiente relación:

$$\rho \leq \begin{cases} 3 + \frac{2}{a^2} & \text{para } a \leq \frac{m_{f,m}}{m_{f_0,m}} \\ 3 + \frac{2}{a(1-a)} & \text{para } a > \frac{m_{f,m}}{m_{f_0,m}} \end{cases}$$

Donde $1 \leq f_0 \leq f \leq m$ son parámetros que dependen de la configuración de las máquinas y la carga de trabajo.

IV.2 Análisis teórico de la estrategia $MCT_a + PS$

IV.2.1 Modelo fuera de línea

Para este análisis (al igual que para el análisis realizado para la estrategia $MLB_a + PS$) se hace la suposición que el tiempo de terminación del calendario de la máquina k es quien define el tiempo de terminación del calendario del Grid ($C_k = C_{max}$). Asimismo, la última tarea asignada a la máquina k es J_d .

Como se analiza el desempeño de la estrategia MCT esto garantiza que la tarea J_d fue asignada a la máquina k debido a que el tiempo de terminación de J_d fue menor o igual que su tiempo de terminación en cualquier otra máquina de su conjunto $M_{admissible}(J_d)$, es decir $c_d^k \leq c_d^i$ para toda $i = f, \dots, r$ (donde c_d^i representa el tiempo de terminación de la tarea J_d en la máquina i). También cumpliéndose que $C_k \leq C_i + p_d$ para toda $i = f, \dots, r$. En la estrategia MCT se considera desde un inicio el trabajo de la tarea J_d dentro de W_k a diferencia con la estrategia MLB .

Para definir el tiempo de terminación de la máquina k es necesario considerar que el tiempo de procesamiento de la tarea como el tiempo de procesamiento máximo (en el peor caso), además de tomar en cuenta el factor de competitividad del algoritmo de calendarización local ($\frac{C_{max}(L)}{C_{max}^*} < 2$). Por lo tanto, el tiempo de terminación del calendario queda expresado, como:

$$C_k \leq C_i + p_d \leq C_i + p_{max} \leq 2 \frac{W_i}{m_i} + p_{max} \quad (34)$$

Posteriormente, es necesario determinar la cota inferior entre las máquinas del intervalo N_f, \dots, N_r para definir finalmente el tiempo C_k . Suponiendo que la cota inferior $(\frac{W_i}{m_i})$ se alcanza en una máquina con índice e esto implica que $\frac{W_e}{m_e} = \min_i \left\{ \frac{W_i}{m_i} \right\}$ para toda $i = f, \dots, r$. Por último, se obtiene:

$$C_k \leq 2 \frac{W_e}{m_e} + p_{max} \quad (35)$$

Para determinar el trabajo $W_{f,r}$ se considera la cota inferior de la máquina e quedando:

$$W_{f,r} = \sum_{i=f}^r W_i = \sum_{i=f}^r \frac{W_i}{m_i} \cdot m_i \geq \sum_{i=f}^r \frac{W_e}{m_e} \cdot m_i = \frac{W_e}{m_e} \sum_{i=f}^r m_i = \frac{W_e}{m_e} \cdot m_{f,r} \quad (36)$$

Considerando una tarea J_b (cuyo grado de paralelismo es menor al de todas las tareas del conjunto N_f, \dots, N_r) se define el tiempo de terminación óptimo tomando en cuenta que $r_b \geq f$, resultando en un tiempo de terminación igual al que se expresa en la fórmula 30.

Sustituyendo la fórmula 36 en la fórmula 30 se que tiene el tiempo de terminación óptimo está dado por:

$$C_{opt} \geq \frac{W_e}{m_e} \cdot \frac{m_{f,r}}{m_{f_0,m}} \quad (37)$$

A continuación se consideran dos casos posibles para el valor del factor de admisibilidad:

$$\text{Caso 1: } a \leq \frac{m_{f,m}}{m_{f_0,m}}$$

De la fórmula original de admisibilidad se deriva $m_{f,r} \geq a^2 \cdot m_{f_0,m}$. Si se sustituye éste último resultado dentro de la fórmula 37 se obtiene que $C_{opt} \geq \frac{W_e}{m_e} \cdot \frac{m_{f,r}}{m_{f_0,m}} \geq \frac{W_e}{m_e} \cdot a^2$.

Para la obtención de la cota inferior del factor de competitividad se parte de la fórmula 35 obteniendo:

$$\rho \leq \frac{2W_e}{m_e \cdot C_{opt}} + \frac{p_{max}}{C_{opt}} \leq 1 + \frac{2}{a^2}$$

Caso 2: $a > \frac{m_{f,m}}{m_{f_0,m}}$

Recordando que $r_b \geq f$ (ver Figura 11) lo que lleva a $m_{f_0,m} \leq m_{f,m} + a \cdot m_{f_0,m}$. Despejando $m_{f_0,m}$ se obtiene $m_{f_0,m} \leq \frac{m_{f,m}}{1-a}$. Para obtener el tiempo de terminación óptimo se sustituye éste despeje en la fórmula 37. Finalmente, el C_{opt} obtenido se sustituye dentro de la fórmula 34 para obtener la cota para el factor de competitividad del caso 2:

$$\rho \leq \frac{2W_e}{m_e \cdot C_{opt}} + \frac{p_{max}}{C_{opt}} \leq 1 + \frac{2}{a \cdot (1-a)}$$

Utilizando un valor del factor de admisibilidad igual $a = 1/2$ se obtiene en ambos casos un factor de competitividad con un valor de $\rho \leq 9$.

El teorema resultante para MCT en un modelo fuera de línea utilizando el esquema de admisibilidad es el siguiente:

Teorema 3: Calendarización de trabajos rígidos en un modelo fuera de línea en un Grid computacional con procesadores idénticos utilizando el esquema $MCT_a + PS$ con un intervalo de admisibilidad de $0 < a \leq 1$ tiene como factor de competitividad:

$$\rho \leq \begin{cases} 1 + \frac{2}{a^2} & \text{para } a \leq \frac{m_{f,m}}{m_{f_0,m}} \\ 1 + \frac{2}{a(1-a)} & \text{para } a > \frac{m_{f,m}}{m_{f_0,m}} \end{cases}$$

donde $1 \leq f_0 \leq f \leq m$ son parámetros que dependen de la configuración de las máquinas y la carga de trabajo.

IV.2.2. Modelo en línea

Siguiendo el mismo procedimiento que se detalló en el apartado IV.1.2 en el análisis de *MLB*. La obtención del factor de competitividad para un modelo en línea para *MCT* es exactamente igual. De este análisis nuevamente se deriva el siguiente teorema para *MCT*:

Teorema 4: Calendarización de trabajos rígidos en un modelo en línea en un Grid computacional con procesadores idénticos utilizando el esquema $MCT_a + PS$ con un intervalo de admisibilidad de $0 < a \leq 1$ tiene como factor de competitividad:

$$\rho \leq \begin{cases} 3 + \frac{2}{a^2} & \text{para } a \leq \frac{m_{f,m}}{m_{f_0,m}} \\ 3 + \frac{2}{a(1-a)} & \text{para } a > \frac{m_{f,m}}{m_{f_0,m}} \end{cases}$$

donde $1 \leq f_0 \leq f \leq m$ son parámetros que dependen de la configuración de las máquinas y la carga de trabajo.

IV.3 Análisis de resultados estrategias $MLB_a + PS$ y $MCT_a + PS$

En ambas estrategias las relaciones para conocer el factor de competitividad dentro de un modelo en línea resultan ser las mismas, independientemente de las características y la forma de selección de cada una de las estrategias de asignación utilizadas.

A continuación se puede observar el comportamiento de las estrategias propuestas basadas en los resultados obtenidos con las siguientes gráficas:

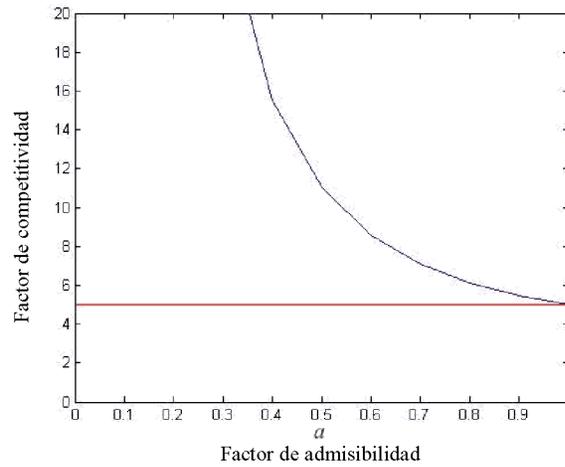
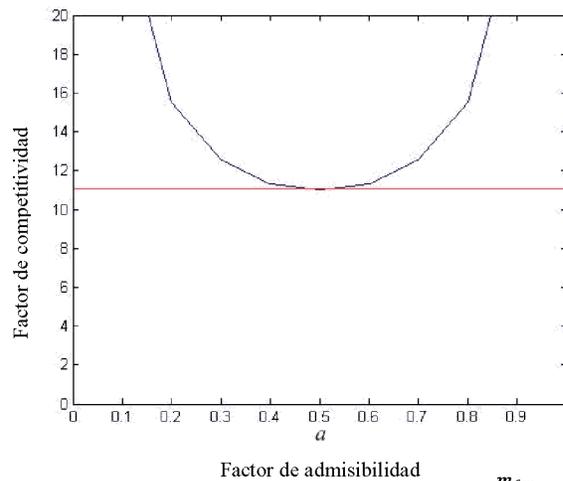


Figura 13. Factor de admisibilidad ($a \leq \frac{m_{f,m}}{m_{f_0,m}}$)

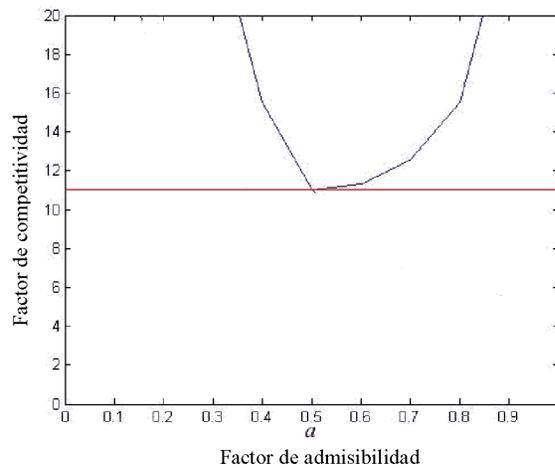
En las figuras 13 y 14 se puede observar la cota para el factor de competitividad para las estrategias de asignación $(MLB_a, MCT_a) + PS$ para valores de admisibilidad de $a \leq \frac{m_{f,m}}{m_{f_0,m}}$ y $a > \frac{m_{f,m}}{m_{f_0,m}}$, respectivamente.

En la Figura 13 es posible apreciar cuando el valor de admisibilidad es $a \leq \frac{m_{f,m}}{m_{f_0,m}}$ la cota para el peor caso varía entre ∞ a 5 en función al rango del factor de admisibilidad. En cambio, en la Figura 14 cuando se tiene un valor de $a > \frac{m_{f,m}}{m_{f_0,m}}$ la cota varía de ∞ a ∞ con un valor mínimo de $\rho \leq 11$ para un valor de $a = 1/2$.



Factor de admisibilidad
Figura 14. Factor de competitividad ($a > \frac{m_{f,m}}{m_{f_0,m}}$)

En la Figura 15 se muestra el límite máximo resultante de todos los límites para los peores casos mostrados en las figuras 13 y 14 en función del valor de admisibilidad. Finalmente, en el punto de cruce de ambos resultados el algoritmo (para un modelo en línea) produce un valor igual a $\rho \leq 11$ para un valor de $a = 1/2$.



Factor de admisibilidad
Figura 15. Factor de competitividad ($a=1/2$) para $(MLB_a, MCT_a) + PS$

Dado que la estrategia de admisibilidad depende de la configuración de la carga de trabajo, se desea analizar el desempeño del algoritmo en el peor caso. Para este fin, se consideran dos intervalos para el factor de admisibilidad a : $(0, \frac{m_{f,m}}{m_{f_0,m}}]$ y $(\frac{m_{f,m}}{m_{f_0,m}}, 1]$. Entonces, tomando en cuenta algunas de las características posibles de la carga de trabajo se muestran los siguientes resultados para el peor caso:

- Cuando $f = m$ y $f_0 = 1$, el intervalo para el factor de admisibilidad está dado por $\frac{m_m}{m_{1,m}} \leq a \leq 1$. Con este rango la relación para el factor de competitividad resulta ser $\rho \leq 3 + \frac{2}{a(1-a)}$. Entonces, si $a = 1$ se está considerando todo el conjunto de máquinas disponibles por lo que no se puede garantizar un factor de competitividad constante ($\rho \rightarrow \infty$). En cambio, con un valor de $a = 1/2$ se obtiene una cota constante de $\rho \leq 11$.
- Un segundo caso considerando una carga *predominantemente secuencial* se daría cuando $f = f_0 = 1$. En tal caso, se considera $\rho \leq 3 + \frac{1}{a^2}$. Si se toma un valor de $a = 1$ resulta en un valor de $\rho \leq 5$. Sin embargo, cuando todos los trabajos son asignados a su primer máquina disponible ($a \rightarrow 0$) no se obtiene un valor constante por lo que $\rho \rightarrow \infty$, ver Figura 13.
- Finalmente, si se considera una carga *predominantemente paralela* ($f = f_0 = m$), esto implica que se tenga la relación $\rho \leq 3 + \frac{1}{a^2}$. En este caso, con un valor de $a = 1$ se obtiene un valor para el factor de competitividad de $\rho \leq 5$.

Los resultados obtenidos en el análisis teórico que se presenta en esta sección, mejoran el resultado de dos trabajos previos, tanto para un modelo fuera de línea como para un modelo en línea. Para un modelo fuera de línea se obtuvo un valor de $\rho \leq 9$ al considerar un mejor algoritmo de calendarización local cuyo factor de competitividad es de

$\rho \leq 2 - 1/m$, mejorando el trabajo previo presentado por Tchernykh *et al.*, (2006). Los autores proponen un algoritmo con un valor de $\rho \leq 10$ utilizando un algoritmo de calendarización local con un valor de $\rho \leq 3$.

Por otro lado, en un modelo en línea se mejoró el resultado propuesto por Tchernykh *et al.*, (2010) donde el algoritmo que los autores presentan tiene un valor de $\rho \leq 17$ (diseñado para un modelo más general). Este resultado se redujo a un valor de $\rho \leq 11$ considerando que los trabajos llegan uno por uno al sistema (restricción en la carga de trabajo).

Capítulo V

V. Análisis experimental

En este capítulo se presenta un análisis experimental de las estrategias *MCT*, *MLB* y *BN* con el objetivo de comparar el comportamiento en promedio de los resultados teóricos (evaluados para el peor caso) de dichas estrategias. Asimismo, se describen a detalle los algoritmos diseñados e implementados dentro del simulador Teikoku para la nueva estrategia de asignación *BN*. Se mencionan las especificaciones de los experimentos realizados para el análisis del desempeño de las diferentes estrategias de asignación que se van a simular. Finalmente, se presentan el análisis de resultados de estas simulaciones.

V.1 Configuración del Grid

Se consideran dos configuraciones de Grid para la realización de las evaluaciones. En la Tabla III, se muestra la configuración para el Grid 1 el cual está constituido por 7 sitios con un total de 4442 procesadores. En cambio el Grid 2 se compone de 9 sitios con 2194 procesadores, ver Tabla IV.

Tabla III. Configuración Grid1

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

Tabla IV. Configuración Grid2

Sitio	Ubicación	Procs.	Registros
1	DAS2 - University of Amsterdam	64	Gwa-t-1-anon_jobs-reduced.swf ,
2	DAS2 - Delft University of Technology	64	1124772 trabajos, 333 usuarios
3	DAS2 - Utrecht University	64	
4	DAS2 - Leiden University	64	
5	KTH - Swedish Royal Institute of Technology	100	KTH-SP2-1996-2.swf, 28489 trabajos, 204 usuarios
6	DAS2- Vrije University Amsterdam	144	Gwa-t-1-anon_jobs-reduced.swf (cont.)
7	HPC2N - High Performance Computing Center North, Sweden	240	HPC2N-2002-1.1-cls.swf , 527371 trabajos, 256 usuarios
8	CTC - Cornell Theory Center	430	CTC-SP2-1996-2.1-cls.swf , 79302 trabajos, 679 usuarios
9	LANL -.Los Alamos National Lab	1024	LANL-CM5-1994-3.1-cls.swf, 201387 trabajos 211 usuarios

V.2 Configuración de la carga de trabajo

Evaluar el desempeño de la ejecución de una tarea dentro de un Grid dedicado (no se consideran las tareas locales de cada sitio) depende en gran medida de la carga de trabajo que se esté utilizando. Si se desconocen las características de la carga de trabajo no se podría determinar si los resultados de las simulaciones fueron buenos o malos.

Algunas estrategias de calendarización ofrecen buenos resultados con trabajos de gran tamaño, en cambio existen otras estrategias que no tienen este comportamiento. Por lo tanto, todas las características de la carga de trabajo afectan el desempeño de las estrategias de calendarización. Por esta razón, para una evaluación precisa de las estrategias de calendarización en un Grid se utilizan cargas de trabajo constituidas por flujos de tareas reales.

Los registros que se utilizan son PWA (ver Parallel Workloads Archive de Dror Feitelson) y GWA (ver Grid Workloads Archive de TU Delft team). Para la creación de un entorno de simulación, se removieron las tareas de los registros utilizados que tenían las siguientes características:

- Identificador de tarea ≤ 0
- Tiempo de llegada < 0

- Tiempo de ejecución ≤ 0
- Número de procesadores asignados ≤ 0
- Tiempo solicitado ≤ 0
- Identificador de usuario ≤ 0
- Según su estatus (0,4 o 5) donde 0 = falla de la tarea; 4 = ejecución parcial; 5 = la tarea fue cancelada

En la carga de trabajo para el Grid 1 constituye un total de 30 experimentos de una semana con un total de tareas de 175337 (150 días). Para el Grid 2 la carga también es para un total de 30 experimentos de una semana, con un total de 512606 tareas (150 días). El tamaño máximo para las tareas (considerando tamaño como la cantidad de procesadores que necesita para su ejecución) en el Grid 1 y Grid 2 es de 1368 y 1024, respectivamente.

V.3 Análisis de resultados

En esta sección se presentan los resultados y el análisis de las simulaciones realizadas a las estrategias de asignación (*MLB*, *MCT*, *MPL*, *MLP*) + *PS* descritas en la sección III.3 así como las seis nuevas estrategias de asignación *BN* diseñadas.

Para analizar el desempeño de las estrategias se utilizan las métricas:

1. *Factor de competitividad* $\rho \leq \frac{C_{\max(A)}}{C_{\text{opt}}}$.
2. *Tiempo de espera promedio*: $t_w = \frac{1}{n} \sum (c_j - p_j - r_j)$
3. *Slowdown promedio limitado*: $SD_b = \frac{1}{n} \sum_{j=1}^n \frac{c_j - r_j}{\max\{10, p_j\}}$.

V.3.1 Desempeño de las estrategias de asignación

A menudo, los proveedores de recursos y los usuarios tienen diferentes metas que con frecuencia entran en conflicto, desde minimizar el tiempo de respuesta (usuario) como optimizar la utilización de los recursos (proveedor). Un buen algoritmo de calendarización debe alcanzar un alto desempeño del Grid y al mismo tiempo satisfacer las necesidades de los usuarios.

A continuación se muestran el desempeño de las diez estrategias de asignación analizadas bajo las métricas de desempeño establecidas:

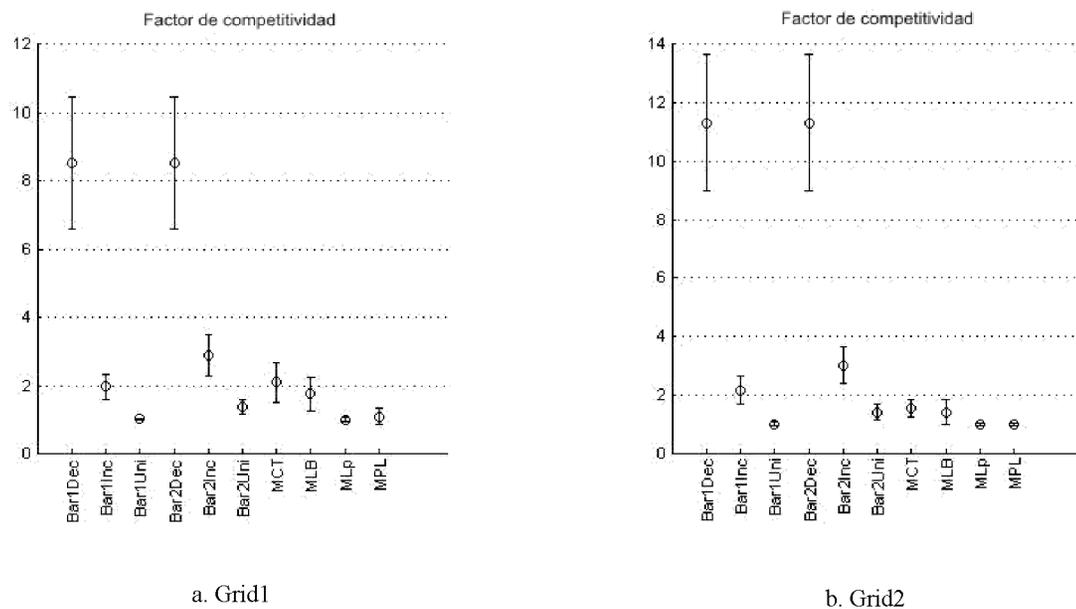
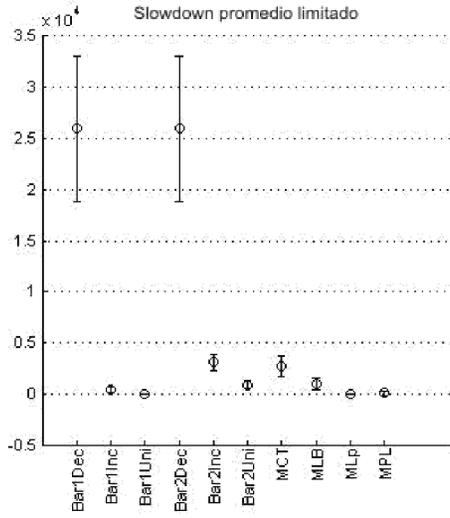
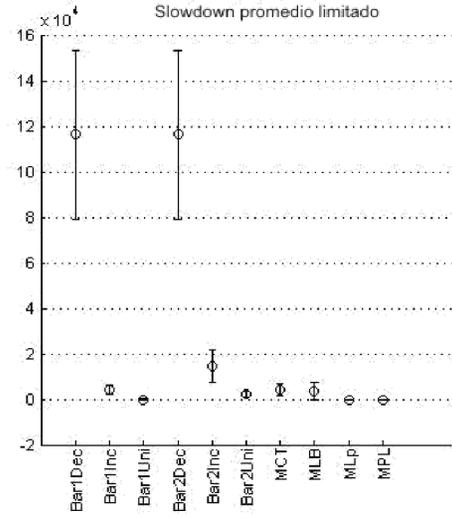


Figura 16. Factor de competitividad

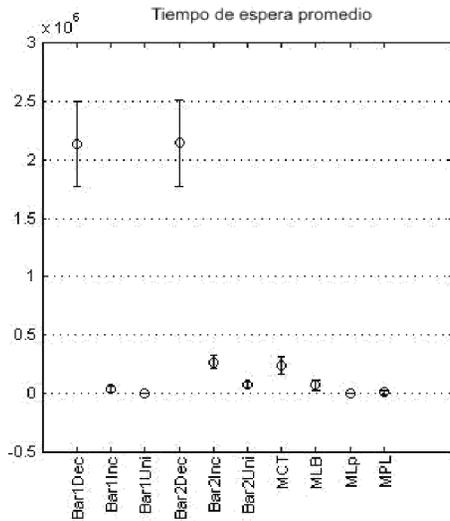


a. Grid1

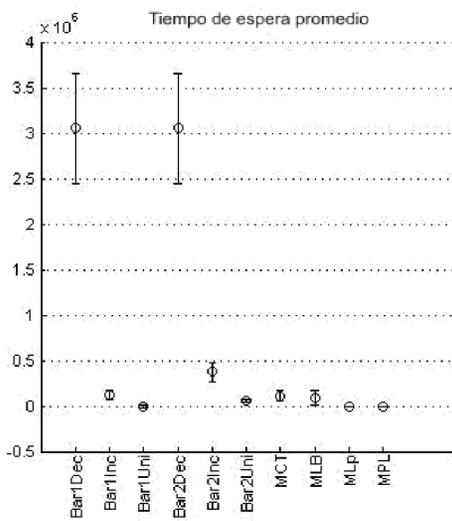


b. Grid2

Figura 17. Slowdown promedio limitado



a. Grid1



b. Grid2

Figura 18. Tiempo de espera promedio

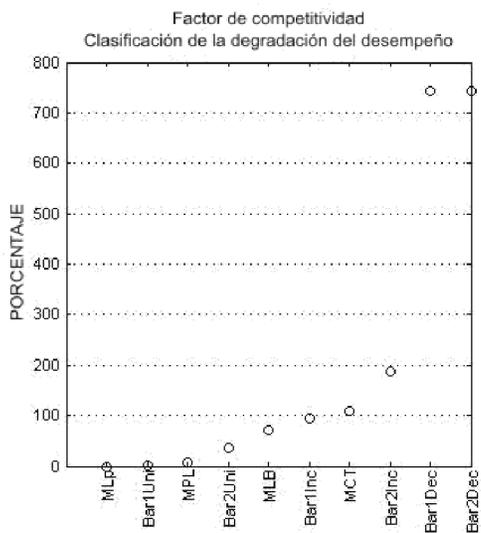
Como se mencionó anteriormente, la administración de recursos en un Grid involucra múltiples objetivos por lo cual se podría utilizar decisiones multi-criterio basadas en una metodología del Óptimo de Pareto ver (Kurowski *et al.*, 2006; Kurowski *et al.*, 2008). Sin embargo en un ambiente Grid es muy difícil obtener soluciones rápidas y eficientes por medio de esta metodología. Una solución comúnmente utilizada es simplificar el problema a un único objetivo o utilizar diferentes métodos en los cuales se hace una combinación de objetivos.

A continuación se detalla la metodología utilizada para determinar el desempeño de las estrategias evaluadas de acuerdo a las métricas establecidas.

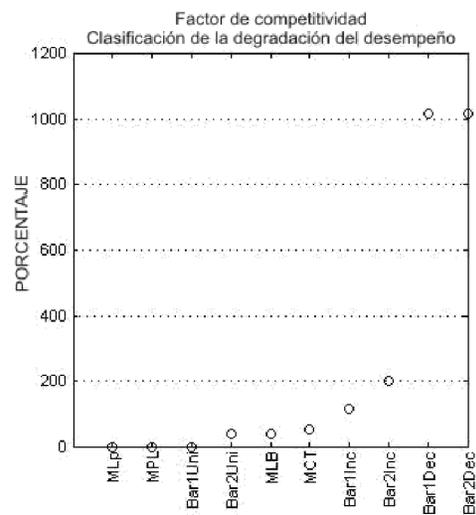
V.3.2 Degradación del desempeño de las estrategias de asignación

Con el fin de establecer un patrón eficaz para la selección de la mejor estrategia se realiza un análisis de tres métricas de acuerdo a la metodología propuesta por Tsafirir *et al.*, (2005). Primero, se evalúa la degradación del desempeño de cada estrategia “*a*” analizada por cada métrica. Después, se selecciona una estrategia “*b*” con el mejor desempeño para cierta métrica dada. Luego de seleccionar ambas estrategias se define a “*P_b*” y “*P_a*” como los desempeños de *b* y *a* respectivamente bajo una métrica “*t*”. Así la degradación en *a* se define por medio de $\frac{100 \cdot P_a}{P_b} - 100$. De esta manera cada estrategia se caracteriza por medio de tres números con los cuales se refleja su degradación relativa bajo cada caso de estudio.

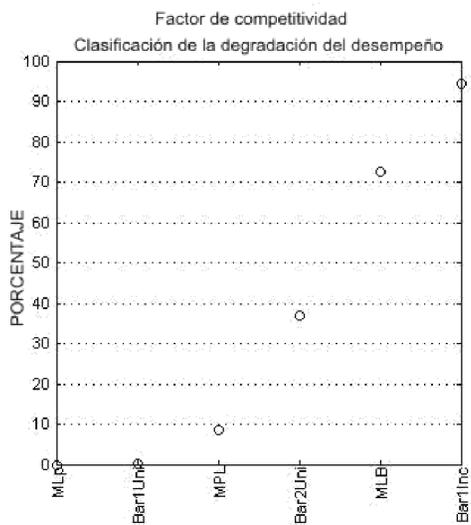
Después, se promedian estos tres valores (suponiendo que las métricas tienen la misma importancia) con estos resultados se clasifican las estrategias de manera que la mejor estrategia (con la menor degradación de desempeño promedio) sea la número 1 y la peor estrategia quede en el número 10. Posteriormente en la sección V.3.3 se muestran los resultados para todos los casos de prueba para ambos Grids.



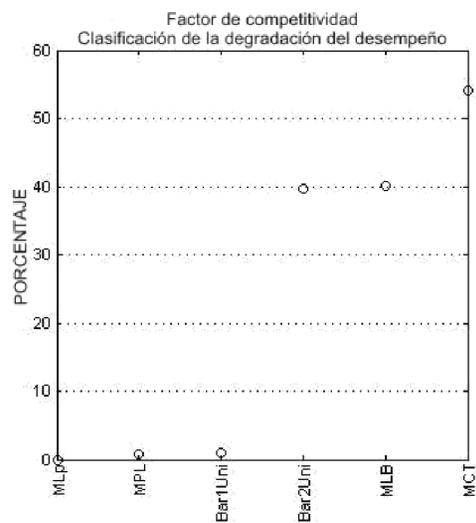
a. Grid 1 (clasificación)



c. Grid 2 (clasificación)

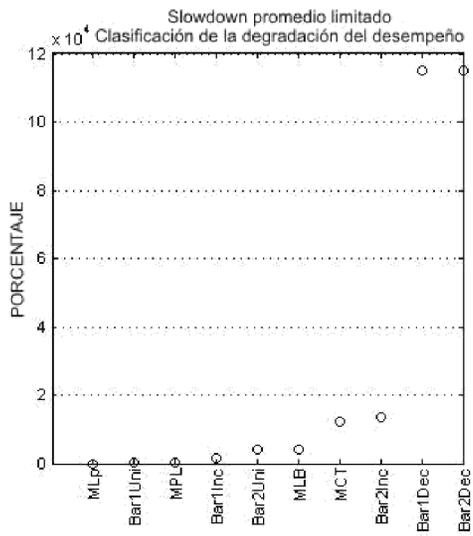


b. Grid 1 (6 mejores estrategias)

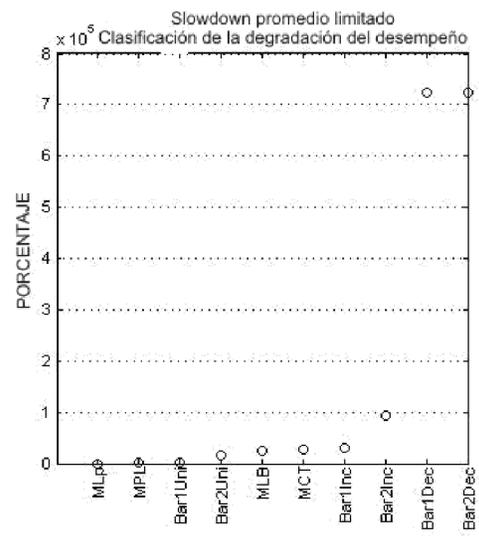


d. Grid 2 (6 mejores estrategias)

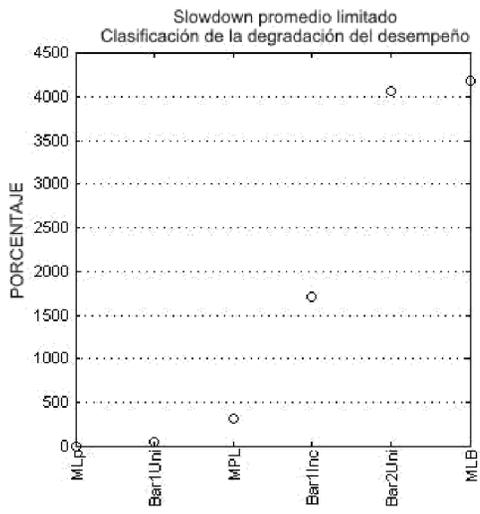
Figura 19. Degradación del factor de competitividad



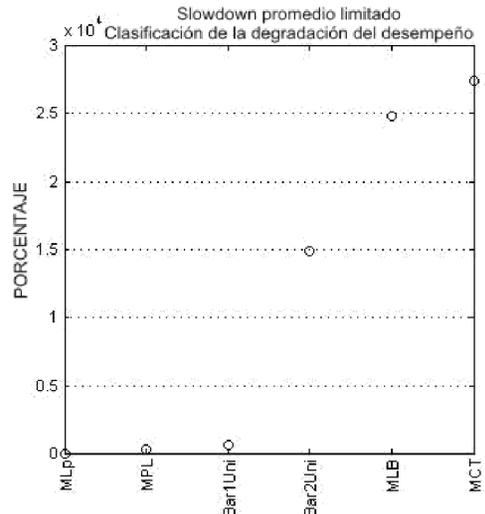
a. Grid 1 (clasificación)



c. Grid 2 (clasificación)

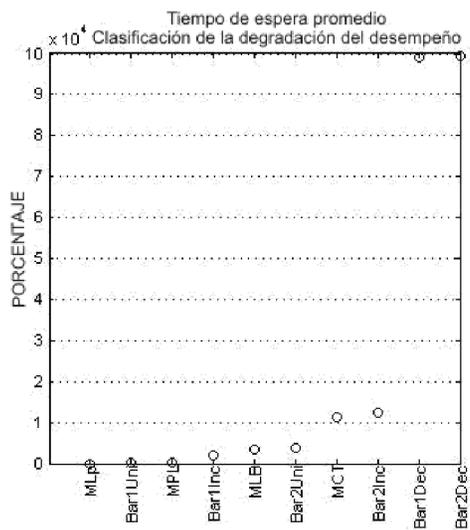


b. Grid 1 (6 mejores estrategias)

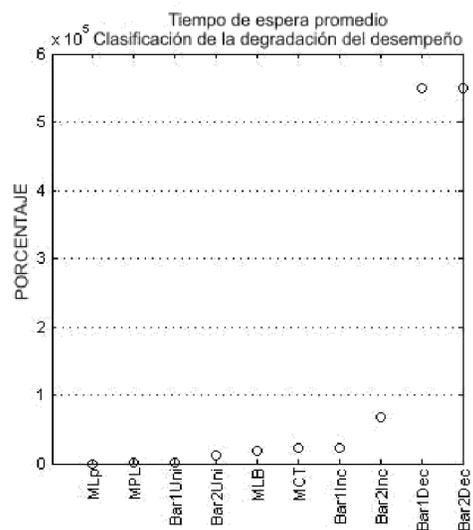


d. Grid 2 (6 mejores estrategias)

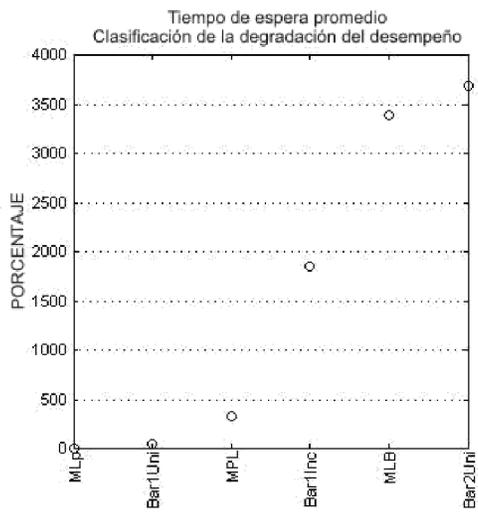
Figura 20. Degradación de Slowdown promedio limitado



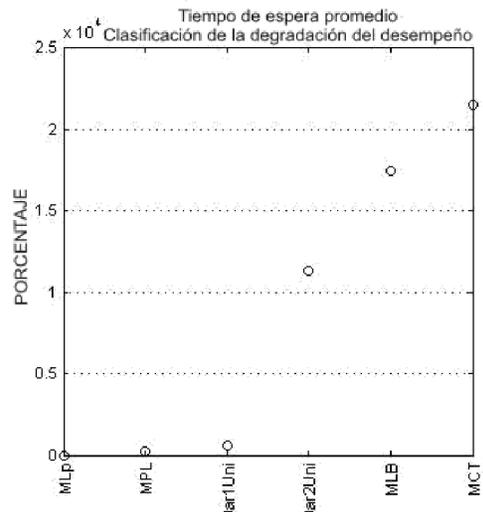
a. Grid 1 (clasificación)



c. Grid 2 (clasificación)



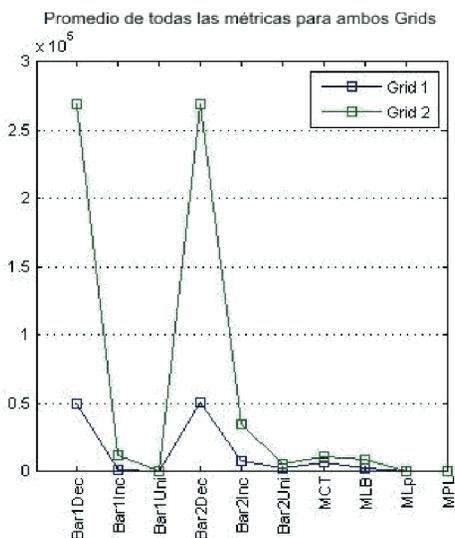
b. Grid 1 (6 mejores estrategias)



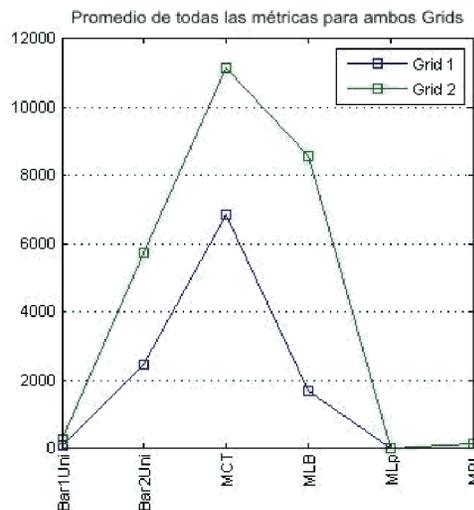
d. Grid 2 (6 mejores estrategias)

Figura 21. Degradación del Tiempo de espera promedio

V.3.3 Degradación promedio para Grid 1 y Grid 2

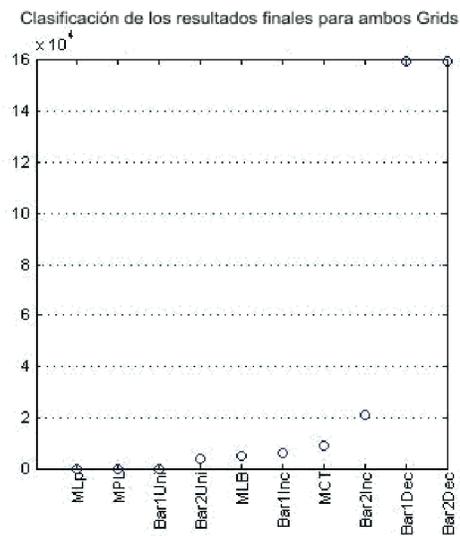


a. Diez estrategias

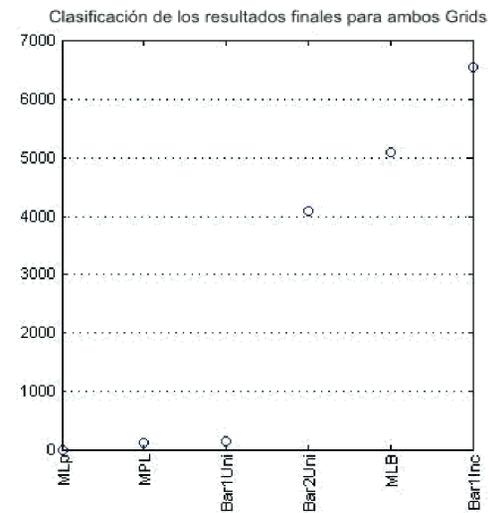


b. Seis mejores estrategias

Figura 22. Degradación promedio de todas las estrategias de asignación del Grid 1 y Grid 2



a. Diez estrategias



b. Seis mejores estrategias

Figura 23. Clasificación de la degradación promedio para todos los casos de estudio

De las estrategias de asignación analizadas bajo las métricas: *factor de competitividad*, *slowdown promedio* y *tiempo de espera promedio*, se puede observar que *MLp* y *MPL* muestran un mejor desempeño ante cualquier caso de estudio analizado. Las ventajas de estas dos estrategias es que no requieren solicitar un precalendario de las máquinas del sistema o de alguna otra información para su implementación. De igual forma, se observa que la versión *Bar1Uni* de la estrategia *BN* presenta un buen resultado ante cualquier caso de estudio.

Las figuras 22 y 23 muestran que el promedio relativo del desempeño de las estrategias de asignación *MLp* y *MPL* y *Bar1Uni* no dependen significativamente de la carga de trabajo, la configuración del Grid o incluso de la métrica de desempeño utilizada. Para más detalles sobre las cinco mejores estrategias de ambos Grids en la Tabla V se muestran los porcentajes de la degradación del desempeño para cada métrica.

Tabla V. Porcentajes de la degradación del desempeño de las cinco mejores estrategias para el Grid 1 y el Grid 2

Estrategia Métrica	Grid	<i>MLp</i>	<i>MPL</i>	<i>MLB</i>	<i>Bar1Uni</i>	<i>Bar2Uni</i>
ρ	Grid 1	0	8.576	72.54	0.114	36.77
	Grid 2	0	0.7689	40.03	1.023	39.77
SD_b	Grid 1	0	318.6	4176	45.8	4056
	Grid 2	0	348.1	2.483×10^4	648.4	1.485×10^4
MWWT	Grid 1	0	79.68	735.8	151.7	3220
	Grid 2	0	7.16	755.6	116	2279

En la Figura 23(b) se puede observar que dentro de las nuevas estrategias de asignación; *Bar1Uni* y *Bar2Uni* a pesar que consideran de forma diferente la carga dentro del sistema ofrecen buenos resultados en todos los casos de estudio.

Tabla VI. Valores teóricos y experimentales para *MCT*, *MLB* y *BN*

Resultados		<i>MCT</i>	<i>MLB</i>	<i>BN</i>
Teóricos	ρ	11	11	6.43
Experimental	%	1.82745	1.5795	1.0163

Por otro lado, en la Tabla VI se muestran los resultados teóricos para las estrategias *MCT*, *MLB* y *BN* así como los resultados observados durante la simulación de acuerdo al comportamiento promedio obtenido para cada estrategia (ver Figura 16). En la tabla se puede apreciar que la estrategia *BN* de acuerdo a sus valores teóricos y experimentales es la estrategia con mejores resultados comparada con las estrategias *MCT* y *MLB*. Asimismo, la estrategia *MCT* en promedio tiene un pobre desempeño en comparación a *MLB* a pesar que teóricamente ambas estrategias tienen una misma cota ($\rho \leq 11$) para el peor caso (ver Capítulo IV).

Capítulo VI

VI. Conclusiones

VI.1 Resumen

En este trabajo se aborda el análisis de calendarización de trabajos paralelos en un Grid computacional jerárquico de dos niveles dentro de un modelo en línea. Se estudia el desempeño de dos estrategias de calendarización $MLB_a + PS$ y $MCT_a + PS$. En ambas estrategias se aplica la estrategia de admisibilidad, el cual permite excluir máquinas de gran tamaño del conjunto de máquinas disponibles para tareas con grado de paralelismo pequeño.

Durante el trabajo se cumplieron con los objetivos planteados en la sección I.4. Se presentaron conceptos básicos relacionados al problema de calendarización en el marco teórico discutido en el Capítulo II. En el Capítulo III se trató el modelo de calendarización en el que se enfoca este trabajo así como, las estrategias y métricas a analizar. El análisis correspondiente se describió en el Capítulo IV, en este mismo capítulo se discuten los resultados obtenidos para ambas estrategias. Finalmente, en el Capítulo V se presenta el análisis experimental de la nueva estrategia de calendarización BN considerando las seis versiones implementadas además, se muestran los resultados obtenidos de las simulaciones para todas las estrategias de asignación evaluadas.

VI.2 Conclusiones

Considerando los resultados obtenidos del análisis teórico realizado en el trabajo se obtienen las siguientes conclusiones:

- A pesar de que la manera de seleccionar una máquina en ambas estrategias difiere, el factor de competitividad resultante es el mismo tanto para un modelo fuera de línea como para un modelo en línea.
- La utilización de la estrategia de admisibilidad permite la obtención de algoritmos con un factor de competitividad constante.
- El factor de admisibilidad puede ser ajustado dinámicamente en respuesta a los cambios que sufra la configuración del sistema dentro de un escenario real.
- El tiempo de ejecución¹³ del algoritmo de las estrategias $(MLB_a, MCT_a) + PS$ resulta en $O(nm + m \log m)$ donde m denota el número de máquinas dentro del sistema y n el número de trabajos.
- Se logró reducir los valores para el factor de competitividad para un modelo fuera de línea de $\rho \leq 10$ a $\rho \leq 9$. Asimismo se redujo el factor de competitividad para un modelo en línea de $\rho \leq 17$ a $\rho \leq 11$.
- En el análisis experimental se observó que las estrategias de asignación MLp y MPL resultan en un mejor desempeño que el resto de las estrategias analizadas.
- Se observó que la estrategia de asignación $BarInc$ no tuvo el desempeño esperado por ser la estrategia más parecida a la estrategia propuesta por Bar-Noy *et al.*, (2001).
- La estrategia de asignación $BarUni$ tuvo mejores resultados que las estrategias MCT y MLB (analizadas en la sección IV).

¹³ El recurso más común es el tiempo. Dentro de la complejidad del tiempo de un conjunto de problemas define qué tanto tiempo es necesario para resolverlos. El tiempo de ejecución es el número de pasos necesarios para resolver el problema y está en función del tamaño de la entrada. (Goddard, 2008)

- Para la selección de una estrategia de asignación adecuada para el Grid depende de las preferencias de los distintos administradores del Grid, proveedores de recursos y usuarios finales, así como la importancia de cada criterio de optimización

VI.3 Trabajo a futuro

Después de realizar los análisis correspondientes para cada estrategia evaluada, se considera el siguiente trabajo a futuro:

- Evaluar las estrategias presentadas de manera experimental para observar su comportamiento bajo cargas de trabajo y configuraciones del Grid reales.
- Considerar la aplicación de nuevas estrategias de asignación para la primer etapa de calendarización que no dependan significativamente de la carga de trabajo, la configuración del Grid o de la métrica de evaluación que se esté utilizando, como en el caso de *MLp* o *Bar1Uni*.
- Considerar en un algoritmo de calendarización local con un factor de competitividad de $\rho \leq 2 - 1/m$ en las simulaciones.

Referencias

- Albers S., (2003) *Online algorithms: a survey*. Mathematical Programming, Springer. 97(1):3-26.
- Azar Y., (1998) *On-line load balancing*. Online Algorithms, Lectures Notes in Computer Science. Springer-Verlag. 1448: 178-195 p.
- Bar-Noy A., Freund A. y Naor J., (2001) *On-line load Balancing in a Hierarchical Server Topology*. Society for Industrial and Applied Mathematics. 31:527-549 p.
- Berman F., (1999) *High-performance schedulers*. En: Foster I. y Kesselman C., The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc. San Francisco. 279-309 p.
- Blazewicz J., Ecker K., Pesch E., Schmidt G. y Weglarz J. (2007) *HANDBOOK ON SCHEDULING: Form theory to applications*. International Handbooks on Information Systems. Springer-Verlag. Primera edición. Berlín. 630 p.
- Braun T.D., Siegel H. J., Beck N., Boloni L. L., Maheswaran M., Reuther A. y Robertson J. P., (2001) *A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems*. Journal of Parallel and Distributed Computing. 61(6):810-837 p.
- Cormen T. H., Leiserson C. E., Rivest R. L. y Stein C., (2001) *Introduction to algorithms*. McGraw-Hill. Segunda edición. Boston. 1126 p.
- El-Rewini H., Lewis T. G. y Hesham H. A., (1994) *Task Scheduling*. IEEE computer. 28(2):27-37 p.
- Ernemann C., Hamscher V., Schwiegelshohn, Yahyapour R. y Streit A., (2002) *On Advantages of Grid Computing for Parallel Job Scheduling*. Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid. CCGRID '02. IEEE Computer Society. Washington. 39:37 p.
- Etsion Y. y Tsafir D., (2005) *A short survey of commercial cluster batch schedulers*. The Hebrew University. School of Computer Science and Engineering. Jerusalem. 1-4 p.
- Feitelson D. G., (2001) *Metrics for Parallel Job Scheduling and their Convergence*. Lectures Notes in Computer Science. Springer-Verlag. 2221:188-205 p.

- Feitelson D. G., Rudolph L. y Schwiegelshohn U., (2005) *Parallel Job Scheduling. A Status Report*. Job Scheduling Strategies for Parallel Processing. Springer-Verlag. 3277:1-16 p.
- Feitelson D. G., Rudolph L., Schwiegelshohn U., Sevcik K. C. y Wong P., (1997) *Theory and Practice in Parallel Job Scheduling*. Proceedings of the Job Scheduling Strategies for Parallel Processing, Springer-Verlag. 1291:1-34 p.
- Foster I. y Kesselman C., (2004) *The Grid in a nutshell*. En: Nabrzyski J., Schopf J. y Weglarz J., Grid resource management: state of the art and future trends. Kluwer Academic Publishers. Norwell. 3-13 p.
- Garey M. R. y Grahams R. L., (1975) *Bounds for multiprocessor scheduling with resource constraints*. SIAM Journal on Computing. 4:187-200 p.
- Garey M. R. y Johnson D., (1979) *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman. New York. 340 p.
- Goddard W., (2008) *Introducing the Theory of Computation*. Jones and Bartlett Publishers, Inc. Primera edición. USA. 228 p.
- Graham R. L., (1966) *Bounds for certain multiprocessor anomalies*. Bell System Technical Journal. 45(9):1563-1581 p.
- Graham R. L., (1969) *Bounds on multiprocessing timing anomalies*. SIAM Journal on applied Mathematics. 17(2):416-429 p.
- Graham R. L., Lawler E., Lenstra J. y Kan A. R., (1979) *Optimization and approximation in deterministic sequencing and scheduling: A survey*. Annals of Discrete Mathematics. Elsevier. 5:287-326 p.
- Hamscher V., Schwiegelshohn U., Streit A. y Yahyapour R., (2000) *Evaluation of Job Scheduling Strategies for Grid Computing*. Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing, GRID '00. Springer-Verlag. London. 191-202 p.
- Iovanella A., (2002) *On-Line algorithms for multiprocessing tasks scheduling*. Tesis doctoral. Universita degli Studi di roma "La Sapienza". Roma.
- Jiang C., Wang C., Liu X. y Zhao Y., (2007) *A survey of Job Scheduling in Grids*. Proceedings of the joint 9th Asia-Pacific web and 8th international conference on web-age information management conference en Advances in data and web management. Junio 16-18. China. 419-427 p.

- Karger D., Stein C. y Wein J., (1997) *Scheduling Algorithms*. CRC Handbook of Computer Science.
- Karp R. M., (1972) *Reducibility among combinatorial problems*. En: Miller R. y Thatcher J.W. Complexity of Computer Scheduling Strategies for Parallel Processing, Primera edición. Springer-Verlag. New York. 98-121 p.
- Kento A., (2000) *Effect of Job Size Characteristics on Job Scheduling Performance*. Job Scheduling Strategies for Parallel Processing. Springer-Verlag. 1911:1-17 p.
- Kento A., Kasahara H. y Narita S., (1998) *Job Scheduling Scheme for Pure Space Sharing Among Rigid Jobs*. Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, Marzo 30-Abril 3. Orlando. 98-121 p.
- Kesselman C. y Foster I., (1998) *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers. San Francisco. 653 p.
- Kurowski J., Nabrzyski J., Oleksiak A. y Weglarz J., (2006) *Scheduling jobs on the grid-multicriteria approach*. Computational methods in science and technology. 12(2): 123-138 p.
- Kurowski K., Nabryzyski J., Oleksiak A. y Weglarz J., (2008) *A multicriteria approach to two-level hierarchy scheduling in grids*. Journal of Scheduling. 11(5): 371-379 p.
- Kwok Y., Maciejewski A., Siegel H. J., Ahmad I. y Ghafoor A., (2006) *A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems*. Journal of Parallel and Distributed Computing. 66(1): 77-98 p.
- Li k., Cheng K. H., (1991) *Job Scheduling in Partitionable Mesh Using a Two-Dimensional Buddy System Partitioning Scheme*. IEEE Trans. Parallel Distributed Systems. 2(4): 413-422 p.
- Lifka D., (1995) *The ANL/IBM SP scheduling system*. Job Scheduling Strategies for Parallel Processing. 949: 205-303 p.
- Moallem A., (2009) *Using Swarm Intelligence for Distributed Job Scheduling on the Grid*. Saskatoon Canada: University of Saskatchewan.
- Naroska E. y Schwiegelshohn U., (2002) *On an on-line scheduling problem for parallel jobs*. Information Processing Letters 81(6): 297-304 p.
- Oyentunji E. O., (2009) *Some Common Performance Measures in Scheduling Problems: Review Article*. Research Journal of Applied Sciences, Engineering and Technology. 1(2): 6-9 p.

- Pascual F., Rzadca K. y Trystram D., (2007) *Cooperation in Multi-organization Scheduling*. Euro-Par 2007 Parallel Processing. Springer. 4641:224-233 p.
- Platform Computing, Inc. (2008) Clusters, Grids, Clouds – Platform Computing. Junio 17. <http://www.platform.com>.
- Rajasekaran S. y Rief J., (2008) *Scheduling in Grid Environments. Handbook of Parallel Computing: Models, Algorithms y Applications*. Computer and Information Science series. Chapman and Hall/ CRC. 2-16 p.
- Ramírez J. A., Rodríguez A., Tchernykh A. y Verduzco J., (2007) *Rendimiento de las Estrategias de Calendarización considerando Fluctuación de Tiempo de Ejecución de tareas en un Grid Computacional*. Conferencia Latinoamericana de Computación de Alto Rendimiento, CLCAR 2007. 13-18 Agosto. Santa María, Colombia. 237-281 p.
- Ramírez J. M., Tchernykh A., Yahyapour R., Schwiegelshohn U., Quezada-Pina A., Gonzalez J. L. y Hiraes A., (2010) *User run time estimate unaware online job scheduling in hierarchical grids*. Journal of Grid Computing.
- Schwiegelshohn U., Tchernykh A. y Yahyapour R., (2008) *Online scheduling in grids*. Parallel and Distributed Processing, IPDPS-2008. IEEE-International symposium. 14-18 Abril. Miami. 1-10 p.
- Schwiegelshohn U. y Yahyapour R., (1998) *Analysis of first-come-first-serve parallel scheduling*. Proceedings on the 9th annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics. 25-27 Enero. San Francisco. 629-638 p.
- Sgall J., (1998) *On-line scheduling*. Online algorithms. 196-231 p.
- Sipser M., (2005) *Introduction to the Theory of Computation*. 2nd ed. Computer Science Series. Primera edición. Cengage Learning. 396 p.
- Srinivasan S., Kettimuthu R. y Subramani V., (2002) *Characterization of Backfilling Strategies for Parallel Job Scheduling*. IEEE International Conference on Parallel Processing Workshops. 18-21 Agosto. 514-519 p.
- Talia D., Yahyapour R. y Ziegler W., (2008) *Grid Middleware and Services, Challenges and Solutions*. Core Grid. Springer Science+Business Media, LLC. 420 p.
- Tchernykh A., Ramírez J. M., Avetisyan A., Kuzjurin N., Grushin D. y Zhuk S., (2006) *Two Level Job-Scheduling Strategies for a Computational Grid*. Parallel Processing and Applied Mathematics, Springer-Verlag. 1911: 774-781 p.

- Tchernykh A., Schwiegelshohn U., Yahyapour R. y Kuzjurin N., (2008) *Online Hierarchical Job Scheduling on Grids*. From Grids to Service and Pervasive Computing. 77-91 p.
- Tchernykh A., Schwiegelshohn U., Yahyapour R. y Kuzjurin N., (2010) *On-line hierarchical job scheduling on grids with admissible allocation*. Journal of Scheduling. 13(5):545-555 p.
- Tsafrir D., Etsion Y. y Feitelson D., (2005) *Modeling User Runtime Estimates*. Job Scheduling Strategies for Parallel Processing. 1834: 1-35 p.
- Ullman J. D., (1975) *NP-Complete scheduling problems*. Journal of Computer and System Sciences. 10(3): 384-393 p.
- University of Chicago, (1997) The Globus Alliance. Enero 1. <http://www.globus.org>.
- University of Chicago, (2002) Open Grid Services Architecture OGSA. Enero 1. <http://www.globus.org>.
- University of Melbourne, (2004) The CLOUDS Lab: Flagship Projects-Gridbus and Cloudbus. Junio 17. <http://www.gridbus.org>.
- University of Virginia, (1993) Legion: A Worldwide Virtual Computer. Agosto 1. <http://www.legion.virginia.edu>.
- University of Wisconsin Madison, (1998) Condor Project Homepage. Enero 1. <http://www.cs.wisc.edu/condor>.
- Vazirani V., (2001) *Approximation algorithms*. Primera edición. Springer. 343 p.
- Wang Q., Gui X., Zheng S. y Xie B., (2004) *De-centralized job scheduling on Computational Grids using Distributed Backfilling*. Grid and Cooperative Computing GCC 2004. 3251:285-292 p.
- Zhu Y., (2003) *A Survey on Grid Scheduling Systems*. Tesis doctoral. Department of Computer Science: Hong Kong, University of Science and Technology.
- Zhuk S., Chernykh A., Avetisyan A., Gaissaryan S., Grushin D., Kuzjurin N., Pospelov A. y Shokurov A., (2004) *Comparison of Scheduling Heuristics for Grid Resource Broker*. Proceedings of the 5th Mexican International Conference in Computer Science. 13-17 Noviembre. IEEE Computer Society. 388-392 p.

APÉNDICE A

Complejidad computacional

De acuerdo a la teoría de la complejidad se pueden clasificar cuatro clases de problemas:

- Problemas \mathcal{P} (Goddard 2008; Cormen *et al.*, 2001): La clase \mathcal{P} consiste de aquellos problemas que se pueden resolver en tiempo polinomial ($O(n^k)$ para alguna constante k , donde n es el tamaño de la entrada del problema).
- Problemas NP (Goddard 2008; Cormen *et al.*, 2001): La colección de problemas que se pueden “verificar” en tiempo polinomial se consideran dentro de la clase NP . En otras palabras, se tiene un “certificado” de una solución dada, se puede verificar si el certificado es correcto en tiempo polinomial con respecto al tamaño de la entrada del problema.
- Problemas NP -Completo (Goddard 2008; Sipser 2005; Cormen *et al.*, 2001): Se considera un problema (\mathcal{A}) dentro de la clase NP -Completo si cumple con las siguientes condiciones:
 1. \mathcal{A} se encuentra dentro de la clase NP
 2. Todos los problemas de la clase NP (\mathcal{A}') se pueden reducir polinomialmente a \mathcal{A} ($\mathcal{A}' \leq_p \mathcal{A}$).
- Problemas NP -Difícil (Cormen *et al.*, 2001): Un problema \mathcal{A} se considera dentro de NP -Difícil si cualquier problema de la clase NP (\mathcal{A}') se puede reducir polinomialmente a \mathcal{A} ($\mathcal{A}' \leq_p \mathcal{A}$). Pero \mathcal{A} no necesariamente pertenece a la clase NP .

Uno de los mayores problemas sin resolver dentro de las matemáticas y de las ciencias computacionales es conocer si:

$$\mathcal{P} \neq \mathcal{NP}$$

Si estas clases fueran iguales, cualquier problema que se puede verificar polinomialmente se podría resolver en tiempo polinomial. Muchos investigadores creen que estas dos clases no son iguales debido a que se ha invertido mucho esfuerzo en encontrar un algoritmo de tiempo polinomial para ciertos problemas en \mathcal{NP} -Completo, sin ningún resultado exitoso. Las siguientes figuras muestran los dos posibles casos:

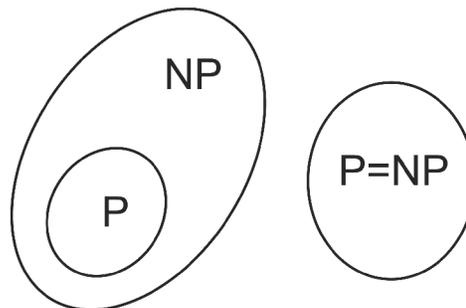


Figura 24. Posibilidades para la problemática si $\mathcal{P} \neq \mathcal{NP}$ (Sipser, 2005)

APÉNDICE B

Notación de tres campos

La notación fue propuesta por Graham *et al.*, (1979) y se compone de tres campos $\alpha|\beta|\gamma$. Los campos denotan las características de los procesadores o máquinas (α), de las tareas o recursos (β) y finalmente el criterio de optimización (γ), ver (Blazewicz *et al.*, 2007).

El primer campo α describe el ambiente de los procesadores. Se compone de dos parámetros α_1 y α_2 . El parámetro $\alpha_1 \in \{\emptyset, P, Q, R\}$ caracteriza el tipo de procesador que se está utilizando:

$\alpha_1 = \emptyset$: Un procesador

$\alpha_1 = P$: Procesadores idénticos

$\alpha_1 = Q$: Procesadores uniformes

$\alpha_1 = R$: Procesadores no relacionados

El parámetro $\alpha_2 \in \{\emptyset, k\}$ denota el número de procesadores involucrados en el problema de calendarización:

$\alpha_2 = \emptyset$: El número de procesadores es variable

$\alpha_2 = k$: El número de procesadores es igual a k (siendo k un entero positivo)

El segundo campo β describe las características de las tareas y/o recursos. El parámetro $\beta \in \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$.

El parámetro $\beta_1 \in \{\emptyset, pmtn\}$ indica la posibilidad de tareas con interrupciones, es decir si se puede dividir una tarea en subtareas. Parámetro $\beta_2 \in \{\emptyset, res\}$ caracteriza recursos adicionales. Parámetro $\beta_3 \in \{\emptyset, prec, uan, tree, chains\}$ refleja las relaciones de precedencia entre las tareas. Denota respectivamente tareas independientes, limitaciones de

precedencia generales, limitaciones de precedencia formando árboles o un conjunto de cadenas. El parámetro $\beta_4 \in \{\emptyset, r_j\}$ describe el instante de tiempo cuando las tareas se encuentran listas para su ejecución. El parámetro $\beta_5 \in \{\emptyset, p_j = p, \underline{p} \leq p_j, p_j \leq \bar{p}\}$ describe las características del tiempo de procesamiento de las tareas. Parámetro $\beta_6 \in \{\emptyset, \bar{d}\}$ describen las fechas límites de las tareas.

El tercer campo γ denota la métrica de desempeño que se está buscando optimizar en el problema. Por ejemplo: $\gamma \in \{C_{max}, \sum C_j, \sum C_j \cdot w_j, -, \dots\}$. El caso de “-“ significa que se está considerando un problema de decisión.