Tesis defendida por

Alonso Mitza Aragón Ayala

Y aprobada por el siguiente comité

Dr. Andrey Chernykh

Director del Comité

Dr. Raúl Rangel Rojo

Miembro del Comité

Dr. Carlos Alberto Brizuela Rodríguez

Miembro del Comité

Dr. Raúl Valente Ramírez Velarde

Miembro del Comité

Dr. José Antonio García Macías

Coordinador del programa de posgrado en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales

Director de la Dirección de Estudios de Posgrado

07 de septiembre de 2012

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR
DE ENSENADA


---

Programa de Posgrado en Ciencias
en Ciencias de la Computación

---

# Diseño y análisis experimental de algoritmos de calendarización de bajo consumo de energía para Grids jerárquicos de dos niveles

Tesis
para cubrir parcialmente los requisitos necesarios para obtener el grado de
Maestro en Ciencias

Presenta:
Alonso Mitza Aragón Ayala

Ensenada, Baja California, México
2012

Resumen de la tesis de Alonso Mitza Aragón Ayala, presentada como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Diseño y análisis experimental de algoritmos de calendarización de bajo consumo de energía para Grids jerárquicos de dos niveles

Resumen aprobado por:

_____

Dr. Andrey Chernykh

En este trabajo nos enfocamos en algoritmos energéticamente eficientes para calendarizar trabajos paralelos en Grids jerárquicos de dos niveles. Nuestras estrategias apagan recursos cuando no son necesarios, y los vuelven a encender cuando así se requiera. Los algoritmos se dividen en una asignación de los recursos a trabajos y una calendarización local en cada sitio. Consideramos cargas de trabajo de sitios reales con el fin de obtener resultados válidos en escenarios reales. Consideramos cuatro criterios para evaluar nuestras estrategias, los cuales están en conflicto entre sí, por lo que una estrategia que se desempeñe aceptablemente en uno de ellos, puede no hacerlo en los demás; estos criterios son el tiempo de espera, suma ponderada de ralentización, consumo de energía y un factor de competitividad en base a un óptimo teórico. Consideramos dos escenarios Grid basándonos en sistemas heterogéneos, y para la carga de trabajo consideramos trabajos paralelos de los cuales no conocemos de antemano el tiempo de ejecución exacto, sólo un tiempo estimado por el usuario. Nuestros resultados obtenidos muestran que para los casos estudiados puede reducirse el consumo energético de un Grid sin perder calidad en las métricas tradicionales.

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

Abstact of the thesis presented by Alonso Mitza Aragón Ayala as a partial requirement to obtain the Master of Science degree in Computer Science.

Diseño y análisis experimental de algoritmos de calendarización de bajo consumo de energía para Grids jerárquicos de dos niveles

Abstract approved by:

_____

Dr. Andrey Chernykh

In this work, we focus on energy-efficient algorithms to schedule parallel jobs in hierarchical two-level Grids. Our strategies turn resources off when they are not needed and turn them back on when they are required. Our algorithms can be divided in two parts: an assignment of resources to jobs and local site scheduling. We consider real-life based workloads in order to get valid results. We evaluate four criteria that are conflicted with each other, meaning that a good strategy in one criterion can perform badly in another. These criteria are waiting time, bounded slowdown, energy consumption and competitive ratio. We consider two heterogeneous Grid scenarios, and our workload consists of parallel jobs with a user run time estimate. Our results show that, for our study cases, energy consumption can be reduced without loss of quality considering traditional metrics.

Keywords: **Grid computing, Scheduling, Energy efficiency, Allocation strategies, Resource management.**

## Dedicatorias

Para Alfonso y Lourdes, grandes como sólo ellos pueden.

**Agradecimientos**

A CICESE, la institución donde se llevó a cabo esta investigación, a CONACyT por la beca para realizar este trabajo, a mi director de tesis por guiarme y a mis padres por su apoyo incondicional.

## Tabla de contenido

## Lista de tablas

## Lista de figuras

# 1. Introducción

En este capítulo se presenta una introducción a los conceptos más comunes del cómputo en Grid. También se define el problema a tratar y se plantean los objetivos a lograr.

## 1.1 Computación en Grids

El término *Computación en Grid* fue utilizado por primera vez a mediados de la década de 1990 para definir una propuesta de infraestructura de cómputo que permitiera la utilización de recursos computacionales geográficamente distribuidos como si fueran uno solo, ejecutando aplicaciones que requiriesen un nivel de desempeño que la infraestructura de cómputo tradicional (incluyendo supercomputadoras) no podía proveer (Foster, Kesselman, 1999, 2003; Berman, Hey, 2004). Estos problemas se llamaron *problemas de gran reto*, entre los que encontramos temas como física de alta energía, meteorología, biología, astronomía, medicina, genética, farmacología, química y economía (CERN, 2008).

Un Grid es considerado como una colección de recursos de cómputo y almacenamiento distribuidos geográficamente y mantenido para cubrir las necesidades computacionales de alguna organización virtual (VO) .

Una organización virtual puede definirse como un conjunto de individuos e instituciones con actividades afines y que han establecido reglas para compartir sus recursos entre ellos (Foster *et al*, 2001).

Los recursos que pueden ser compartidos en este contexto de computación distribuida pueden ser unidades de procesamiento, sistemas de almacenamiento y dispositivos especializados de entrada y salida y datos.

La idea de la Computación en Grid surgió como una analogía a la red de energía eléctrica, donde los generadores de energía están distribuidos y los usuarios son capaces de acceder a la energía eléctrica sin preocuparse acerca del lugar de origen de esa energía, del medio de transporte o de los mecanismos de almacenamiento y regulación (Buyya, 2005). Así pues, los usuarios participantes

de una comunidad de Grid son capaces de utilizar grandes recursos mediante el uso de interfaces relativamente simples, sin tener la necesidad de conocer a detalle el funcionamiento tecnológico de este servicio.

El problema de administrar de forma óptima un conjunto de recursos distribuidos ha sido es un tema de estudio de múltiples investigaciones.

Existen tres áreas principales en el desarrollo del cómputo distribuido: centrados en cómputo, centrados en datos y centrados en comunidad. Los primeros son del dominio del cómputo de alto rendimiento, donde el usuario requiere una alta capacidad de procesamiento; la segunda área corresponde a problemas que manejan una cantidad muy grande de datos (Petabytes y Exabytes); y la tercera, a la que también se le llama *aplicaciones colaborativas,* intenta reunir comunidades para realizar trabajo colaborativo sobre medios electrónicos. Esta última clasificación es donde se encuentran los proyectos @home como SETI@home (California, 2009b), Rosetta@home (Washington, 2009), Einstein@home (MIT, 2009) y Folding@home (Beberg *et al*, 2009). Este tipo de proyectos de cómputo voluntario y colaborativo puede ser considerado como el precursor del Grid.

Los Grids fueron creados principalmente para sufragar las dos primeras tendencias y, en base a esto, pueden diferenciarse los Grids computacionales y los Grids de datos.

Aun cuando la creación del Grid se debió a las necesidades existentes en el ámbito de la investigación científica, han sido rápidamente adoptados por grandes empresas y organizaciones que buscan aprovechar las ventajas que este tipo de plataforma les ofrece para compartir sus propios recursos geográficamente distribuidos.

### 1.2 Planteamiento del problema

Debido a la creciente disponibilidad y decreciente costo de los sistemas de cómputo, es cada vez más fácil que empresas e instituciones tengan su centro de supercómputo. Esto genera problemas relacionados con la utilización de estos centros, como el incremento en gastos de consumo eléctrico y en las emisiones

generadas, relacionadas con los gases del efecto invernadero. El consumo energético de las tecnologías de información (excluyendo el requerido en su fabricación) fue alrededor del 8% del consumo energético global en el año 2006, y se cree que será un 14% para el año 2020 (Develder *et al*, 2007).

### 1.3 Cómputo ecológico en Grids

Los problemas mencionados previamente generaron un área de investigación completamente nueva en el cómputo distribuido: la *computación ecológica,* que busca generar sistemas de cómputo energéticamente eficientes y así minimizar estos problemas sin perder mucha calidad en los otros aspectos del desempeño del sistema tales como la velocidad y el espacio.

Existen varias agrupaciones que se dedican a investigar en el área del cómputo ecológico o verde, tales como Green Touch (2010), un consorcio que busca reducir los gases del efecto invernadero que resultan de la siempre creciente red de computadoras mundial; y Green-Net (2008), un proyecto colaborativo que busca crear plataformas de software para sistemas distribuidos de gran escala.

### 1.4 Investigación previa relevante

En Ghandi *et al* (2009) se presenta el siguiente problema: *cómo distribuir la potencia disponible entre servidores en una granja de tal forma que se minimice el tiempo promedio de respuesta.* En el artículo se trabaja con un presupuesto predefinido de potencia, el cual debe ser distribuido entre servidores de tal forma que se minimice el tiempo de respuesta. Se utilizan tres esquemas principales: *PowMax*, *PowMin* y *PowMed*. El primero considera pocos servidores trabajando a su máxima capacidad, el segundo considera a todos los servidores disponibles funcionando a poca frecuencia, y el tercero es un nivel intermedio entre los dos anteriores.

Los experimentos consideran un servidor IBM BladeCenter, y muestran que ninguno de los tres esquemas es superior ya que el desempeño de cada uno depende de los trabajos entrantes. Los resultados muestran que al utilizar el

esquema óptimo para cada caso, el tiempo de respuesta promedio puede ser reducido por hasta un factor de 5.

Una restricción de este trabajo es que consideran solamente llegadas estáticas de trabajos al sistema, lo cual no representa completamente un escenario realista en el que los trabajos pueden llegar en cualquier momento. Además, al estar trabajando en servidores, trabajan variando la frecuencia de operación de los procesadores, lo cual es muy complejo de hacer en un modelo de Grid.

En Develder *et al* (2008) se utiliza el modelo presentado en Vereecken *et al* (2008) para representar el consumo de energía en los servidores. Se trabaja con un modelo de Grid jerárquico de dos niveles que debe procesar trabajos independientes. La estrategia presentada en este artículo consiste en asignar trabajos a un subconjunto de todos los servidores disponibles, apagando los servidores que no estén trabajando por un lapso de tiempo predefinido. Los resultados mostrados muestran que en escenarios con poca carga, el consumo de energía puede reducirse a un nivel cercano al que tendría un sistema dimensionado para esa carga. Los experimentos se realizaron con cargas de trabajo reales extraídas de los Grids EGEE/LCG.

La deficiencia principal de este trabajo es que solamente considera encender o apagar sitios completos, no sus componentes. Además, no se consideran trabajos paralelos, los cuales son comunes en el cómputo en Grids.

En este trabajo diseñamos y analizamos un modelo energético para Grids computacionales que no tiene los problemas de los trabajos mencionados y que busque optimizar el consumo energético del equipo.

## 1.5 Objetivos de la investigación

### 1.5.1 Objetivo general

Diseñar y realizar un análisis experimental de algoritmos de calendarización amigables con el ambiente para Grids jerárquicos de dos niveles.

### 1.5.2 Objetivos específicos

- Desarrollar un modelo de consumo de energía para Grids.
- Diseñar el manejo de energía para algoritmos de calendarización en Grids.
- Crear una estrategia que considere el consumo energético.
- Agregar factores de eficiencia energética y velocidad de los sitios a estrategias ya existentes.

## 1.6 Preguntas de investigación

- ¿Cómo puede minimizarse la energía cuando no se están utilizando todos los recursos?
- ¿De qué forma puede ser evaluado el consumo de energía en un Grid jerárquico de dos niveles?
- ¿Cómo puede un análisis multicriterio ayudar a encontrar el mejor resultado en un modelo de Grid consciente de la energía?
- ¿Cómo puede el considerar el consumo de energía cambiar la calidad de los calendarios?
- ¿Cómo afectan las políticas de calendarización locales o globales el consumo de energía y la calidad de los calendarios generados?
- ¿Cómo afecta el considerar el consumo de energía a la complejidad de los algoritmos de calendarización?

## 1.7 Justificación

Debido a las prioridades tanto de los consumidores como de los proveedores de los servicios de cómputo en Grids, la investigación en esta área se ha enfocado principalmente en aspectos como maximizar la velocidad de procesamiento o minimizar la utilización de recursos, por nombrar algunos; el impacto ambiental ha sido relegado a un segundo plano hasta años recientes, cuando problemas como la contaminación y el cambio en las pautas meteorológicas se han convertido en problemas prioritarios a nivel mundial.

El incremento en la cantidad de gases de invernadero está alterando el clima global, creando problemas como inundaciones y altas temperaturas. Para detener la acumulación de estos gases, es necesario detener o al menos reducir el incremento global de emisiones, en el que la generación de energía tiene un papel importante debido a que libera contaminantes como azufre en la atmósfera.

Además de estas repercusiones ambientales, la economía también se ve afectada mediante incrementos en tarifas de energía eléctrica. Este es un gran problema para empresas que tienen una gran cantidad de equipo de cómputo, como los centros de datos (*datacenters*) que, al estar trabajando constantemente, necesitan de poderosos sistemas de refrigeración, los cuales consumen grandes cantidades de energía.

El cómputo consciente de la energía se presenta como un cambio en el punto de vista y funcionamiento de los sistemas de cómputo de alto nivel, pasando de sistemas “principalmente desempeño” a sistemas “desempeño-energía” equilibrados que sean capaces de reducir costos e impacto en el medio ambiente mediante mejoras a la utilización de recursos, manteniendo siempre una calidad predeterminada del servicio (Moreno, Xu, 2011).

Los Grids de computadoras están diseñados para manejar la carga máxima de un sistema; pero durante periodos de menor carga el exceso de recursos puede apagarse.

Estudios recientes en la utilización de recursos de Grids muestran que la utilización de un sitio en particular puede variar mucho diariamente (entre menos que un 20% y más que un 90%) (Orgerie *et al*, 2008), así que se busca tener encendidos solamente los recursos que sean necesarios.

## 2. Descripción del modelo

En este capítulo se describe la arquitectura de nuestro modelo y se detallan los elementos que lo componen. Posteriormente se describen formalmente las métricas y parámetros que consideramos necesarios.

### 2.1 Arquitecturas de calendarización

Las arquitecturas de calendarización pueden clasificarse en distribuidas, centralizadas y jerárquicas.

En la arquitectura centralizada todos los sitios son administrados por una instancia central, la cual es la que recibe los trabajos y mantiene información sobre el estado de las máquinas que componen el Grid. Tiene -como cualquier sistema centralizado- desventajas como poca escalabilidad, baja tolerancia a fallos y una incapacidad de trabajar con distintas políticas de calendarización. Como principal ventaja cuenta con la capacidad de crear calendarios eficientes, dado que el calendarizador principal tiene toda la información necesaria para trabajar.

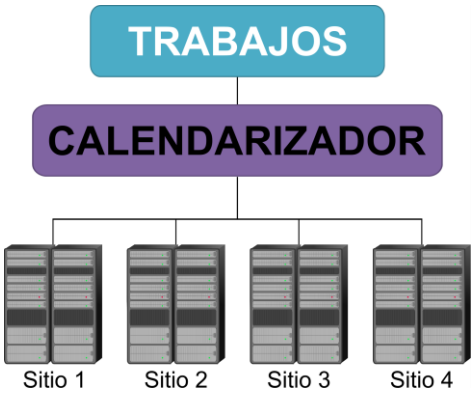


**Figura 1: Arquitectura de sistemas centralizados**

La arquitectura distribuida utiliza un calendarizador para cada máquina del Grid, los cuales se comunican entre sí para decidir dónde ejecutar cada trabajo. Al no contar con un administrador principal, es más estable y es más sencillo escalar el sistema (comparándolo con la arquitectura centralizada), y dado que cada sitio

tiene su propio calendarizador, se pueden implementar diferentes estrategias para cada sitio dependiendo de las políticas y necesidades que tengan. En contraste y como principal desventaja, los sistemas distribuidos mantienen una gran carga de comunicación entre los sitios; además, dado que ningún calendarizador cuenta con la información de todos los sitios, los sistemas distribuidos suelen obtener calendarios menos eficientes que los sistemas centralizados.

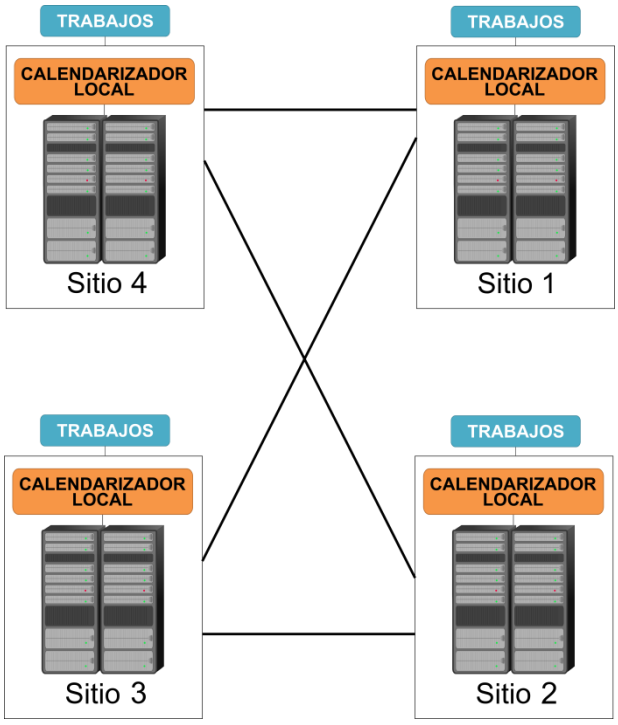


**Figura 2: Arquitectura de sistemas distribuidos**

La arquitectura jerárquica es la que mejor representa a los Grids actuales. Como su nombre lo dice, en esta arquitectura la administración está estructurada en niveles. Un caso común –y el que utilizamos en este trabajo– es el de dos niveles; en él, el nivel superior corresponde al calendarizador del Grid, que es el que recibe los trabajos y, por ello, tiene información sobre el tamaño y los requisitos de los trabajos, mas no suele tener información en tiempo real sobre el estado de los sitios del Grid.

El nivel inferior está compuesto por los sistemas locales de administración; cada uno de ellos conoce con detalle el estado de los recursos de ese sitio y de los trabajos que recibe del administrador general.

En sistemas más grandes la arquitectura jerárquica puede incluir más niveles para facilitar el funcionamiento; por ejemplo, Días de Assunçao *et al* (2008) consideran un metacalendarizador que actúa como selector de Grids.


**Figura 3: Arquitectura de sistemas jerárquicos**

La Figura 4 muestra el modelo jerárquico que se utiliza en esta investigación. Consideramos un sistema administrador de los recursos del Grid (GRMS por sus siglas en inglés) el cual recibe los trabajos de los usuarios. Dicho sistema (*broker)* tiene una cola global, que es a donde llegan los trabajos recibidos. El Calendarizador del Grid se encarga posteriormente de tomar los recursos de esa cola y asignarlos a los recursos o sitios disponibles. Una vez que se elige el sitio, se envía el trabajo al sistema administrador local de recursos (LRMS por sus siglas en inglés) de dicho sitio. El LRMS recibe los trabajos y los coloca en una cola local, de la cual serán tomados por el calendarizador local para su asignación final a los procesadores que ejecutarán el trabajo. Consideramos los recursos locales como recipientes rectangulares de ancho limitado por el número de

procesadores y altura ilimitada correspondiente al tiempo que el recurso está disponible. Consideramos trabajos paralelos, y los representamos como bloques rectangulares cuyo ancho corresponde al número de procesadores asignados y su altura corresponde a su tiempo de ejecución. Este tipo de problemas se conoce como Strip Packing (Martello *et al*, 2003) y, en el ambiente del Grid: Multiple Strip Packing.

Los trabajos paralelos que utilizamos se consideran rígidos; es decir, el trabajo no puede ejecutarse si no están disponibles el número de procesadores que se requieren, los cuales deben estar asignados durante un periodo ininterrumpido de tiempo a éste hasta que finalice su ejecución.

Existen otros tipos de trabajos, denominados maleables (Dutton, Mao, 2007) y moldeables (Cirne, 2001), los cuales pueden ejecutarse con un número inferior de procesadores al requerido y continuar así hasta su terminación (moldeables) o cambiar el número de procesadores que utilizan durante su ejecución (maleables). Estos trabajos no están considerados en la presente investigación.

### 2.2 Definición formal

Abordamos un problema de asignación de trabajos en línea. Consideramos $n$ trabajos paralelos $J_1, J_2, \dots, J_n$, los cuales deben ser calendarizados en $m$ sitios (máquinas) $N_1, N_2, \dots, N_m$. Si utilizamos $m_i$ para denotar el número de procesadores idénticos de la máquina $N_i$, definimos el número total de procesadores pertenecientes a las máquinas desde $N_1$ hasta $N_m$ por

$$m_{i,m} = \sum_{i=1}^{m} m_i \qquad (1)$$

Se supone, sin pérdida de generalidad, que las máquinas están organizadas en orden no-decreciente respecto al número de procesadores, es decir, se cumple que $m_1 \leq m_2 \leq \cdots \leq m_m$. Además, cada sitio $N_i$ tiene una velocidad de procesamiento $v_i$ y una eficiencia energética $eff_i$; estos valores son constantes.

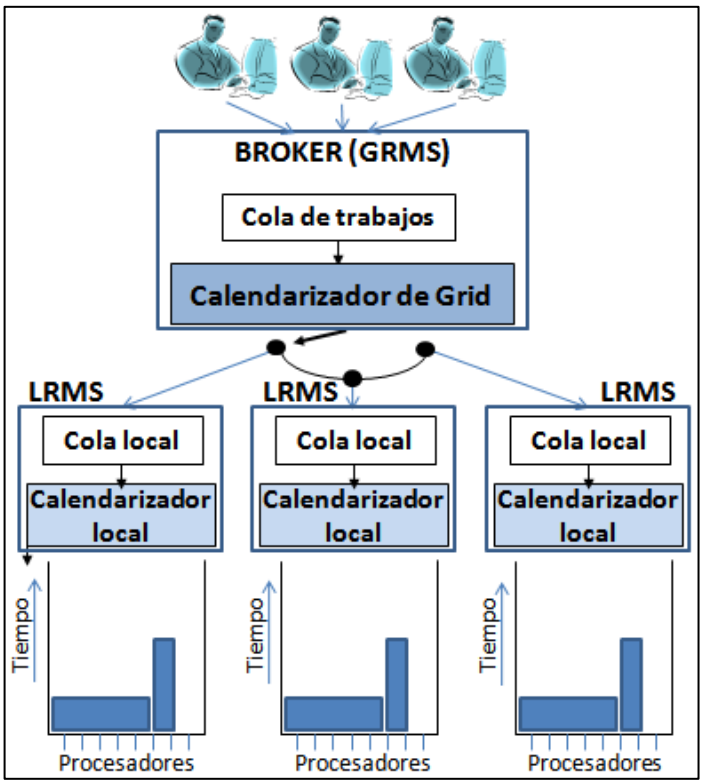


**Figura 4. Modelo jerárquico utilizado**

Cada trabajo $J_j$ es definido por la tupla $(r_j, size_j, p_j, p'_j)$: su fecha de llegada $r_j \geq 0$; su tamaño $1 \leq size_j \leq m_m$, también denominado su grado de paralelismo, se refiere al número de procesadores requeridos por ese trabajo; su tiempo de procesamiento $p_j$ y el tiempo de procesamiento estimado por el usuario al momento de enviar el trabajo al sistema $p'_j$. La fecha de llegada de cada trabajo es desconocida antes de que el trabajo sea enviado, y su tiempo de procesamiento no es conocido hasta que el trabajo ha terminado su ejecución (modelo no-clarividente). El tiempo estimado provisto por el usuario para cada uno de sus trabajos es utilizado como límite de ejecución, después del cual el trabajo será interrumpido por el sistema si aún no ha finalizado su procesamiento, esto previene el desperdicio de recursos debido a alguna falla en la ejecución de los trabajos.

Definimos $t_w^j$ como el tiempo que el trabajo $J_j$ debe esperar desde su llegada hasta el momento de iniciar su ejecución, y $s_j$ como el instante en que su ejecución inicia. Definimos también $w_j = p_j \cdot size_j$ y $w_j' = p_j' \cdot size_j$ como el total de recursos consumidos y la estimación por parte del usuario de los recursos consumidos para el trabajo $J_j$, respectivamente.

Cuando un trabajo llega al sistema, este es inmediata e irrevocablemente asignado a un único sitio. Si ese sitio no tiene el número requerido de procesadores disponibles, la asignación del trabajo puede ser retrasada hasta que este requisito se cumpla.

Un sitio debe ejecutar un trabajo asignando exclusiva y exactamente el número de procesadores requeridos ($size_j$) por un periodo de tiempo ininterrumpido $p_j$. Dado que no consideramos la ejecución de un trabajo en múltiples sitios, para que el trabajo $J_j$ se procese en el sitio $N_i$ debe cumplirse que $size_j \leq m_i$. Suponemos que todos los recursos disponibles son estables y dedicados al Grid.

Utilizaremos $g_j = i$ para representar que el trabajo $J_j$ es asignado al sitio $N_i$ y $n_i$ para denotar la cantidad de trabajos asignados al sitio $N_i$. El tiempo de terminación del trabajo $J_j$ del caso $I$ en un calendario $S$ es denotado por $c_j(S, I)$ y el tiempo de terminación del calendario $S$ en la instancia $I$ es $c_{\max}(S, I) = max_{J_j}\{c_j(S, I)\}$. Utilizamos $c_{max}^*(I)$ para representar el calendario óptimo para el caso $I$. Cuando sea posible y sin causar ambigüedad, omitiremos $I$ y $S$.

En nuestras simulaciones utilizamos cuatro métricas (ver Tabla 1), tres de las cuales se consideran comúnmente al buscar expresar los objetivos de diferentes participantes del Grid, como lo son los usuarios y los proveedores de recursos. La cuarta estrategia representa el consumo energético del Grid; esta métrica está en conflicto con las demás, ya que al buscar reducir el consumo energético, el desempeño en las métricas tradicionales debe deteriorarse.

Por tanto, se busca un equilibrio entre desempeño y consumo energético: obtener un consumo bajo sin reducir considerablemente el desempeño del Grid.

Tabla 1: Métricas consideradas

| | Métrica | Descripción |
|---|---|---|
| | ***Centradas en el algoritmo*** | |
| 1 | Factor de competitividad | $\rho = \frac{C_{max}}{C^*_{max}}$ |
| | ***Centradas en el usuario*** | |
| 2 | Promedio del tiempo de espera | $t_w = \frac{1}{n}\sum_{j=1}^{n}(c_j - p_j - r_j)$ ó $t_w = \frac{1}{n}\sum_{j=1}^{n}(s_j - r_j)$ |
| 3 | Suma ponderada de la ralentización. | $SD_b = \frac{1}{n}\sum_{j=1}^{n}\frac{c_j - r_j}{\max\{10, p_j\}}$ |
| | ***Centradas en el sistema*** | |
| 4 | Consumo de energía | $E_{Grid} = \sum P_{Grid}(t)$ |

No consideramos otras métricas debido a que son muy similares a estas. Por ejemplo, la métrica de suma de tiempos de espera

$$SWT = \sum_{j=1}^{n}(c_j - p_j - r_j) \quad (2)$$

cuya diferencia con $t_w$ es solamente la constante $1/\mathrm{n}$.

Debido a que no es viable determinar un *makespan* óptimo, para evaluar nuestros experimentos utilizamos la mínima cota inferior posible de éste, representada por $\check{C}^*_{max}$ , la cual definimos como

$$C^*_{max} \geq \check{C}^*_{max} = max\left\{max_j(r_j + p_j), \frac{\sum_{j=1}^{n} w_j}{m}\right\} \quad (3)$$

Como se demostró en Schwiegelshohn *et al* (2008), no existe ningún algoritmo que produzca -en tiempo polinomial- un calendario S con $\frac{C_{max}}{C^*_{max}} < 2$ para este problema a menos que $P = NP$.

Denotamos nuestro modelo de Grid como $GP_m$ y, utilizando la notación de tres campos $(\alpha|\beta|\gamma)$ introducida por Graham *et al* (1979), caracterizamos nuestro problema de calendarización como $GP_m|r_j, size_j|\{\rho, t_w, SD_b, E_{Grid}\}$, que describe el ambiente de máquinas $(\alpha)$, las características de los trabajos $(\beta)$ y las funciones

objetivo $(\gamma)$. Finalmente, utilizamos la notación MPS (*Multiple Parallel Scheduling*: Múltiple calendarización paralela) para referirnos a nuestro problema, y la notación PS (*Parallel Scheduling*. Calendarización paralela) para describir la calendarización paralela en una única máquina de múltiples procesadores o núcleos.

## 3. Modelo de energía

Dado que el propósito principal de este proyecto es crear y evaluar estrategias de calendarización energéticamente eficientes, se deben tener valores claros del consumo de los componentes del sistema.

### 3.1 Definición formal

Nuestro modelo de energía es muy similar al utilizado en Barrondo *et al* (2012) Cada sitio $N_i$ con $m_i$ procesadores está compuesto por $CH_i$ chasises, $BO_i$ tarjetas en cada chasís y $CR_i$ procesadores en cada tarjeta. Por simplicidad, se supondrá que todos los procesadores de cada máquina son idénticos. Cada uno de los componentes consumen una cantidad determinada de energía si están encendidos, y los procesadores consumen una cantidad extra si se encuentran trabajando.

En este modelo se le da un periodo de gracia $T_{cr}^{shutdownCore}$ a los procesadores: si el procesador $CR_i$ se encuentra ocioso por un determinado intervalo de tiempo, se apagará para ahorrar energía. Cuando todos los procesadores contenidos en una tarjeta $BO_i$ son apagados, la tarjeta $BO_i$ también se apagará. Finalmente, cuando todas las tarjetas contenidas en el sitio $N_i$ se encuentren apagadas, comenzará un segundo periodo de gracia $T_i^{shutdownSite}$; si al finalizar este periodo ninguna tarjeta ha vuelto a encenderse, el sitio $N_i$ se apagará también.

Si el *broker* asigna un trabajo a un sitio $N_i$ que en ese momento está apagado, dicho trabajo deberá esperar un periodo $T_i^{startSite}$ durante el cual el sitio se encenderá y preparará para procesar. Durante ese periodo, el sitio consume una cantidad $P_i^{startSite}$ de energía. De esta manera, podemos calcular el consumo energético que un sitio en particular consume solamente por las veces en que fue encendido con la fórmula:

$$E_i^{startSite} = n_i^{startSite} \cdot T_i^{startSite} \cdot P_i^{startSite} \quad (4)$$

donde $n_i^{startSite}$ representa el número de veces que el sitio $i$ fue encendido.

El consumo energético del Grid puede ser calculado mediante la suma del consumo al estar operacional y el consumo total de los sitios al ser encendidos. Esto puede representarse mediante la fórmula:

$$E^{grid} = E^{opGrid} + E^{startGrid} \tag{5}$$

donde:

$$E^{opGrid} = \sum_{t=1}^{C_{max}} P^{opGrid}(t)$$

denota la energía consumida por el Grid al encontrarse operacional y:

$$E^{startGrid} = \sum_{i=1}^{m} E_i^{startSite}$$

denota la energía que se consume al iniciar cada sitio. Se utiliza la fórmula:

$$P^{opGrid}(t) = \sum_{i=1}^{m} q_i(t) \cdot \left(P_i^{idlesite} + P_i^{opSite}(t)\right) \tag{6}$$

para representar el consume energético del Grid en un momento $t$. $q_i(t)$ es una variable binaria que toma el valor de 1 si el sitio $i$ se encuentra encendido en un momento $t$, y 0 (cero) en caso contrario. $P_i^{idlesite}$ es un valor constante que representa el consumo de energía del sitio $i$ cuando éste está encendido. $P_i^{opSite}(t)$ se utiliza para representar el consumo energético del sitio $i$ cuando está trabajando:

$$P_i^{opSite}(t) = \sum_{ch}^{CH_i} r_{ch}(t) \cdot P_{ch}^{opBoard}(t) \tag{7}$$

donde $r_{ch}(t)$ es igual a 1 si el chasís $ch = 1, \ldots, CH_i$ está encendido en un momento dado $t$, y 0 (cero) en caso contrario. $P^{idleChassis}$ es un valor constante que representa el consumo energético de un chasís al estar encendido. $P_{ch}^{opBoard}(t)$ representa un consume extra de energía del chasís $ch$ al encontrarse trabajando:

$$P_{CH_i}^{opBoard}(t) = \sum_{bo=1}^{BO_i} s_{bo}(t) \cdot \left(P^{idleBoard} + P_{bo}^{opCore}(t)\right) \quad (8)$$

$s_{bo}(t)$ es igual a 1, si la tarjeta $bo = 1, \ldots, BO_i$ está encendida en un momento dado $t$, y 0 (cero) en caso contrario. $P^{idleBoard}$ representa el consumo energético de una tarjeta al estar encendida, $P_{bo}^{opCore}(t)$ se utiliza para representar el consumo extra de energía de la tarjeta $bo$ al encontrarse trabajando. Finalmente:

$$P_{BO_i}^{opCore}(t) = \sum_{cr=1}^{CR_i} w_{cr}(t) \cdot \left(P^{idleCore} + v_{cr}(t) \cdot P^{workCore}\right) \quad (9)$$

se utiliza para denotar el consumo energético de todos los procesadores contenidos en la tarjeta $BO_i$, donde $w_{cr}(t)$ es igual a 1 si el procesador $cr = 1, \ldots, CR_i$ está encendido en un momento dado $t$, y 0 (cero) en caso contrario. $P^{idleCore}$ es un valor constante que representa el consumo energético de un procesador al encontrarse ocioso (sin procesar ningún trabajo). $v_{cr}(t)$ es igual a 1 si el procesador $cr$ está trabajando (procesando un trabajo) en un momento dado $t$, y 0 (cero) en caso contrario. $P^{workCore}$ es un valor constante que representa el consumo energético extra de un procesador trabajando.

# 4. Estrategias y algoritmos de calendarización

Las estrategias de asignación de trabajos que se implementaron en esta investigación fueron clasificadas en base al tipo de información que utiliza cada una de ellas. La clasificación de los niveles de información, las estrategias de asignación implementadas y los algoritmos de calendarización utilizados son descritos en este capítulo.

## 4.1 Estrategias de asignación de trabajos

Como se mencionó previamente, el esquema utilizado para calendarización en Grid consta de dos partes: una asignación global y un calendarizador local, así pues, consideramos MPS como una estrategia de calendarización de dos etapas:
Como comentamos antes, nuestro esquema de calendarización en Grid consta de dos partes, una asignación global y una calendarización local, así pues, consideramos *MPS* como una estrategia de calendarización de dos etapas: *MPS* = *MPS_Alloc* + *PS*. En la primera etapa se asigna un sitio para cada trabajo que ingrese al sistema; en la segunda, el algoritmo PS se aplica independientemente en cada sitio para los trabajos asignados durante la etapa anterior.
Distinguimos las estrategias de asignación dependiendo del tipo de información requerida. La información de velocidad y eficiencia energética de cada sitio es conocida de antemano. Clasificamos cuatro niveles de información disponibles para la asignación de trabajos:
Nivel 0: No se cuenta con información del sistema o de los trabajos.
Nivel 1: Se cuenta con información sobre los sitios del Grid (número total de procesadores, y número de trabajos que cada sitio está procesando).
Nivel 2: Se cuenta con la información del nivel anterior, así como información sobre el tamaño $size_j$ de los trabajos que ingresan al sistema.
Nivel 3: Se cuenta con la información de los niveles anteriores así como el tiempo estimado $p'_j$ de procesamiento de cada trabajo.

Nivel 4: Se cuenta con la información de los niveles anteriores, así como con información sobre los calendarios locales. El GRMS tiene información sobre el calendario local de las máquinas, y la puede considerar cuando asigna trabajos a una máquina

En la Tabla 2 se listan 31 de las estrategias implementadas en el simulador de calendarización utilizado en este trabajo. La fórmula utilizada en cada una de ellas toma en cuenta la información correspondiente de los trabajos previamente asignados a ese sitio más el que está por asignarse. Entonces, el valor resultante es un estimado del valor de la métrica en cuestión al asignar el nuevo trabajo a ese sitio.

Tabla 2: Estrategias de asignación

| Estrategia | Nivel | Descripción |
|---|---|---|
| $Random$ | 0 | Asigna aleatoriamente cada uno de los trabajos a uno de los sitios admisibles |
| $MaxAR_v$ | 0 | Asigna el trabajo $j$ al sitio más rápido y con mayor fracción de recursos disponibles en el momento $r_j$: $$\max_{i=1\ldots m}\left\{\frac{avail_i * v_i}{m_i}\right\}$$ donde $avail_i$ denota la cantidad de procesadores disponibles del sitio $i$ en el momento $r_j$, $v_i$ es la velocidad del sitio $i$ y $m_i$ es el número de procesadores del sitio $i$. $MaxAR$ no considera los requerimientos de los trabajos en la cola (Hirales-Carbajal *et al*, 2010) |
| $MaxAR_{eff}$ | 0 | Asigna el trabajo $j$ al sitio con mayor eficiencia energética y mayor fracción de recursos disponibles en el momento $r_j$: $$\max_{i=1\ldots m}\left\{\frac{avail_i * eff_i}{m_i}\right\}$$ donde $avail_i$ denota la cantidad de procesadores disponibles del sitio $i$ en el momento $r_j$ y $eff_i$ representa la eficiencia energética del sitio $i$. |

Tabla 2 (continuación)

| | | |
|---|---|---|
| $MaxAR_{veff}$ | 0 | Asigna el trabajo $j$ al sitio más rápido, con mayor eficiencia energética y mayor fracción de recursos disponibles en el momento $r_j$: $$\max_{i=1\ldots m}\left\{\frac{avail_i * v_i * eff_i}{m_i}\right\}$$ donde $avail_i$ denota la cantidad de procesadores disponibles del sitio $i$ en el momento $r_j$. |
| $LBal_E$ | 0 | Asigna el trabajo $j$ al sitio con la menor desviación estándar de consume energético por procesador de todos los trabajos asignados (tomando en cuenta todos los sitios) cuando el trabajo es asignado a dicho sitio: $$\min_{q=1..m}\left\{\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(E_i^q-\bar{E}\right)^2}\right\}$$ donde $E_{i=1..m}^q=\frac{1}{m_i}\sum_{g_k=i}(e_k+e_j^q)$ y $e_j^q$ representa el consume energético del trabajo $j$ cuando es asignado al sitio $q$ |
| $MLp$ | 1 | Asigna el trabajo $j$ al sitio con la menor carga por procesador *(Min Load per processor)* en el tiempo $r_j$. La motivación para utilizar esta estrategia es para equilibrar la carga entre los procesadores de las máquinas. $$min_{i=1..m}\left(\frac{n_i}{m_i}\right)$$ donde $n_i$ representa al sitio $i$. |
| $MLp_v$ | 1 | Asigna el trabajo $j$ al sitio más rápido y con la menor carga por procesador en el tiempo $r_j$: $$\min_{i=1\ldots m}\left\{\frac{n_i}{v_i * m_i}\right\}$$ |

Tabla 2 (continuación)

| | | |
|---|---|---|
| $MLp_{eff}$ | 1 | Asigna el trabajo $j$ al sitio con mayor eficiencia energética y con la menor carga por procesador en el tiempo $r_j$: $$\min_{i=1\ldots m}\left\{\frac{n_i}{eff_i * m_i}\right\}$$ |
| $MLp_{veff}$ | 1 | Asigna el trabajo $j$ al sitio más rápido y con mayor eficiencia energética y con la menor carga por procesador en el tiempo $r_j$: $$\min_{i=1\ldots m}\left\{\frac{n_i}{eff_i * v_i * m_i}\right\}$$ |
| $MPL$ | 1 | Asigna el trabajo $j$ al sitio con el menor grado de paralelismo por procesador (Min Parallel Load) considerando todos los trabajos previamente asignados al sitio en el tiempo $r_j$. La intuición detrás de MPL es mantener todos los procesadores tan ocupados como sea posible. Una ventaja de esta estrategia es su simplicidad, no toma en cuenta el tiempo de ejecución del trabajo ni su estimado. $$min_{i=1..m}\left\{\sum_{g_k=i}\frac{size_k}{m_i}\right\}$$ |
| $MPL_v$ | 2 | Asigna el trabajo $j$ al sitio más rápido y con el menor grado de paralelismo por procesador considerando todos los trabajos previamente asignados al sitio en el tiempo $r_j$: $$\min_{i=1\ldots m}\left\{\sum_{g_k=i}\frac{size_k}{v_i * m_i}\right\}$$ donde $size_k$ representa el tamaño (número de procesadores requeridos) del trabajo $k$. |
| $MPL_{eff}$ | 2 | Asigna el trabajo $j$ al sitio con mayor eficiencia energética y con el menor grado de paralelismo por procesador considerando todos los trabajos previamente asignados al sitio en el tiempo $r_j$: $$\min_{i=1\ldots m}\left\{\sum_{g_k=i}\frac{size_k}{eff_i * m_i}\right\}$$ |

Tabla 2 (continuación)

| | | |
|---|---|---|
| $MPL_{veff}$ | 2 | Asigna el trabajo $j$ al sitio más rápido, con mayor eficiencia energética y con el menor grado de paralelismo por procesador considerando todos los trabajos previamente asignados al sitio en el tiempo $r_j$: $$\min_{i=1\ldots m}\left\{\sum_{g_k=i}\frac{size_k}{eff_i * v_i * m_i}\right\}$$ |
| $LBal_S$ | 2 | Asigna el trabajo $j$ al sitio con la menor desviación estándar del grado de paralelismo por procesador *(Load Balance Size)* de todos los trabajos asignados (tomando en cuenta todos los sitios) cuando el trabajo es asignado a dicho sitio. $$\min_{q=1..m}\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(PL_i^q-\overline{PL}\right)^2}$$ Donde $PL_{i=1..m}^q=\frac{1}{m_i}\sum_{g_k=i}(size_k+size_j^q)$ y $size_j^q$ es el tamaño del trabajo $j$ agregado al sitio $q$ (Kurowski *et al*, 2008). |
| $LBal\_T$ | 2 | Asigna el trabajo $j$ al sitio con la menor desviación estándar del tiempo de ejecución por procesador *(Load Balance Time)* de todos los trabajos asignados (tomando en cuenta todos los sitios) cuando el trabajo es asignado a dicho sitio. $$\min_{q=1..m}\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(T_i^q-\overline{T}\right)^2}$$ Donde $T_{i=1..m}^q=\frac{1}{m_i}\sum_{g_k=i}(p_k+p_j^q)$, $p_k$ es el tiempo de ejecución del trabajo $k$ y $p_j^q$ es el tiempo de ejecución del trabajo $j$ agregado al sitio $q$. |

Tabla 2 (continuación)

| | | |
|---|---|---|
| $LBal_W$ | 2 | Asigna el trabajo $j$ al sitio con la menor desviación estándar de los recursos consumidos por procesador *(Load Balance Work)* de todos los trabajos asignados (tomando en cuenta todos los sitios) cuando el trabajo es asignado a dicho sitio. $$\min_{q=1..m} \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(W_i^q - \overline{W}\right)^2}$$ Donde $W_{i=1..m}^q = \frac{1}{m_i}\sum_{g_k=i}(w_k + w_j^q)$, $w_k$ es el producto de $p_k$ y $size_k$ y $w_j^q$ son los recursos consumidos del trabajo $j$ agregado al sitio $q$. |
| $MLB$ | 2 | Asigna el trabajo $j$ al sitio con los menores recursos consumidos por procesador *(Min Lower Bound)* en el tiempo $r_j$: $$min_{i=1..m}\left\{\sum_{g_k=i}\frac{size_k \cdot p'_k}{m_i}\right\}$$ donde $p_k^{'}$ representa el tiempo de ejecución estimado del trabajo $k$. |
| $MLB_v$ | 3 | Asigna el trabajo $j$ al sitio con más rápido y con los menores recursos consumidos por procesador en el momento $r_j$: $$\min_{i=1...m}\left\{\sum_{g_k=i}\frac{size_k * p'_k}{v_i * m_i}\right\}$$ |
| $MLB_{eff}$ | 3 | Asigna el trabajo $j$ al sitio con mayor eficiencia energética y con los menores recursos consumidos por procesador en el momento $r_j$: $$\min_{i=1...m}\left\{\sum_{g_k=i}\frac{size_k * p'_k}{eff_i * m_i}\right\}$$ |
| $MLB_{veff}$ | 3 | Asigna el trabajo $j$ al sitio más rápido, con mayor eficiencia energética y con los menores recursos consumidos por procesador en el momento $r_j$: $$\min_{i=1...m}\left\{\sum_{g_k=i}\frac{size_k * p_k^{'}}{eff_i * v_i * m_i}\right\}$$ |

Tabla 2 (continuación)

| | | |
|---|---|---|
| $MCT$ | 3 | Asigna el trabajo $j$ al sitio con el menor tiempo de finalización del Grid *(Min Completion Time)*<br>$$min\{C_{max}^{i}\}$$<br>donde $C_{max}^{i} = max_{g_k=i}(C_k^i)$, y $C_k^i$ es el tiempo de finalización del trabajo $J_k$ en el sitio $i$. Esto ocasiona que algunas tareas sean asignadas a máquinas que no tienen el mínimo tiempo de finalización para ella. Esta estrategia intenta minimizar el tiempo de finalización total. |
| $MCT_{eff}$ | 4 | Asigna el trabajo $j$ al sitio con la mayor eficiencia energética y el menor tiempo de finalización del Grid<br>$$min\left\{\frac{C_{max}^{i}}{eff_i}\right\}$$<br>donde $C_{max}^{i} = \max_{g_k=i}\{C_k^i\}$ y $C_k^i$ es el tiempo de terminación del trabajo $k$ en el sitio $i$. |
| $MWT$ | 4 | Asigna el trabajo $j$ al sitio con el menor promedio de tiempo de espera *(Min Waiting Time)* de los trabajos.<br>$$min_{i=1..m}\left\{\sum_{g_k=i}\frac{t_w^k}{n_i}\right\}$$<br>donde $t_w^k$ representa el tiempo de espera del trabajo $k$. |
| $MWT_{eff}$ | 4 | Asigna el trabajo $j$ al sitio con mayor eficiencia energética y el menor promedio de tiempo de espera de los trabajos:<br>$$\min_{i=1...m}\left\{\frac{1}{n_i * eff_i}\sum_{g_k=i}(t_w^k)\right\}$$ |
| $MWWT_S$<br>$MWWT_T$<br>$MWWT_W$ | 4 | Asignan el trabajo $j$ al sitio con el menor promedio de tiempo de espera ponderado (con peso). *(Min Weighted Waiting Time_Size) (Min Weighted Waiting Time_Time) (Min Weighted Waiting Time_Work).*<br>$$min_{i=1..m}\left\{\sum_{g_k=i}\frac{t_w^k \cdot weight_k}{n_i}\right\}$$<br>donde $weight_k = \{size_k, p'_k, w_k\}$ |

Tabla 2 (continuación)

| | | |
|---|---|---|
| $MST$ | 4 | Asigna el trabajo $j$ al sitio que pueda iniciar más temprano su ejecución *(Min Start Time)*:<br>$$min_{\ i=1..m}\{s_j^i\}$$<br>donde $s_j^i$ representa el tiempo de inicio del trabajo $j$ en el sitio $i$.<br>Para plataformas homogéneas, esta estrategia asigna cada tarea a la máquina con el menor tiempo de finalización esperado para tal tarea (*Earliest-Finish-Time*) |
| $MST_v$ | 4 | Asigna el trabajo $j$ al sitio más rápido y que pueda iniciar más temprano su ejecución<br>$$\min_{i=1...m}\left\{\frac{s_j^i}{v_i}\right\}$$<br>donde $s_j^i$ es el tiempo de inicio del trabajo $j$ en el sitio $i$. |
| $MST_{eff}$ | 4 | Asigna el trabajo $j$ al sitio con mayor eficiencia energética y que pueda iniciar más temprano su ejecución:<br>$$\min_{i=1...m}\left\{\frac{s_j^i}{eff_i}\right\}$$ |
| $MST_{veff}$ | 4 | Asigna el trabajo $j$ al sitio más rápido, con mayor eficiencia energética y que pueda iniciar más temprano su ejecución:<br>$$\min_{i=1...m}\left\{\frac{s_j^i}{v_i * eff_i}\right\}$$ |

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

### 4.2 Estrategias de calendarización local

Ya que el trabajo ha sido asignado al sitio, el LRMS de esta máquina generará un calendario en base a la aplicación de algún algoritmo de calendarización local. Existen diferentes algoritmos que pueden ser aplicados en esta fase; muchos sistemas reales aplican el algoritmo FCFS (First Come, First Served), que

calendariza los trabajos basándose en el orden de sus momentos de llegada al sistema. Si no hay suficientes recursos disponibles en ese momento, el trabajo espera hasta que se liberen recursos y pueda iniciarse; mientras, el resto de trabajos de la cola debe esperar. Este algoritmo tiene muy malos resultados para el peor caso; esto se debe a que se pueden retardar trabajos cortos que se quedan bloqueados detrás de los trabajos grandes. Sin embargo, este algoritmo cuenta con la ventaja de ser muy fácil de implementar.

Una variación a este algoritmo es el algoritmo de *Backfilling* propuesto por Lifka (Lifka, 1995; Skovira *et al.*, 1996). El algoritmo de *Backfilling* es una muy buena opción para incrementar el desempeño para el usuario y para el sistema. Este algoritmo permite que algunos trabajos puedan iniciar su ejecución antes del tiempo que les correspondería en base a su posición en la cola. Esto se logra realizando una búsqueda de trabajos más pequeños en la cola de trabajos disponibles para su ejecución cuando el trabajo al frente de la cola no puede iniciar su ejecución por carencia de recursos en ese momento.

El algoritmo *Backfilling* tiene varias implementaciones conocidas, pero existen dos que son las más populares: Conservative *Backfilling* y EASY *Backfilling* (D. Feitelson *et al.*, 2005b; Feitelson, Weil, 1998). La primera implementación busca garantizar que ninguno de los trabajos de la cola de espera retrase el inicio de su ejecución. Esto se logra haciendo una reservación de tiempo de inicio de ejecución para cada trabajo cuando este llega, de tal manera que al buscar trabajos en la cola, se establecen estas reservaciones como limitantes para poder elegir los trabajos que pueden ser considerados para rellenar. En el caso de *EASY Backfilling*, la única limitante para iniciar otros trabajos es que no retrasen el inicio de la ejecución del trabajo que se encuentra al frente de la cola. Es importante notar que un trabajo pequeño (con menor cantidad de procesadores requeridos) no es necesariamente corto (con menor tiempo de ejecución), por lo que un trabajo pequeño pero largo puede ser ejecutado como relleno en el caso de que existan recursos disponibles más allá de la reservación (*EASY*) o reservaciones (*Conservative*) establecidas para los demás trabajos.

Como puede notarse, la base de los algoritmos *Backfilling* es el tiempo de ejecución de los trabajos, siempre que para realizar una reservación de tiempo de inicio de ejecución es necesario saber cuándo finalizarán los trabajos que se encuentran actualmente en ejecución.

De la misma forma, para seleccionar los trabajos para relleno, es necesario conocer el tiempo de ejecución de cada uno de ellos. Dado que es imposible saber el tiempo de ejecución de cada trabajo antes de que termine, se necesita contar con una estimación del tiempo de ejecución provista por el usuario.

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

# 5. Configuración experimental

Buscando realizar una evaluación lo más cercana posible a casos reales, se consideraron los siguientes parámetros para realizar la experimentación.

## 5.1 Trabajos

Un aspecto muy importante para obtener una simulación acertada son los datos de entrada. Si estos no son representativos de valores reales, los resultados no serían certeros. En este caso se hizo uso de cargas de trabajos paralelos construidas a partir de varias bitácoras extraídas de sistemas reales. Este capítulo contiene la descripción de la metodología empleada para la generación de la carga, así como un análisis de ella.

Dado que se busca realizar un análisis probabilístico, se dividieron los trabajos en 30 bloques de una semana; de esta forma, para cada estrategia se ejecutaron 30 experimentos –uno por cada carga de trabajo– (60 para ambos casos). Esto da una perspectiva más acertada del comportamiento de cada estrategia al trabajar con ambientes distintos.

### 5.1.1 Bitácoras

Como se mencionó previamente, uno de los aspectos principales de la simulación de sistemas es la selección de los datos de entrada para la ejecución de los experimentos, dado que de ellos depende la precisión de los resultados obtenidos (Frachtenberg, Feitelson, 2005). Para el caso del Grid, y aún con los esfuerzos realizados por el equipo de *The Grid Workloads Archive* (Anoep *et al*, 2007) para ofrecer un estándar de cargas de Grid, aún no existe una carga con características generales para esta arquitectura. Algunas de las cargas que se encuentran en el sitio son cargas secuenciales; otras consideran una monitorización de sólo unos días o solamente de pocos sitios.

Existen trabajos que utilizan una carga generada en base a funciones probabilísticas, como en Maheswaran *et al* (1999); sin embargo, se necesita hacer un estudio del comportamiento de las cargas de sistemas reales para proveer un

sistema generador de cargas eficientes en base a probabilidad, como se ha demostrado en Tsafrir *et al* (2006), en que los autores generan un modelo que utilizan para reproducir tiempos estimados de bitácoras reales.

Reproducir características de una carga de trabajos para un sistema paralelo, como tiempos de llegada, tiempos de ejecución, grado de paralelismo, etc., implica un trabajo muy complejo que termina siendo una carga dependiente de un sitio específico. Si se extiende esto al ámbito de Grids, la complejidad aumenta.

Una alternativa aceptable es el uso de bitácoras reales de sistemas en producción, como se muestra en Grimme *et al* (2007, 2008) y Hamscher *et al* (2000), las cuales contienen una secuencia de trabajos reales que pueden ser utilizadas para evaluar el desempeño de algoritmos basados en simulación. Para generar una carga para el Grid basada en bitácoras reales, se pueden mezclar las cargas intentando reproducir la participación de los usuarios de cada sitio en un ambiente de Grid.

Para la fase experimental de este trabajo se utilizó una mezcla de trabajos extraídos de bitácoras reales de sistemas en producción, las cuales se tomaron de *Parallel Workloads Archive* (Feitelson, D. G., 2005b); posteriormente se aplicó el siguiente proceso para generar una carga más representativa:

1. Selección de bitácoras que tengan los siguientes datos válidos para todos los trabajos: tiempo de llegada, número de procesadores solicitado, tiempo de ejecución, tiempo estimado de ejecución y un identificador de usuario. Estos valores son los necesarios para el funcionamiento correcto de las estrategias y algoritmos de calendarización descritos previamente.
2. *Normalización por zona horaria*. Al unir las bitácoras, es necesario conocer la zona horaria en que se generó cada una. Esto se hace para mantener una consistencia con las características del envío de trabajo: durante la madrugada se reciben menos trabajo que a mediodía, por ejemplo. Entonces, si el primer trabajo de una zona horaria GMT-0 llegó a las 12:00 y el primer trabajo de una zona horaria GMT-6 llegó a las 12:00, no pueden

considerarse como que llegaron al mismo tiempo al sistema. Al crear la bitácora, se utilizaron la zona horaria menor y se ajustaron todas las demás con respecto a ese valor. Para ello se modificaron los datos del encabezado de la bitácora correspondientes a la zona horaria, a la hora y al momento de inicio. Con esto se puede deducir que el *Broker* de los experimentos se considera ubicado en la zona horaria menor.

3. *Exclusión de los primeros trabajos* de cada bitácora. Esto se hace para descartar las condiciones iniciales del sistema para medir sólo el desempeño cuando éste esté estable (Frachtenberg, Feitelson, 2005).
4. *Normalización por día de la semana*. Así como existe un comportamiento del envío de trabajos durante las distintas horas del día, también existe un comportamiento particular para los días de la semana; por ejemplo, no se envía la misma cantidad de trabajos un miércoles que un domingo. Para respetar esta característica, se ajustó el inicio de trabajos válidos a las 0 horas del primer lunes y, como final, el último domingo registrado en cada una.
5. *Normalización de números de identificación de usuario*. Dado que cada bitácora es independiente, los registros de usuarios también lo son; por esto se podría encontrar al usuario 21 en más de una de ellas, lo cual impediría identificar a cada usuario participante del Grid. Para esto, se modificaron los números de identificación de usuario por una secuencia global. De esta forma, los usuarios de la primera bitácora tienen numeración de $1$ a $m$, los de la segunda de $m + 1$ a $n$, y así sucesivamente.
6. *Filtrado de trabajos no válidos*. Aun cuando las bitácoras fueron previamente filtradas y adaptadas al formato estándar SWF (*Standard Workload Format*) propuesto inicialmente por Chapin *et al* (1999) y cuya versión actualizada se encuentra en Feitelson D.G. (2005b), existen trabajos irrelevantes para esta investigación, por lo cual se eliminan los trabajos que tengan alguna de las siguientes características:
    a. número de trabajo menor o igual a cero (job_number <= 0)

b. tiempo de llegada menor que cero (release_time < 0)
c. tiempo de ejecución menor o igual a cero (runtime <= 0)
d. número de procesadores asignados menor o igual a cero (allocated_processors <= 0)
e. tiempo estimado menor o igual a cero (requested_time <= 0)
f. Identificador de usuario menor o igual a cero (user_id <= 0)
g. Estatus fallido (status = 0)
h. Estatus fallido en la última ejecución parcial (status = 4)
i. Estatus cancelado por el usuario (status = 5)

7. *Unión*. Finalmente, cuando todas las bitácoras han pasado por las operaciones previamente descritas, se procede a tomar los trabajos en orden no decreciente de su tiempo de llegada hasta completar el número de días requerido.

Debe aclararse que la unión de varias bitácoras independientes para simular la carga de un Grid computacional no representa con precisión la carga de un Grid real con esas mismas máquinas y usuarios. Por ejemplo, si un sitio forma parte de un Grid con sitios más grandes en él, los usuarios pueden enviar trabajos más grandes, los cuales no estarían representados en la bitácora original. No obstante, y dada la falta de cargas de trabajo de uso público para Grids, esta es considerada como una buena aproximación para evaluar estrategias de calendarización basadas en bitácoras reales.

### 5.1.2 Configuración del Grid

Para evaluar las estrategias se consideraron dos escenarios llamados Grid1 y Grid2. En cada escenario se incluyen las características de 7 y 9 sitios, respectivamente. La configuración de cada sitio se muestra en las tablas Tabla 3: Configuración de Grid1 y Tabla 4: Configuración de Grid2.

Tabla 3: Configuración de Grid1

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

Tabla 4: Configuración de Grid2

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

La Tabla 5 muestra las características básicas de las mezclas finales para Grid1 y para Grid2. Podemos notar que el número de usuarios finales en Grid2 es menos de la mitad que el número de usuarios en Grid1, sin embargo, el número de trabajos casi se triplica en Grid2 respecto a Grid1; esto puede indicar una alta actividad de los usuarios de Grid2, lo cual puede reflejarse en una mayor utilización de los recursos.

Tabla 5: Resumen de características de las mezclas para Grid 1 y Grid 2

| Grid | #Sitios | #Procesadores | #Trabajos | #Usuarios | Zona horaria | Periodo (días) |
|---|---|---|---|---|---|---|
| 1 | 7 | 4442 | 152396 | 2652 | -8 | 210 |
| 2 | 9 | 2194 | 429938 | 1630 | -7 | 210 |

Las siguientes figuras muestran detalles de las cargas de trabajo para Grid1 y Grid2. La Figura 5 muestra el número de trabajos por semana. La Figura 6 muestra el promedio de recursos consumidos por día del mes. Podemos ver que la demanda de recursos está distribuida en los días del mes en ambos Grids, lo cual difiere si consideramos las bitácoras originales independientemente.

También podemos observar que la demanda de recursos es mayor en \ Grid1 a pesar de contar con un número menor de trabajos, lo cual indica que son trabajos muy largos o con un alto grado de paralelismo. Esto nos da dos escenarios diferentes para simulación.

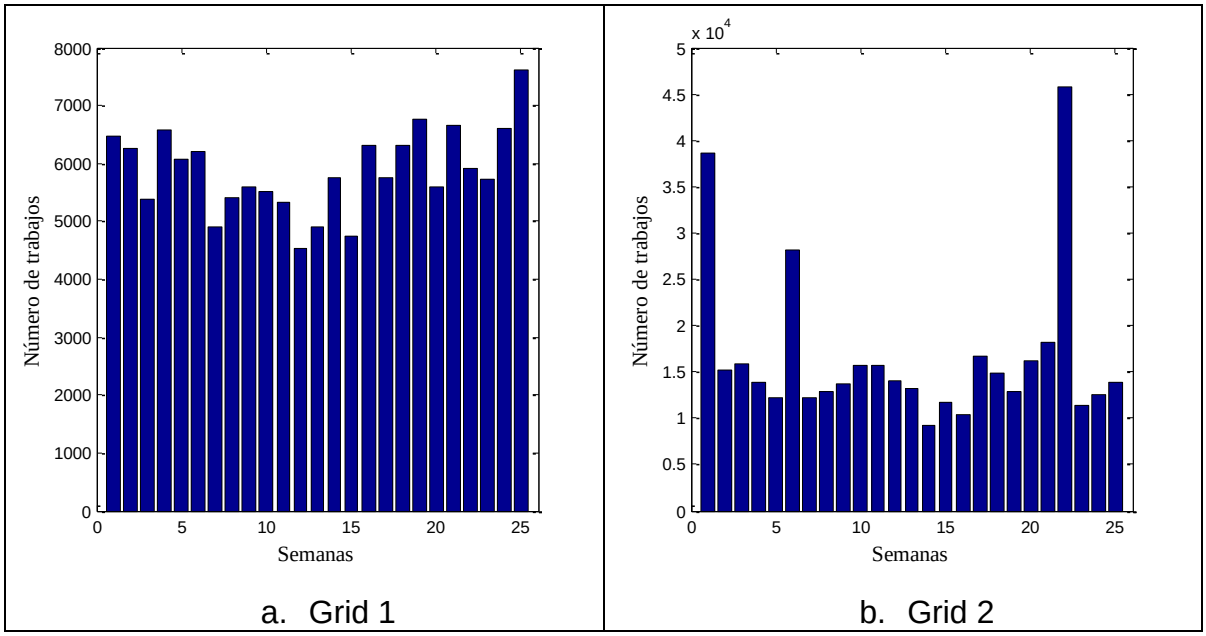


**Figura 5. Número de trabajos por cada semana**

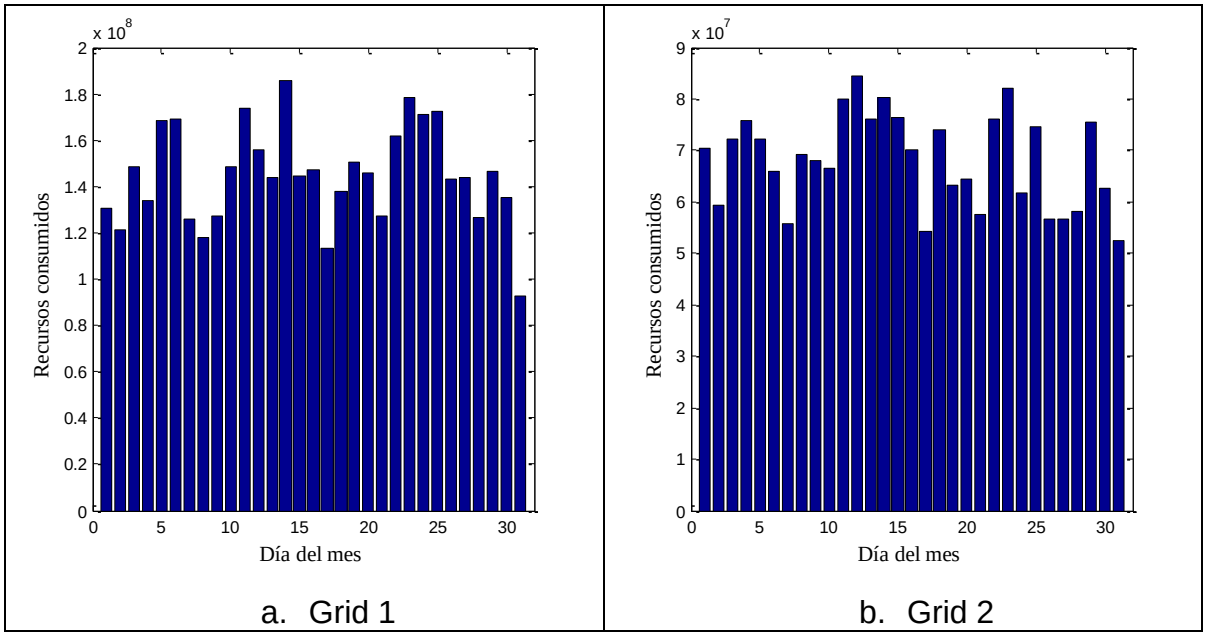


**Figura 6. Promedio de recursos consumidos por día del mes**

## 5.2 Configuración energética

Las

Tabla 6 yTabla 8 muestran la configuración del consumo y eficiencia energética de los sitios contenidos en los dos casos de prueba considerados en este trabajo, así como su velocidad. Las

Tabla 7 y

Tabla 9 muestran los mismos datos, pero en una proporción relativa (el valor mínimo de cada columna es 1, y los demás se calculan en base a éste).

Al realizar los experimentos, los valores de eficiencia energética y velocidad se normalizaron; el sitio más lento tiene una velocidad de 1, y los demás una velocidad relativa a éste. De forma similar, el sitio menos eficiente tiene una eficiencia de 1, mientras que el resto una eficiencia relativa a éste.

Tabla 6: Detalles de velocidad y energía de Grid1

| Sitio | Procs | Consumo energético | | | Eficiencia energética | Velocidad |
|---|---|---|---|---|---|---|
| | | $P^{Idle}$ | $P^{Working}$ | $Board^{Idle}$ | | |
| KTH—Swedish Royal Institute of Technology | 100 | 17.8W | 26W | 110W | 1.75 MFLOPS/W | 18.6 GFLOPS/s |
| SDSC-SP2—San Diego Supercenter SP2 | 128 | 17.8W | 26W | 130W | 1.43 MFLOPS/W | 57.24 GFLOPS/s |
| HPC2N—High Performance Computing Center North, Sweden | 240 | 58.9W | 66W | 140W | 0.89 MFLOPS/W | 481 GFLOPS/s |
| CTC—Cornell Theory Center | 430 | 17.8W | 26W | 135W | 1.64 MFLOPS/W | 88.4 GFLOPS/s |
| LANL—Los Alamos National Lab | 1024 | 24.7 W | 31 W | 120W | 1.45 MFLOPS/W | 65.4 GFLOPS/s |
| SDSC-BLUE—San Diego Supercenter Blue Gene | 1152 | 76 W | 78 W | 220W | 0.79 MFLOPS/W | 2662.31 GFLOPS/s |
| SDSC-DS—San Diego Supercenter Data Star | 1368 | 115 W | 125 W | 240W | 0.76 MFLOPS/W | 5223 GFLOPS/s |

Los valores de eficiencia energética y velocidad se obtuvieron de los sitios web de cada máquina, mientras que para obtener los valores del consumo energético de sus componentes se revisaron las hojas de especificación del fabricante de cada uno de ellos.

Tabla 7: Detalles relativos de velocidad y energía de Grid1

| Sitio | Procs | Consumo energético | | | Eficiencia energética | Velocidad |
|---|---|---|---|---|---|---|
| | | $P^{Idle}$ | $P^{Working}$ | $Board^{Idle}$ | | |
| KTH—Swedish Royal Institute of Technology | 1 | 1 | 1 | 1 | 2.30263158 | 1 |
| SDSC-SP2—San Diego Supercenter SP2 | 1.28 | 1 | 1 | 1.18181818 | 1.88157895 | 3.07741935 |
| HPC2N—High Performance Computing Center | 2.4 | 3.30898876 | 2.53846154 | 1.27272727 | 1.17105263 | 25.8602151 |
| CTC—Cornell Theory Center | 4.3 | 1 | 1 | 1.22727273 | 2.15789474 | 4.75268817 |
| LANL—Los Alamos National Lab | 10.24 | 1.38764045 | 1.19230769 | 1.09090909 | 1.90789474 | 3.51612903 |
| SDSC-BLUE—San Diego Supercenter Blue Gene | 11.52 | 4.26966292 | 3 | 2 | 1.03947368 | 143.134946 |
| SDSC-DS—San Diego Supercenter Data Star | 13.68 | 6.46067416 | 4.80769231 | 2.18181818 | 1 | 280.806452 |

Los valores de consumo energético son inexactos, ya que no se considera el consumo requerido para disipación de calor, entre otras cosas. Aunque es claro que estos valores no son irrelevantes (Orgerie *et al*, 2010), calcularlos sería muy complejo, así que se optó por utilizar sólo el consumo requerido para trabajar de cada componente.

Tabla 8: Detalles de velocidad y energía de Grid2

| Sitio | Procs | Consumo energético | | | Eficiencia energética | Velocidad |
|---|---|---|---|---|---|---|
| | | $P^{Idle}$ | $P^{Working}$ | $Board^{Idle}$ | | |
| DAS2—University of Amsterdam | 64 | 17.8 W | 35.35 W | 130W | 1.36 MFLOPS/W | 126 GFLOPS/s |
| DAS2—Delft University of Technology | 64 | 17.8 W | 35.35 W | 130W | 1.36 MFLOPS/W | 126 GFLOPS/s |
| DAS2—Utrecht University | 64 | 17.8 W | 35.35 W | 130W | 1.36 MFLOPS/W | 126 GFLOPS/s |
| DAS2—Leiden University | 64 | 17.8 W | 35.35 W | 130W | 1.36 MFLOPS/W | 126 GFLOPS/s |
| KTH—Swedish Royal Institute of Technology | 100 | 17.8 W | 26 W | 110W | 1.75 MFLOPS/W | 18.6 GFLOPS/s |
| DAS2—Vrije University Amsterdam | 144 | 17.8 W | 35.35 W | 130W | 1.32 MFLOPS/W | 230 GFLOPS/s |
| HPC2N—High Performance Computing Center North, Sweden | 240 | 58.9 W | 66 W | 140W | 0.89 MFLOPS/W | 481 GFLOPS/s |
| CTC—Cornell Theory Center | 430 | 17.8 W | 26 W | 135W | 1.64 MFLOPS/W | 88.4 GFLOPS/s |
| LANL—Los Alamos National Lab | 1024 | 24.7 W | 31 W | 120W | 1.45 MFLOPS/W | 65.4 GFLOPS/s |

Para los experimentos, se consideraron los siguientes valores para los tiempos de gracia de apagado y encendido: $T_i^{startSite} = 20s$, $T_i^{shutdownSite} = 10s$ y $T_{cr}^{shutdownCore} = 5s$.

Tabla 9: Detalles relativos de velocidad y energía de Grid2

| Sitio | Procs | Consumo energético | | | Eficiencia energética | Velocidad |
|---|---|---|---|---|---|---|
| | | $P^{Idle}$ | $P^{Working}$ | $Board^{Idle}$ | | |
| DAS2—University of Amsterdam | 1 | 1 | 1.35961538 | 1.18181818 | 1.52808989 | 6.77419355 |
| DAS2—Delft University of Technology | 1 | 1 | 1.35961538 | 1.18181818 | 1.52808989 | 6.77419355 |
| DAS2—Utrecht University | 1 | 1 | 1.35961538 | 1.18181818 | 1.52808989 | 6.77419355 |
| DAS2—Leiden University | 1 | 1 | 1.35961538 | 1.18181818 | 1.52808989 | 6.77419355 |
| KTH—Swedish Royal Institute of Technology | 1.5625 | 1 | 1 | 1 | 1.96629213 | 1 |
| DAS2—Vrije University Amsterdam | 2.25 | 1 | 1.35961538 | 1.18181818 | 1.48314607 | 12.3655914 |
| HPC2N—High Performance Computing Center North, Sweden | 3.75 | 3.30898876 | 2.53846154 | 1.27272727 | 1 | 25.8602151 |
| CTC—Cornell Theory Center | 6.71875 | 1 | 1 | 1.22727273 | 1.84269663 | 4.75268817 |
| LANL—Los Alamos National Lab | 16 | 1.38764045 | 1.19230769 | 1.09090909 | 1.62921348 | 3.51612903 |

## 6. Análisis experimental

En este capítulo se presentan los resultados de simulación de las estrategias de asignación de trabajos mencionadas previamente, utilizando las métricas descritas en la sección 2.2. Los detalles de la configuración de Grids y las cargas de trabajos utilizadas en nuestros experimentos son descritos en la sección 5.2. Para llevar a cabo los experimentos se utilizó un simulador llamado tGSF (Teikoku Grid Scheduling Framework).

### 6.1 Objetivos de los experimentos

En los experimentos realizados se buscó evaluar tres factores.

- El primero es comparar las estrategias mencionadas en la Tabla 2 considerando las cuatro métricas mencionadas en la Tabla 1.
- El segundo consiste en comparar las estrategias mencionadas en la Tabla 2 en base a su eficiencia energética. Esto se calcula con la fórmula:

$$EnEff = \frac{work}{E_{Grid}} \tag{10}$$

donde $work$ representa el trabajo realizado por el Grid:

$$\text{work} = \sum_{j=1}^{n} \left( size_j * p_j \right) \tag{11}$$

y $E_{Grid}$ el consumo energético del Grid.

- Finalmente se buscó comparar el desempeño del modelo de energía propuesto contra un modelo en el que los componentes de un sitio estuviesen siempre encendidos. Para esto se consideraron de nuevo las métricas propuestas en la Tabla 1, comparando solamente la mejor estrategia del primer conjunto de experimentos contra el desempeño que tendría esa misma estrategia sin considerar el encendido y apagado dinámico de los recursos.

### 6.2 Método de evaluación

Ya que el problema abordado es un problema multiobjetivo, se consideraron diversos criterios para su evaluación. Un buen algoritmo debe calendarizar

trabajos logrando un alto rendimiento del Grid mientras satisface diversas demandas. Como se mencionó previamente, minimizar el consumo energético del Grid está en conflicto con las métricas tradicionales de desempeño de un Grid. El objetivo es encontrar una estrategia que tenga un buen desempeño bajo todos los casos de prueba, con la expectativa de que tendrá un buen desempeño bajo otras condiciones. La administración de recursos del Grid puede usar soporte para decisiones multicriterio.

Para evaluar el desempeño de las estrategias en cada métrica, se evalúa la degradación (error relativo) en rendimiento de cada estrategia en cada métrica. Esto se hace tomando como base la mejor estrategia de cada métrica utilizando la fórmula

$$\max\left(\left(\frac{metric\ value}{metric\ best\ value} - 1\right) \cdot 100, 0\right) \qquad (12)$$

que es una adaptación de la utilizada en Quezada-Pina *et al* (2012).

En la fórmula 10 $metric\ value$ representa el valor de cada estrategia para esa métrica y $metric\ best\ value$ representa el valor de la mejor estrategia para esa métrica.

El método utilizado para evaluar el desempeño considerando todas las métricas está basado en la optimalidad de Pareto. Para esto, se obtuvo un frente no dominado para cada estrategia, considerando las cuatro métricas. Posteriormente se compararon los frentes utilizando la estrategia *set coverage* (Coello Coello *et al*, 2007), la cual consiste en comparaciones binarias para determinar qué proporción de un frente es dominada débilmente por otro frente**.**

Si se consideran A y B como dos frentes distintos, *set coverage* ($SC$) se define como un emparejamiento del par ordenado $(A, B)$ al intervalo $[0,1]$:

$$SC(A, B) \triangleq |\{b \in B; \exists a \in A: a \preceq b\}|/|B| \qquad (13)$$

Si al menos un punto en A domina o es igual a todos los puntos en B, entonces por definición $SC(A, B) = 1$. $SC(A, B) = 0$ implica lo opuesto. En general, deben considerarse tanto $SC(A, B)$ como $SC(B, A)$ dado que las intersecciones no están vacías. Si $SC(A, B) > SC(B, A)$, decimos que $A$ es mejor que $B$.

Al comparar todos los frentes con este método se puede crear una clasificación de las estrategias para ambos escenarios, mostrándonos así una evaluación precisa del desempeño.

### 6.3 Resultados

A continuación se muestran los resultados de las estrategias consideradas con cuatro métricas para los escenarios Grid1 y Grid2. Las condiciones en Grid 1 son caracterizadas por 7 grandes sitios con 634 procesadores por sitio en promedio, y 21770 trabajos por sitio en promedio. Las condiciones en Grid 2 son caracterizadas por 9 sitios de tamaño más pequeño con 243 procesadores por sitio en promedio, y 47770 trabajos por sitio en promedio.

La

Tabla 10: Clasificación multiobjetivo de las estrategias. muestra una clasificación de las estrategias basada en $SC$. En ella, la columna etiquetada como $SC$ muestra qué porción del frente de una estrategia es dominada por los frentes de las demás. Por ejemplo: el valor de la celda donde se intersectan la fila $MaxAR_v$ y la columna $SC$ se obtuvo utilizando la fórmula 11, donde $B$ es el frente no dominado de $MaxAR_v$ y $A$ es el frente no dominado de las demás estrategias. La siguiente columna muestra una clasificación de estrategias para cada escenario. Las últimas dos muestran el promedio de $SC$ considerando ambos escenarios y una clasificación de las estrategias basada en este promedio.

Como se puede ver, la estrategia con mejor resultado es $MaxAR_v$, la cual está clasificada en la Tabla 2 como de nivel 0; es decir, que no considera información sobre los calendarios locales o los trabajos para realizar la asignación. Podría creerse que las estrategias de nivel superior arrojarían un mejor resultado, pero como la información que se utiliza no es precisa –al considerar el tiempo de ejecución de los trabajos, sólo se cuentan con estimaciones–, esto hace que el tener más información empeore la calidad de los calendarios generados.

Tabla 10: Clasificación multiobjetivo de las estrategias.

| Estrategia | Grid | $SC$ | Rango | Promedio | Rango promedio |
|---|---|---|---|---|---|
| $LBal_E$ | 1 | 0.37096774 | 14 | 0.21082949 | 12 |
| | 2 | 0.05069124 | 3 | | |
| $LBal_S$ | 1 | 0.30875576 | 10 | 0.18291386 | 9 |
| | 2 | 0.05707196 | 5 | | |
| $LBal_T$ | 1 | 0.50179211 | 17 | 0.41353047 | 20 |
| | 2 | 0.32526882 | 21 | | |
| $LBal_W$ | 1 | 0.45362903 | 16 | 0.37427995 | 7 |
| | 2 | 0.29493088 | 18 | | |
| $MaxAR_{eff}$ | 1 | 0.92059553 | 30 | 0.80131159 | 31 |
| | 2 | 0.68202765 | 30 | | |
| $MaxAR_v$ | 1 | 0.06451613 | 1 | 0.05913978 | 1 |
| | 2 | 0.05376344 | 4 | | |
| $MaxAR_{veff}$ | 1 | 0.06451613 | 2 | 0.19758065 | 10 |
| | 2 | 0.33064516 | 23 | | |
| $MCT$ | 1 | 0.86021505 | 29 | 0.47556207 | 25 |
| | 2 | 0.09090909 | 10 | | |
| $MCT_{eff}$ | 1 | 0.22939068 | 7 | 0.14874552 | 5 |
| | 2 | 0.06810036 | 7 | | |
| $MLB$ | 1 | 0.66308244 | 25 | 0.690681 | 30 |
| | 2 | 0.71827957 | 31 | | |
| $MLB_{eff}$ | 1 | 0.71612903 | 27 | 0.61075269 | 27 |
| | 2 | 0.50537634 | 29 | | |
| $MLB\_v$ | 1 | 0.34408602 | 12 | 0.40188172 | 18 |
| | 2 | 0.45967742 | 28 | | |
| $MLB_{veff}$ | 1 | 0.38709677 | 15 | 0.37211982 | 15 |
| | 2 | 0.35714286 | 24 | | |
| $MLp$ | 1 | 0.24193548 | 8 | 0.15020161 | 6 |
| | 2 | 0.05846774 | 6 | | |
| $MLp_{eff}$ | 1 | 0.359447 | 13 | 0.23718318 | 13 |
| | 2 | 0.11491935 | 13 | | |
| $MLp_v$ | 1 | 0.16129032 | 5 | 0.09811828 | 3 |
| | 2 | 0.03494624 | 2 | | |
| $MLp_{veff}$ | 1 | 0.1733871 | 6 | 0.13407258 | 4 |
| | 2 | 0.09475806 | 11 | | |

Tabla 10 (continuación)

| | | | | | |
|---|---|---|---|---|---|
| $MPL$ | 1 | 0.54516129 | 20 | 0.41612903 | 21 |
| | 2 | 0.28709677 | 16 | | |
| $MPL_{eff}$ | 1 | 0.58312655 | 23 | 0.37220844 | 16 |
| | 2 | 0.16129032 | 14 | | |
| $MPL_{v}$ | 1 | 0.25806452 | 9 | 0.17741935 | 8 |
| | 2 | 0.09677419 | 12 | | |
| $MPL_{veff}$ | 1 | 0.31854839 | 11 | 0.20326246 | 11 |
| | 2 | 0.08797654 | 9 | | |
| $MST$ | 1 | 0.60703812 | 24 | 0.31964809 | 14 |
| | 2 | 0.03225806 | 1 | | |
| $MST_{eff}$ | 1 | 1 | 31 | 0.64956012 | 29 |
| | 2 | 0.29912023 | 19 | | |
| $MST_{v}$ | 1 | 0.07096774 | 3 | 0.07695853 | 2 |
| | 2 | 0.08294931 | 8 | | |
| $MST_{veff}$ | 1 | 0.07096774 | 4 | 0.16451613 | 7 |
| | 2 | 0.25806452 | 15 | | |
| $MWT$ | 1 | 0.51612903 | 18 | 0.40524194 | 19 |
| | 2 | 0.29435484 | 17 | | |
| $MWT_{eff}$ | 1 | 0.54569892 | 21 | 0.43413978 | 22 |
| | 2 | 0.32258065 | 20 | | |
| $MWWT_{S}$ | 1 | 0.56129032 | 22 | 0.44423963 | 23 |
| | 2 | 0.32718894 | 22 | | |
| $MWWT_{T}$ | 1 | 0.54032258 | 19 | 0.4681085 | 24 |
| | 2 | 0.39589443 | 25 | | |
| $MWWT_{W}$ | 1 | 0.66451613 | 26 | 0.55219941 | 26 |
| | 2 | 0.4398827 | 27 | | |
| $Rand$ | 1 | 0.8266129 | 28 | 0.62567204 | 28 |
| | 2 | 0.42473118 | 26 | | |

En la tabla también puede verse que las estrategias que consideran la velocidad de los sitios ($v_i$) obtienen en la mayoría de los casos mejores resultados que sus contrapartes que no consideran este factor. Así pues, se tiene que el conocer previamente la velocidad de los sitios del Grid es conveniente para la calidad del servicio otorgado.

Las Figura 7, Figura 8, Figura 9 y Figura 10 clasifican –utilizando la fórmula 12– las seis mejores estrategias para cada métrica en particular. En el apéndice A1 se

muestran los valores promedio y la desviación estándar de las estrategias para cada métrica.

Como se ve en la **Figura 7** y aunque sólo se muestren los resultados de las seis mejores estrategias, la estrategia MST está en primer lugar para el escenario Grid1 y en segundo lugar para el escenario Grid2. Es un resultado esperado, dado que dicha estrategia busca minimizar los tiempos de inicio de los trabajos, minimizando así la longitud del calendario.

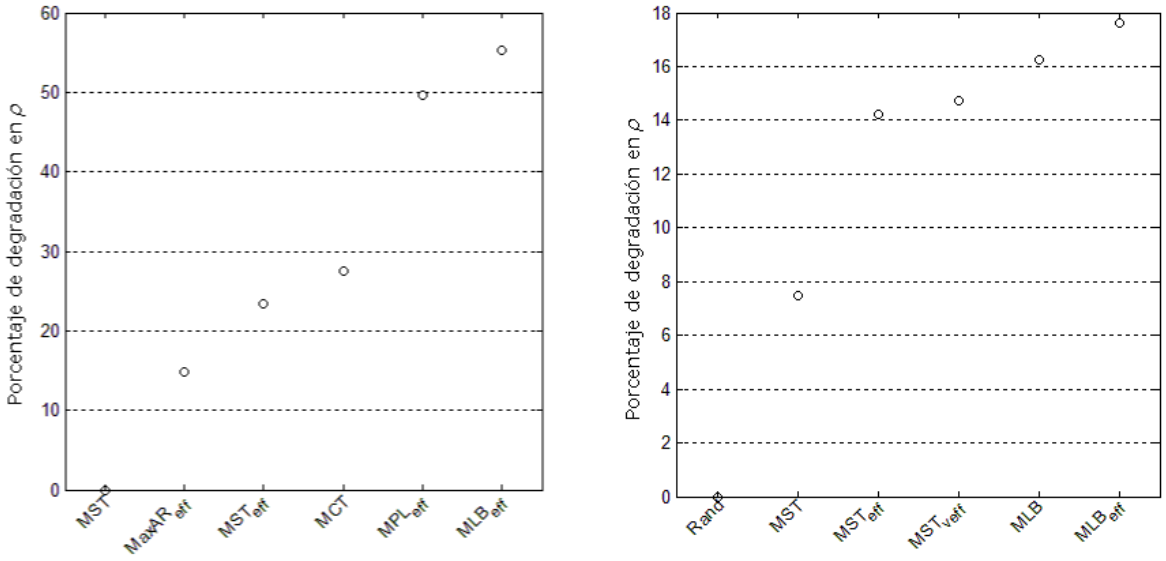


**Figura 7: Porcentaje de degradación en $\rho$ para Grid1 (izquierda) y Grid2 (derecha). Seis mejores estrategias.**

La Figura 8 muestra los resultados considerando el tiempo promedio de espera ($t_w$) de los trabajos. En el escenario Grid1 podemos ver que las mejores estrategias tienen un porcentaje de degradación 0, lo cual contrasta con lo obtenido en el escenario Grid2, que muestra una diferencia de más de $10^4$% entre las dos mejores estrategias. Esto puede deberse a la diferencia entre los tamaños de los Grids considerados en nuestros casos de estudio (4442 para Grid1 y 2194 para Grid2).

De igual manera podemos ver en la Figura 9 que la diferencia entre las mejores estrategias cuando se evalúa la suma de ralentizaciones ($SD_b$) es menor en Grid1

que en Grid2. Podemos ver que en Grid1 la estrategia $MLp_v$ es 4% peor que la mejor estrategia para esa métrica ($MST_v$), mientras que en Grid2 $MLp_v$ obtiene el mejor resultado. Al buscar asignar los trabajos al sitio con menor proporción de carga, ésta estrategia minimiza la ralentización de los trabajos, como era de esperarse.

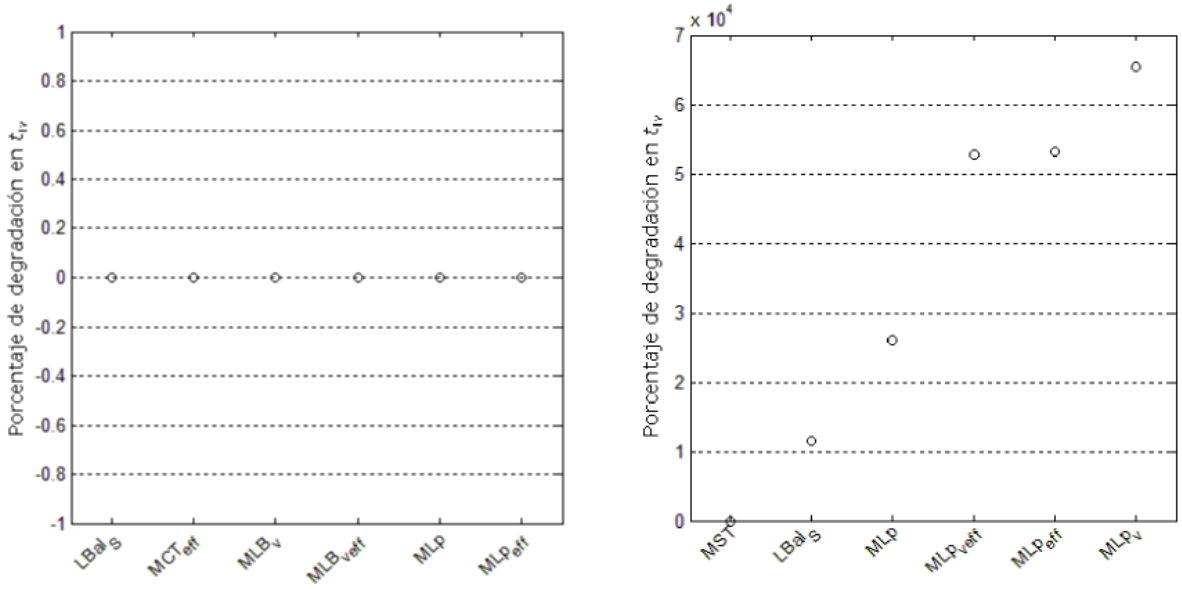


**Figura 8: Porcentaje de degradación en $t_w$ para Grid1 (izquierda) y Grid2 (derecha). Seis mejores estrategias.**

La Figura 10 compara el desempeño de las estrategias considerando su consumo energético ($E_{Grid}$). La diferencia entre ambos escenarios se nota claramente. En Grid1 hay una diferencia menor al 2.5% entre la mejor estrategia y la que ocupa el sexto puesto. Para Grid2 esta diferencia crece a más de 180%. Podemos notar que en Grid2 la estrategia que busca balancear el consumo energético de los sitios ($LBal_E$) es la que obtiene el mejor resultado.

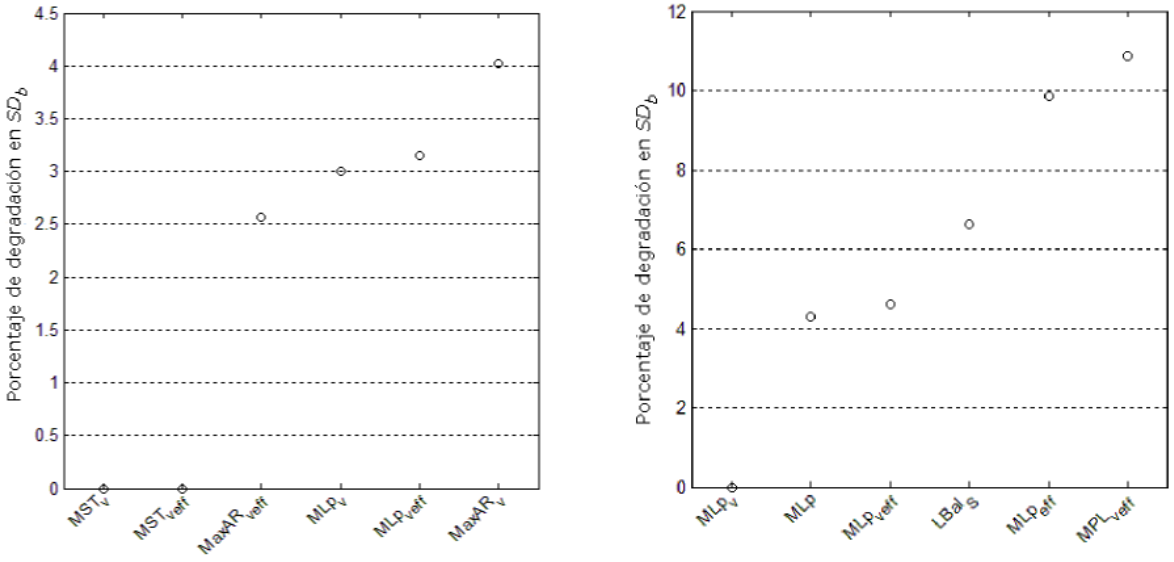


**Figura 9: Porcentaje de degradación en $SD_b$ para Grid1 (izquierda) y Grid2 (derecha). Seis mejores estrategias.**

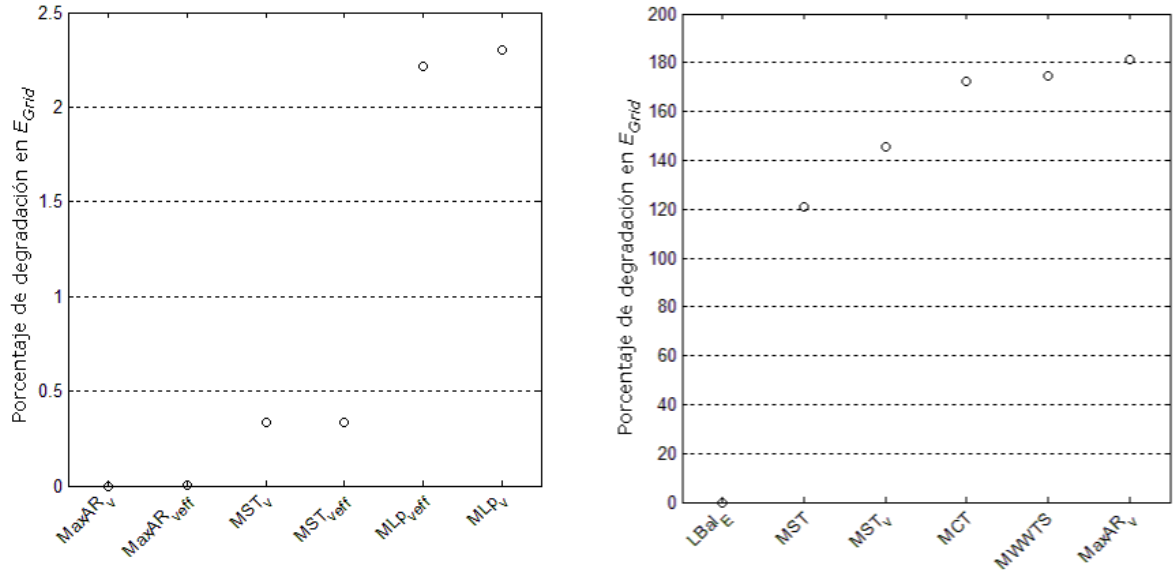


**Figura 10: Porcentaje de degradación en $E_{Grid}$ para Grid1 (izquierda) y Grid2 (derecha). Seis mejores estrategias.**

Como segundo paso de la experimentación, se procedió a comparar las estrategias en base a su eficiencia energética mediante la fórmula 10. La Tabla

11: Clasificación de las estrategias con base a su eficiencia energética muestra la clasificación obtenida. La columna de eficiencia energética muestra los porcentajes de degradación en esta métrica.

Tabla 11: Clasificación de las estrategias con base a su eficiencia energética

| Estrategia | Grid1 | Eficiencia energética | Clasificación |
|---|---|---|---|
| $LBal_E$ | 1 | 47.38 | 3 |
| | 2 | 0 | |
| $LBal_S$ | 1 | 24.62 | 14 |
| | 2 | 69.18 | |
| $LBal_T$ | 1 | 50.22 | 16 |
| | 2 | 69.42 | |
| $LBal_W$ | 1 | 28.46 | 15 |
| | 2 | 69.61 | |
| $MCT$ | 1 | 84.31 | 20 |
| | 2 | 37.43 | |
| $MCT_{eff}$ | 1 | 14.45 | 28 |
| | 2 | 65.59 | |
| $MLB$ | 1 | 74.38 | 26 |
| | 2 | 71.9 | |
| $MLB_{eff}$ | 1 | 81.21 | 28 |
| | 2 | 68.71 | |
| $MLB\_v$ | 1 | 10.1 | 9 |
| | 2 | 70.5 | |
| $MLB_{veff}$ | 1 | 15.91 | 10 |
| | 2 | 67.58 | |
| $MLp$ | 1 | 21.39 | 13 |
| | 2 | 71.23 | |
| $MLp_{eff}$ | 1 | 49.28 | 17 |
| | 2 | 70.39 | |
| $MLp_v$ | 1 | 1.96 | 5 |
| | 2 | 67.39 | |
| $MLp_{veff}$ | 1 | 1.88 | 6 |
| | 2 | 68.59 | |
| $MPL$ | 1 | 74.04 | 25 |
| | 2 | 71.19 | |

Tabla 12 (continuación)

| | | | |
|---|---|---|---|
| $MPL_{eff}$ | 1 | 82.55 | 29 |
| | 2 | 69.69 | |
| $MPL_v$ | 1 | 16.23 | 11 |
| | 2 | 69.23 | |
| $MPL_{veff}$ | 1 | 21.06 | 12 |
| | 2 | 68.14 | |
| $MST$ | 1 | 88.21 | 18 |
| | 2 | 31.53 | |
| $MST_{eff}$ | 1 | 92.16 | 31 |
| | 2 | 74.77 | |
| $MST_v$ | 1 | 0.34 | 2 |
| | 2 | 25.73 | |
| $MST_{veff}$ | 1 | 0.34 | 7 |
| | 2 | 74.64 | |
| $MWT$ | 1 | 61.13 | 21 |
| | 2 | 64.51 | |
| $MWT_{eff}$ | 1 | 58.5 | 23 |
| | 2 | 70 | |
| $MWWT_S$ | 1 | 60.37 | 19 |
| | 2 | 61.14 | |
| $MWWT_T$ | 1 | 61.23 | 22 |
| | 2 | 65.41 | |
| $MWWT_W$ | 1 | 67.72 | 24 |
| | 2 | 66.12 | |
| $MaxAR_{eff}$ | 1 | 87.87 | 30 |
| | 2 | 73.48 | |
| $MaxAR_v$ | 1 | 0 | 1 |
| | 2 | 5.99 | |
| $MaxAR_{veff}$ | 1 | 0.001 | 4 |
| | 2 | 64.72 | |
| $Rand$ | 1 | 81.07 | 27 |
| | 2 | 68.53 | |

Podemos ver que la estrategia $MaxAR_v$ es la mejor en este aspecto para los escenarios evaluados, obteniendo el mejor resultado en Grid1 y una degradación de 5.99% para Grid2. Esto era de esperarse, ya que fue la mejor en las otras métricas y $EnEff$ está basada en $\rho$ y $E_{Grid}$. Las Figura 11 y Figura 12 muestran la degradación del desempeño de las estrategias para ambos casos.

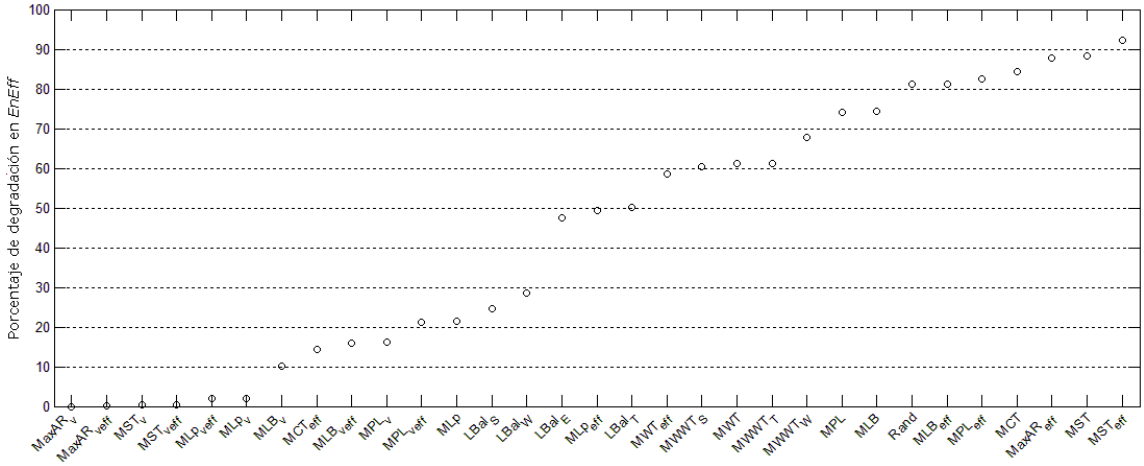


**Figura 11: Porcentaje de degradación promedio de la eficiencia energética de todas las métricas para Grid1.**

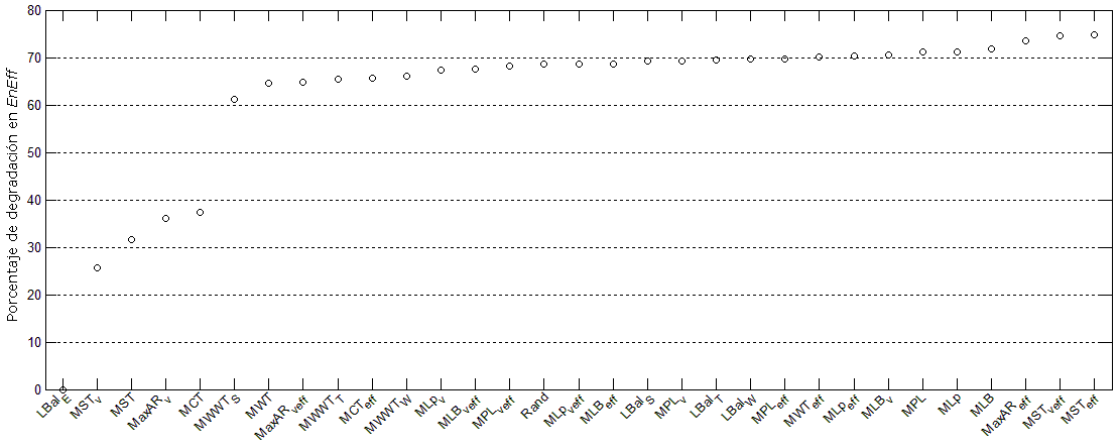


**Figura 12: Porcentaje de degradación promedio de la eficiencia energética de todas las métricas para Grid2.**

Teniendo $MaxAR_v$ como la mejor estrategia, se comparó con otra estrategia que llamaremos $MaxAR_{vc}$; esto es, con $MaxAR_v$ sin considerar apagar y encender recursos dinámicamente. La Figura 13, Figura 14 y Figura 15 muestran los resultados comparativos en las tres métricas tradicionales.

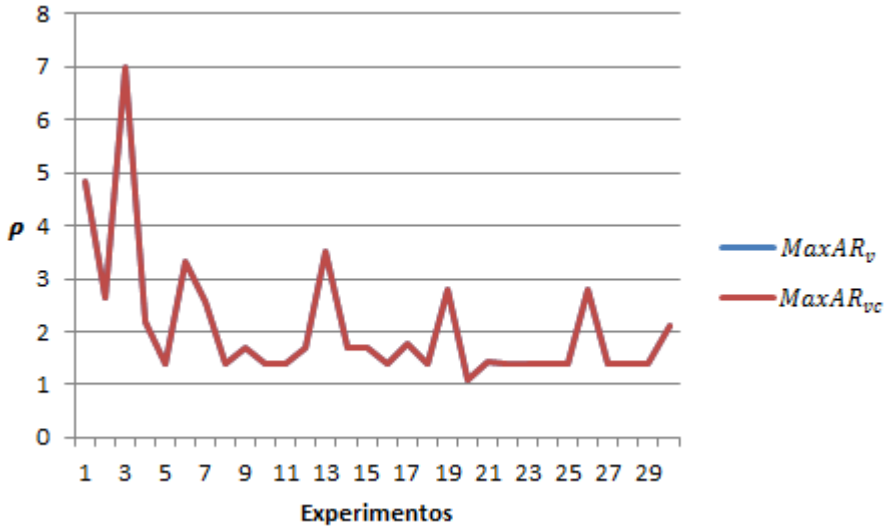


**Figura 13: Comparación de $\rho$ entre $MaxAR_v$ y $MaxAR_{vc}$.**

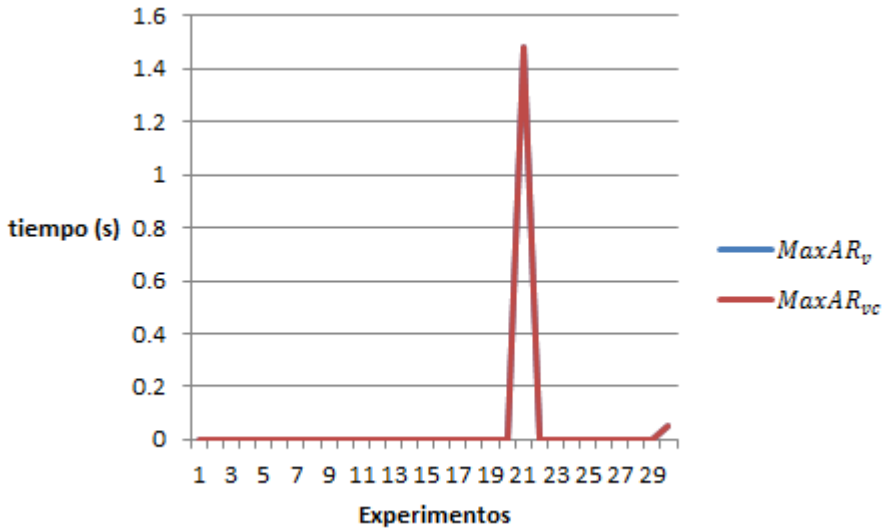


**Figura 14: Comparación de $t_w$ entre $MaxAR_v$ y $MaxAR_{vc}$.**

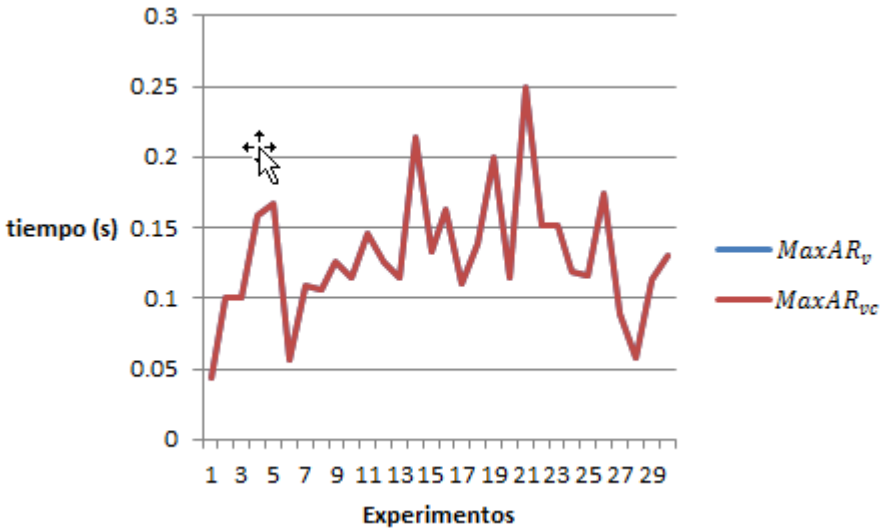


**Figura 15: Comparación de $SD_b$ entre $MaxAR_v$ y $MaxAR_{vc}$.**

Como puede verse, en $\rho$, $t_w$ y $SD_b$ ambas estrategias dan los mismos resultados. En $E_{Grid}$ y $EnEff$ es donde se muestra una ventaja significativa de $MaxAR_v$ (Figura 16 y Figura 17). Se esperaba reducir el consumo energético perdiendo ligeramente la calidad del servicio entregado y medido por las tres métricas tradicionales $\rho$, $t_w$ y $SD_b$, pero los resultados experimentales muestran que para estos casos de estudio es posible reducir el consumo energético sin pérdida de calidad. Esto es un resultado muy satisfactorio, ya que abre la posibilidad a seguir reduciendo el consumo energético hasta comenzar a notar cambios en las demás métricas.

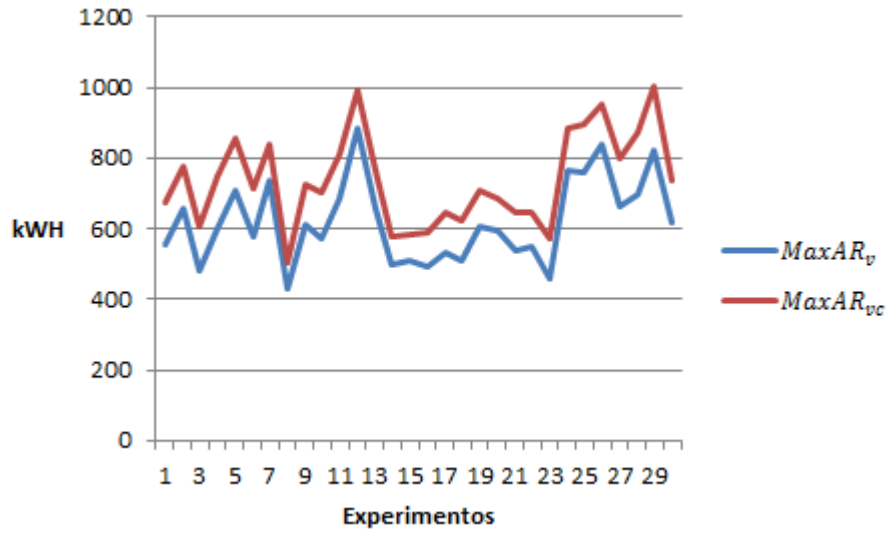


**Figura 16: Comparación de $E_{Grid}$ entre $MaxAR_v$ y $MaxAR_{vc}$**

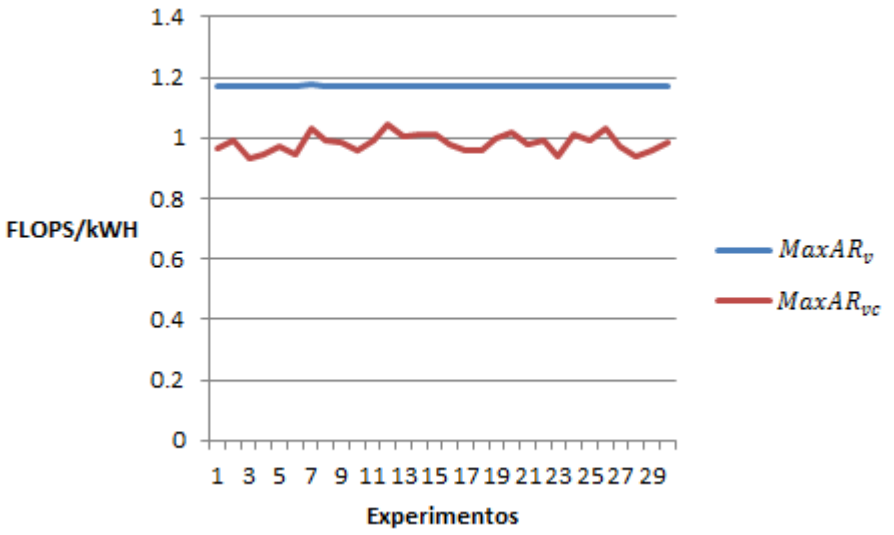


**Figura 17: Comparación de $EnEff$ entre $MaxAR_v$ y $MaxAR_{vc}$**

## Conclusiones

Dado que los Grids se están volviendo cada vez más accesibles, son necesarias estrategias para utilizar eficientemente (en todos los aspectos) sus recursos. El problema de asignación de recursos debe verse ahora desde un punto de vista más: el energético. Ya no basta con satisfacer las demandas de los usuarios o administradores.

En esta tesis se abordó la calendarización en línea sin interrupciones y no clarividente de trabajos paralelos en un Grid jerárquico de dos niveles, buscando una buena estrategia que, sin descuidar los objetivos tradicionales del sistema, fuera más eficiente energéticamente. Se presentó una evaluación detallada del rendimiento de algoritmos de asignación. Se consideraron estrategias ya conocidas y, para hacer un mejor análisis, se les agregaron los valores de eficiencia energética y velocidad de cada sitio.

La elección de la estrategia para asignación de trabajos sigue dependiendo de las preferencias de quien toma las decisiones (usuarios, proveedores de recursos, administradores), y de la importancia que tenga cada criterio para ellos, o ya de una importancia relativa entre ellos.

Para obtener una guía basada en estos criterios, se realizó un análisis de cuatro métricas, basándonos en la optimalidad de pareto de cada estrategia.

Los resultados presentados muestran lo siguiente para los casos de estudio considerados:

- En términos de los criterios considerados, las estrategias $MaxAR_v$, $MST_v$ y $MLp_v$ obtienen los mejores resultados.
- Si se busca utilizar una estrategia cuya prioridad sea la eficiencia energética, podemos contar con $MaxAR_v$, que obtiene los mejores resultados también en este aspecto
- Puede reducirse el consumo energético utilizando la metodología propuesta sin reducir la calidad de los calendarios en las demás métricas.

**Trabajo Futuro**

- Extender este modelo energético a otras arquitecturas de cómputo distribuido (la nube).
- Seguir reduciendo el consumo energético –posiblemente reduciendo los tiempos de gracia de apagado de recursos– hasta que la calidad del servicio comience a bajar.
- Ampliar el modelo actual para considerar consumo de enfriamiento.
- Agregar otra métrica para evaluar la calidad, como puede ser costo económico del calendario.

## Referencias bibliográficas


Anoep, S., Dumitrescu, C., Dick, E., Iosup, A., Jan, M., Li, H., *et al*. (2007). Grid Workloads Archive. Retrieved April, 2011, from http://gwa.ewi.tudelft.nl

Barrondo, A., Tchernykh, A., Schaeffer, E. & Pecero, J. (2012). Energy Efficiency of Knowledge-Free Scheduling in Peer-to-Peer Desktop Grids. 2012 Workshop on Optimization Issues in Energy Efficient Distributed Systems (OPTIM 2012). In conjunction with International Conference on High Performance Computing & Simulation (HPCS 2012) , to be held July 2 - July 6, 2012, in Madrid, Spain.

Beberg, A., Ensign, D., Jayachandran, G., Khaliq, S., & Pande, V. (2009). Folding@home: lessons From Eight Years of Volunteer Distributed Computing. 2009 IEEE International Symposium on Parallel & Distributed Processing

Beloglazov, A., Buyya, R., Lee, Y.C. & Zomaya, A. (2010). A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. Young, Vol. 82(CLOUDS-TR-2010-3), 49.

Berman, F., & Hey, T. (2004). The Scientific Imperative. In I. Foster & C. Kesselman (Eds.), The Grid 2. Blueprint for a New Computing Infrastructure (Vol. 2). San Francisco, CA, USA. Morgan Kaufmann.

Buyya, R. (2005). Grid Computing Info Centre (GRID Infoware). Retrieved October 2011, 2011, from http://www.gridcomputing.com/

California, U. o. (2009b). SETI@home. Retrieved January 2012, from http://setiweb.ssl.berkeley.edu/

CERN. (2008). Grid Café. Retrieved January 2012, from http://www.gridcafe.org

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

Chow, Y.-C. & Kohler, W.H. (1979). Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System. IEEE Trans. Comput., IEEE Computer Society, Vol. 28, 354-361.

Cirne, W. (2001) A model for moldable supercomputer jobs. Parallel and Distributed Processing Symposium., Proceedings 15th International

Coello Coello, C., Lamont, G. & Veldhuizen, D. Van (2007). Evolutionary Algorithms for Solving Multi-Objective Problems. 2nd ed., 2007, XXII.

Develder, C., Pickavet, M., Dhoedt, B. & Demeester, P. (2008). A power-saving strategy for Grids. Proc. 2nd Int. Conf. on Networks for Grid Applications

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

Dutton, R., Mao, W. (2007) Online scheduling of malleable parallel jobs. PDCS '07 Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems, 136-141

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

Feitelson, D. G., & Weil, A. M. a. (1998). Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. Paper presented at the 12th. International Parallel Processing Symposium.

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

Foster, I., & Kesselman, C. (2003). The Grid in a Nutshell. In J. Nabrzyski, J. M. Schopf & J. Weglarz (Eds.), Grid Resource Management, State of the Art and Future Trends (1a ed.): Kluwer Academic Publisher.

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

Foster, I., Kesselman, C., & Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15(3), 200-222.

Frachtenberg, E., & Feitelson, D. (2005). Pitfalls in Parallel Job Scheduling Evaluation. In D. Feitelson, E. Frachtenberg, L. Rudolph & U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing (Vol. 3834, pp. 257-282): Springer Berlin / Heidelberg.

Gandhi, A., Harchol-Balter, M., Das, R. & Lefurgy, C. (2009). Optimal power allocation in server farms. Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems, ACM.

Gelenbe, E. & Silvestri, S. (2009). Reducing power consumption in wired networks. ISCIS IEEE, 292-297

Grimme, C., Lepping, J., & Papaspyrou, A. (2007, Nov). Identifying Job Migration Characteristics in Decentralized Grid Scheduling Scenarios. Paper presented at the Parallel and Distributed Computing and Systems, Cambridge, MA, USA.

Grimme, C., Lepping, J., & Papaspyrou, A. (2008). Prospects of collaboration between compute providers by means of job interchange, Proceedings of the 13th international conference on Job scheduling strategies for parallel processing. Seattle, WA, USA: Springer-Verlag.

Hamscher, V., Schwiegelshohn, U., Streit, A., & Yahyapour, R. (2000). Evaluation of Job-Scheduling Strategies for Grid Computing. Paper presented at the First IEEE/ACM International Workshop on Grid Computing (GRID 2000).

Hirales-Carbajal, A.; Tchernykh, A.; Roblitz, T.; Yahyapour, R.; , "A Grid simulation framework to study advance scheduling strategies for complex workflow applications," Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW), 2010 IEEE International Symposium on , vol.,

no., pp.1-8, 19-23 April 2010, doi: 10.1109/IPDPSW.2010.5470918. Print ISBN: 978-1-4244-6533-0, ATLANTA (Georgia) USA

Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., Röblitz, T., Ramírez-Alcaraz, J. & González-García, J. (2012). Multiple Parallel Workflow Scheduling Strategies on a Grid. Journal of Grid Computing, Springer -Verlag, Netherlands, 2012. DOI: 10.1007/s10723-012-9215-6, ISSN: 1570-7873.

Lifka, D. A. (1995). The ANL/IBM SP Scheduling System. Paper presented at the Job Scheduling Strategies for Parallel Processing, Santa Barbara, CA, USA.

Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., & Freund, R. F. (1999). Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. Paper presented at the Proceedings of the Eighth Heterogeneous Computing Workshop, San Juan, Puerto Rico.

Martello, S., Monaci, M., * Vigo, D. (2003) An Exact Approach to the Strip-Packing Problem. INFORMS Journal on Computing Summer 2003 15:310-319;

MIT, U. o. (2009). Einstein@Home. Retrieved June 2012, from http://hdl.handle.net/1721.1/56299

Moreno, I.S. & Xu, J. (2011). Energy-Efficiency in Cloud Computing Environments: Towards Energy Savings without Performance Degradation. IJCAC Vol. 1.

Orgerie, A., Lefèvre, L. & Gelas, J. (2010). Demystifying Energy Consumption in Grids and Clouds. Work in Progress in Green Computing (WIPGC) Workshop, in conjunction with the first International Green Computing Conference (IGCC), Chicago, USA, pages 335-342.

Orgerie, A., Lefèvre, L. & Patrick Gelas, J. (2008) How an experimental Grid is used: The Grid5000 case and its impact on energy usage. Proc. 8 the IEEE Int. Symp. on Cluster Computing and the Grid.

Quezada-Pina, A., Tchernykh, A., González-García, J. Hirales-Carbajal, A., Miranda-López, V., Ramírez-Alcaráz, J., Scwiegelshohn, U. & Yahyapour, R. (2012). Adaptive job scheduling on hierarchical Grids. Future Generation Computer Systems 28/7 (2012) pp. 965-976. Elsevier Science, DOI:10.1016/j.future.2012.02.004, ISSN: 0167-739X.

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

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

Reyes-Sierra, M. & Coello Coello C. (2005). Improving PSO-based Multi-Objective Optimization using Crowding, Mutation and ∈-Dominance. EMO'05 Proceedings of the Third international conference on Evolutionary Multi-Criterion Optimization pp. 505-519.

Schwiegelsohn, U., Tchernykh, A. & Yahyapour, R. (2008). Online Scheduling in Grids. IPDPS 2008, IEEE International Symposium on Parallel and Distributed Processing, April 14-18, 2008, Miami, Florida USA pp. 1-10

Schwiegelshohn, U., Yahyapour, R. (2004). Attributes for communication between Grid scheduling instances. Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid Resource Management: State of the Art and Future Trends, pp. 41–52. Kluwer, Norwell

Skovira, J., Chan, W., & Zhou, H. (1996, Abril 16). The EASY-LoadLeveler API Project. Paper presented at the Job Scheduling Strategies for Parallel Processing, Honolulu, Hawaii.

Subrata, R., Zomaya, A.Y. & Landfeldt, B. (2010). Cooperative power-aware scheduling in Grid computing environments. J. Parallel Distrib. Comput., Academic Press, Inc. Vol. 70, 84-91.

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

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

W. Vereecken *et al* (2008). Energy efficiency in telecommunication networks. Proc. 13th European Conf. on Networks & Optical Commun.

Washington, U. o. (2009). Rosetta@home. Retrieved September 2009, from http://boinc.bakerlab.org/rosetta/

Yao, F., Demers, A. & Shenker, S. (1995). A scheduling model for reduced CPU energy. Proceedings of the 36th Annual Symposium on Foundations of Computer Science IEEE Computer Society.

# Apéndices

## A.1 Rendimiento de las estrategias de asignación

A continuación se presentan los valores promedio y la desviación estándar de las 31 estrategias consideradas para las cuatro métricas en ambos casos de prueba:

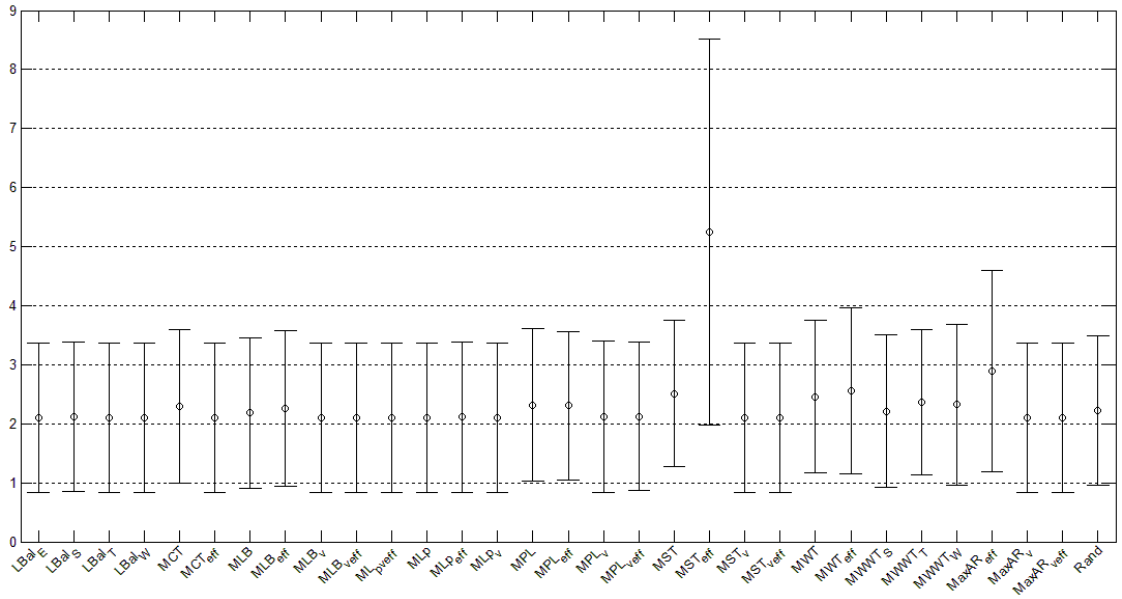


**Figura 18: Valores promedio para $\rho$ en Grid1**

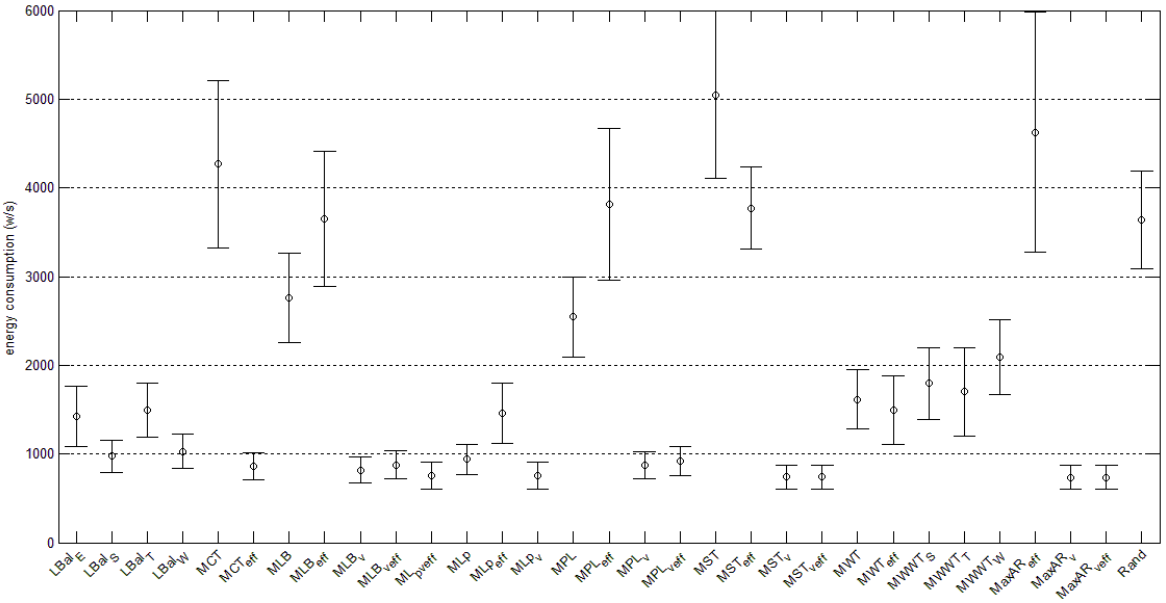


**Figura 19: Valores promedio para $E_{Grid}$ en Grid1**

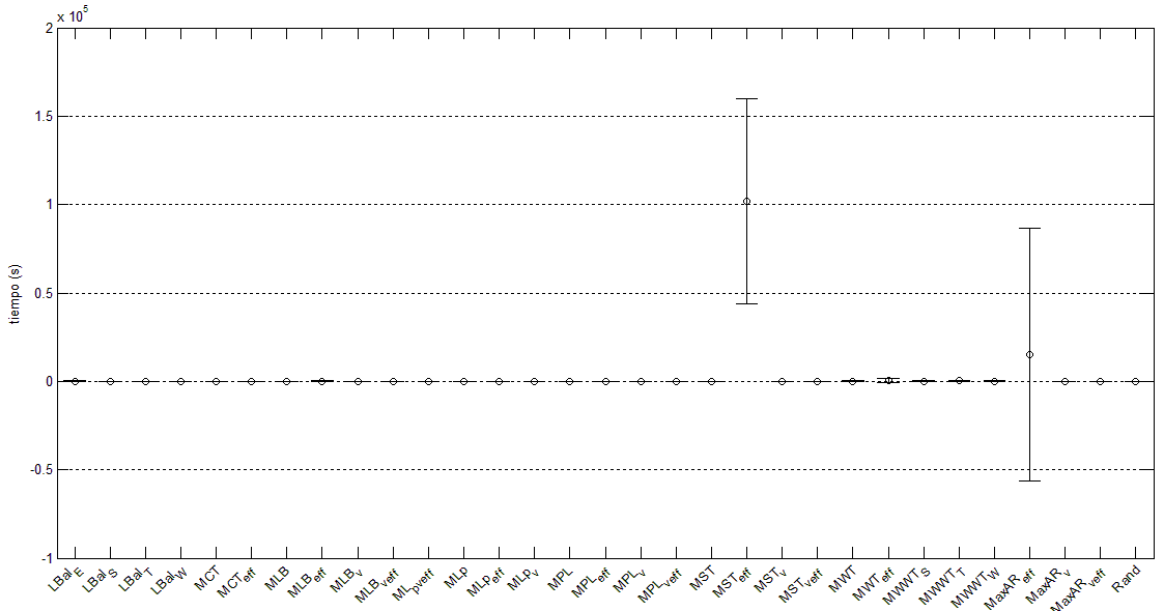


**Figura 20: Valores promedio para $t_w$ en Grid1**

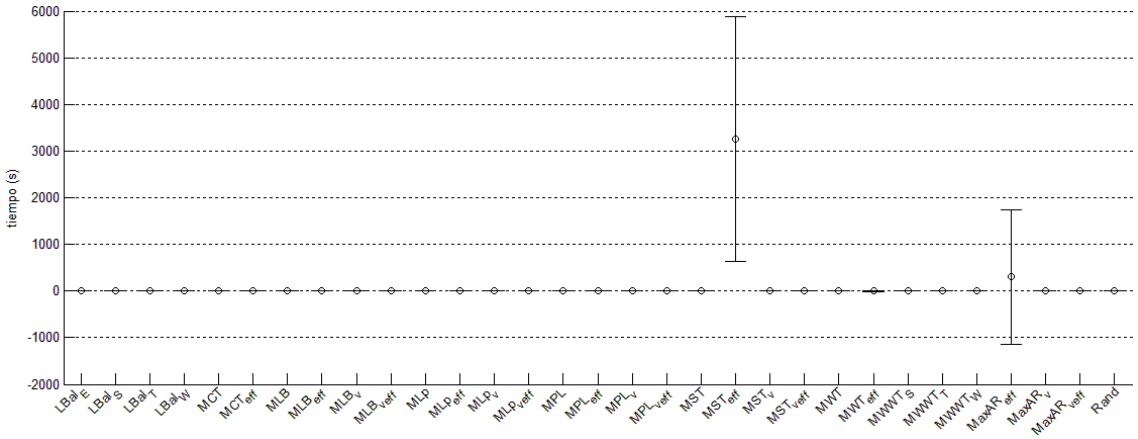


**Figura 21: Valores promedio para $SD_b$ en Grid1**

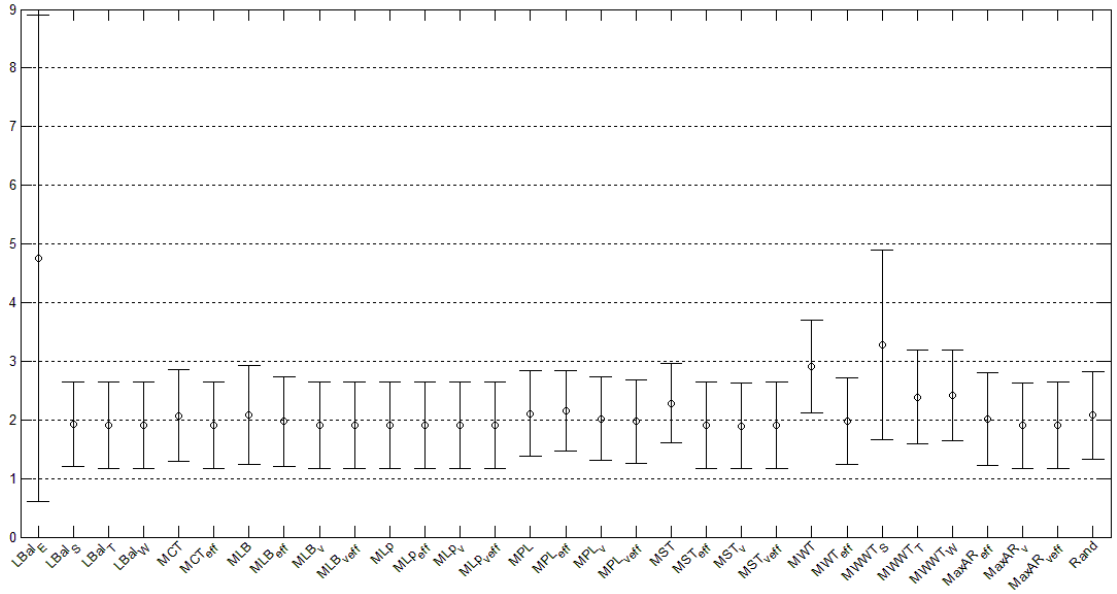


**Figura 22: Valores promedio para $\rho$ en Grid2**

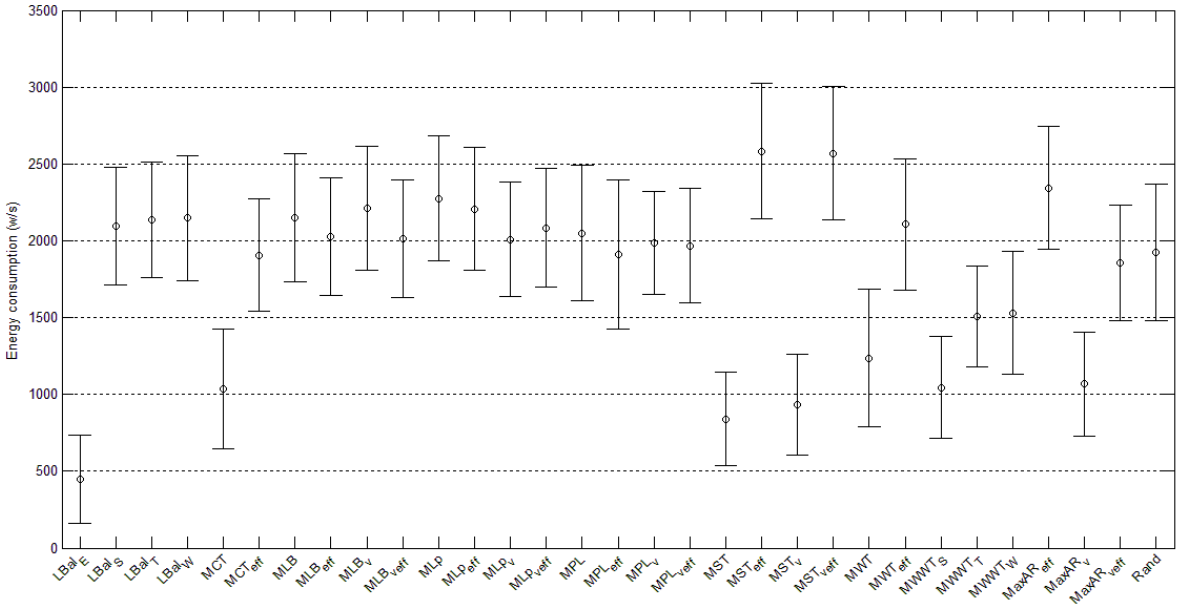


**Figura 23: Valores promedio para $E_{Grid}$ en Grid2**

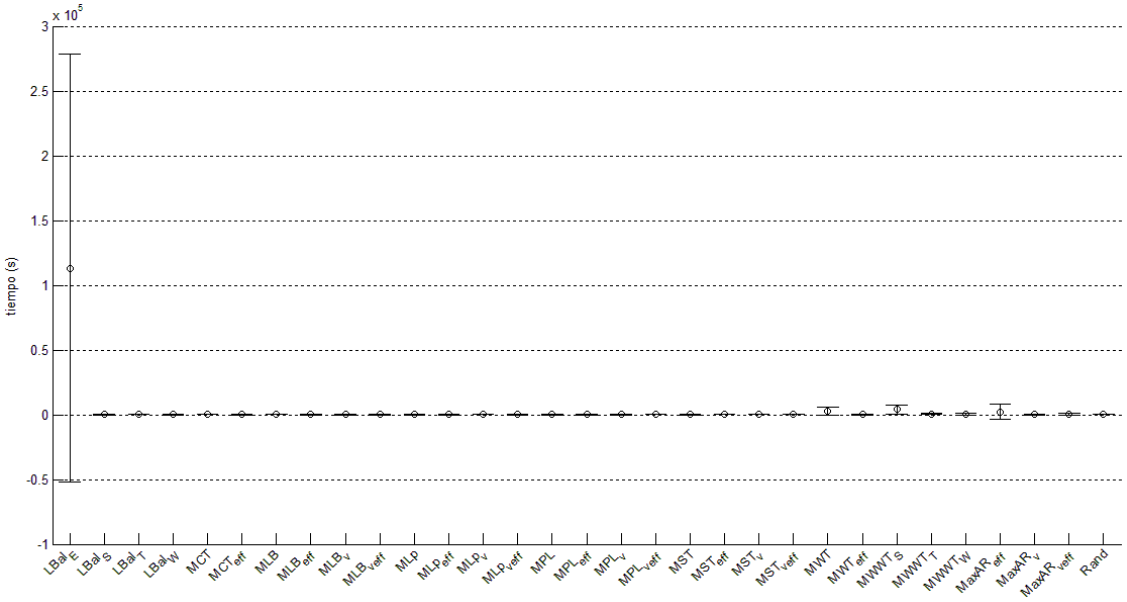


**Figura 24: Valores promedio para $t_w$ en Grid2**

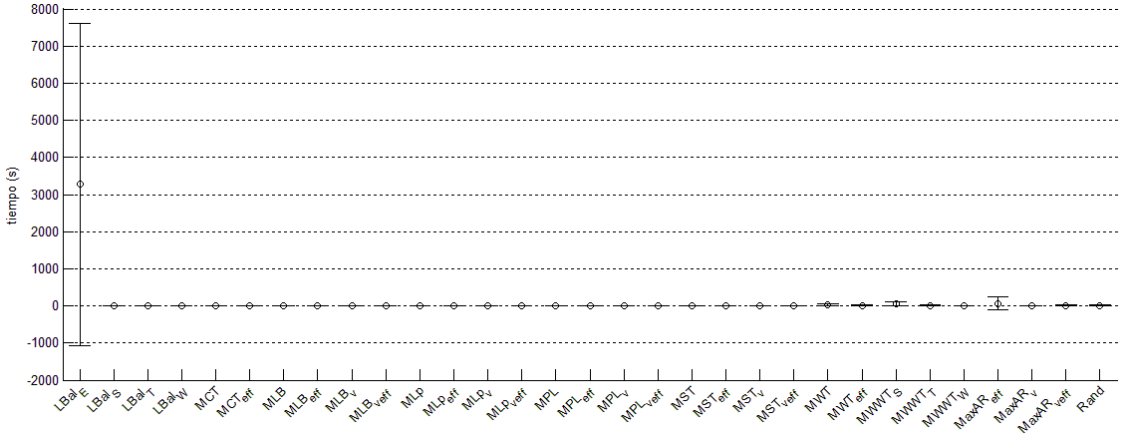


**Figura 25: Valores promedio para $SD_b$ en Grid2**

## A.2 Arquitectura del simulador

La arquitectura del simulador está dividida en cuatro capas, como puede verse en la Figura 26 (Hirales-Carbajal *et al*, 2010):

**Capa base:** La base de tGSF (teikoku Grid Scheduling Framework) es un núcleo responsable de administrar el tiempo y administrar eventos. Este núcleo utiliza un reloj interno para aislar las capas superiores del reloj en tiempo real y proveer abstracción para periodos e instantes. Además, provee un sistema de tipo extensible que permite el manejo de eventos personalizados para cada dominio. El comportamiento de tGSF, sin embargo, no depende del núcleo en sí; está determinado por el Ambiente de Ejecución (*Runtime Environment)*, el cual (dependiendo de la implementación) permite la ejecución de simulaciones y sistemas de prueba.

**Capa común:** Para proveer abstracciones generales para administración, esta capa provee métricas, un modelo de trabajos y servicios de persistencia. El modelo de trabajos utiliza un enfoque holístico de unidades de trabajo dentro de un ambiente Grid: consiste en una descripción de características estáticas de los trabajos y recursos requeridos; un ciclo de vida que representa el estado actual del trabajo, así como un historial, y una procedencia que denota la ruta que un trabajo ha recorrido desde su llegada al sistema hasta la asignación y ejecución. El servicio de métricas provee una interfaz unificada para medir los datos en tiempo de ejecución: valores relacionados a distintos aspectos pueden ser almacenados y recuperados para, por ejemplo, utilizarse para la toma de decisiones. Tales métricas pueden ser objetivos de desempeño como la utilización y el tiempo de respuesta. Los servicios de persistencia ofrecen un acceso común a distintos mecanismos de almacenamiento como archivos y bases de datos relacionales.

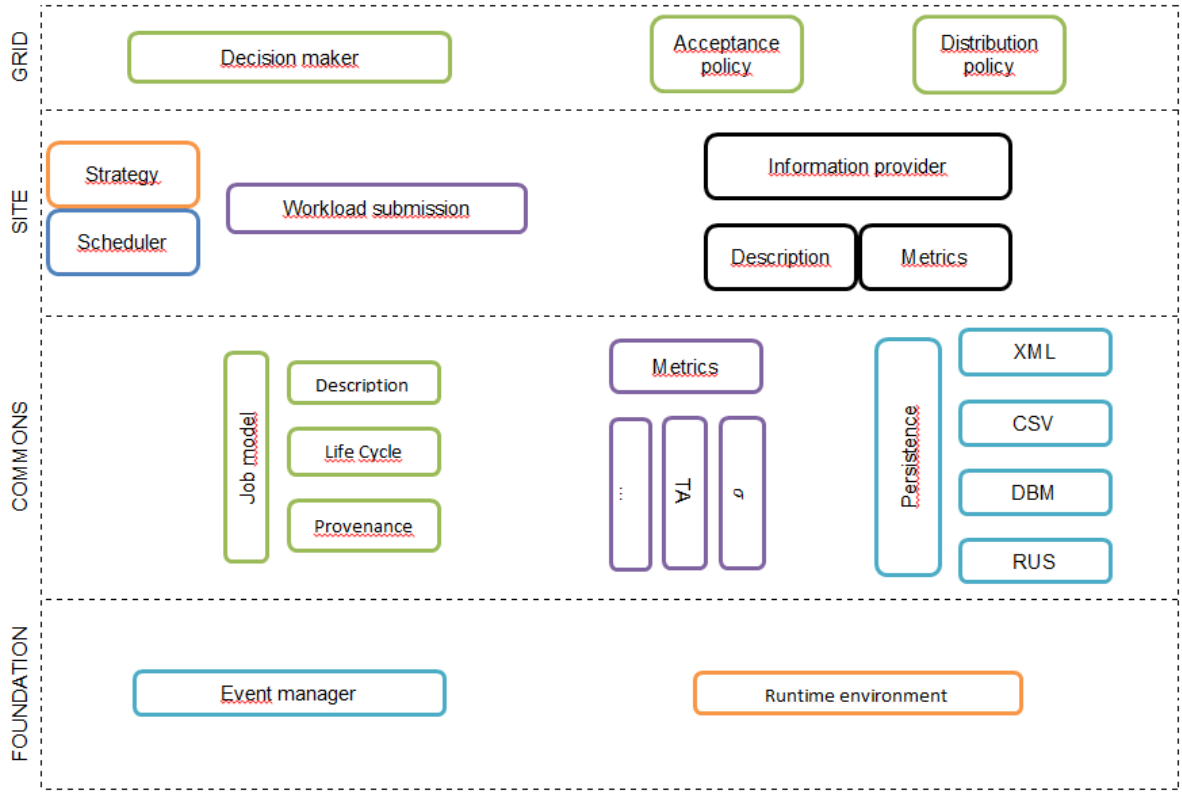


**Figura 26: Arquitectura del simulador**

**Capa de sitio:** La administración de un conjunto de recursos se lleva a cabo aquí. Esta capa abstrae la implementación de la funcionalidad de un calendarizador. Además de manejar estructuras de datos para tener información del calendario actual, permite la aplicación de estrategias de asignación en tiempo de ejecución y ofrece una interfaz de proveedor de servicios (SPI, *Service Provider Interface*) hacia sistemas tradicionales de administración de recursos locales. Aunque las estrategias calculan la asignación de trabajos a recursos, sus resultados son meramente en calidad de asesor: proponen sus soluciones al calendarizador, el cual decide qué acciones tomar. Esto permite al calendarizador consultar múltiples estrategias a la vez y elegir la solución más apropiada. El proveedor de información agrupa información tanto estática como dinámica del sitio tal como la cantidad de recursos, la utilización actual, etc., y presenta estos datos a los demás componentes de esta capa y (parcialmente) al ambiente global.

**Capa de Grid:** Para permitir una interacción necesaria para la distribución de la carga de trabajo a los otros sitios participantes dentro de un ambiente Grid, esta capa provee servicios para el intercambio de trabajos. Dependiendo en la topología del calendarizador de Grid, el Tomador de Decisiones (*Decision Maker)* tiene distintas responsabilidades. En sistemas jerárquicos o centralizados su principal tarea es la de aceptar trabajos y asignarlos a recursos o administradores subordinados. En escenarios no centralizados, la redistribución a otros tomadores de decisiones es incluida; la decisión de las modalidades se basa en un conjunto de políticas de acuerdo a la aceptación y distribución de trabajos.

## A.3 Ejemplo de un archivo de configuración del simulador

```
# Identificadores de los sitios del Grid. El sitio 0 es el administrador del Grid
# y no ejecuta ningún trabajo, sólo administra a los demás sitios. En este ejemplo se considera un
Grid de dos sitios.
sites.id = site0
sites.id = site1
sites.id = site2

# Información de la ejecución
runtime.workloadSource.id = site0
runtime.workloadSource.id = site1
runtime.workloadSource.id = site2

runtime.site0.associatedSite.ref = site0
runtime.site1.associatedSite.ref = site1
runtime.site2.associatedSite.ref = site2

# Carga de trabajo para cada sitio. Se le pasa toda al sitio 0, éste se encargará de redistribuirla
en tiempo de ejecución.
runtime.site0.url = file:G:\\Users\\mitza\\workspace\\tGSF\\wl\\Carga01.swf
runtime.site1.url = file:G:\\Users\\mitza\\workspace\\tGSF\\wl\\emptyWorkload.swf
runtime.site2.url = file:G:\\Users\\mitza\\workspace\\tGSF\\wl\\emptyWorkload.swf

# Site 0: configuration section
# El tamaño es el número total de recursos del Grid
sites.site0.numberOfProvidedResources = 228
sites.site0.gridActivityBroker.class = mx.cicese.dcc.teikoku.broker.CentralizedGridActivityBroker
sites.site0.gridActivityBroker.composite.strategy.class                                          =
mx.cicese.dcc.teikoku.scheduler.strategy.composite.DAS
sites.site0.gridActivityBroker.dss.policy.class                                                  =
mx.cicese.dcc.teikoku.scheduler.priority.DownwardRank
sites.site0.gridActivityBroker.dss.strategy.class                                                =
mx.cicese.dcc.teikoku.scheduler.strategy.rigid.MLB

# Aquí se define la estrategia a utilizar
#sites.site0.gridStrategyRigid.class = mx.cicese.dcc.teikoku.scheduler.strategy.rigid.LBal_E

sites.site0.gridInformationBroker.class                                                          =
mx.cicese.dcc.teikoku.information.broker.GridInformationBrokerImpl
sites.site0.activitybroker.distributionpolicy.class                                              =
de.irf.it.rmg.core.teikoku.Grid.distribution.StandardDistributionPolicy
sites.site0.activitybroker.transferpolicy.class                                                  =
de.irf.it.rmg.core.teikoku.Grid.transfer.FittingTransferPolicy
sites.site0.activitybroker.acceptancepolicy.class                                                =
de.irf.it.rmg.core.teikoku.Grid.acceptance.StandardAcceptancePolicy
sites.site0.activitybroker.locationpolicy.class                                                  =
de.irf.it.rmg.core.teikoku.Grid.location.MetricLocationPolicy
sites.site0.activitybroker.locationpolicy.metric = energy_Grid
sites.site0.activitybroker.public = size
sites.site0.resourcebroker.class = de.irf.it.rmg.core.teikoku.Grid.resource.TestResourceBrokerTake
sites.site0.resourcebroker.public = size
sites.site0.scheduler.class = de.irf.it.rmg.core.teikoku.scheduler.ParallelMachineScheduler
sites.site0.scheduler.localstrategy.class                                                        =
de.irf.it.rmg.core.teikoku.scheduler.strategy.FCFSStrategy
sites.site0.scheduler.localqueuecomparator.class                                                 =
de.irf.it.rmg.core.teikoku.scheduler.queue.NaturalOrderComparator
sites.site0.scheduler.localqueuecomparator.ordering = ascending
sites.site0.registeredMetric.ref = energy_Grid

#Configuración de los sitios
# Site 1 : configuration section, KTH
sites.site1.numberOfProvidedResources = 100
sites.site1.informationBroker.class                                                              =
mx.cicese.dcc.teikoku.information.broker.SiteInformationBrokerImpl
```

```
sites.site1.informationBroker.refreshRate = -1
sites.site1.activitybroker.class = de.irf.it.rmg.core.teikoku.Grid.activity.StandardActivityBroker
sites.site1.activitybroker.distributionpolicy.class                                                    =
de.irf.it.rmg.core.teikoku.Grid.distribution.StandardDistributionPolicy
sites.site1.activitybroker.transferpolicy.class                                                        =
de.irf.it.rmg.core.teikoku.Grid.transfer.FittingTransferPolicy
sites.site1.activitybroker.acceptancepolicy.class                                                      =
de.irf.it.rmg.core.teikoku.Grid.acceptance.StandardAcceptancePolicy
sites.site1.activitybroker.locationpolicy.class                                                        =
de.irf.it.rmg.core.teikoku.Grid.location.MetricLocationPolicy
sites.site1.activitybroker.locationpolicy.metric = parallel_site
sites.site1.activitybroker.public = size
sites.site1.scheduler.class = de.irf.it.rmg.core.teikoku.scheduler.ParallelMachineScheduler
sites.site1.scheduler.localstrategy.class                                                              =
de.irf.it.rmg.core.teikoku.scheduler.strategy.EASYStrategy
sites.site1.scheduler.localqueuecomparator.class                                                       =
de.irf.it.rmg.core.teikoku.scheduler.queue.NaturalOrderComparator
sites.site1.scheduler.localqueuecomparator.ordering = ascending
sites.site1.registeredMetric.ref = parallel_site
sites.site1.speed = 1

# Administración de energía
sites.site1.energy.sitePowerIdle = 32
sites.site1.energy.siteTurnOnDelay = 20000
sites.site1.energy.siteTurnOffDelay = 10000
sites.site1.energy.numberChassises=2
sites.site1.energy.chassisPowerIdle = 0
sites.site1.energy.boardPowerIdle = 24
sites.site1.energy.corePowerIdle = 17.8
sites.site1.energy.corePowerWorking = 26
sites.site1.energy.coreTurnOffDelay = 5000
sites.site1.energy.energyEfficiency = 1

# Site 2 : configuration section, SDSC-SP2
sites.site2.numberOfProvidedResources = 128
sites.site2.informationBroker.class                                                                    =
mx.cicese.dcc.teikoku.information.broker.SiteInformationBrokerImpl
sites.site2.informationBroker.refreshRate = -1
sites.site2.activitybroker.class = de.irf.it.rmg.core.teikoku.Grid.activity.StandardActivityBroker
sites.site2.activitybroker.distributionpolicy.class                                                    =
de.irf.it.rmg.core.teikoku.Grid.distribution.StandardDistributionPolicy
sites.site2.activitybroker.transferpolicy.class                                                        =
de.irf.it.rmg.core.teikoku.Grid.transfer.FittingTransferPolicy
sites.site2.activitybroker.acceptancepolicy.class                                                      =
de.irf.it.rmg.core.teikoku.Grid.acceptance.StandardAcceptancePolicy
sites.site2.activitybroker.locationpolicy.class                                                        =
de.irf.it.rmg.core.teikoku.Grid.location.MetricLocationPolicy
sites.site2.activitybroker.locationpolicy.metric = parallel_site
sites.site2.activitybroker.public = size
sites.site2.scheduler.class = de.irf.it.rmg.core.teikoku.scheduler.ParallelMachineScheduler
sites.site2.scheduler.localstrategy.class                                                              =
de.irf.it.rmg.core.teikoku.scheduler.strategy.EASYStrategy
sites.site2.scheduler.localqueuecomparator.class                                                       =
de.irf.it.rmg.core.teikoku.scheduler.queue.NaturalOrderComparator
sites.site2.scheduler.localqueuecomparator.ordering = ascending
sites.site2.registeredMetric.ref = parallel_site
sites.site2.speed = 3.077
#ENERGY MANAGEMENT
sites.site2.energy.sitePowerIdle = 50
sites.site2.energy.siteTurnOnDelay = 20000
sites.site2.energy.siteTurnOffDelay = 10000
sites.site2.energy.numberChassises=2
sites.site2.energy.chassisPowerIdle = 0
#sites.site2.energy.chassisPowerTurnOn
#sites.site2.energy.numberBoards
sites.site2.energy.boardPowerIdle = 40
```

```
#sites.site2.energy.boardPowerTurnOn
sites.site2.energy.corePowerIdle = 17.8
sites.site2.energy.corePowerWorking = 26
#sites.site2.energy.corePowerTurnOn
#sites.site2.energy.coreTurnOnDelay
sites.site2.energy.coreTurnOffDelay = 5000
sites.site2.energy.energyEfficiency = 1.881

# Metric : parallel job : Esta es una lista de métricas definidas en el código fuente del simulador.
metrics.id = awrtdel
metrics.id = art
metrics.id = rt
metrics.id = cu
metrics.id = awrt
metrics.id = swf
metrics.id = util
metrics.id = waiting_time_indp
metrics.id = parallel_site
metrics.id = workflow_Grid
metrics.id = parallel_Grid
metrics.id = parallel_site
metrics.id = energy_Grid

metrics.parallel_site.class = de.irf.it.rmg.core.teikoku.metrics.ParallelJob_Site
metrics.parallel_site.writer.class = de.irf.it.rmg.core.util.persistence.CSVFilePersistentStore
metrics.parallel_site.writer.mayOverwrite = true
metrics.parallel_site.writer.outputfile = 01
# Para evaluar el desempeño, utilizamos la métrica Energy_Grid
metrics.energy_Grid.class = mx.cicese.mcc.teikoku.metrics.Energy_Grid
metrics.energy_Grid.writer.class = de.irf.it.rmg.core.util.persistence.CSVFilePersistentStore
metrics.energy_Grid.writer.mayOverwrite = true
metrics.energy_Grid.writer.outputfile = 01
```

## A.4 Modificaciones al simulador

### de.irf.it.rmg.core.teikoku.Constants.java

```
//Site speed
final public static String CONFIGURATION_SITES_SPEED = "speed";

//Core power when it's working
final public static String CONFIGURATION_SITES_ENERGY_POWER_SITE = "energy.corePowerWorking";

//Energy efficiency
final public static String CONFIGURATION_SITES_ENERGY_EFFICIENCY_SITE = "energy.energyEfficiency";

//Site on power
final public static String CONFIGURATION_SITES_ENERGY_POWER_IDLE_SITE = "energy.sitePowerIdle";

//Power to turn on a site
final public static String CONFIGURATION_SITES_ENERGY_POWER_TURN_ON_SITE = "energy.sitePowerTurnOn";

//Waiting time while the site is turning on
final public static String CONFIGURATION_SITES_ENERGY_TURN_ON_DELAY_SITE = "energy.siteTurnOnDelay";

//Waiting time to turn off a site
final public static String CONFIGURATION_SITES_ENERGY_TURN_OFF_DELAY_SITE =
"energy.siteTurnOffDelay";

//Chassises in that site
final public static String CONFIGURATION_SITES_ENERGY_NUMBER_CHASSISES = "energy.numberChassises";

//Energy consumption by chassis
final public static String CONFIGURATION_SITES_ENERGY_POWER_IDLE_CHASSIS =
"energy.chassisPowerIdle";

//Boards in that site
final public static String CONFIGURATION_SITES_ENERGY_NUMBER_BOARDS = "energy.numberBoards";

//Energy consumption by board
final public static String CONFIGURATION_SITES_ENERGY_POWER_IDLE_BOARD = "energy.boardPowerIdle";

//Energy consumed by idle core
final public static String CONFIGURATION_SITES_ENERGY_POWER_IDLE_CORE = "energy.corePowerIdle";

//Energy consumed by working core
final public static String CONFIGURATION_SITES_ENERGY_POWER_WORKING_CORE =
"energy.corePowerWorking";

//Waiting time to turn off core
final public static String CONFIGURATION_SITES_ENERGY_TURN_OFF_DELAY_CORE =
"energy.coreTurnOffDelay";
```

### de.irf.it.rmg.core.teikoku.site.ComputeSiteEventHandler.java

```
@AcceptedEventType(value=JobQueuedEvent.class)
//When a job arrives, we see if there's enough resources on to process it.
int jobSize = ((SWFJob) job).getRequestedNumberOfProcessors();
int idleCores = this.site.getSiteEnergyManager().numberOfIdleOnCore();
int sub = jobSize - idleCores;

if(this.site.getSiteEnergyManager().isOn()) {
    if(idleCores >= jobSize) {
        ((AbstractScheduler) this.site.getScheduler()).activate();
else {
    this.site.getSiteEnergyManager().requestSiteTurnOn(event);
```

```
    //If not, we must turn them on
    for (int i = 1; i < sub; i++) {
        this.site.getSiteEnergyManager().requestCoreTurnOn(event);
    }

@AcceptedEventType(value=JobStartedEvent.class)
//When a job is queued, mark the resources assigned to it as “working”
this.site.getSiteEnergyManager().setWorkingCore(event.getTimestamp().timestamp(),
job.getResources());

@AcceptedEventType(value=JobCompletedEvent.class)
//When a job finishes, get job area (size*time) to calculate strategy energy efficiency.
long jobSize = ((SWFJob)completedJob).getRequestedNumberOfProcessors();
long jobLength = ((SWFJob)completedJob).getRunTime();
long work = (jobSize * jobLength);
```

## mx.cicese.mcc.teikoku.energy

```
//Se agregaron las siguientes clases y todo su contenido, para manejar el consumo energético del
//Grid en las simulaciones
AbstractEnergyManager.java
BoardEnergyManager.java
ChassisEnergyManager.java
CoreEnergyManager.java
EnergyConfiguration.java
GridEnergyManager.java
SiteEnergyManager.java
```