CALENDARIZACIÓN ENERGÉTICAMENTE EFICIENTE EN *MILLICLUSTERS*

EDSON ALEJANDRO FLÓREZ SUÁREZ
Ingeniero de Sistemas

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
MAESTRÍA EN INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2016

CALENDARIZACIÓN ENERGÉTICAMENTE EFICIENTE EN *MILLICLUSTERS*

EDSON ALEJANDRO FLÓREZ SUÁREZ
Ingeniero de Sistemas

Trabajo de Investigación para optar al título de
Magíster en Ingeniería de Sistemas e Informática

Director
PhD. CARLOS JAIME BARRIOS HERNÁNDEZ
Supercomputación y Cálculo Científico UIS
Bucaramanga, Colombia

Codirector
PhD. JOHNATAN E. PECERO SÁNCHEZ
Universidad de Luxemburgo
Luxemburgo, Luxemburgo

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
MAESTRÍA EN INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2016

DEDICATORIA

A Dios

A mis padres, Jaime y Luz Mary

A mis hermanos, Jaime Andrés y Mary

A toda mi familia

A mis amigos

Gracias por apoyarme para alcanzar esta meta

Edson Flórez

AGRADECIMIENTOS

A la Universidad Industrial de Santander y a la Escuela de Ingeniería de Sistemas e Informática, a la que está vinculado el grupo de investigación Supercomputación y Cálculo Científico (SC3), que me permitió desarrollar uno de sus trabajos de investigación.

A los directores del proyecto, PhD. Carlos Barrios y PhD. Johnatan Pecero, por sus valiosos aportes y orientaciones para llevar a cabo el trabajo de investigación.

Al grupo de investigación Parallel Computing and Optimization Group (PCOG) de la Universidad de Luxemburgo, dirigido por el Prof. Pascal Bouvry, donde se realizó una pasantía con el asesoramiento de PhD. Joseph Emeras, y el apoyo de PhD. Sébastien Varrette (administrador de la infraestructura HPC) en el acceso y uso del millicluster Viridis.

A mis compañeros y amigos de la universidad, por sus orientaciones durante el desarrollo del proyecto y el agradable tiempo compartido.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

TITULO: Calendarización Energéticamente Eficiente en *Milliclusters*[1]

AUTOR: Edson Alejandro Flórez Suárez[2]

PALABRAS CLAVE: Computación de alto rendimiento, heurísticas, calendarización, optimización combinatoria, eficiencia energética, *millicluster.*

El aumento de la necesidad de Computación de Alto Rendimiento ha llevado al incremento de la demanda de energía y el tamaño de los *datacenters*, y consecuentemente también lo hacen sus emisiones de carbono y costos de operación. Por esto, se deben desarrollar heurísticas adecuadas para el problema de calendarización (*scheduling*) de tareas, y se utiliza un clúster de procesadores de bajo consumo (*milliWatts*), que es energéticamente eficiente y ocupa poco espacio, conocido como *millicluster*. Para estimar el consumo de energía de los nodos del *millicluster*, se propone un modelo de energía basado en el uso de los recursos. El problema de calendarización de tareas es un problema de optimización combinatoria, que es abordado con el multi-objetivo de maximizar el rendimiento del sistema (minimización del *makespan*) y minimizar el consumo de energía, para satisfacer los requerimientos de los usuarios y del administrador del sistema. Los casos de estudio son instancias de calendarización de tareas de diferente consistencia y heterogeneidad de tareas y máquinas [8], que se basan en el modelo ETC [31]. El algoritmo de calendarización más destacado en las instancias más comunes del modelo ETC, es la metaheurística evolutiva *Parallel Micro CHC*.

---

[1] Trabajo de Investigación

[2] Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas e Informática. Director: Carlos Barrios. Codirector: Johnatan Pecero

ABSTRACT

TITLE: Energy Efficient Scheduling on Milliclusters[3]

AUTHOR: Edson Alejandro Flórez Suárez[4]

KEYWORDS: High Performance Computing, heuristics, scheduling, combinatorial optimization, energy efficient, millicluster.

The growing demand for High Performance Computing has led to the increased of energy demand and the datacenters size, therefore, its carbon emissions and operating costs also grow. Thus, appropriate heuristics must be developed for the job scheduling problem, and we use a cluster of low-power processors (milliWatts), which is energy efficient and takes less space, known as millicluster. To estimate the power consumption of millicluster nodes, an energy model is proposed based on use of resources. The job scheduling problem is a combinatorial optimization problem, which is approached with the multi-objective to maximize system performance (minimizing the makespan) and minimize energy consumption, to satisfy the requirements of users and system administrator. The study cases are job scheduling instances of different consistency and heterogeneity of tasks and machines [8], which are based on the ETC model [31]. The scheduling algorithm most outstanding is the evolutionary metaheuristic Parallel Micro CHC, in the most common instances of ETC model.

---

[3] Research Work

[4] Faculty of Physical-Mechanical Engineering, School of Systems Engineering and Informatic. Director: Carlos Barrios. Codirector: Johnatan Pecero

INTRODUCCIÓN

La planificación de recursos es transcendental en la Computación de Alto Rendimiento (HPC), porque debe decidir cómo utilizar los limitados recursos para ejecutar los *jobs* de los usuarios en el tiempo determinado. Por lo tanto, se requiere gestionar eficientemente (optimizar) el uso de recursos HPC, generando un plan (calendario) adecuado según el rendimiento, disponibilidad y otros criterios relacionados con la calidad del servicio (QoS) brindado al usuario y los requerimientos del administrador del sistema (costos de operación, consumo de energía). Los sistemas HPC (como las *grid* de computadores) son administrados con un sistema de gestión de recursos (RMS), que típicamente tiene un administrador de recursos y un calendarizador (*scheduler*), que utilizan algoritmos de *scheduling* inmediato (*on-line*), sin una auténtica planificación que permita aprovechar al máximo los recursos de los sistemas HPC.

Las plataformas HPC pueden estar centradas en el sistema o en el usuario. Cuando está centrada en el sistema, se optimiza el rendimiento general del clúster como el espacio de disco, intervalo de acceso y consumo de energía, sin tener en cuenta las necesidades del usuario. En cambio, cuando está centrado en el usuario se evalúan criterios relacionados con la utilidad lograda por los usuarios, como el costo y tiempo de ejecución de sus tareas, lo que puede degradar el rendimiento del sistema. El punto de equilibrio que permite cubrir los intereses del usuario y del administrador del sistema está en maximizar la utilización de recursos manteniendo buenos tiempos de respuesta para las peticiones de los usuarios [9]. Es decir, reducir el *makespan*[5] del calendario de tareas del sistema (indicador de rendimiento [2]), que a su vez permite reducir el *flowtime*[6] (indicador de la calidad del servicio

[5] *Makespan* es el lapso de tiempo transcurrido desde el inicio hasta el fin de un plan de trabajo

[6] *Flowtime* (tiempo de flujo) es el tiempo de procesamiento de las tareas de un plan de trabajo

QoS [9]), relacionado con el consumo de energía.

Para el desarrollo sostenible de plataformas HPC, los *milliclusters* surgen como una alternativa prometedora. Un *millicluster* es un centro de datos pequeño (*Datacenter-in-a-box*), de alta densidad y escalabilidad, conformado por *millicomputers* o procesadores de bajo consumo (por ejemplo, procesadores ARM [1]), que se utilizan principalmente en dispositivos móviles como tabletas y teléfonos inteligentes. Su consumo de energía se puede obtener con un modelo teórico, basado en el nivel de trabajo del procesador [7].

1. PRESENTACIÓN DEL PROYECTO

1.1 JUSTIFICACIÓN

La eficiencia energética es un aspecto relevante tanto en arquitecturas de computación de propósito general, como en arquitecturas un poco más complejas. En el caso de los centros de recursos de Computación de Alto Rendimiento (HPC), un problema de gran interés actualmente es el alto consumo de energía, que afecta el desarrollo sostenible de estas Tecnologías de la Información y la Comunicación (TIC), por lo que se ha estado buscando su reducción desde diferentes enfoques. Una limitación común en los esfuerzos para hacer frente a problemas críticos, es que los requerimientos de simulación y análisis de datos científicos están superando las capacidades *petascale* de la computación de alto rendimiento actual, por lo que se necesita un aumento de mil veces en la potencia de cálculo utilizable, es decir, computación *exascale.* Sin embargo, debido a las limitaciones tecnológicas proyectadas, los enfoques de diseño de software y hardware HPC tendrán que ser lo más energéticamente eficiente posibles.

La Lista Green500 que calcula la eficiencia energética de los 500 supercomputadores más poderosos del mundo (Lista Top500), en términos de rendimiento (MFLOPS) y potencia (kW), tuvo en primer lugar en Noviembre de 2015 a un supercomputador que requiere una potencia de 50.32 kW para alcanzar su máximo rendimiento, mientras que el supercomputador más poderoso requiere 17808 kW, esto refleja el alto consumo de energía de las infraestructuras de HPC. El consumo de energía de los numerosos procesadores *multicore* de los *datacenters* es costoso, además de requerir de refrigeración para disipar el calor producido, lo que ha llevado al surgimiento de una nueva infraestructura computacional conocida como *millicluster* [1]. Los procesadores de los *milliclusters* son reconocidos como tecnologías verdes al consumir poca energía (en *milliwatts* o milivatios) y no demandar refrigeración, también ocupan menos espacio lo que permite reducir los

costos de operación de los *datacenters*.

Sin embargo, sus procesadores de menor capacidad de procesamiento no pueden ejecutar las aplicaciones con un rendimiento equivalente al de un clúster convencional [1], por lo que requiere de un algoritmo de calendarización de tareas adecuado para atender satisfactoriamente los *jobs* de los usuarios, generando un calendario (*schedule*) de la asignación de los *jobs* en los procesadores del *millicluster*, que a su vez considere la eficiencia energética de la calendarización. Los algoritmos que encuentran soluciones para este problema de calendarización de tareas, han sido implementados principalmente en clústeres y *grids*, por lo cual se debe hacer un análisis de su desempeño con tareas de *millicluster*.

1.2 PLANTEAMIENTO DEL PROBLEMA

En el problema de calendarización de trabajos en los recursos del *millicluster*, los trabajos o tareas son unidades de *workload* [9]. El *scheduler* del *millicluster* determina la fecha de inicio de la ejecución de cada tarea, buscando encontrar el calendario más adecuado, para procesar los trabajos rápidamente en los procesadores. La calendarización de tareas es un problema de optimización combinatoria, multi-objetivo en este caso porque se pretende reducir el consumo de energía sin detrimento del rendimiento de la plataforma, para lo cual se debe proponer un modelo de energía, que permita estimar el consumo de energía de tareas ejecutadas en los nodos *millicluster.*

Las instancias del problema se basan en el modelo ETC (*Expected Time to Compute*) [2], muy utilizado en la literatura de HPC. ETC es el tiempo estimado para procesar una tarea en una máquina determinada, calculado a partir del *workload* de las tareas (que es la estimación o predicción de carga computacional de cada tarea [2]), definida en millones de instrucciones, y la capacidad de computación de las máquinas, especificada en millones de instrucciones por segundo (MIPS). El

calendario o plan para maximizar la eficiencia energética y el rendimiento del sistema, se evalúa con indicadores como *makespan* y/o *flowtime* para el rendimiento [3], y el consumo de energía consumida para calcular la eficiencia energética. Generalmente el *scheduling* en clúster se enfoca principalmente en el *makespan,* y se tiene en cuenta el *flowtime* para garantizar un buen nivel en la calidad del servicio prestado a los usuarios. Al reducir el *makespan* del calendario también se puede reducir el *flowtime* de los trabajos.

1.3 OBJETIVOS

1.3.1 Objetivo general

Evaluar el rendimiento y la eficiencia energética de una infraestructura computacional *millicluster*, a partir de algoritmos de calendarización de tareas.

1.3.2 Objetivos específicos

1. Identificar las heurísticas que presentan mejor rendimiento y eficiencia energética para el problema de calendarización de tareas en infraestructuras de Computación de Alto Rendimiento (HPC).

2. Diseñar y desarrollar los algoritmos de calendarización más eficientes identificados, para la gestión de tareas de una infraestructura computacional *millicluster*.

3. Evaluar el rendimiento de los algoritmos de calendarización de tareas desarrollados, con otros algoritmos relacionados del estado del arte.

4. Proponer un modelo de energía para determinar la eficiencia energética de algoritmos de calendarización de tareas en *milliclusters*.

1.4 CONTRIBUCIONES

Los clústeres de Computación de Alto Rendimiento deben manejar la concurrencia de los diversos trabajos de numerosos usuarios, haciendo necesario el desarrollo de nuevas estrategias para gestión de colas y asignación de recursos a los trabajos. La comunidad científica ha abordado desde diferentes enfoques este problema para mejorar la calidad del servicio (reduciendo tiempos de ejecución, tiempos en cola, etc.), y a la vez reducir los costos de operación de estas infraestructuras (consumo de energía, refrigeración, espacio requerido, etc.), siendo uno de los enfoques el diseño e implementación de metaheurísticas para la asignación planificada de recursos. Por lo tanto, este proyecto es un aporte al estado del arte de las metaheurísticas, centrado en el análisis de rendimiento y consumo de energía en *clusters* de bajo consumo (*milliclusters*), en un ámbito que requiere el uso eficiente de los recursos.

1.4.1 Resultados o productos obtenidos

- **Relacionados con la generación de conocimiento y/o nuevos desarrollos tecnológicos:**

-Estudio de los algoritmos del estado del arte de *job scheduling* en HPC, presentado en el artículo “METHODS FOR JOB SCHEDULING ON COMPUTATIONAL GRIDS: REVIEW AND COMPARISON”, publicado en la revista especializada *Communications in Computer and Information Science* (CCIS 565), High Performance Computing, pp. 19–33 (2015), indexada y homologada por COLCIENCIAS en la categoría A2. Este artículo permite comparar muchos algoritmos en un mismo contexto, facilitando a los nuevos investigadores la identificación de los mejores métodos actualmente, para implementar sus estrategias de búsqueda en los problemas de optimización combinatoria relacionados.

-Modelo de energía basado en datos experimentales recopilados en la ejecución de *benchmarks* en el *millicluster* Viridis. Este facilita el análisis del consumo de energía del *millicluster* con diversas aplicaciones, lo que podría ayudar a la inclusión de *milliclusters* en plataformas de HPC, que actualmente tienen como gran limitante el alto consumo de energía.

-Algoritmo evolutivo *Parallel Micro CHC* para *job scheduling*, adaptado para optimizar rendimiento y consumo de energía simultáneamente.

-Publicación de artículo en revista de Computación de Alto Rendimiento, que se está culminando actualmente, sobre el modelo de energía para *millicluster* propuesto y el algoritmo de *scheduling* implementado.

- **Conducentes al fortalecimiento de la capacidad científica nacional:**

-Cooperación científica internacional con la Universidad de Luxemburgo (Luxemburgo), en la co-dirección del trabajo de investigación por parte de PhD. Johnatan Pecero, y realización de pasantía (de Marzo 6 a Junio 3 de 2015) en el grupo de investigación *Parallel Computing and Optimization Group* (PCOG) de la Universidad de Luxemburgo, dirigido por el Prof. Pascal Bouvry, con el asesoramiento de PhD. Joseph Emeras y el apoyo de PhD. Sébastien Varrette (administrador del departamento HPC de la Universidad de Luxemburgo), para el desarrollo de uno de los objetivos del trabajo de investigación que requirió el uso del *millicluster* Viridis. Además la facilitación de algoritmos para probar su desempeño con instancias del *benchmark* de Braun et al. [8].

-Trabajo de investigación que aporta al avance del país en una línea de investigación de gran proyección, computación de alto rendimiento (HPC), y que promueve los *milliclusters* como nuevo paradigma para el desarrollo sostenible de las plataformas de HPC.

- **Dirigidos a la apropiación social del conocimiento:**

-Ponencia “METHODS FOR JOB SCHEDULING ON COMPUTATIONAL GRIDS: REVIEW AND COMPARISON” en Conferencia Latinoamericana de *High Performance Computing* (CARLA 2015), realizada en Petrópolis (Brasil), entre el 26 y 28 de Agosto de 2015.

Los algoritmos de calendarización de tareas en HPC deben cumplir con las restricciones del problema para generar calendarios válidos, y buscar optimizar los objetivos en los casos de estudio que se presentan de manera formal en el siguiente capítulo.

2. MARCO TEORICO

Los problemas de planificación de recursos se presentan en múltiples actividades humanas, desde la sencilla organización que hace una persona (de su tiempo) en su agenda semanal, hasta problemas complejos como la generación de horarios de estudiantes en una universidad. Tiene aplicaciones en labores que requieren de una gestión optima del tiempo (con estrictas fechas de entrega), como en la programación para la entrega de paquetes, la planificación de los despegues y aterrizajes de aviones a través de las pistas de un aeropuerto, envío de datos en una red de computadores, computación (multitarea y multiprocesamiento), gestión de proyectos (plan o cronograma), procesos de producción y administración (en líneas de ensamblaje por ejemplo).

2.1 PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA

La optimización combinatoria es una rama de la optimización matemática, la cual es un área muy activa de las matemáticas aplicadas relacionadas con la Investigación de Operaciones (I.O.). El propósito de este estudio es obtener la mejor solución factible de un problema, tomando las decisiones más adecuadas para maximizar o minimizar la función objetivo según el problema, por ejemplo, aumentar la eficiencia en la ejecución de un proyecto o reducir los costos de producción de una empresa.

Este propósito está presente especialmente en actividades económicas, administrativas y militares, que buscan incrementar ganancias, rendimiento, y/o disminuir perdidas, riesgos, costos, etc. “La optimización combinatoria trata una clase de problemas en los que el número de soluciones candidatas es de tamaño combinatorio. Cada posible solución tiene un costo asociado y el objetivo consiste

en encontrar la solución con el menor costo" [50], donde las variables son de naturaleza discreta y solo pueden tomar valores dentro de un conjunto de soluciones factibles que también es discreto. Si fueran variables continuas como en Optimización Numérica, las soluciones factibles podrían ser infinitas y estarían dentro de una región limitada por las restricciones. Para los problemas de optimización, la Investigación de Operaciones genera modelos de donde se puede obtener la configuración óptima de un sistema, para toda área en que la toma de decisiones no puede depender exclusivamente de la experiencia de las personas.

Encontrar la mejor solución posible para un problema de optimización combinatoria se vuelve muy difícil cuando la función objetivo depende de muchas variables de decisión, las cuales pueden tomar cualquier valor perteneciente a un espacio de búsqueda, que es limitado por un conjunto de restricciones a las que está sujeto el problema a resolver. En un problema de *scheduling* de gran tamaño es imposible realizar todas las combinaciones de las soluciones factibles en un tiempo razonable (polinómico) [18]. La complejidad del problema de *job scheduling* (*NP-hard* [2]) no permite determinar la solución óptima, así que los mejores algoritmos ejecutados en las computadoras actuales solo pueden obtener soluciones aproximadas, que podrían ser la mejor solución, aunque no se puede tener certeza de esto sin explorar por completo el enorme espacio de búsqueda.

En un problema de optimización combinatoria se establece:
La función objetivo a optimizar: minimizar[7] $f: S \rightarrow \mathbb{R}_0^+$

- Ω Un conjunto de restricciones entre las variables definido por el problema a solucionar.
- *S* El espacio de búsqueda definida sobre un conjunto finito de variables de decisión discretas.

[7] Maximizar sobre *f* es lo mismo que minimizar sobre *–f*, por consiguiente, cualquier problema de optimización combinatoria puede ser descrito como un problema de minimización.

El espacio de búsqueda (o dominio) *S* se define como un conjunto de variables discretas $X_i,\ i = 1, \dots, n$, en donde $X_i \leftarrow v_i^j$ y los valores $v_i^j \in D_i = \{v_i^1, \dots, v_i^{|D_i|}\}$. Los elementos de *S* son asignaciones completas, es decir, asignaciones en las que cada variable X_i tiene un valor asignado v_i^j de su dominio D_i. El conjunto de soluciones factibles S_Ω está dado por los elementos de *S* que satisfacen todas las restricciones en el conjunto Ω. Una solución $s^* \in S_\Omega$ se denomina óptimo global si y sólo si $f(s^*) \leq f(s)\ \forall_s \in S_\Omega$.
El conjunto de todas las soluciones óptimas a nivel global se denota por $S_\Omega^* \in S_\Omega$. Resolver un problema de optimización combinatoria exige encontrar al menos una $s^* \in S_\Omega^*$.

Los problemas de optimización combinatoria se clasifican en varias grupos [57], entre los cuales están los problemas de asignación, como el Problema de Asignación Cuadrática (QAP) y el problema de organización de cursos de una universidad. También se tienen problemas de enrutamiento como el Problema de Ruteo de Vehículos (VRP). Dentro de los problemas de calendarización (o programación de trabajos) se encuentran problemas como el *Job Shop Scheduling Problem* (JSP), *Flow Shop Problem* (FSP) y *Open Shop Problem* (OSP).

2.2 DEFINICIÓN FORMAL DEL PROBLEMA DE *JOB SCHEDULING*

Un *job* puede ser una aplicación monolítica (es decir, una sola tarea) o estar compuesto de varias tareas, que pueden ser dependientes o independientes [14]. Los trabajos dependientes tienen una secuencia de tareas predefinida para cumplir dependencias entre ellas, y los trabajos independientes se pueden ejecutar simultáneamente. La intercomunicación entre tareas de un *job* se realiza con

modelos de programación en paralelo, por ejemplo, MPI[8], empezando todas sus tareas al mismo tiempo en diferentes procesadores del clúster. Una aplicación tiene tareas secuenciales y/o paralelas, pero no es completamente paralela. En nuestro caso de estudio, se tienen trabajos independientes (enviadas por los usuarios) que deben ser asignados en los nodos del clúster y procesadas de forma secuencial.

Un clúster está compuesto de $T = \{t_1, t_2, t_3, \dots, t_n\}$, un conjunto de t trabajos independientes para ser asignadas y $M = \{m_1, m_2, m_3, \dots, m_m\}$, un conjunto de *m* máquinas disponibles para la planificación, con un tiempo de ejecución $\mathrm{ETC}[t_i][m_i]$ previamente conocido [2], de cada trabajo en cada máquina $\tau_{jk} = \{\tau_{j1}, \tau_{j2}, \tau_{j3}, \dots, \tau_{jk_i}\}$, que representa las características de los trabajos y máquinas, como la capacidad de computo de las máquinas y el *workload* de los trabajos. El *workload* de aplicaciones reales se puede obtener de especificaciones proveídas por el usuario, de datos históricos o predicciones [20].

Función objetivo: Minimizar el consumo de energía E y *makespan* C_{max}:
-Minimización del consumo de energía según el modelo de energía propuesto.
-Minimización del *makespan*:

$$C_{max} = \max_{i \in tarea} (C_i) \qquad (1)$$

El *makespan* es el máximo tiempo de finalización C_i (*completion time*) de los trabajos [16], [17], que es el tiempo cuando termina la último trabajo, es decir, el tiempo de finalización (o longitud) del calendario [6]. La minimización del *makespan* también puede reducir el *flowtime* total (FT), que es la suma de los tiempos de

[8] MPI por sus siglas en inglés de *Message Passing Interface*, que significa Interfaz de Paso de Mensajes.
www.mpi-forum.org

ejecución de todos los trabajos:

$$FT = \sum_{i \in tarea} C_i \qquad (2)$$

- **Restricciones**: El problema está sujeto a restricciones que aseguran que cada trabajo sea asignado una sola vez [6], [19], y procesada en su totalidad en una única máquina sin ser interrumpida, y una máquina solo puede procesar un trabajo a la vez. Además el problema está sujeto a la restricción de tiempos de inicio de cada operación ($t_{ik} \geq 0$), y se define que los trabajos tienen la misma prioridad para ser asignadas.

Casos de estudio: las Instancias del problema son matrices ETC de tamaño $t\ x\ m$ [17]. El modelo ETC es caracterizado por los parámetros consistencia, heterogeneidad de las máquinas y heterogeneidad de las tareas [31]:

-heterogeneidad de la máquina: es el grado de variación de los tiempos de ejecución para una tarea determinada a través de todas las máquinas [9]. Alta heterogeneidad de la máquina representa los sistemas de computación integrados por recursos de diferente tipo y capacidad; baja heterogeneidad de la máquina se presenta en sistemas que poseen recursos de computación similares o casi homogéneos.

-heterogeneidad de la tarea: es el grado de variación entre los tiempos de ejecución de todas las tareas en una máquina determinada [9]. Alta heterogeneidad de la tarea son casos donde los requerimientos computacionales de las tareas varían ampliamente, porque se tienen diferentes tipos de aplicaciones, desde simples a complejos programas que requieren bastante tiempo de procesamiento. Baja heterogeneidad de la tarea se da cuando las tareas de los usuarios son casi homogéneas, es decir, tareas con requerimientos computacionales y complejidad similares, tendrán tiempos de ejecución similares en una máquina determinada.

En estas instancias definidas mediante una matriz cada fila corresponde a una tarea ($t_1, t_2, t_3, \dots, t_n$), y se muestra el tiempo de procesamiento estimado de cada una en las máquinas ($m_1, m_2, m_3, \dots, m_m$). La variación a lo largo de una fila de la matriz ETC es la heterogeneidad de la máquina (Ver tabla 1.b) y la variación a lo largo de una columna es la heterogeneidad de la tarea (Ver tabla 1.a) [31].

Tabla 1: Alta heterogeneidad de (a) tareas y (b) máquinas en matriz ETC de tamaño 10 *x* 5

(a)

	m_1	m_2	m_3	m_4	m_5
t_1	11648,23	11803,25	13198,95	14208,43	15309,41
t_2	12826,31	13439,28	13326,27	15145,01	15323,84
t_3	10394,73	10543,99	10629,78	12025,45	14339,22
t_4	508,99	561,11	567,35	766,93	858,48
t_5	5084,65	5288,79	5872,92	6503,83	7001,72
t_6	1808,62	1869,03	1936,83	1987,72	2229,49
t_7	877,99	901,28	956,57	1039,97	1044,57
t_8	5331,69	5858,57	6379,28	6985,41	7339,16
t_9	25250,93	25747,22	25785,37	26322,56	26332,69
t_{10}	3905,32	4012,28	4016,58	4511,21	4521,13

(b)

	m_1	m_2	m_3	m_4	m_5
t_1	896,03	1033,62	3276,71	16061,46	25993,39
t_2	913,99	1573,82	2928,01	18939,34	27081,67
t_3	802,42	1220,04	2489,74	17588,49	25076,18
t_4	764,43	1389,37	2733,12	17863,56	27848,96
t_5	987,75	1524,07	3622,65	16750,89	24889,36
t_6	658,35	1379,73	2940,43	16916,91	23134,42
t_7	844,28	1437,73	2571,79	14899,55	25771,68
t_8	702,05	1504,82	2955,61	17555,64	25156,15
t_9	642,51	1053,21	3156,67	15995,97	26244,13
t_{10}	866,42	1589,23	2233,13	15738,73	26766,26

La matriz ETC también es definida por su consistencia dada por la relación entre tareas y como estas son ejecutadas en la máquina de acuerdo a la heterogeneidad de cada una [9]:

-Consistencia: si una máquina m_j ejecuta la tarea t_i más rápido que la máquina m_k, luego la máquina m_j ejecuta todas las tareas más rápido que m_k.

-Inconsistencia: es el caso más común, donde se recibe diferentes tareas, desde aplicaciones sencillas hasta programas paralelos complejos [9]. Este escenario representa un caso donde una máquina es más rápida para algunas tareas y más lenta para algunas otras.

-Semi-consistente: es una matriz inconsistente que contiene una sub-matriz consistente (de un tamaño predefinido). Esta sub-matriz puede estar compuesta de cualquier subconjunto de filas y cualquier subconjunto de columnas. Como un ejemplo, en una matriz parcialmente consistente, 50% de las tareas y 25% de las máquinas pueden definir una sub-matriz consistente [31].

Las instancias son etiquetadas con x_ttmm [17], donde x indica el tipo de consistencia de la matriz (c para consistente, s para semi-consistente, i para inconsistente), tt indica la heterogeneidad de las tareas y mm indica la heterogeneidad de las máquinas. Para la heterogeneidad, “hi” significa alto (*high*), y “lo” significa bajo (*low*). Con estas características de heterogeneidad y consistencia se obtiene doce combinaciones (Ver tabla 2).

Tabla 2: Clases de instancias ETC

Consistencia		
Consistente	Semi-consistente	Inconsistente
c_lolo	s_lolo	i_lolo
c_lohi	s_lohi	i_lohi
c_hilo	s_hilo	i_hilo
c_hihi	s_hihi	i_hihi

Cada grupo de tareas ejecutadas en una máquina obtiene el mismo makespan (para esa máquina) sin importar el orden de ejecución de las tareas, por lo que si se realizan combinaciones con *t* tareas y *m* máquinas se tiene m^t soluciones posibles, que corresponde al espacio de búsqueda (S) del problema. Para un problema de 512 tareas y 16 máquinas se tienen $16^{512} = 3{,}23 * 10^{616}$ soluciones posibles, un número enorme de combinaciones si se compara con la edad del universo estimada en segundos (comprendida entre 13761 y 13835 millones de años), que es aproximadamente:

$(14 * 10^9)$ $años * 365.25$ $días * 24$ $horas * 60$ $minutos * 60$ $segundos = 4.418 * 10^{17}$ seg.

2.2.1 Representación de la solución

El cumplimiento de las restricciones del problema se puede ver mediante un diagrama de Gantt (Ver Figura 1), que representa una de las soluciones factibles para la instancia de ejemplo de la Tabla 3. En el diagrama se muestra el tiempo de ejecución (eje horizontal) de las tareas en las máquinas (eje vertical), cada máquina con una fila (o renglón) correspondiente, donde se pueden ver las operaciones realizadas en ellas, por ejemplo, se puede ver que en el intervalo [50, 70] de la máquina 3 se realizó la tarea t_2, es decir la operación O_{23}, que es la última tarea en terminar su ejecución, estableciendo el makespan en 70 unidades de tiempo. Utilizando heurísticas como Min Min y Max Min se encuentran mejores soluciones, con makespan de 32 y 30 respectivamente.

Tabla 3: Instancia de ejemplo de 9 tareas x 3 máquinas

	m_1	m_2	m_3
t_1	1	10	19
t_2	2	11	20
t_3	3	12	21
t_4	4	13	22
t_5	5	14	23
t_6	6	15	24
t_7	7	16	25
t_8	8	17	26
t_9	9	18	27

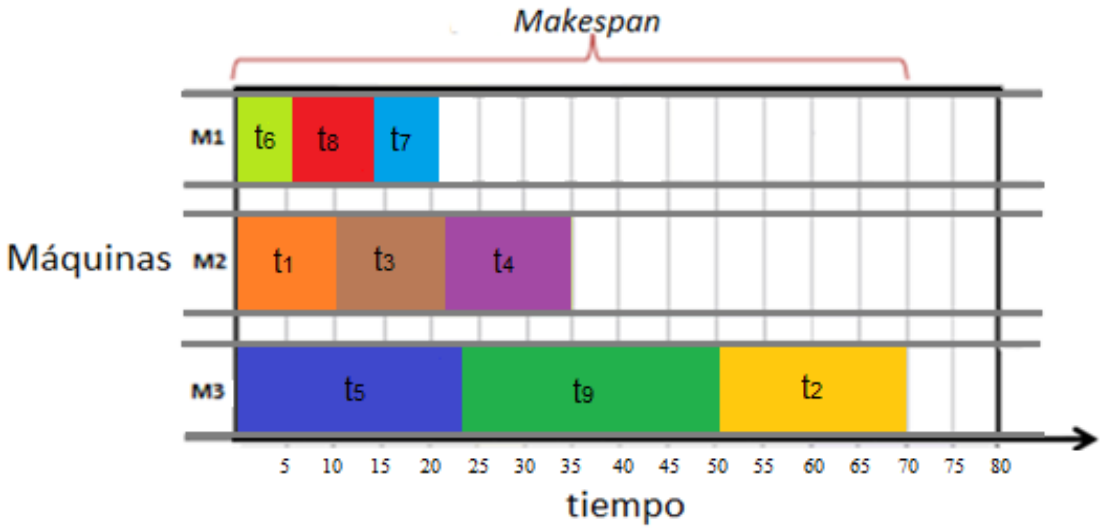

Figura 1: Diagrama de Gantt de un calendario de trabajos factible

Se puede obtener la mejor solución para esa instancia si se ubican la mayoría de tareas en la máquina 1, donde todas las tareas tienen el menor tiempo de ejecución (Ver Figura 2), así se logra obtener la solución óptima para la instancia de la Tabla 3, que tiene el menor makespan posible de 27 unidades de tiempo. Para asegurar que esta es la solución óptima, se deben realizar todas las combinaciones posibles, algo que es muy difícil para instancias de gran tamaño. En este pequeño ejemplo de 9 tareas y 3 máquinas se tienen $3^9 = 19683$ soluciones posibles.

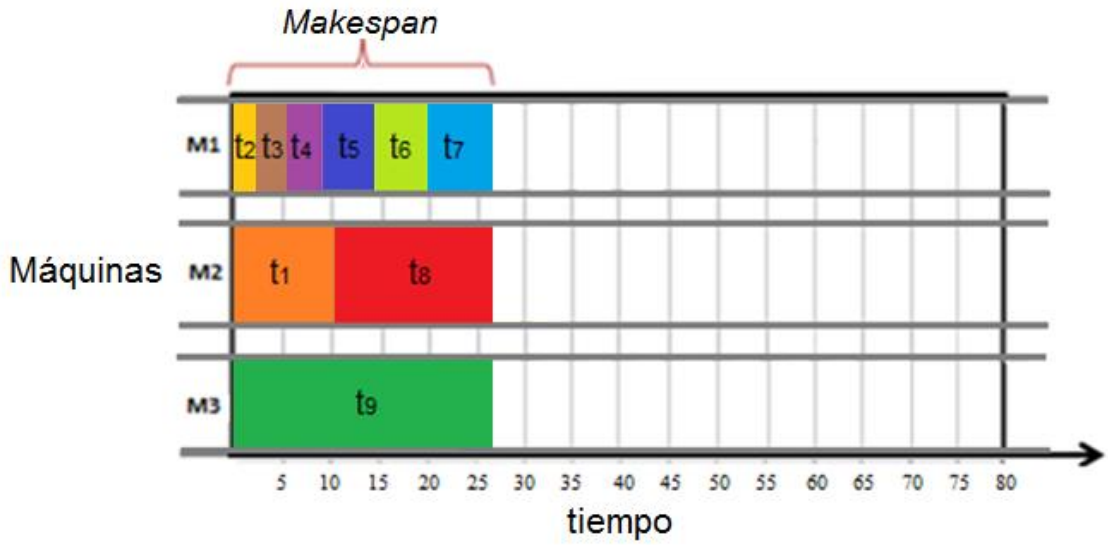

Figura 2: Diagrama de Gantt del calendario de trabajos óptimo.

2.3 POLÍTICAS DE *SCHEDULING*

El escenario de *scheduling* puede ser dinámico cuando modifica el calendario *on-line* [9], [11], agregando o quitando procesadores a un *job*, o estático, cuando el calendario se genera antes de empezar su ejecución y no se modifica durante la ejecución, es decir, es *off-line* porque todos los *jobs* son conocidos y están disponibles al inicio [6]. Y el procesamiento del *job* puede ser de tipo *batch* (lote) o inmediato [20]; el *scheduling* inmediato produce un *makespan* más largo que el *batch* porque asigna *jobs* individualmente, sin tener en cuenta los *jobs* que llegan después. En nuestro caso, el *scheduling* es tipo *batch* estático, donde las tareas se agrupan en lotes (como una bolsa de tareas) para luego ser periódicamente asignadas a los recursos [20].

Los sistemas distribuidos de HPC son administrados por el Sistema de Gestión de

Recursos (RMS, por sus siglas en inglés) [14], como OAR[9], SLURM (*Simple Linux Utility for Resource Management*), PBS (*Portable Batch System*), LSF (*Load Sharing Facility*) y Condor, que contienen el *job scheduler* encargado de la gestión de los *jobs* de los múltiples usuarios. Estos generalmente realizan *scheduling* inmediato y sus políticas de *scheduling* pueden tener en cuenta diferentes criterios como la prioridad del *job*, la hora de llegada del *job* o su tiempo de procesamiento (tamaño) [6]. Si se sigue la regla FCFS (*First Come, First Serve*, primero en llegar, primero en ser atendido), cada *job* se coloca de ultimo en la cola y son asignados cuando los recursos solicitados estén disponibles.

Entre otras reglas que pueden ser parte de las políticas de *scheduling* se encuentran, *backfilling* (relleno) que mueve *jobs* adelante en el calendario para aprovechar los lapsos de tiempo vacíos (ociosos) de las máquinas, y *fairness* (justicia) que busca asignar los recursos de manera equilibrada entre los usuarios, con tiempos de computación similares para cada *job*, acordes con su prioridad y *workload*. Otra reglas son el algoritmo *Shortest Job First* (SJF, atender de primero al trabajo más corto) que genera una cola de tareas con tiempo de procesamiento creciente, o su opuesto, el algoritmo *Longest Job First* (LJF, atender de primero al trabajo más largo) [5]. Para clústeres con poca concurrencia que tendrán buena disponibilidad de recursos, un algoritmo FCFS de *scheduling* inmediato es más adecuado, porque asignará los *jobs* al momento de ser enviados por el usuario, sin ningún retardo.

2.4 MODELO DE ENERGÍA

Los modelos de energía para plataformas HPC pueden realizarse al nivel de componente, máquina o sistema distribuido [51], mediante *benchmarks* y

[9] https://oar.imag.fr

mediciones de consumo de energía, mediciones que se pueden hacer usando software para registrar las lecturas de los sensores del sistema, o dispositivos de medición colocados directamente en el conector eléctrico. En el trabajo [52] se estudió la variación de la energía a nivel de componentes (CPU, memoria, almacenamiento, red, PCI *slots*, etc.), con la ejecución de un *workload* conformado por los *benchmarks* Stream, Matrix, SPEC (CPU int y fp, jbb, web). El modelo se derivó usando programación lineal sobre los datos recopilados, para obtener los parámetros de las métricas utilización de CPU, conteo de accesos a memoria *Off-chip*, velocidad (*rate*) de I/O de disco y red. El modelado por componentes tiene limitaciones porque modela el consumo de energía de un sistema complejo solo como la suma del consumo de cada subsistema [51]. Basmadjian y de Meer presenta un modelo basado en componentes de procesadores *multi-core* [53], resaltando las diferencias entre el modelado del consumo de energía del sistema *multi-core* (hasta *Quad-core*) y *single-core* (un solo núcleo), demostrando que el consumo de procesadores *multi-core* no se puede reducir a simple suma del consumo de los *cores*.

El nivel de máquina, se trata de adaptar un modelo al consumo de todo el nodo, se evalúa con diversos tipos de *workload* para CPU, memoria, red, I/O [51]. En los modelos al nivel de máquina, algunos trabajos muestran que la energía del sistema varía casi linealmente con la utilización de CPU.

En sistemas distribuidos el monitoreo del consumo de energía no se hace solamente de componentes locales, incluye componentes de la red como *routers*, *switches*, PDU (*Power Distribution Unit*), aire acondicionado, luces, etc. [51]. Allí el modelado es muy complejo por la heterogeneidad de los nodos y redes, y la configuración del software en ellos, por lo que se debe hacer mediciones directamente en el conector eléctrico.

Los modelos se pueden elaborar con técnicas como árboles de decisión, para

predecir con mayor precisión el consumo de energía de cualquier aplicación de la amplia gama de aplicaciones de HPC, con esta técnica se selecciona automáticamente el modelo más apropiado al *workload* actual [56]. El modelo de regresión lineal se utiliza frecuentemente para modelar el consumo de energía, la regresión lineal multi-variable se utiliza para construir modelos con las métricas de múltiples componentes [52, 54, 55].

La energía consumida en un intervalo de tiempo por los procesadores de un nodo, se puede definir de la siguiente forma [2]:

$$E = P * \sum_{i=1}^{m} CT_i \qquad (3)$$

El consumo de energía está en función de la potencia P de los procesadores (vatios) y del tiempo que están en funcionamiento CT (*Completion Time*). Por lo tanto, reducir la energía implica ejecutar las tareas en procesadores más rápidos para tener tiempos de finalización cortos, lo que a su vez reduce el *flowtime* de las tareas. El Sistema Operativo de los procesadores puede autorregular el voltaje suministrado y la frecuencia de reloj del procesador de forma dinámica con técnicas como DVFS (*Dynamic Voltage Frequency Scaling*) [13], para ahorrar energía y producir menos calor. Esto se representa con valores discretos de potencia, que depende de la frecuencia y voltaje suministrado [40], y sus niveles se ajustan a una potencia mínima cuando está ocioso (*idle*), y cambia a la potencia máxima cuando está procesando una tarea.

En el trabajo de Pinel y Bouvry [7] se planteó un modelo de energía teórico (Ver Ecuación 4), que está en función de un término de potencia constante (BL), el número N de máquinas encendidas (una máquina que no tiene tareas asignadas es

considerada apagada) [7], y la potencia P_{high} y P_{low} de la CPU cuando funciona al máximo y mínimo voltaje (y frecuencia) respectivamente.

$$E = BL * N * C_{max} + \sum_{i=1}^{m} (P_{high} * CT_i + P_{low} * (C_{max} - CT_i)) \qquad (4)$$

Mientras que la potencia en estado activo de los procesadores de un clúster puede ser de 200 W, la potencia de un *millicomputer* es 5 W [2], aunque son 15 veces más lentos que el procesador más lento del clúster.

La complejidad del problema de *job scheduling* (*NP-hard* [2]) exige abordarla con métodos aproximados, que encuentran soluciones de buena calidad (de bajo consumo de energía y makespan) con gran eficiencia computacional, en lugar de algoritmos que realizan búsquedas exhaustivas que generan un alto costo en tiempo y recursos computacionales. Entre ellos se encuentran diversos métodos heurísticos de Inteligencia Artificial, como la heurística Min-Min [1], [6], que es uno de los algoritmos más usados en la literatura como base de comparación [9], y técnicas más elaboradas como la metaheurística Algoritmo Genético (GA)[10] [2].

[10] En una *grid* de computación que se extendió con un *millicluster*

3. ESTADO DEL ARTE

3.1 MÉTODOS DE SOLUCIÓN DEL PROBLEMA DE *JOB SCHEDULING*

Los algoritmos que encuentran soluciones a problemas de *Job Scheduling* se clasifican en métodos exactos y aproximados. Los métodos exactos buscan determinar la solución óptima, para lo cual realizan búsquedas exhaustivas de escasa eficiencia computacional, por tanto, para problemas de gran tamaño, se deben usar los métodos aproximados, que implementan diversas estrategias para no tener que explorar todo el espacio de búsqueda.

3.1.1 Métodos exactos

Vuelta atrás *(Backtracking):* realiza una búsqueda en profundidad dentro del grafo dirigido para encontrar soluciones al problema, por lo que va generando un árbol de búsqueda en el que construye soluciones parciales que limitan las regiones en las que se puede encontrar un estado de aceptación. Si no puede encontrar una solución, retrocede eliminando los elementos que se hubieran añadido en cada paso. Cuando llega a un nodo con algún nodo vecino sin explorar, reanuda la búsqueda de una solución.

Ramificación y poda *(B&B, Branch and Bound):* propuesto por Land y Doig en 1960 [49], es una variante del *Backtracking*. Además de ramificar (*branch*) una solución padre en hijos, se trata de eliminar de consideración los hijos cuyos descendientes tienen un costo que supera al óptimo, acotando (*bound*) el costo de los descendientes del hijo. Esta “poda” reduce el tiempo de búsqueda, cortando las ramas del árbol que tienen descendientes costosos.

3.1.2 Métodos aproximados

Encuentran una buena aproximación al valor óptimo de problemas con un espacio de búsqueda muy amplio. Los métodos aproximados difieren básicamente en la

manera como exploran del espacio de búsqueda de soluciones, mientras que unos van construyendo incrementalmente una sola solución, otros realizan iteraciones en las que evalúan múltiples soluciones.

3.1.2.1 Heurísticas de *Job Scheduling*

Algunos algoritmos heurísticos generan una solución desde cero mediante la adición de componentes a la solución parcial, paso a paso, de un modo sencillo y rápido de acuerdo a una regla de transición y una función de costo, hasta que se completa la solución. El problema de calendarización de tareas presente en *clusters* se ha resuelto con heurísticas de baja complejidad, que consumen menos tiempo y memoria para generar un calendario. Una heurística muy conocida es Min-Min, la cual inicia con un conjunto de todas las tareas no asignadas, y luego establece el tiempo de finalización mínimo esperado para toda tarea del conjunto, de donde selecciona la tarea con el menor tiempo de finalización mínimo esperado y asigna a la máquina correspondiente; después se elimina la tarea asignada del conjunto y se vuelve a repetir el procedimiento anterior hasta mapear todas las máquinas [8]. Max-Min funciona de la misma forma que Min-Min, pero según el máximo tiempo de finalización esperado.

En un trabajo reciente, Diaz et al. [17] comparan Min-Min con las heurísticas Max-Max-Min, Avg-Max-Min y Min-Max-Min en sistemas de computación heterogéneos, e implementaron la técnica *Task Priority Diagram* (TPD). TPD define un grafo para establecer la precedencia de las tareas en base al valor de ETC, por medio de un diagrama de Hasse. De acuerdo a la métrica makespan, las heurísticas de baja complejidad fueron mejores en los escenarios inconsistentes y semi-consistentes, en los escenarios consistentes la heurística basada en TPD fue mejor. Díaz et al. [15] compararon Min-Min con los algoritmos Min-Min-Min, Min-Mean-Min y Min-Max-Min, evaluando rendimiento, eficiencia energética y escalabilidad en sistemas de gran escala. Entre esta familia de algoritmos no se presentaron significativas diferencias en los indicadores de rendimiento (makespan y flowtime) y escalabilidad, en cambio, respecto a la eficiencia energética Min-Min se destacó sobre los demás.

Otras heurísticas especializadas para el problema de asignación de tareas en sistemas distribuidos son Balanceo de Carga Oportunista (OLB por sus siglas en inglés)[11], *Minimum Execution Time* (MET), *Minimum Completion Time* (MCT), *Sufferage* y *High Standard Deviation First*. En el amplio estudio de Braun et al. [8], donde se evaluaron once heurísticas usando el modelo ETC, se concluyó: un algoritmo genético obtuvo el menor makespan, la heurística MCT superó a MET, y OLB obtuvo el peor makespan. OLB intenta mantener todas las máquinas siempre ocupadas, asignando cada tarea a la próxima máquina que se espera esté disponible, sin embargo, como no considera el tiempo de ejecución esperado para la tarea, puede producir un makespan muy largo [8]. MET trata de asignar cada tarea a la máquina donde se ejecute más rápido, es decir, la máquina con el mejor tiempo de ejecución para la tarea, pero como no tiene en cuenta la disponibilidad de la máquina (carga de trabajo actual), puede resultar un balanceo de carga de trabajo desequilibrado entre las máquinas. MCT trata de asignar cada tarea a la máquina con el tiempo de finalización mínimo esperado para ella, de esta manera trata de evitar los casos en que MET y OLB tienen un bajo rendimiento [8], pero esto puede llegar a asignar tareas a máquinas que no corresponden a su tiempo de finalización mínimo.

High Standard Deviation First (MaxStd) asigna primero la tarea con la desviación estándar más alta del tiempo de ejecución esperado de esa tarea, a la máquina que tiene el tiempo de finalización mínimo, ya que el retardo generado en su asignación no afectará demasiado el makespan total. La desviación estándar representa la cantidad de variación en el tiempo de ejecución de la tarea en diferentes máquinas [2].

Sufferage es la diferencia entre el primero y segundo mejor tiempo de finalización

[11] Opportunistic Load Balancing

mínimo de la tarea [2], [32]. La tarea con mayor *sufferage* es asignada a la segunda máquina más favorable para ella, porque de otra manera esa tarea sería la de mayor retardo.

3.1.2.2 Metaheurísticas de *Job Scheduling*

En la literatura de HPC se encuentran técnicas más complejas conocidas como metaheurísticas, enfoques que se han utilizado para resolver múltiples problemas de optimización donde las heurísticas no son muy efectivas, y que podrían ser base para diseñar eficientes *schedulers* para *grid* [20]. Estas evalúan numerosas soluciones para encontrar soluciones sub-óptimas de alta calidad (es decir, cercana a la óptima) o en ocasiones la óptima, sin embargo, normalmente requieren largos tiempos de ejecución [20], tiempos muy superiores a los de las heurísticas. Las principales metaheurísticas que se han aplicado en calendarización de tareas se muestran en la Tabla 4, junto a sus características fundamentales y trabajos relacionados. Algunos de estos trabajos siguen el modelo ETC y la mayoría son sobre calendarización de tareas en *grid*.

Algunas de estas metaheurísticas tienen componentes aleatorios, como las mutaciones en Algoritmos Evolutivos, e información adicional generada por sí mismo, como la feromona en Optimización por Colonia de Hormigas (ACO), para guiar y diversificar la búsqueda de soluciones. Aun así, no pueden garantizar que la solución óptima fue encontrada, solo pueden encontrar soluciones aproximadas al óptimo. Estos métodos dependen mucho de la calidad y diversidad de la solución encontrada en la primera iteración, que generalmente se genera de forma aleatoria para asegurar la diversidad. Algunos son multi-arranque, para explorar otras soluciones que permitan direccionar la búsqueda hacia las regiones del espacio donde está el óptimo global, en vez de estancarse en óptimos locales. Las metaheuristicas pueden ser basadas en búsqueda local o en población.

Tabla 4: Características principales de las metaheurísticas [12]

Metaheurística	Características	Referencias
Recocido Simulado	Criterio de aceptación Calendario de enfriamiento	30
Búsqueda Tabú	Elección de vecino (lista tabú) Criterio de aspiración	29
Algoritmos evolutivos	Recombinación Mutación Selección	2, 8, 21, 22, 23, 24
Optimización por Colonia de Hormigas	Construcción probabilística Actualización de feromona	25, 26, 27
Optimización por Enjambre de Partículas	Basado en población Coeficiente social	28

Metaheurísticas basadas en búsqueda local

Una metaheurística de búsqueda local comienza desde alguna solución inicial e iterativamente tratan de reemplazar la solución actual por una mejor solución en una vecindad de la solución actual apropiadamente definida [12]. Búsqueda Local (LS por sus siglas en inglés) se realiza hasta que se cumpla un criterio de parada, como un número determinado de iteraciones consecutivas sin cambios de solución actual o hasta que se agote el tiempo máximo de ejecución. Sólo requiere de unas cuantas especificaciones como una función de evaluación y un método eficiente para explorar entornos. Este método determinístico y sin memoria, puede encontrar soluciones rápidamente pero su solución final depende fuertemente de la solución inicial para no estancarse en óptimos locales.

Búsqueda Tabú (TS por sus siglas en inglés, de *Tabu Search*): En cada iteración puede aceptar soluciones de mayor costo para explorar otras áreas del espacio de búsqueda (diversidad) [35], teniendo en cuenta una lista tabú que impide los movimientos repetidos. Xhafa et al. implementaron este método siguiendo el modelo ETC [29].

Greedy Randomized Adaptive Search Procedure (GRASP) es un método de búsqueda iterativa aleatoria [31], que cambia la solución actual con una lista limitada de candidatos (RCL, *Restricted Candidate List*) de las mejores opciones disponibles, y termina cuando completa un criterio de finalización, por ejemplo al alcanzar un número determinado de iteraciones.

Recocido Simulado (SA por sus siglas en inglés, de *Simulated Annealing*) es un algoritmo de búsqueda estocástico sin ningún tipo de memoria [30], inspirado en el proceso de recocido en Metalurgia. En este proceso se calienta un material (como el acero) hasta una temperatura específica, el calor causa que los átomos aumenten su energía y así pueden desplazarse fácilmente de sus posiciones iníciales para explorar el espacio de búsqueda, y luego se enfría progresivamente hasta la temperatura ambiente, buscando alcanzar el óptimo global donde el material adquiere las propiedades físicas (como ductilidad, tenacidad, etc.) deseadas. El algoritmo parte de una solución inicial aleatoria y una probabilidad alta (temperatura inicial) para permitir cualquier movimiento aleatorio, que puede ser una solución de peor calidad que la solución actual, con el fin de escapar de los mínimos locales y explorar más el espacio de búsqueda. La probabilidad de aceptar cualquier movimiento se reduce (enfriamiento) gradualmente durante la búsqueda, hasta convertirse en un algoritmo iterativo que solo acepta cambios de solución actual si hay una mejora. La regla de enfriamiento puede variar durante la ejecución del algoritmo, con el objetivo de ajustar el equilibrio entre la diversificación de la búsqueda e intensificación para converger a una solución [12].

Metaheurísticas basadas en población

En las metaheurísticas basadas en población, el espacio de solución es explorado a través de una población de individuos. Las principales metaheuristicas de esta categoría son los Algoritmos Evolutivos, Optimización por Colonia de Hormigas y Optimización por Enjambre de Partículas.

Los Algoritmos Evolutivos (EA) están inspirados en la evolución de los seres vivos, por lo que utiliza mecanismos de selección y combinación. Los algoritmos más utilizados de esta familia son los Algoritmos Genéticos (GA por sus siglas en inglés), en que a partir de una población inicial de cromosomas (solución) se busca encontrar la más apta (solución con el mejor costo en la función objetivo) en el transcurso de varias generaciones, por medio de recombinaciones (*crossover*) de los cromosomas, mutaciones aleatorias de sus genes y selección de los cromosomas que sobrevivirán para producir la siguiente generación. Los algoritmos genéticos para el problema de calendarización en *grid* han sido bastante utilizados, por ejemplo por Braun et al. [8], Zomaya y Teh [21], Gao et al. [22] y Carretero et al. [23]. Pinel et al. [2] implementaron un Algoritmo Genético en un clúster convencional, al cual se le añadieron *millicomputers* para reducir el consumo de energía. Este algoritmo se denomina PA-CGA (*Parallel Asynchronous – Cellular Genetic Algorithm*), y fue propuesto junto a una heurística llamada 2PH (*Two Phase Heuristic*) que se compone de dos fases, Min-Min seguido de *Local Search*. Ambos algoritmos fueron evaluados frente a Min-Min, logrando un mejor rendimiento (makespan) con un runtime menor. En el trabajo de Nesmachnow et al. [37], propusieron el algoritmo evolutivo *Parallel Micro CHC* (*Cross generational elitist selection, Heterogeneous recombination and Cataclysmic mutation*) con el que obtuvieron un buen makespan para calendarización en *grid*.

Otro algoritmo de computación evolutiva es el Algoritmo Memético (MA por sus siglas en inglés), algoritmo hibrido que combina las ideas de la evolución con la búsqueda local, por medio de memes (unidad de información cultural) semejantes a los genes, se transmite información común de una población a la siguiente generación. Existen pocos trabajos que implemente este algoritmo para el problema de calendarización en *grid* porque es un algoritmo relativamente nuevo, como el de Xhafa et al. [24], que propone un *Cellular Memétic Algorithm* (cMA) para calendarización bajo el modelo ETC.

En la literatura se han propuesto diversos algoritmos que siguen la técnica probabilística de Optimización por Colonia de Hormigas (ACO por sus siglas en inglés, de *Ant Colony Optimization*), para encontrar soluciones aproximadas a los problemas complejos de optimización combinatoria. El primer algoritmo ACO fue el *Ant Sytem* (AS), propuesto por Marco Dorigo y utilizado para resolver un problema similar denominado *Job Shop Scheduling* [26]. Recientes desarrollos dieron mejores resultados, como el *Max-Min Ant Sytem* (MMAS) [27]. Una implementación de ACO para calendarización de tareas en *grid* fue realizada por Chang et al. [25].

Optimización por Enjambre de Partículas (PSO por sus siglas en inglés, de *Particle Swarm Optimization*) es un algoritmo similar a ACO, que busca copiar el comportamiento en enjambre de diferentes seres vivos (abejas, aves, peces, etc.). Abraham et al. plantearon una aproximación al problema de calendarización usando un algoritmo de *PSO fuzzy* [28]. También se han implementado metaheurísticas híbridas, principalmente con Búsqueda Tabú. Primero se ejecuta otra metaheurística, por ejemplo un algoritmo genético, que busca una solución de buena calidad, y luego Búsqueda Tabú trata de mejorarla explorando el vecindario de esa solución. En los trabajos que presentan resultados con las instancias de Braun et al. [8], se han implementado las metaheurísticas híbridas MA+TS [34] y ACO+TS [36].

Las metaheurísticas que se han utilizado para abordar el problema de *job scheduling*, presentan diversos métodos de búsqueda de soluciones, centrados principalmente en la generación de información a medida que se evalúan múltiples soluciones (para guiar la búsqueda hacia mejores soluciones), con lo cual pueden encontrar soluciones de más calidad que las halladas por las heurísticas y los ineficientes métodos exactos. Las más promisorias son inspiradas en fenómenos de la naturaleza como la teoría de la evolución y la inteligencia de enjambre. En el siguiente capítulo se comparan estos algoritmos según un conjunto de instancias representativas del problema de *job scheduling* en sistemas de computación

distribuida [8].

4. COMPARACIÓN DE LOS ALGORITMOS DE *JOB SCHEDULING*

Los algoritmos de *job scheduling* se pueden comparar utilizando las instancias más usadas del modelo ETC, generadas por Braun et al. [8], son 12 instancias de 512 tareas y 16 máquinas que corresponden a los doce tipos diferentes de matrices ETC. Las métricas analizadas en este trabajo son makespan, como el indicador de rendimiento, y consumo de energía para establecer la eficiencia energética según el rendimiento alcanzado. En la mayoría de artículos revisados solo se reporta el makespan, por lo que la eficiencia energética solo se pudo determinar para los algoritmos implementadas por la Universidad de Luxemburgo [15], [17], mediante la ejecución de los algoritmos con las doce instancias y usando el modelo de energía (y valores de los parámetros) definido por Guzek et al. [40].

El mejor makespan reportado de los algoritmos se compara en la Tabla 5, donde se destacan los algoritmos evolutivos *Parallel CHC* [33] y *Parallel Micro CHC* [37], este último logra el mejor makespan en todas las instancias. También se puede destacar la heurística Min-Min [1], [32], por requerir un tiempo de ejecución muy bajo para obtener soluciones de buena calidad, aspecto en el cual las metaheurísticas evolutivas no son muy fuertes. Los resultados completos de makespan de todas las heurísticas y metaheurísticas se encuentran en la página web http://forge.sc3.uis.edu.co/redmine/documents/1. Allí se puede ver que Min-Min es mejor que todas las heurísticas como Sufferage y Max-Min, y se conoce que Min-Min también es mejor que las heurísticas MET, MCT y OLB según las gráficas de comparación presentadas en el trabajo de Braun et al. [8]. Los resultados del makespan de Min-Min y Max-Min reportados en [32], concuerdan con los obtenidos en la ejecución de los algoritmos proporcionados por la Universidad de Luxemburgo [15], [17]

Tabla 5: Mejor *makespan* de los algoritmos

Instancia 512x16	Min Min [32]	Sufferage [32]	TS [35]	cMA [24]	GA [33]	Parallel GA [33]	PA-CGA [38]	CHC [33]	Parallel CHC [33]	Parallel Micro-CHC [37]	MA+TS [34]	ACO+ TS [36]
u_c_hihi.0	8.460.674	10.908.698	7.448.641	7.700.930	7.659.879	7.577.922	7.437.591	7.599.288	7.461.819	**7.381.570**	7.530.020	7.497.201
u_c_hilo.0	161.805	167.483	153.263	155.335	155.092	154.915	154.393	154.947	153.792	**153.105**	153.917	154.235
u_c_lohi.0	275.837	349.746	241.673	251.360	250.512	248.772	242.062	251.194	241.513	**239.260**	245.289	244.097
u_c_lolo.0	5.441	5.650	5.155	5.218	5.239	5.208	5.248	5.226	5.178	**5.148**	5.174	5.178
u_i_hihi.0	3.513.919	3.391.758	2.957.854	3.186.665	3.019.844	2.990.518	3.011.581	3.015.049	2.952.493	**2.938.381**	3.058.475	2.947.754
u_i_hilo.0	80.756	78.828	73.693	75.857	74.143	74.030	74.477	74.241	73.640	**73.378**	75.109	73.776
u_i_lohi.0	120.518	125.689	103.866	110.621	104.688	103.516	104.490	104.546	102.123	**102.051**	105.809	102.446
u_i_lolo.0	2.786	2.674	2.552	2.624	2.577	2.575	2.603	2.577	2.549	**2.541**	2.597	2.554
u_s_hihi.0	5.160.343	5.574.358	4.168.796	4.424.541	4.332.248	4.262.338	4.229.018	4.299.146	4.198.780	**4.103.500**	4.321.015	4.162.548
u_s_hilo.0	104.375	103.401	96.181	98.284	97.630	97.506	97.425	97.888	96.623	**95.787**	97.177	96.762
u_s_lohi.0	140.285	153.094	123.407	130.015	126.438	125.717	125.579	126.238	123.237	**122.083**	127.633	123.922
u_s_lolo.0	3.807	3.728	3.451	3.522	3.510	3.480	3.526	3.492	3.450	**3.434**	3.484	3.455
Promedio	1.502.545	1.738.759	1.281.544	1.345.414	1.319.317	1.303.875	1.290.666	1.311.153	1.284.600	**1.268.353**	1.310.475	1.284.494

Nota: En negrilla los mejores resultados

Analizado el *makespan* por cada tipo de consistencia, las heurísticas Min-Min y *Sufferage* tienen un *makespan* largo en las instancias consistentes y semi-consistentes, *Parallel CHC* es el segundo algoritmo de mejor *makespan* en seis de las doce instancias, que pertenecen en su mayoría al tipo de instancias semi-consistentes e inconsistentes. En las instancias restantes es superado por Búsqueda Tabú [35], ACO+TS [36] y PA-CGA [38]. Y aunque las metaheurísticas híbridas (ACO+TS y cMA+TS) no son las mejores en este caso, son una buena alternativa que debe seguirse investigando.

En los algoritmos donde se tiene la información necesaria para evaluar la función multi-objetivo, que relaciona el consumo de energía y el *makespan*, se utiliza una función de *score* SF [7], [17], también conocida como función de *fitness* (aptitud) [20], y se fija la importancia de los dos objetivos (*makespan* respecto a la energía) con un parámetro de peso α de la siguiente manera:

$$SF = \alpha * C_{max} + (1 - \alpha) * E \quad (5)$$

Por lo tanto, el objetivo será minimizar la función score. Esta puede representar la eficiencia energética si la prioridad de ambos objetivos es igual, para lo cual se fija el valor de α equilibrado en 0.5. Además, se requiere normalizar los valores de cada métrica para realizar un correcto cálculo del score (porque las métricas tienen diferentes unidades de medida), por lo que el valor de *makespan* y energía obtenido por cada algoritmo es dividido por el máximo valor de todos los algoritmos [15], así los valores normalizados estarán en el intervalo [0,1], donde 1 es el valor de peor desempeño. El *score* obtenido de los algoritmos ejecutados se comparado en la Figura 3, que muestra que Min-Min es el algoritmo de mejor desempeño en todas las instancias, aunque es superado respecto a la energía por su versión especializada en la reducción del consumo de energía (Min-Min *with Energy*). También es destacable la eficiencia energética de Min-Mean-Min y Min-Max-Min. En contraste, Max-Min tiene uno de los peores desempeños, principalmente en las instancias inconsistentes donde presenta el más alto consumo de energía y *makespan*.

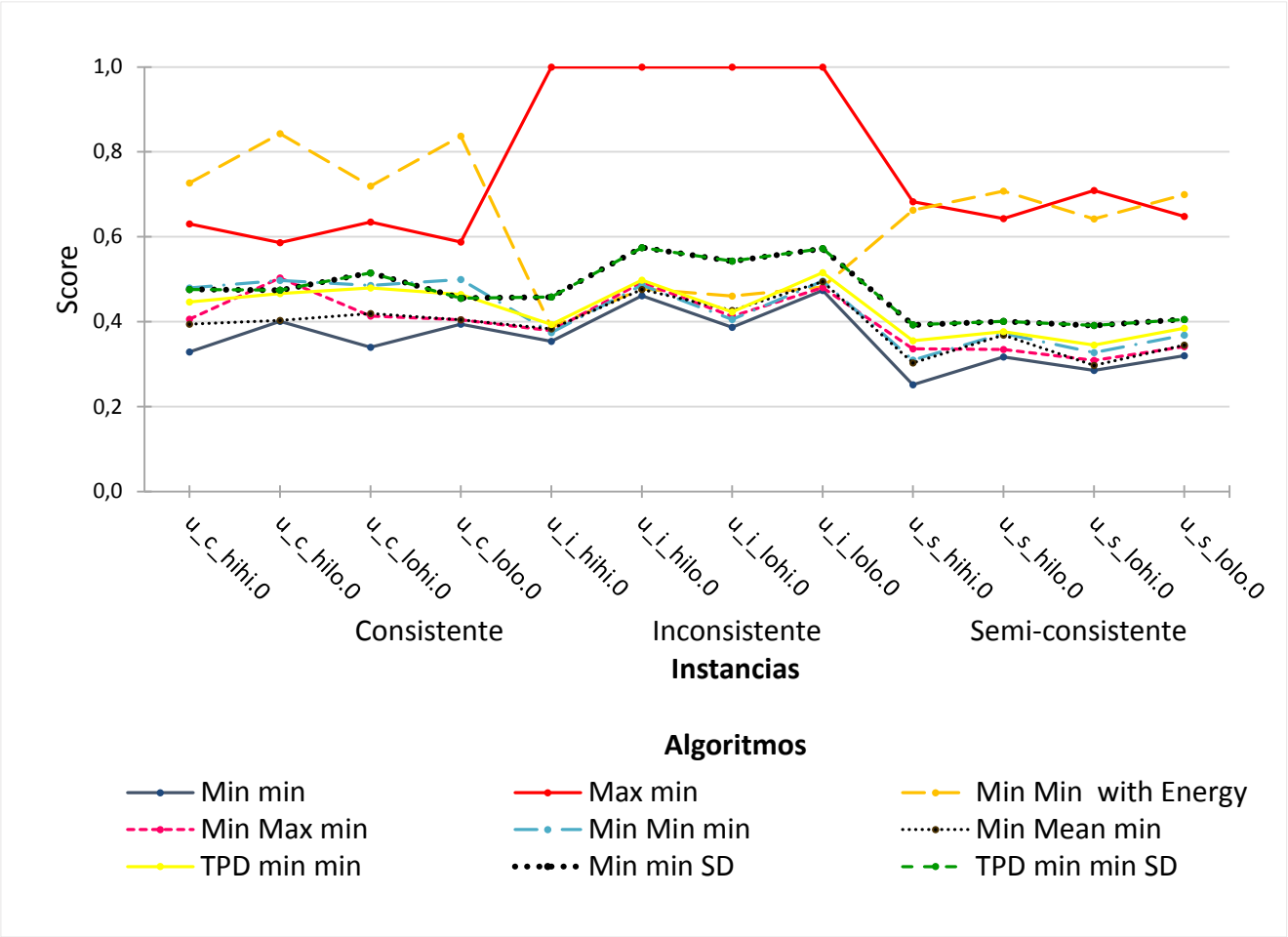

Figura 3: Eficiencia energética de los algoritmos

4.1 ANÁLISIS DE LOS ALGORITMOS DESTACADOS

Los algoritmos destacados según makespan son *Parallel Micro CHC* y *Parallel CHC*. Estos son algoritmos evolutivos que tienen en común sus características básicas (selección, recombinación, mutación), y logran encontrar las mejores soluciones para el problema de calendarización en *grid* porque tienen un buen balance entre búsqueda aleatoria y guiada. A partir de una solución inicial aleatoria (u obtenida con las heurísticas rápidas como Min-Min, MCT, *Sufferage*), exploran el espacio de búsqueda guiados por la información contenida en una población de individuos

(genes). Para esto se define una regla de selección de los individuos más aptos, los cuales se recombinan para formar nuevos individuos (descendientes), y con el paso de las generaciones (iteraciones) permiten llegar a una solución de buena calidad (o que tenga el valor óptimo de la función objetivo). Las mutaciones que se realizan aleatoriamente en los algoritmos evolutivos tradicionales, para superar los óptimos locales y diversificar la búsqueda, en estos algoritmos son reemplazadas por una restricción de emparejamiento (*mating*) entre individuos muy parecidos y un proceso de re-inicialización cuando la búsqueda tiende a converger rápidamente a un óptimo local.

Los dos algoritmos se diferencian principalmente en que *Parallel Micro CHC* incorpora conceptos adicionales del Algoritmo Micro Genético (μ-GA) [41], para no estancarse por la falta de diversidad en las soluciones cuando se usan pequeñas poblaciones, mediante el mantenimiento de una población elite que se utiliza para reinicializar la población principal cada cierto número de generaciones.

Entre las heurísticas, Min-Min tiene el mejor equilibrio entre el consumo de energía y makespan, debido a que siempre asigna primero la tarea a la máquina con el tiempo de finalización mínimo (MCT) global [8], por lo tanto, el sistema tiene más máquinas disponibles para la ejecución de tareas en la mejor máquina correspondiente, es decir, la máquina con el ETC más bajo para la tarea. Min-Min probablemente puede asignar más tareas a su mejor ETC que Max-Min en instancias grandes, que asigna las tareas ordenadas según el tiempo de finalización mínimo, a la máquina con el máximo tiempo de finalización. La heurística Min-Min asigna la primera tarea t_i a la máquina que lo termina más rápido, y para cada tarea asignada después de t_i, Min-Min cambia el estado de disponibilidad de la máquina la menor cantidad posible para cada asignación. La expectativa es que se puede obtener un makespan más pequeño si más tareas se asignan a las máquinas que las finalizan más rápido y también las ejecutan en menos tiempo [8].

Los algoritmos han sido comparados para el problema de *job scheduling* en términos de las instancias más comunes del modelo ETC. En estos términos, el mejor método identificado según el makespan es el algoritmo evolutivo *Parallel Micro CHC*, y los resultados completos sugieren que los algoritmos evolutivos son los más adecuados para encarar la complejidad este problema. En la comparación de la eficiencia energética de las principales heurísticas, se destaca el algoritmo Min-Min. Por lo tanto, se implementaran estos algoritmos para solucionar el problema de *job scheduling* en *milliclusters*.

5. DESARROLLO DEL ALGORITMO

El algoritmo evolutivo a implementar, *Parallel Micro CHC*, empieza su búsqueda desde una generación inicial de cromosomas (soluciones) obtenida con otras heurísticas, y guarda el cromosoma más apto (solución con el mejor costo en la función objetivo) encontrado en el transcurso de múltiples generaciones, por medio de recombinaciones (*crossover*) selectivas de los cromosomas y mecanismos de diversificación de la búsqueda.

Para minimizar simultáneamente makespan y consumo de energía, se debe implementar la función multi-objetivo (*score*) en la etapa donde se toman decisiones en el algoritmo. En *Parallel Micro CHC* es en la evaluación de las nuevas soluciones o cromosomas, mientras que en Min Min el *score* se calcula en la asignación de cada trabajo a las máquinas, según los valores parciales de los objetivos (makespan actual y energía acumulada), porque solo construyen una solución. El pseudocódigo resumido es el siguiente:

Algoritmo Parallel Micro CHC
Establecer parámetros
Generar poblaciones de cromosomas iniciales (*Generación* 0)
Bucle de generaciones
    Selección de parejas de cromosomas
    Reproducción de parejas de cromosomas
    Evaluación de cromosomas descendientes
    Reemplazo de población de cromosomas
    Reinicio de cromosomas
    Migración entre poblaciones
Fin − de generaciones
Retornar mejor cromosoma encontrado
Fin − Algoritmo

5.1 ALGORITMO *PARALLEL MICRO CHC*

Parallel Micro CHC se basa en los conceptos de los algoritmos Micro Genético (µ-GA) [41], CHC y *Parallel CHC* [33].

En el **algoritmo Micro Genético** (µ-GA) [41], se divide la población principal de cromosomas en pequeñas poblaciones, y se realizan mutaciones cuando se detecta convergencia nominal (cuando todos los individuos tienen sus genotipos idénticos o muy similares), por ejemplo, en 2 iteraciones seguidas del algoritmo de un total de 3000 [41]. En un algoritmo genético la selección de las parejas de cromosomas que se van a reproducir se hace de acuerdo a la probabilidad de cruce [23], y luego la mutación de los descendientes de acuerdo a la probabilidad de mutación.

En el **algoritmo CHC** sólo los padres que se diferencian entre sí por un número determinado de genes (distancia) se les permite reproducirse [33]. Esta regla impide el incesto, es decir, la reproducción de cromosomas muy similares [37], con el fin de producir hijos bastante diferentes de sus dos padres, en lugar de seguir obteniendo soluciones muy similares que probablemente ya fueron evaluadas. Es una regla flexible porque a medida que avanzan las generaciones los cromosomas tienden a asemejarse, así que se debe reducir la distancia mínima exigida para permitir la reproducción, y cuando llega a cero, se reinicia la población porque todos los cromosomas son muy similares entre sí (convergencia nominal). La mutación de los algoritmos genéticos es reemplazada por el reinicio de la población para reintroducir diversidad a la búsqueda, generando una nueva población pero manteniendo los mejores individuos encontrados hasta el momento, que son los cromosomas elite. CHC logra converger usando una pequeña población externa de tres cromosomas elite [37].

En el **algoritmo *Parallel CHC*** se propone otra fuente de diversidad, las migraciones entre poblaciones “vecinas”, donde a intervalos definidos se intercambian cromosomas entre poblaciones [33].

Todos estos algoritmos conforman el *Parallel Micro CHC* [37] (Ver Figura 4), que en la etapa de reemplazo de soluciones, usa la regla de reemplazar solo si mejora, formando la nueva generación con los mejores individuos de la generación de padres y los nuevos hijos [23]. Mediante la metodología Prototipado Evolutivo se construyen prototipos del algoritmo, para ir desarrollándolo desde la versión más básica hasta el diseño completo, es decir, partiendo del algoritmo genético hasta alcanzar el diseño del algoritmo *Parallel Micro CHC*, a través de cambios graduales en el prototipo inmediatamente anterior. La construcción del prototipo asegura que las soluciones obtenidas sean siempre correctas, comparando el calendario generado con los datos en la instancia de entrada. La calidad de las soluciones se evalúa con los indicadores definidos anteriormente, y se determinan modificaciones pertinentes del algoritmo para obtener mejores resultados. El diseño del algoritmo implementado junto a su descripción se presenta a continuación.

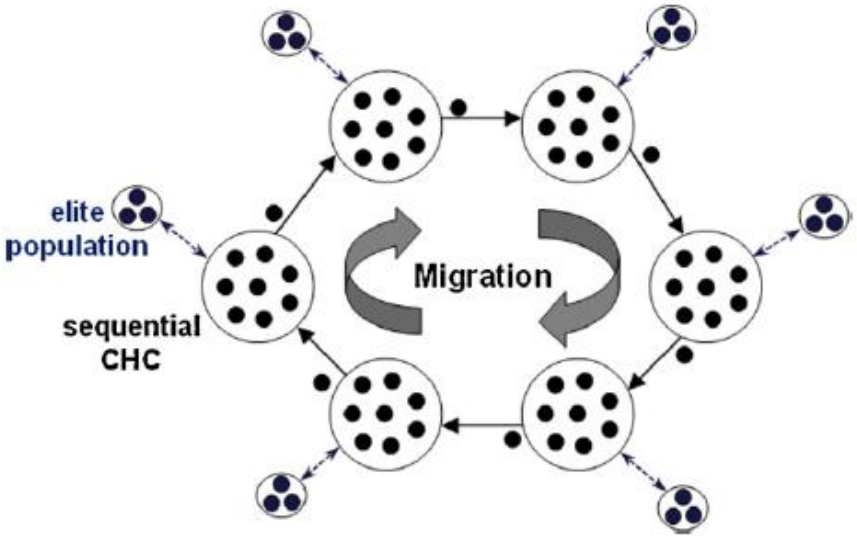

Figura 4: Esquema del algoritmo *Parallel Micro CHC* [37]

5.2 DISEÑO IMPLEMENTADO PARA EL PROBLEMA DE *JOB SCHEDULING*

La capacidad de exploración de este algoritmo depende de las migraciones y reinicios, porque las poblaciones por si solas solo pueden realizar búsquedas locales desde los cromosomas iniciales. Para buscar en nuevas regiones del espacio se requieren reinicios diversos pero de buena calidad, de lo contrario no llega a mejores óptimos locales, por esto usamos cromosomas nuevos diferentes junto a todos los cromosomas elites, tratando de tener reinicios que equilibran la importancia de la información conocida (por experiencia) y las nuevas soluciones. Como los reinicios solo consideran la convergencia nominal, se malgastan generaciones (sin poder realizar cambios) en el caso en que todos los cromosomas convergen en una misma solución. Por lo tanto, al detectar convergencia de la población, inmediatamente reiniciamos los cromosomas, sin esperar más generaciones a que la *distanciaMínima* se reduzca a cero.

Las migraciones sirven para compartir información entre poblaciones, pero solo es aprovechado al máximo cuando las poblaciones que intercambian cromosomas son diferentes, por esto utilizamos dos algoritmos distintos entre poblaciones vecinas.

Con estas modificaciones se pretende obtener soluciones buenas y diversas (de otras regiones del espacio de búsqueda), a partir de las cuales se pueden obtener soluciones que sobrepasen los óptimos locales hallados en las primeras iteraciones, y se detiene la búsqueda en el número generaciones donde ya no se registran mejoras frecuentes de la solución, que es similar al número definido en algoritmos genéticos [41]. El siguiente es el pseudocódigo del *Parallel Micro CHC* implementado para solucionar el problema de *job scheduling* en HPC:

Algoritmo Parallel Micro CHC

1. *Inicializar variables*:
$K\ (Cromosomas), P\ (Poblaciones), G\ (Generaciones)$, $distanciaMínima(P)$, p_c, $Trabajos, Máquinas$
2. *Lectura de entradas*:
Para cada Trabajo t y Máquina m hacer:
$\tau iempo_{tm} = matrizETC[t][m]$
$Energía_{tm} = \tau iempo_{tm} * matrizPower[t][m]$
3. *Generación g inicial*:
$g = 0$
Para cada Población P hacer:
Para cada Cromosoma K hacer:
3.1 *Asignación aleatoria del primer Trabajo*
3.2 *Mientras* $P(g)$ *no este completo, hacer*:
Asignar Trabajos segun regla de MinMin o MaxMin
3.3 $PadresP(g+1) =$ *Selección*$(P(g))$
4. *Mientras* $g <$ *Generaciones hacer*:
$g++$
Para cada Población P hacer:
Para cada Cromosoma K hacer:
4.1 ***Recombinar***: *Si* $distancia(PadresP(g)) \geq distanciaMínima(P)$:
$HijoP(g) = Reproducción(Padres\ P(g))$
4.2 ***Evaluar***: $score(HijoP(g))$
4.2.1 *Si* $HijoP(g)$ *mejor que* $Elite(P)$:
$Elite(P) = HijoP(g)$
4.3 ***Reemplazar***: $P(g) = mejor(HijoP(g) \cup P(g-1))$
4.4 ***Selección***: *Si* $p_c(P(g)) > p_c$:
$PadresP(g+1) = P(g)$
4.5 *Si* $P(g) = P(g-1)$:
$distanciaMinima(P)--$
4.5.1 *Si* $(distanciaMinima == 0$ ***OR*** $CromosomasIguales(P) == Cromosomas)$:
$Reiniciar(P(g)) = True$
$distance(PadresP(g)) = distanciaMínima(P)$
4.6 *Si* ***Reiniciar***$(P(g)) == True$:
$P(g) = Reinicialización(P(g)) \cup P(Elite(P))$
Fin − de cromosomas
Fin − de poblaciones
4.7 ***Migración***: *Si* $enviarMigrante == True$:
$Inmigrante(P) = SelecciónMigración(P(g))$
$recibirMigrante(Inmigrante(P) \rightarrow P(g)vecino)$
4.8 *Guardar mejor plan de la generación g*
Fin − bucle de generaciones
5. *Mostrar mejor plan de las generaciones*

$Fin - Algoritmo$

Descripción del pseudocódigo:

1. Los resultados se obtuvieron con las variables inicializadas en:

$G = 2000$ generaciones (iteraciones del bucle generacional)

$P = 16$ poblaciones independientes

$K = 8$ cromosomas (soluciones) por población, por lo que 128 es el tamaño de la población global

$p_c = 0.9$ Probabilidad de reproducción (*crossover*)

$Trabajos = 512$ trabajos (*jobs*) por procesar

$Máquinas = 16$ máquinas disponibles para procesar los trabajos

$distanciaMinima(P) = \frac{1}{4} Genes = 128$ es la distancia mínima entre padres para permitir su reproducción, es decir, el número de genes (longitud del cromosoma) que deben ser diferentes (25%). Los genes son iguales al número de trabajos en este problema.

2. Lectura de instancia ETC y la instancia de uso de recursos para estimar el *power* de las máquinas, con los que se calcula el consumo de energía por tarea en cada máquina.

3. Se obtiene la primera generación de cromosomas (soluciones iniciales) usando el algoritmo Min Min y Max Min, intercalados entre poblaciones vecinas, que serán los padres de la siguiente generación de cromosomas, la primera tarea de cada cromosoma es asignada de forma aleatoria.

4. Empieza el bucle de generaciones del algoritmo evolutivo (en g=1), y todo su procedimiento se repite hasta llegar al número total de generaciones (criterio de parada), para cada cromosoma en cada población. Las poblaciones de cromosomas evolucionan en paralelo mediante la librería *Concurrent* de Java, que

hace uso de memoria compartida.

4.1 Recombinación de las parejas de cromosomas padres, si tienen al menos el número mínimo de máquinas diferentes (*distanciaMínima*) asignadas a los trabajos. Esta es la restricción que evita el “incesto” entre cromosomas, es decir, impide la reproducción de individuos muy similares [37]. En la reproducción si la máquina asignada a un trabajo (que conforman un gen) es el misma en los dos cromosomas, se agrega igual en la secuencia de genes del cromosoma hijo, pero si es diferente, la máquina asignada para el trabajo se escoge aleatoriamente entre las máquinas que tienen los dos cromosomas. Esto se repite hasta completar la secuencia de genes (trabajos) del cromosoma hijo.

En el algoritmo la secuencia de genes es una matriz (Ver Tabla 6), que contiene las operaciones conformadas por los trabajos y máquinas en orden de asignación.

Tabla 6: Ejemplo de secuencia de genes de instancia de Tabla 3 usando Min Min

Trabajo	t1	t2	t3	t4	t5	t6	t7	t8	t9
Máquina	M1	M1	M1	M1	M2	M1	M1	M3	M1

4.2 Evaluación de la aptitud de las nuevas soluciones (cromosomas hijos), según la función multi-objetivo representada por el *score* (Ver Ecuación 5), para lo cual se obtiene el makespan y energía consumida del calendario de trabajos.

4.3 Reemplazar la generación de cromosomas anterior con los cromosomas hijos y padres más aptos (mejores soluciones) según el *score* (regla de aptitud). Mantiene a los padres que sean mejores que los hijos de la población, si todos los padres sobreviven significa que no hubo cambios entre dos generaciones sucesivas.

4.4 Selección de nuevos padres para la próxima generación, usando la probabilidad de reproducción (*crossover*) p_c para todas las combinaciones posibles de padres.

4.5 Si dos generaciones seguidas de una población son iguales, porque todos los cromosomas padres sobrevivieron al ser mejores que sus hijos o por no poderse reproducir al ser similares (distancia menor a la *distanciaMínima* actual), se reduce la distancia permitida (*distanciaMínima*) entre padres para dejar que se reproduzcan parejas de cromosomas que tienen menos máquinas diferentes.

4.5.1 Si la *distanciaMínima* se reduce hasta cero, significa que los cromosomas de una población han pasado muchas generaciones sin cambiar por convergencia nominal (falta de diversidad), los genes de los cromosomas de una población son muy similares, haciendo imposible obtener nuevos y mejores cromosomas hijos. Entonces se debe reiniciar la población de cromosomas y también se reiniciar la *distanciaMínima* permitida porque ya se cuenta con cromosomas muy diferentes.

También si todos los cromosomas de una población son iguales entonces se debe reiniciar la población de cromosomas y la *distanciaMínima*. Detectando la convergencia de la población hacia una solución (cromosomas homogéneos), se evita pasar generaciones sin cambios, esperando a que se reduzca la *distanciaMínima* hasta cero para reiniciar la población.

4.6 Reiniciar población usando los cromosomas de la subpoblación elite y los algoritmos Min Min o Max Min, para crear una nueva generación de cromosomas. Es la diversificación de cromosomas sin perder los mejores cromosomas hallados hasta el momento (guardados en población Elite).

4.7 Migración de cromosoma (mejor inmigrante) de la población cada determinado número de generaciones, que se envía a la población vecina según un grafo unidireccional, en reemplazo del mejor cromosoma que ya fue guardado en población élite.

4.8 Guardar la mejor solución de la generación con su respectivo *score*, mientras no terminen las generaciones, esta se incrementa para empezar a construir nuevas soluciones en la siguiente iteración.

5. Muestra el mejor calendario (solución) de todas las generaciones de cromosomas.

El código del algoritmo (desarrollado en lenguaje JAVA) está disponible en el repositorio del grupo de investigación SC3: http://forge.sc3.uis.edu.co/calendarizacion-energeticamente-eficiente-en-milliclusters.git

El algoritmo implementado permite resolver el problema de *job scheduling* en HPC, en función del makespan y consumo de energía. Debido a que nuestro propósito es estudiar la eficiencia energética en *milliclusters*, se requiere de un modelo de energía que pueda estimar el *power* de tareas (trabajos) ejecutados sobre los nodos (máquinas) del *millicluster*, el cual se propone en el siguiente capítulo.

6. MODELO DE ENERGIA

El modelo de consumo de energía en *millicluster* se obtiene mediante un proceso de benchmarking, donde se utilizó el clúster Viridis basado en arquitectura de hardware ARM (versión 7), ubicado en la Universidad de Luxemburgo (Ver Tabla 7). Es un clúster muy reciente de alta densidad con procesadores de bajo consumo ARM CortexA9 *Quad-core*, que cuenta con 4GB de DRAM por nodo (Ver Figura 5). Los 96 nodos del clúster están distribuidos en 2 módulos (*enclosures*) *de 48 SoC, c*ada módulo con 12 Calxeda EnergyCard, agrupando 4 nodos cada una. El *power* actual de los grupos de 4 nodos se puede ver en https://hpc.uni.lu/viridis-energy/last/minute/ y se puede obtener el *power* de cada nodo con la herramienta de gestión IPMI. Un nodo consume de 0.5 a 6 watts (dependiendo de la carga) y los nodos se interconectan a través de un enlace de 10 GbE. La comunicación Ethernet es manejada internamente por el ancho de banda de 80 GB en el *switch* de fábrica del *EnergyCore*, evitando así la necesidad de *switches* adicionales que consumen más energía y generan más latencia [58]. Usando el *benchmark* HPL este clúster alcanza un rendimiento máximo (R_{peak}) de 0.42 TFlops[12].

Tabla 7: Especificaciones del clúster Viridis

Procesadores	Nodos	Cores	RAM (GB)	R_{Peak} (TFlops)
ARM A9 Cortex @ 1.1GHz, 4 Cores	96	384	384	0.422

Con aplicaciones específicas, el rendimiento combinado general del clúster Viridis puede superar a todo un rack de servidores x86 estándar, y solo consumiendo 1/10 de la potencia y ocupando 1/10 de espacio [58].

[12] Fuente: Plataforma HPC de la Universidad de Luxemburgo, https://hpc.uni.lu/systems/nyx

El clúster Viridis tiene el sistema operativo Debian GNU/Linux 7.0 (wheezy), y contiene software requerido por los *benchmarks* como GCC 4.6.3, gFortran 4.6.3, OpenMP 1.6.3 y ATLAS 3.8.4-arm.

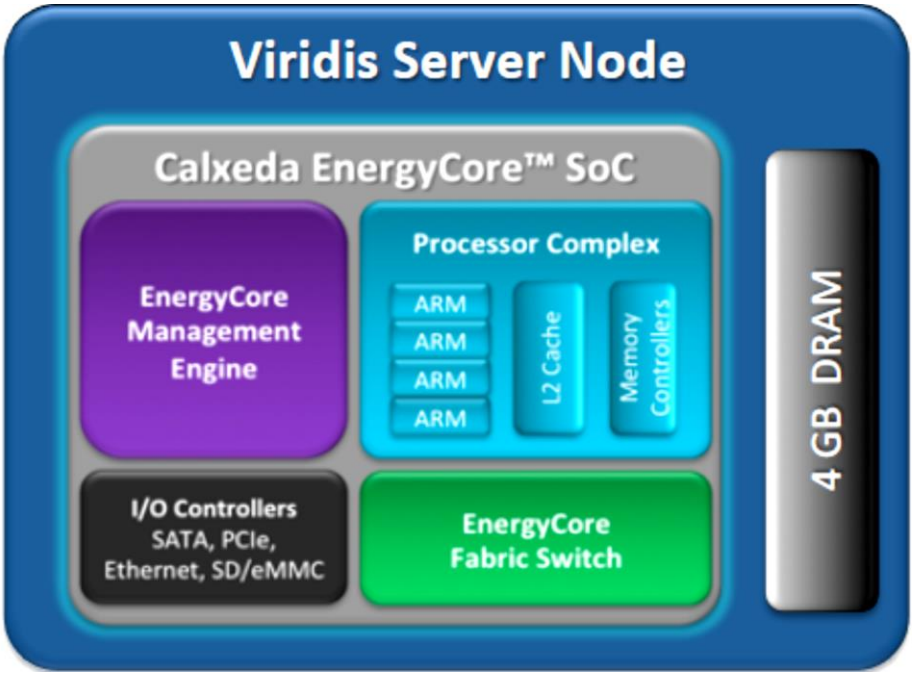

Figura 5: Esquema de nodo del clúster Viridis [58]

6.1 *BENCHMARKS* USADOS

Para modelar el consumo de energía del clúster, se consideran *benchmarks* que tienen diversas métricas de rendimiento ya analizadas en clústeres de bajo consumo [42], [43], [45], por representar los diferentes tipos de *workload* que se procesan comúnmente en infraestructuras HPC. Con el fin de facilitar la reproducibilidad de los experimentos, todos los *benchmarks* son *open source* y tuvieron como datos de entrada los *datasets* distribuidos junto a ellos mismos o datos que generan con los valores de los parámetros especificados más adelante.

El conjunto de *benchmarks* ejecutado se compone de HPL 2.1, Whetstone 1.2,

Fhourstone 3.2, Linpack, Coremark 1.0, NBench 2.2.3, Phoenix 2.0, y OSU Micro-Benchmarks 4.3. En la Tabla 8 se indica los componentes principales del clúster (disco, memoria, red o CPU) que utilizan con mayor intensidad.

Tabla 8: Tipo de *workload*

Benchmark	**Uso intensivo de componente**			
	CPU	**Memoria**	**Red**	**Disco**
Whetstone	●			
Linpack	●	●		
HPL	●	●		
Coremark	●			●
NBench	●			
Fhourstone	●			
Phoenix	●			●
OSU Micro-Benchmarks			●	

Una breve descripción de cada uno de los benchmarks junto a los parámetros de ejecución se presenta a continuación.

OSU Micro-Benchmarks: se utiliza para probar la capacidad de la interfaz de red, por lo que se mide la latencia (micro-seg) y ancho de banda (MB/s) en las comunicaciones entre nodos. En la versión One-sided MPI, un nodo envía y recibe paquetes (MPI_Send y MPI_Recv) de otro, y se mide la latencia y ancho de banda con la operación de iniciación de la ventana MPI_Win_create y sincronización pasiva con MPI_Win_lock/unlock. La latencia es el tiempo promedio para las llamadas MPI_Lock + MPI_Get + MPI_Unlock, necesarias para intercambiar paquetes de tamaños que van desde 2^0 hasta 2^{22} (4194304) bytes. El ancho de banda (MBytes/seg) se calcula con el número de bytes enviados por el remitente y el tiempo transcurrido desde que el emisor envía el primer mensaje (de tamaño igual al de la ventana) hasta el momento en que recibe la respuesta de vuelta desde el receptor, que la envía sólo después de recibir todos los mensajes.

CoreMark: es un *benchmark* diseñado principalmente para probar la capacidad del procesador. CoreMark mide las iteraciones por segundo (Iteraciones/Seg) al procesar un *workload* compuesto de varios algoritmos comúnmente utilizados, incluyendo la manipulación de matrices, manipulación de listas (para el uso de punteros), operación de máquinas de estados (de uso común con datos dependientes), y la comprobación de redundancia cíclica (CRC, es una función común utilizada en sistemas embebidos). Se realizaron 160000 iteraciones con la ejecución estándar de CoreMark, usando las semillas por defecto 0x3415, 0x3415, 0x66, y el tamaño del búfer ajustado a 2000 bytes.

LINPACK: registra los Flops (*FLoating Point Operations Per Second*) alcanzados al solucionar un sistema de ecuaciones lineales N x N mediante eliminación gaussiana, usando la librería BLAS (*Basic Linear Algebra Subprograms*) para realizar las operaciones con vectores y matrices. Este *benchmark* se ejecutó con una matriz de tamaño 200 x 200 y 10 repeticiones para recopilar mínimo 30 segundos de actividad, y la complejidad computacional es del orden de n^3 operaciones.

Whetstone: es un *benchmark* sintético que mide principalmente el rendimiento de la aritmética de punto flotante, con las operaciones básicas como funciones trigonométricas, saltos condicionales y llamadas de procedimientos. Fue derivado de estadísticas sobre el comportamiento de los programas recogidos del computador KDF9 en el NPL (*National Physical Laboratory*) de Reino Unido, utilizando una versión modificada de su compilador ALGOL 60 Whetstone. El *workload* en la máquina se representa como un conjunto de frecuencias de ejecución de las 124 instrucciones del Código de Whetstone. En la máquina se ejecutan 1000000 bucles de operaciones de doble precisión para medir los Millones de Instrucciones de Whetstone Por Segundo (MWIPS).

Fhourstones: *benchmark* que maneja datos tipo entero para resolver posiciones

en el juego Conecta Cuatro (*Connect Four*, c4). Este se juega en un tablero vertical (de tamaño 7x6) de 7 columnas por 6 filas, donde 2 jugadores se turnan (42 veces máximo) en la caída de piedras (*stones*) en una columna, y el primer jugador que consigue 4 piedras en una fila horizontal, vertical o diagonal, gana el juego. El algoritmo de búsqueda Alpha-Beta ordena movimientos dinámicamente desde cada posición, que se representan como *bitboards* de 64 bits, dando a las máquinas de 64 bits una ligera ventaja. La métrica de Fhourstones es el número de posiciones de juego exploradas por segundo (kpos/Seg).

HPL (*High Performance Linpack*): *benchmark* usado como métrica de rendimiento en el ranking Top500, mide al igual que el Linpack los Flops al solucionar un sistema de ecuaciones lineales N x N pero en un computadores con memoria distribuida, por lo que requiere la disponibilidad de una implementación de la interfaz de paso de mensajes (MPI). El tamaño N del problema debe ser soportable por la memoria RAM disponible, es luego dividido en NB x NB bloques que se reparten en una *grid* de P x Q procesos a distribuir entre los nodos. Para la ejecución del HPL sobre 2 nodos se distribuyeron 1 x 2 procesos, y con el N fijado en 20832, el NB más adecuado fue 96.

Nbench-byte: conjunto de 10 benchmarks basado en el *BYTEmark Native Mode Benchmark* que mide una variedad de operaciones diseñadas para determinar las capacidades de CPU y memoria. Los benchmarks son *String sort*, *Numeric sort*, Bitfield, IDEA, *Assignment*, Huffman, *FP Emulation*, Fourier, Descomposición LU y Redes Neuronales. Nbench compara las iteraciones por segundo del nodo con dos sistemas (un Dell Pentium 90 con 256 KB de cache y un AMD K6/233 con 512 KB de cache).

Phoenix: *Benchmark* basado en modelo *MapReduce* para tareas de procesamiento intensivo de datos, realizando numerosas operaciones de disco (leer, escribir, re-lectura, re-escritura), con 7 aplicaciones de áreas como el procesamiento de

imágenes e Inteligencia Artificial**.** Se ejecutó la versión secuencial de 3 aplicaciones que operan con datos generados al azar, Kmeans sobre 100.000 puntos con 100 K clústeres, PCA (*Principal Component Analysis*) con matriz de 5000 x 5000 y Multiplicación de Matriz con matrices de 2000 x 2000, y los otros 4, Histograma (RGB de una imagen), Regresión Lineal, coincidencia de secuencias de palabras en textos (*String Match*) y contar número de apariciones de palabras (*Word Count*), se ejecutaron sobre un conjuntos de datos proveído para cada aplicación de tamaño 1406 MB (large.bmp), 500 MB, 500 MB y 100 MB respectivamente.

6.2 MODELAMIENTO DEL CONSUMO DE ENERGÍA

El modelado del consumo de energía se realizó al nivel de nodos, que tiene en cuenta los elementos más importantes para el análisis de la eficiencia energética en sistemas HPC, sin incluir factores ambientales (como la temperatura, humedad, etc.), y depende de la asignación de tareas en las máquinas. Por lo tanto, el modelo de energía se deriva de la medición de la utilización de recursos e información de la tarea ejecutada.

Los *n* recursos de los nodos tienen diferentes métricas para cada modo de uso de este, y juntos conforman un vector que representa la cantidad de operaciones (carga) o porcentaje de uso (en cada estado):

$$Uso\_recursos = \left(r_{11}, \dots, r_{1y}, \dots, r_{n1}, \dots, r_{nz}\right) \quad (6)$$

Las métricas de utilización tomadas en cuenta son de los recursos CPU, memoria, disco y red, en la unidad de medida correspondiente (porcentaje de uso de CPU, Bytes de RAM, Bytes/seg en operaciones de I/O de disco o red), de esta forma, cada nodo se puede analizar individualmente. Los modelos se obtuvieron mediante la siguiente metodología de trabajo.

6.2.1 Flujo de trabajo

6.2.1.1 Recopilación de datos

Con el propósito de obtener los parámetros del modelo de energía del clúster, se debe recopilar métricas de la ejecución de aplicaciones y mediciones del *power* de los nodos, observaciones realizadas con las herramientas IPMI (*Intelligent Platform Management Interface*) y Dstat.

El consumo de energía se recopila con la aplicación IPMI, que puede monitorear los sensores del EnergyCore SoC. Usando un script en Python se ejecutan las sentencias de IPMI necesarias para acceder a la información de los sensores de los nodos determinados, especificando solo los datos que se desean recopilar (*power, epoch,* etc.).

Las métricas de cada uno de los tipos de recursos se recopilaron por medio de la herramienta de estadísticas de recursos Dstat[13]:

-CPU (%): usuario, sistema, ocioso (*idle*), espera (por operaciones de I/O)
-Memoria (Bytes): usada, *buffered, cached*, libre
-Disco (Bytes/s): lectura, escritura
-Red (Bytes/s): recibir, enviar

La ejecución de los *benchmarks* se realiza sobre los nodos de computación Viridis101 y Viridis102, si usa memoria distribuida, con la configuración presentada en el párrafo a continuación.

En el script que controla todo el proceso de benchmarking, se establece en 4 el número de ejecuciones por cada *benchmark*. El script se lanza desde el *front-end*

[13] Software para estadística de recursos, disponible en http://dag.wiee.rs/home-made/dstat/

del clúster Viridis, y activa un IPMI *deamon* (en segundo plano del *front-end*) en cada ejecución del *benchmark*, el cual realiza lecturas de los sensores de los nodos monitoreados cada 5 segundos.

Luego el script ejecuta el script de cada *benchmark*, donde se definen los parámetros de las variables propias del *benchmark* (para los nodos usados) y los archivos de entrada si se requieren. Justo antes de empezar la ejecución del *benchmark*, la herramienta Dstat se lanza en segundo plano para recoger estadísticas de uso de recursos de los nodos empleados. Las lecturas de Dstat se toman cada segundo durante mínimo 30 segundos de actividad del *benchmark*.

Se puede ver el progreso de las ejecuciones de los *benchmarks* mientras se van guardando sus resultados en el archivo de salida. Al final de todas las ejecuciones se detiene el monitoreo de datos del IPMI *deamon* y de la herramienta Dstat, que son guardados en los archivos respectivos ubicados en el *home* del usuario en el NFS compartido.

6.2.1.2 Pre-procesamiento

Las ejecuciones generan un total de 45721 registros (observaciones), que se pre-procesan (usando Java) para tener los archivos de salida de IPMI y Dstat en una disposición adecuada para el modelamiento, uniendo todos los archivos en uno solo por ejecución de forma sincronizada según el tiempo (*epoch*), además de estandarizar los datos y agregar información categórica o del entorno como el nombre del *benchmark*, nombre del nodo, y numero de nodos y *cores* usados. Los recursos más utilizados por los benchmarks según los datos recopilados del *millicluster* se indican en la Tabla 9.

Tabla 9: Clasificación de los *benchmarks* según uso de recursos del *millicluster*

Recurso	Benchmark
CPU	Whetstone Linpack HPL Coremark NBench Fhourstone Phoenix Benchmark
Memoria	HPL, Phoenix (memoria cached)
Red	OSU Micro-Benchmarks, Phoenix (String Match y Regresión Lineal)
Disco	Coremark, Phoenix (Kmeans, String Match y Word Count)

OSU (latencia y ancho de banda) se enfoca en el uso de la red, por lo que realiza muchas más operaciones de enviar y recibir de datos que los demás *benchmarks*, y realiza pocas operaciones de I/O de disco y hace un uso mínimo de la memoria con 85,6% aproximadamente en estado *libre*. OSU tiene casi ociosa (*idle*) la CPU, 94% y 92,6% para latencia y ancho de banda respectivamente, superado solo por el 98,86% obtenido con el *millicluster* en estado ocioso, donde solo se realizó el monitoreo de los nodos con IPMI y Dstat, por lo que únicamente el 0,71% es CPU *usuario* y el 0,22% es CPU *sistema*. Todos los *benchmarks* hacen un uso de CPU muy similar (cercano al 25%) excepto OSU (Ver Figura 6), Coremark usando solo un procesador tiene 25,02% en CPU *usuario*, y utilizando el doble de carga en 2 procesadores del nodo en paralelo fue casi el doble (50,2%).

Los *benchmarks* que realizan más operaciones de I/O en Disco son Kmeans, String Match y Word Count, que pertenecen a Phoenix, además de Coremark. Entre los 8 *benchmarks* de Phoenix, solo String Match y PCA tienen un bajo uso de memoria (83,6% y 91,3% memoria *libre* respectivamente). HPL (usando 2 nodos) es el *benchmark* que requiere más memoria (Memoria usada 44%), por lo que está entre los de menos memoria *libre*, junto a la mayoría de *benchmarks* de Phoenix. También

tiene un uso de red considerable, debido al manejo de las comunicaciones (sincronización de tareas en paralelo) entre 2 nodos.

El consumo de energía al ejecutar los *benchmarks* es muy similar, variando en promedio solo entre 4,06 W cuando esta ocioso y 4,99 W con PCA de Phoenix, llegando a un máximo de 6,1 W con HPL.

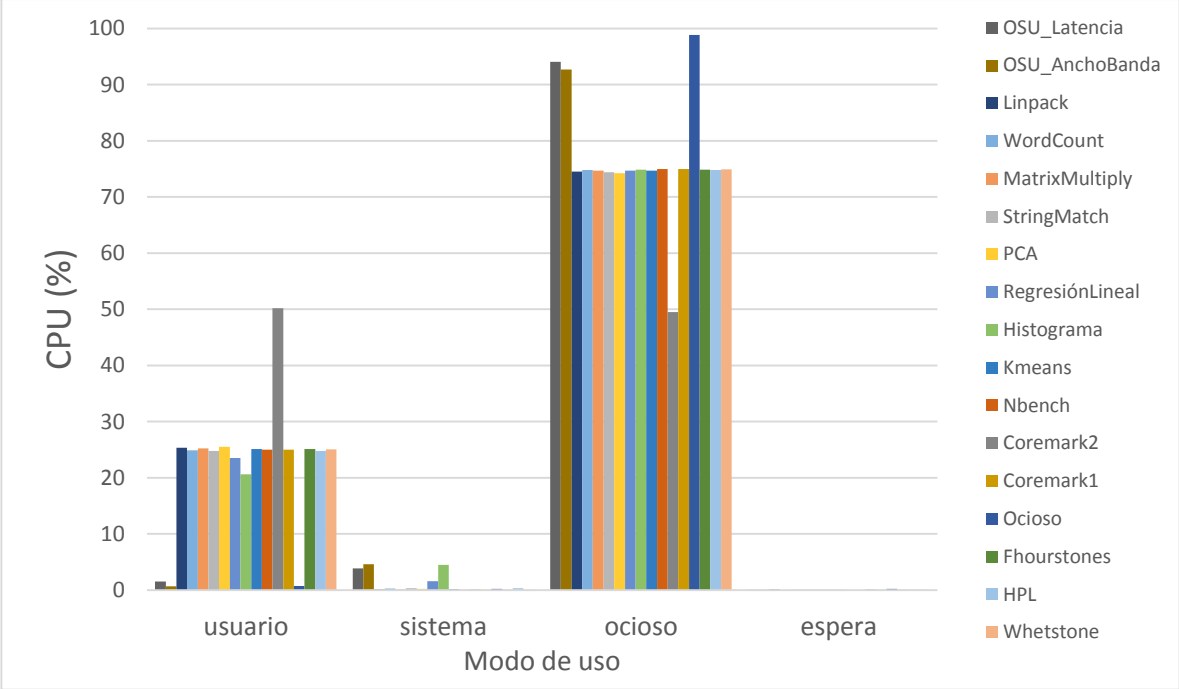

Figura 6: Uso de CPU durante los *benchmarks*

El análisis estadístico de los datos se hizo con la herramienta R[14], importando el paquete Zoo. La intensidad y dirección de la relación lineal de las variables explicativas con el *power* se pueden evaluar con el Coeficiente de correlación de Pearson *r* (ver Tabla 10), que es independiente de la escala de medida de las variables por lo que se ubica entre -1 y 1.

[14] Software para estadística informática, disponible en www.r-project.org

Tabla 10: Correlación entre el *power* y las demás variables

Recurso	Componente	Correlación
CPU	Usuario	0,345850914
	Sistema	0,125160933
	Ocioso	-0,434363295
	Espera	-0,109210512
Memoria	Usada	0,61196202
	Buffered	0,184552767
	Cached	-0,276555575
	Libre	-0,432251964
Disco	Lectura	0,001060889
	Escritura	-0,015088856
Red	Recibir	0,095604122
	Enviar	0,094414862
Ambiente	Nodos	0,476219635
	Cores	-0,06132666

Las variable Disco lectura tiene una correlación casi nula con el *power* (es decir, no existe relación lineal), en cambio, Memoria usada tiene alta correlación positiva (relación directa) y CPU ociosa y Memoria libre tienen alta correlación negativa (relación inversa). Revisando la correlación entre las demás parejas de variables, se puede ver que Memoria libre tiene -0,83 con Memoria usada, Disco lectura tiene nula correlación con todos, solo tiene coherentemente algo con 0,05 con Disco escritura y 0,01 con CPU espera (aumenta cuando se hacen operaciones de Disco). Nodos está altamente relacionado con Memoria usada 0,87, porque se usa más memoria (aunque el doble está a disposición), y *cores* está relacionado con CPU usada 0,31, porque se usa más CPU al activar más *cores* del nodo.

6.2.1.3 División del *dataset*

Luego del pre-procesamiento de los datos, se separan en dos *datasets* independientes con el propósito de modelar y predecir el consumo de energía (según la tarea y máquina usada). Las ejecuciones de cada *benchmark* son divididas en *datasets* de entrenamiento y prueba, mediante dos métodos de

selección de datos, muestreo secuencial y aleatorio. Usando toda la información disponible, se determinó los mejores tamaños para los *datasets*, según el coeficiente de determinación R^2 del *dataset* de entrenamiento y el error de predicción con el *dataset* de prueba (Ver Figura 7).

6.2.1.4 Selección del modelo de predicción

Utilizando el método de regresión múltiple se evalúan diferentes modelos usando el mejor *dataset* de entrenamiento y prueba. Regresión múltiple permite condensar las relaciones lineales de todos los predictores con la variable a modelar, y tiene la ventaja de tener baja complejidad computacional [47].

El modelo está en función de las variables numéricas y categóricas recopiladas en el *dataset* de entrenamiento, con el cual se calculan los coeficientes para cada variable. La Ecuación 7 es la regresión donde P es el *power* obtenido con los valores de las variables $x_1, \dots, x_k$, β_0 es el intercepto de la función lineal y los demás coeficientes $\beta_1, \dots, \beta_k$ son las pendientes.

$$P(y \mid x_1, \dots, x_k) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k \qquad (7)$$

Utilizando diferentes métricas de evaluación se puede verificar la calidad de los modelos. El nivel de adaptación del modelo a los datos del *dataset* de entrenamiento se calcula con el coeficiente de determinación R^2. En este indicador estadístico se tiene en cuenta los residuos del modelo, que son la diferencia entre el valor observado de la variable y el valor predicho por el modelo. El error de predicción del modelo es el promedio de los residuos obtenidos del *power* estimado en el *dataset* de prueba.

Con muestreo secuencial se divide por ejecuciones completas, por lo tanto, el *dataset* de entrenamiento consiste en 3 ejecuciones mientras que el *dataset* de

prueba es la ejecución restante, obteniendo 4 combinaciones posibles de las ejecuciones (Ver Tabla 11). El otro método de división del *dataset* es muestreo aleatorio, donde 3/4 de las observaciones son destinadas al *dataset* de entrenamiento, mientras que con el resto se conforma el *dataset* de prueba. Así se tienen *datasets* con cantidad de observaciones muy similares a la de muestreo secuencial.

Asignando una cantidad menor de observaciones al *dataset* de entrenamiento (solo 25% o 50% de las observaciones), se obtienen valores de R^2 promedio muy similares pero los errores relativos de predicción se acercan más o superan el 1%, por lo que es un modelo menos apto para nuevos datos de prueba (Ver Figura 7). Con una mayor cantidad de observaciones en el *dataset* de entrenamiento, se presentaría el problema contrario, sobreajuste *(overfitting)* a los datos evidente en un R^2 mejor, por lo que igualmente no permite predecir con nuevos datos de prueba. En consecuencia, los modelos con 75% de datos para entrenamiento, son los más aptos para la predicción del *power*.

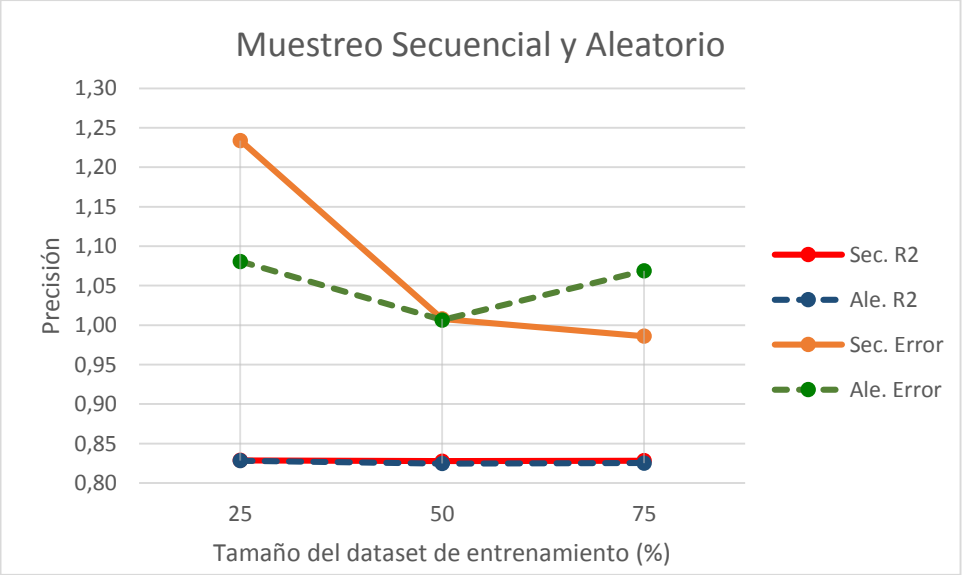

Figura 7: Comparación de tamaños del *dataset* de entrenamiento y prueba según el R^2 y Error de predicción respectivamente.

El muestreo secuencial resulto un poco mejor para dividir el *dataset* que el muestro aleatorio. El muestreo secuencial tiene en el promedio de la mejor combinación posible, R^2 de 0,8284 y error absoluto relativo de 0,986% (Ver Tabla 11), mientras que muestreo aleatorio tiene en el promedio de 10 divisiones, un peor R^2 (0,8247) y peor error absoluto relativo (1,0069%).

Tabla 11: Combinaciones de ejecuciones en muestreo secuencial

TRAIN (75%)	**R^2**	**TEST (25%)**	**|Error| (Relativo%)**
135	0,8317	4	0,9925
134	0,8227	5	1,0380
145	0,8276	3	0,9488
345	**0,8317**	1	**0,9636**
Promedio	0,8284		0,986

El *dataset* tiene variables de tipo numérico, que son las métricas de uso del nodo, número de *cores* y nodos activos, y variables categóricas son información adicional de los datos recopilados, como el nombre del *benchmark* (tipo de *workload*) y el nombre del nodo. Varios modelos se obtienen con las variables agrupadas de las siguientes maneras:

- Básico: Todas las variables disponibles (categóricas y numéricas).
- Sin fase: Todas las variables excepto la información del nombre del *benchmark* (o fase del *workload*).
- Sin grupo: Solo las variables numéricas.
- Solo grupo: Solo las variables categóricas que agrupan los *benchmarks* y los nodos.
- CPU Homogénea: Todas las variables excepto el nombre del nodo, y solo se tiene en cuenta CPU usuario y CPU ociosa en las variables numéricas de CPU.

La calidad de los modelos propuestos se compara en la Tabla 12, usado la mejor

combinación del muestreo secuencial, que tiene las ejecuciones 345 y 1 en *dataset* de entrenamiento y prueba respectivamente (Ver Tabla 11). La Tabla 12 presenta la distribución de residuos del *dataset* de entrenamiento, con los valores máximos (Max), mínimos (Min), la mediana y el primer cuartil (1Q) y tercer cuartil (3Q), e incluyendo la Raíz del Error Cuadrático Medio (RMSE por sus siglas en inglés). El valor promedio del error de predicción (Error) y su valor absoluto (|Error|) son presentados en Watts, junto a sus errores relativos porcentuales (%) del *dataset* de prueba.

Tabla 12: Calidad de los modelos obtenidos con muestreo secuencial

Modelo	**R^2**	**Residuos**						**|Error|**		**Error**	
		RMSE	Min	1Q	Mediana	3Q	Max	Watt	%	Watt	%
Básico	**0,8317**	**0,1210**	**-1,3249**	**-0,0293**	**0,0043**	**0.01638**	**1,3426**	**0,0470**	**0,9636**	**-0,0126**	**-0,1924**
Sin Fase	0,6961	0,1626	-1,3297	-0,0354	-0,0050	0,0174	1,3388	0,0822	1,7260	-0,0124	-0,1334
Sin Grupo	0,6923	0,1636	-1,3239	-0,0333	-0,0096	0,0013	1,3525	0,0816	1,7130	-0,0078	-0,0389
Solo Grupo	0,7321	0,1527	-0,9850	-0,0356	0,0019	0,0019	1,3451	0,0631	1,3140	-0,0013	0,0774
CPU Hom.	0,8317	0,1210	-1,3264	-0,0299	-0,0039	0,0163	1,3434	0,0467	0,9587	-0,0095	-0,1316

Los indicadores de calidad de los modelos presentan mejores valores en el modelo que incluye todos los predictores disponibles (modelo Básico), obteniendo el valor más alto para el R^2 (0,8317) y el más bajo para el Error Cuadrático Medio (0,1210), indicando que más información permite un modelamiento del sistema más preciso.

La baja calidad del modelo Sin fases señala la importancia de aportar la información del *benchmark* (fase del *workload*). El modelo de Sin fase tiene una calidad similar al modelo Sin grupo, apuntando a que la información del nombre del nodo no es significativa, lo que es acorde a un clúster de nodos homogéneos. El modelo de Solo grupo es el segundo modelo de mejor calidad, demostrando que al tener la información del *benchmark* y nodo, la información de las métricas de uso de recursos no es totalmente necesaria porque ya está representada por estas.

Por último, el modelo CPU Homogénea que no incluye la variable categórica del nombre del nodo y solo se tiene en cuenta CPU usuario y CPU ociosa, muestra que esa variable categórica no es indispensable cuando se tienen nodos homogéneos (solo existen pequeñas diferencias en *power* consumido con el mismo *workload*), y tampoco lo son los otros modos de CPU (sistema y espera) porque dependen del CPU usuario y CPU ociosa.

En el mejor modelo (Básico), los coeficientes muestran que el mayor incremento del consumo de energía del nodo se debe al nivel de trabajo del procesador (CPU *usuario, sistema, ocioso, espera*) (Ver Tabla 13). El modo de operación de más consumo de energía es CPU *usuario* como se esperaba, y los de menos consumo son CPU ociosa y CPU espera porque la CPU está en una frecuencia baja.

Los demás componentes del nodo influyen en el *power* en menor medida. La memoria en estado *buffered* tiene mayor consumo de energía que los otros estados (usada, cached y libre). En el estado *recibir* de la red se aumenta el consumo de energía un poco más que en el estado *enviar*. En operaciones de escritura y/o lectura en disco se reduce el consumo de energía, puede ser causada porque la CPU entra en un modo de frecuencia inferior durante las operaciones de escritura de las salidas de las aplicaciones, o por la espera de datos de entrada (*read*) para continuar el procesamiento.

Algunos coeficientes están en NA (*Not Available*) porque se presenta una fuerte correlación (lineal) entre variables explicativas del modelo (multi-colinealidad), haciendo que algunas sean innecesarias porque no aportan nueva información (lineal), ya que ha sido obtenida de las demás variables. Los coeficientes de los *benchmarks* muestran que tienen bastante peso para el cálculo del *power*, y que el nodo Viridis101 tiene una diferencia de 0,525W más que el Viridis102.

Tabla 13: Coeficientes del modelo para uso de recursos, fases y nodos

Inter-cepto	CPU				Memoria				Red		Disco		Nodo	*Core*
	Usua-rio	Sistema	Ociosa	Espe-ra	Usada	Buffered	Cached	Libre	Recibir	Enviar	Escri-tura	Lectu-ra		
5,274 E+00	1,482 E-03	-3,856 E-05	-1,799 E-02	-7,749 E-03	2,477 E-10	1,25 E-08	-2,012 E-10	NA	NA	-6,846 E-07	2,672 E-09	1,487 E-09	NA	NA

Phoenix							NBench	Coremark		Ocioso
Word Count	Matrix Multiply	String Match	PCA	Regresión Lineal	Histo-grama	Kmeans		2cores	1core	
3,380 E-01	1,012 E-01	-9,863 E-02	7,964 E-02	3,583 E-01	4,225 E-01	3,343 E-01	-7,052 E-01	0	-4,996 E-01	4,415 E-01

OSU Micro-Benchmarks		Linpack	HPL	Whetstone	Fhourstone	Viridis101	Viridis102
Latencia	Ancho de Banda						
5,613 E-01	5,157 E-01	2,993 E-01	3,619 E-01	-4,732 E-01	-1,908 E-01	0	-5,252 E-01

6.3 APLICACIÓN DEL MODELO

El mejor modelo logra estimar con buena precisión (0,96% de error de predicción) el consumo de energía de los nodos del *millicluster* estudiado, para una configuración especificada en función del tipo de *workload* y los recursos usados. Comparado con un trabajo similar realizado sobre *clusters* convencionales [47], se tiene un mejor error de predicción que el modelo más apto en ese trabajo (que es de aproximadamente 4%), a pesar de que obtuvieron un R^2 más alto (de 0,96). Con el modelo de energía se pueden generar instancias definiendo el uso de recursos del nodo, además del tipo de fase (*workload*) para el modelo más preciso.

6.3.1 Adaptación al problema de *Job Scheduling* basado en el modelo ETC

La comparación de la eficiencia energética de los algoritmos de calendarización de tareas en *milliclusters,* se realiza implementando el modelo de energía propuesto en el escenario de *job scheduling* basado en el modelo computacional ETC. Para

esto se define la cantidad de máquinas de cada nodo y utilizando el generador de instancias del modelo ETC [34], se obtiene una instancia tipo consistente de alta heterogeneidad de las tareas y baja heterogeneidad de las máquinas, c_hilo, del tamaño definido por la instancia de tareas (*workload* o uso de recursos) en *millicluster* y el número de máquinas determinadas.

El consumo de energía (Joules) se estima según el *power* calculado para la tarea en la máquina seleccionada (tipo de nodo), y el tiempo de ejecución (ETC) en *job scheduling* de la tarea en esa máquina, de la siguiente forma:

$$E_{tm} = P_{tm} * ETC_{tm} \quad (8)$$

Cada tarea es un registro estimado de uso de recursos de una aplicación, nosotros usamos un conjunto de datos seleccionados aleatoriamente del *dataset* de prueba, y se calcula el *power* con el modelo de energía.

La función de costo *score* anteriormente presentada, que relaciona el consumo de energía y makespan del calendario obtenido, con un alfa equilibrado (0.5) su inverso representa la eficiencia energética, es decir, reducir el score significa aumentar la eficiencia energética y viceversa. Por lo tanto, el objetivo de los algoritmos es minimizar el score (reducir el consumo de energía y makespan total), que sería maximizar la eficiencia energética (Ver Ecuación 3). Se debe normalizar los valores de cada métrica porque tienen diferentes unidades de medida, por lo que el valor de *makespan* y energía normalizados estarán en el intervalo [0,1], donde 1 es el peor valor de las métricas. De esta forma en el próximo capítulo se analiza la eficiencia energética de algoritmos de calendarización de tareas en *milliclusters.*

7. ANALISIS DE RESULTADOS

La salida del algoritmo es la secuencia de trabajos (J) que conforman las operaciones (O) de cada máquina, junto al tiempo de ejecución estimado (ETC) y energía consumida acumulada al asignar un trabajo (si esta es estimada). Por ejemplo, la Tabla 6 tiene la mejor solución encontrada por *Parallel Micro CHC* para la instancia de ejemplo de 9 tareas x 3 máquinas (Ver Tabla 14), con makespan de 27 unidades de tiempo, indicado por el tiempo de finalización más largo de las máquinas, que en este caso es igual para las tres máquinas.

Tabla 14: Calendario de trabajos en las máquinas

Máquinas	**O_1**	**O_2**	**O_3**	**O_4**	**O_4**	**O_4**	**Tiempo de Finalización**
M1	J2 (2)	J3 (3)	J4 (4)	J5 (5)	J6 (6)	J7 (7)	27
M2	J1 (10)	J8 (17)					27
M3	J9 (27)						27

7.1 COMPARACIÓN DE ALGORITMOS SEGÚN MAKESPAN

Los algoritmos se pueden comparar respecto al makespan usando las instancias del modelo ETC, muy comunes en el estado del arte de *job scheduling*. Los resultados se obtuvieron al realizar 10 ejecuciones del algoritmo *Parallel Micro CHC* por cada una de las 12 instancias ETC generadas por Braun et al. [8] (Ver Tabla 15), una instancia representativa por cada uno de los diferentes escenarios del modelo ETC. El algoritmo *Parallel Micro CHC* original [37] que genera la mejor solución conocida (BKS por sus siglas en inglés, de *Best Knowed Solution*) hasta ahora (Ver Tabla 5).

Para las ejecuciones de *Parallel Micro CHC* se determinó el mejor makespan y se calculó el promedio y desviación estándar del makespan en 10 ejecuciones. En la Tabla 15 se muestra el mejor makespan encontrado con *Parallel Micro CHC* y su error relativo porcentual respecto al BKS, que es en promedio de 9,2%. Nuestra versión del *Parallel Micro CHC* no pudo repetir los mejores resultados de la versión original, aunque si logra superar todas las soluciones obtenidas con Min Min y Max Min.

Tabla 15: Mejor *makespan* de los algoritmos

Instancia 512x16	Min Min [32]	Max Min [32]	BKS Parallel Micro-CHC [37]	Parallel Micro-CHC	ER(%) vs BKS
u_c_hihi.0	8.460.674	12.385.672	7.381.570	8.224.917,50	11,4
u_c_hilo.0	161.805	204.055	153.105	159.693,00	4,3
u_c_lohi.0	275.837	392.567	239.260	266.383,38	11,3
u_c_lolo.0	5.441	6.945	5.148	5.361,48	4,1
u_i_hihi.0	3.513.919	8.018.378	2.938.381	3.391.661,20	15,4
u_i_hilo.0	80.756	151.924	73.378	78.578,28	7,1
u_i_lohi.0	120.518	251.529	102.051	113.500,58	11,2
u_i_lolo.0	2.786	5.178	2.541	2.673,86	5,2
u_s_hihi.0	5.160.343	9.208.811	4.103.500	4.729.680,00	15,3
u_s_hilo.0	104.375	172.823	95.787	101.894,17	6,4
u_s_lohi.0	140.285	282.086	122.083	136.420,22	11,7
u_s_lolo.0	3.807	6.232	3.434	3.659,51	6,6
Promedio	1.502.545	2.590.517	1.268.353	1434535	9,18

Agrupando las instancias por el nivel de heterogeneidad de la tarea y máquina respectivamente (*high* o *low*), se puede observar que la menor calidad de las soluciones se presenta para instancia con alta heterogeneidad de la máquina (casos hihi y lohi), que sobrepasan el 10% en promedio de error relativo porcentual (Ver Figura 8). En cambio, agrupando por tipo de consistencia, no se presentan grandes diferencias, con error promedio de 7,8%, 9,7% y 10% para instancias consistentes, inconsistentes y semi-consistentes respectivamente.

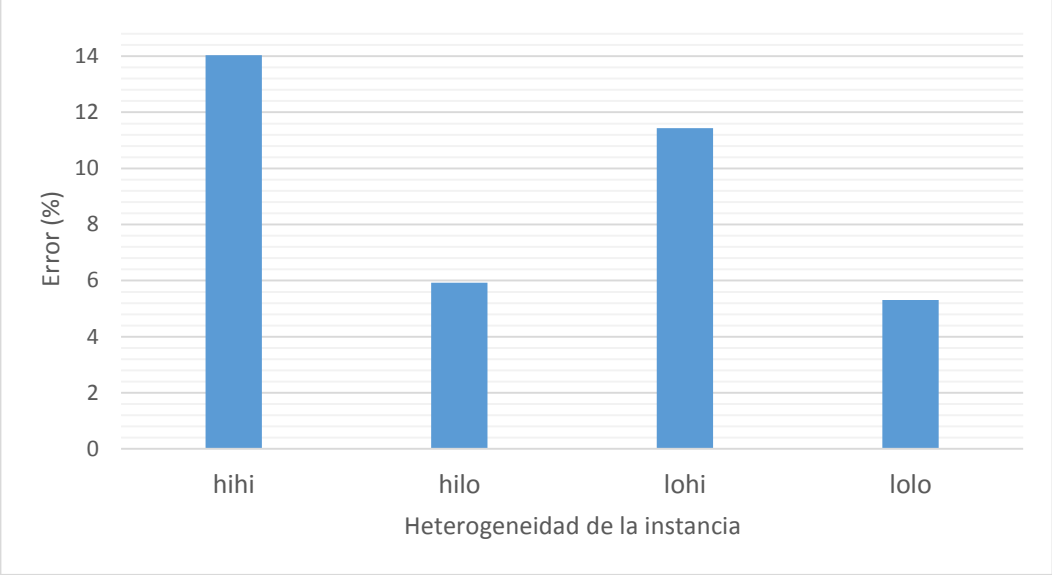

Figura 8: Error relativo según heterogeneidad de la tarea y máquina

La complejidad es muy alta debido al considerable número de soluciones factibles, por lo que se obtienen soluciones de una menor calidad. Aunque la eficacia de *Parallel Micro CHC* para encontrar la mejor solución conocida no es alta, obtiene soluciones aproximadas de buena calidad para problemas de *job scheduling*.

7.2 COMPARACIÓN DE LA EFICIENCIA ENERGÉTICA DE LOS ALGORITMOS EN *MILLICLUSTERS*

Se obtuvo una instancia ETC de 32 tareas x 8 máquinas utilizando el generador de instancias del modelo ETC [34], que para el *millicluster* modelado debe ser de alta heterogeneidad de las tareas y baja heterogeneidad de las máquinas, y si es una instancia del tipo consistente se representa como c_hilo, con tiempos de procesamiento de tipo punto flotante.

La instancia de 32 tareas de uso de recursos (*workload*) de los nodos del *millicluster*, se recopiló aleatoriamente del *dataset* de prueba, y mediante el modelo se estima el *power* de cada tarea en cada nodo. Las 8 máquinas son de los dos nodos del

millicluster, la mitad por cada nodo. El consumo de energía (*Joules*) se estima con el *power* calculado para la tarea en el nodo de la máquina asignada, y el tiempo de ejecución (ETC) de la tarea en esa máquina.

La eficiencia energética del algoritmo *Parallel Micro CHC*, estimada con el makespan y consumo de energía, se compara con la mejor heurística identificada Min Min, y con Max Min, todos adaptados para maximizar la eficiencia energética (minimizar el *score*). Con *Parallel Micro CHC* se logra más eficiencia (menor *score* de 10 ejecuciones), con menos consumo de energía y menor makespan en el calendario de tareas más equilibrado (Ver Tabla 16).

Tabla 16: Eficiencia energética de los algoritmos

Instancia 32x8 B.u_c_hilo	**Makespan (seg)**	**Energía (Joules)**	***Score***
Min Min	20750,6	401829,4	0,74
Max Min	22485,7	724268,2	1
Parallel Micro-CHC	18837,4	393044,8	0,69

Parallel Micro CHC es un algoritmo que tiene la capacidad para obtener soluciones de buena calidad en instancias de *job scheduling*, cuando se enfoca simultáneamente en la reducción del consumo de energía y makespan, logra una mejor eficiencia que las mejores heurísticas.

8. CONCLUSIONES

Actualmente la eficiencia energética es un aspecto primordial para el desarrollo de los sistemas de Computación de Alto Rendimiento (HPC), por consiguiente, se evalúan algoritmos de calendarización de tareas en una plataforma computacional de bajo consumo de energía (*millicluster*), buscando optimizar el uso de recursos y consumo de energía. Para este propósito, se desarrolló un modelo de energía que permite estimar con precisión el consumo de energía de tareas ejecutadas sobre nodos del *millicluster*.

La complejidad del problema de calendarización de tareas lleva a abordarlo con métodos aproximados, que encuentran soluciones de buena calidad sin realizar búsquedas exhaustivas. Entre ellos se encuentran numerosos y diversos métodos heurísticos y metaheurísticas, que fueron comparados para el problema de *job scheduling* usando las instancias más comunes del modelo ETC. En este escenario el mejor método identificado según el makespan es el algoritmo evolutivo *Parallel Micro CHC*, y respecto a la eficiencia energética Min Min se destaca entre las heurísticas.

Se implementaran los algoritmos de calendarización de tareas *Parallel Micro CHC* y Min Min, adaptándolos para reducir el makespan y consumo de energía simultáneamente en la gestión de tareas de un *millicluster*. El algoritmo *Parallel Micro CHC* implementado tiene un buen rendimiento, con soluciones de mejor makespan que la heurística más destacada (Min Min) en las instancias más comunes del modelo ETC, aunque no logra alcanzar los mejores resultados obtenidos por el algoritmo *Parallel Micro CHC* original. En la calendarización de tareas del *millicluster*, *Parallel Micro CHC* logra menor makespan y consumo de energía que Min Min, por lo que obtiene mayor eficiencia energética.

9. RECOMENDACIONES

Las principales líneas para trabajo futuro incluyen comparar la eficiencia energética en calendarización de tareas del *millicluster* con otras plataformas HPC (basadas en ARM) de tecnología avanzada de eficiencia energética.

Diseñar un algoritmo de *Parallel Micro CHC* que permita conseguir un mejor equilibrio en todos los tipos de instancias de *job scheduling* del modelo ETC, evitando la convergencia prematura del algoritmo. Para tal fin se pueden añadir otros mecanismos de reinicio de la búsqueda, por ejemplo, realizar mutaciones de forma selectiva, para explorar regiones del espacio de búsqueda poco frecuentados. Y es pertinente aprovechar plenamente su mecanismo de migraciones, para diversificar la búsqueda cuando se estanca en óptimos locales.

REFERENCIAS BIBLIOGRÁFICAS

[1] Pinel, F., Pecero, J. E., Khan, S. U., & Bouvry, P. (2011, August). Energy-efficient scheduling on milliclusters with performance constraints. In Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications (pp. 44-49). IEEE Computer Society.

[2] Pinel, F., Dorronsoro, B., Pecero, J. E., Bouvry, P., & Khan, S. U. (2013). A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids. Cluster computing, 16(3), 421-433.

[3] Izakian, H., Abraham, A., & Snasel, V. (2009, April). Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In International Joint Conference on Computational Sciences and Optimization (Vol. 1, pp. 8-12). IEEE Computer Society.

[4] He, X., Sun, X., & Von Laszewski, G. (2003). QoS guided min-min heuristic for grid task scheduling. Journal of Computer Science and Technology, 18(4), 442-451.

[5] Iqbal, S., Gupta R. & Lang Y. (2005). Job scheduling in HPC clusters. Power Solutions, 133–135.

[6] Dutot, P. F., Eyraud, L., Mounié, G., & Trystram, D. (2004, June). Bi-criteria algorithm for scheduling jobs on cluster platforms. In Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures (pp. 125-132). ACM.

[7] Pinel, F., & Bouvry, P. (2011). A Model for Energy-efficient Task Mapping on Milliclusters. In Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering. Civil-Comp Press.

[8] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., ...& Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed computing, 61(6), 810-837.

[9] Diaz, C. O., Guzek, M., Pecero, J. E., Danoy, G., Bouvry, P., & Khan, S. U. (2011, July). Energy-aware fast scheduling heuristics in heterogeneous computing systems. In International Conference on High Performance Computing and Simulation (pp. 478-484). IEEE Computer Society.

[10] Leung, J. Y. (Ed.). (2004). Handbook of scheduling: algorithms, models, and performance analysis. CRC Press.

[11] Ali, S., Braun, T. D., Siegel, H. J., Maciejewski, A. A., Beck, N., Bölöni, L., ...& Yao, B. (2005). Characterizing resource allocation heuristics for heterogeneous computing systems. Advances in computers, 63, 91-128.

[12] Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys (CSUR), 35(3), 268-308.

[13] Valentini, G. L., Lassonde, W., Khan, S. U., Min-Allah, N., Madani, S. A., Li, J., ...& Bouvry, P. (2013). An overview of energy efficiency techniques in cluster computing systems. Cluster Computing, 16(1), 3-15.

[14] Hussain, H., Malik, S. U. R., Hameed, A., Khan, S. U., Bickler, G., Min-Allah, N., ...& Rayes, A. (2013). A survey on resource allocation in high performance distributed computing systems. Parallel Computing, 39(11), 709-736.

[15] Diaz, C. O., Guzek, M., Pecero, J. E., Bouvry, P., & Khan, S. U. (2011, August). Scalable and energy-efficient scheduling techniques for large-scale systems. In 11th International Conference on Computer and Information Technology (pp. 641-647). IEEE Computer Society.

[16] Barrondo, A., Tchernykh, A., Schaeffer, E., & Pecero, J. (2012, July). Energy efficiency of knowledge-free scheduling in peer-to-peer desktop Grids. In International Conference on High Performance Computing and Simulation (pp. 105-111). IEEE Computer Society.

[17] Diaz, C.O., Pecero, J.E., Bouvry, P. (2014). Scalable, low complexity, and fast greedy scheduling heuristics for highly heterogeneous distributed computing systems. J. Supercomputing 67(3), 837–853.

[18] Dong, F., & Akl, S. G. (2006). Scheduling algorithms for grid computing: State of

the art and open problems. School of Computing, Queen's University, Kingston, Ontario.

[19] Lindberg, P., Leingang, J., Lysaker, D., Bilal, K., Khan, S. U., Bouvry, P., ...& Li, J. (2011). Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids. Energy-Efficient Distributed Computing Systems, 189-214.

[20] Xhafa F, Abraham A (2010) Computational models and heuristic methods for grid scheduling problems. Future generation computer systems, 26(4), 608–621

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

[22] Gao, Y., Rong, H., & Huang, J. Z. (2005). Adaptive grid job scheduling with genetic algorithms. Future Generation Computer Systems, 21(1), 151-161.

[23] Carretero, J., Xhafa, F., & Abraham, A. (2007). Genetic algorithm based schedulers for grid computing systems. International Journal of Innovative Computing, Information and Control, 3(6), 1-19.

[24] Xhafa, F., Alba, E., Dorronsoro, B., Duran, B., & Abraham, A. (2008). Efficient batch job scheduling in grids using cellular memetic algorithms. In Metaheuristics for Scheduling in Distributed Computing Environments (pp. 273-299). Springer Berlin Heidelberg.

[25] Chang, R. S., Chang, J. S., & Lin, P. S. (2009). An ant algorithm for balanced job scheduling in grids. Future Generation Computer Systems, 25(1), 20-27.

[26] Colorni, A., Dorigo, M., Maniezzo, V., & Trubian, M. (1994). Ant system for job-shop scheduling. Belgian Journal of Operations Research, Statistics and Computer Science, 34(1), 39-53.

[27] Stützle, T., & Hoos, H. (2000). MAX–MIN ant system. Future generation computer systems, 16(8), 889-914.

[28] Liu, H., Abraham, A., & Hassanien, A. E. (2010). Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future

Generation Computer Systems, 26(8), 1336-1343.

[29] Xhafa, F., Carretero, J., Dorronsoro, B., & Alba, E. (2009). A tabu search algorithm for scheduling independent jobs in computational grids. Computing and Informatics, 28, 237-250.

[30] Kirkpatrick, S., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680.

[31] Ali, S., Siegel, H. J., Maheswaran, M., Hensgen, D., & Ali, S. (2000). Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang Journal of Science and Engineering, 3(3), 195-208.

[32] Xhafa, F., Barolli, L., Durresi, A. (2007). Batch mode scheduling in grid systems. International Journal of Web and Grid Services, 3(1), 19-37

[33] Nesmachnow, S., Cancela, H., Alba, E. (2010). Heterogeneous computing scheduling with evolutionary algorithms. Soft Computing, 15(4), 685-701.

[34] Xhafa, F. (2007). A hybrid evolutionary heuristic for job scheduling on computational grids. In Hybrid Evolutionary Algorithms, pp. 269-311.

[35] Xhafa, F., Carretero, J., Alba, E., Dorronsoro, B. (2008). Design and evaluation of tabu search method for job scheduling in distributed environments. In Proceedings of the 22th International Parallel and Distributed Processing Symposium, pp. 1–8.

[36] Ritchie, G., Levine, J. (2004). A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, pp. 178–183.

[37] Nesmachnow, S., Cancela, H., Alba, E. (2012). A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. Applied Soft Computing, 12(2), 626-639.

[38] Pinel, F., Dorronsoro, B., Bouvry, P. (2010). A new parallel asynchronous cellular genetic algorithm for scheduling in grids. In Parallel Distributed Processing, Workshops and PhD Forum, 2010 IEEE International Symposium. pp. 1-8.

[39] Bardsiri, A. K., Hashemi, S. M. (2012). A Comparative Study on Seven Static Mapping Heuristics for Grid Scheduling Problem. International Journal of Software Engineering and Its Applications, 6(4), 247-256.

[40] Guzek, M., Pecero, J. E., Dorronsoro, B., Bouvry, P. (2014). Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems. Applied Soft Computing, 24, 432-446.

[41] Coello, C., Pulido, G. (2001). A micro-genetic algorithm for multiobjective optimization. In Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, pp. 126–140.

[42] Plugaru, V., Varrette, S., & Bouvry, P. (2014). Performance Analysis of Cloud Environments on Top of Energy-Efficient Platforms Featuring Low Power Processors. In 6th International Conference on Cloud Computing Technology and Science (CloudCom'14). IEEE Computer Society.

[43] Jarus, M., Varrette, S., Oleksiak, A., & Bouvry, P. (2013). Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD and ARM processors. In Energy Efficiency in Large Scale Distributed Systems (pp. 182-200). Springer Berlin Heidelberg.

[44] Delplace, V., Manneback, P., Pinel, F., Varette, S., & Bouvry, P. (2013, September). Comparing the performance and power usage of gpu and arm clusters for map-reduce. In Third International Conference on Cloud and Green Computing (pp. 199-200). IEEE Computer Society.

[45] Frederic Pinel and Pascal Bouvry (2012). Energy-efficient data centers with Millicomputing opportunity and challenges. Symposium on Future Generations of Processors and Systems (FGPS'175).

[46] Varrette, S., Guzek, M., Plugaru, V., Besseron, X., & Bouvry, P. (2013, October). HPC Performance and Energy-Efficiency of Xen, KVM and VMware Hypervisors. In 25th International Symposium on Computer Architecture and High Performance Computing (pp. 89-96). IEEE Computer Society.

[47] Guzek, M., Varrette, S., Plugaru, V., Pecero, J. E., & Bouvry, P. (2013). A Holistic Model of the Performance and the Energy-Efficiency of Hypervisors in an HPC

Environment. In Energy Efficiency in Large Scale Distributed Systems (pp. 133-152). Springer Berlin Heidelberg.

[48] Varrette, S., Plugaru, V., Guzek, M., Besseron, X., & Bouvry, P. (2014). HPC Performance and Energy-Efficiency of the OpenStack Cloud Middleware. In 43rd International Conference on Parallel Processing (ICPP-2014). IEEE Computer Society.

[49] Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. Econometrica: Journal of the Econometric Society, 497-520.

[50] Flake, G. W. (1998). The computational beauty of nature: Computer explorations of fractals, chaos, complex systems, and adaptation. MIT press.

[51] Costa, GD., Hlavacs, H., Hummel KA & Pierson, JM. (2012). Modeling the energy consumption of distributed applications. Handbook of Energy-Aware and Green Computing, Ahmad, I., Ranka, S. (eds.). Chapman and Hall CRC.

[52] Economou, D., Rivoire, S., Kozyrakis, C., & Ranganathan, P. (2006). Full-system power analysis and modeling for server environments. IEEE International Symposium on Computer Architecture.

[53] Basmadjian, R., & de Meer, H. (2012, May). Evaluating and modeling power consumption of multi-core processors. In Third International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (pp. 1-10)

[54] Contreras, G., & Martonosi, M. (2005, August). Power prediction for intel XScale processors using performance monitoring unit events. In ISLPED'05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design (pp. 221-226).

[55] Witkowski, M., Oleksiak, A., Piontek, T., & Węglarz, J. (2013). Practical power consumption estimation for real life HPC applications. Future Generation Computer Systems, 29(1), 208-217.

[56] Jarus, M., Oleksiak, A., Piontek, T., & Węglarz, J. (2014). Runtime power usage estimation of HPC servers for various classes of real-life applications. Future Generation Computer Systems, 36, 299-310.

[57] Moraga, R., Whitehouse, G. & Depuy. G. (2013). Meta-raps: Un enfoque de solución eficaz para problemas combinatorios. Revista Ingeniería Industrial, 2(1).

[58] Introducing the Viridis 2.0: Boston Viridis, data sheet (2012). Disponible en URL: *http://www.boston.co.uk/solutions/viridis/introducing-the-viridis-2.aspx*

BIBLIOGRAFÍA

Ali, S., Braun, T. D., Siegel, H. J., Maciejewski, A. A., Beck, N., Bölöni, L., ...& Yao, B. (2005). Characterizing resource allocation heuristics for heterogeneous computing systems. Advances in computers, 63, 91-128.

Ali, S., Siegel, H. J., Maheswaran, M., Hensgen, D., & Ali, S. (2000). Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang Journal of Science and Engineering, 3(3), 195-208.

Bardsiri, A. K., Hashemi, S. M. (2012). A Comparative Study on Seven Static Mapping Heuristics for Grid Scheduling Problem. International Journal of Software Engineering and Its Applications, 6(4), 247-256.

Barrondo, A., Tchernykh, A., Schaeffer, E., & Pecero, J. (2012, July). Energy efficiency of knowledge-free scheduling in peer-to-peer desktop Grids. In International Conference on High Performance Computing and Simulation (pp. 105-111). IEEE Computer Society.

Basmadjian, R., & de Meer, H. (2012, May). Evaluating and modeling power consumption of multi-core processors. In Third International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (pp. 1-10).

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys (CSUR), 35(3), 268-308.

Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., ...& Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems.

Journal of Parallel and Distributed computing, 61(6), 810-837.

Carretero, J., Xhafa, F., & Abraham, A. (2007). Genetic algorithm based schedulers for grid computing systems. International Journal of Innovative Computing, Information and Control, 3(6), 1-19.

Chang, R. S., Chang, J. S., & Lin, P. S. (2009). An ant algorithm for balanced job scheduling in grids. Future Generation Computer Systems, 25(1), 20-27.

Coello, C., Pulido, G. (2001). A micro-genetic algorithm for multiobjective optimization. In Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, pp. 126–140.

Colorni, A., Dorigo, M., Maniezzo, V., & Trubian, M. (1994). Ant system for job-shop scheduling. Belgian Journal of Operations Research, Statistics and Computer Science, 34(1), 39-53.

Contreras, G., & Martonosi, M. (2005, August). Power prediction for intel XScale processors using performance monitoring unit events. In ISLPED'05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design (pp. 221-226).

Costa, GD., Hlavacs, H., Hummel KA & Pierson, JM. (2012). Modeling the energy consumption of distributed applications. Handbook of Energy-Aware and Green Computing, Ahmad, I., Ranka, S. (eds.). Chapman and Hall CRC.

Delplace, V., Manneback, P., Pinel, F., Varette, S., & Bouvry, P. (2013, September). Comparing the performance and power usage of gpu and arm clusters for map-reduce. In Third International Conference on Cloud and Green Computing (pp. 199-200). IEEE Computer Society.

Diaz, C. O., Guzek, M., Pecero, J. E., Bouvry, P., & Khan, S. U. (2011, August). Scalable and energy-efficient scheduling techniques for large-scale systems. In 11th International Conference on Computer and Information Technology (pp. 641-647). IEEE Computer Society.

Diaz, C. O., Guzek, M., Pecero, J. E., Danoy, G., Bouvry, P., & Khan, S. U. (2011, July). Energy-aware fast scheduling heuristics in heterogeneous computing systems. In International Conference on High Performance Computing and Simulation (pp. 478-484). IEEE Computer Society.

Diaz, C.O., Pecero, J.E., Bouvry, P. (2014). Scalable, low complexity, and fast greedy scheduling heuristics for highly heterogeneous distributed computing systems. J. Supercomputing 67(3), 837–853.

Dong, F., & Akl, S. G. (2006). Scheduling algorithms for grid computing: State of the art and open problems. School of Computing, Queen's University, Kingston, Ontario.

Dutot, P. F., Eyraud, L., Mounié, G., & Trystram, D. (2004, June). Bi-criteria algorithm for scheduling jobs on cluster platforms. In Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures (pp. 125-132). ACM.

Economou, D., Rivoire, S., Kozyrakis, C., & Ranganathan, P. (2006). Full-system power analysis and modeling for server environments. IEEE International Symposium on Computer Architecture.

Flake, G. W. (1998). The computational beauty of nature: Computer explorations of fractals, chaos, complex systems, and adaptation. MIT press.

Frederic Pinel and Pascal Bouvry (2012). Energy-efficient data centers with

Millicomputing opportunity and challenges. Symposium on Future Generations of Processors and Systems (FGPS'175).

Gao, Y., Rong, H., & Huang, J. Z. (2005). Adaptive grid job scheduling with genetic algorithms. Future Generation Computer Systems, 21(1), 151-161.

Guzek, M., Pecero, J. E., Dorronsoro, B., Bouvry, P. (2014). Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems. Applied Soft Computing, 24, 432-446.

Guzek, M., Varrette, S., Plugaru, V., Pecero, J. E., & Bouvry, P. (2013). A Holistic Model of the Performance and the Energy-Efficiency of Hypervisors in an HPC Environment. In Energy Efficiency in Large Scale Distributed Systems (pp. 133-152). Springer Berlin Heidelberg.

He, X., Sun, X., & Von Laszewski, G. (2003). QoS guided min-min heuristic for grid task scheduling. Journal of Computer Science and Technology, 18(4), 442-451.

Hussain, H., Malik, S. U. R., Hameed, A., Khan, S. U., Bickler, G., Min-Allah, N., ...& Rayes, A. (2013). A survey on resource allocation in high performance distributed computing systems. Parallel Computing, 39(11), 709-736.

Introducing the Viridis 2.0: Boston Viridis, data sheet (2012). Disponible en URL: http://www.boston.co.uk/solutions/viridis/introducing-the-viridis-2.aspx

Iqbal, S., Gupta R. & Lang Y. (2005). Job scheduling in HPC clusters. Power Solutions, 133–135.

Izakian, H., Abraham, A., & Snasel, V. (2009, April). Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In

International Joint Conference on Computational Sciences and Optimization (Vol. 1, pp. 8-12). IEEE Computer Society.

Jarus, M., Oleksiak, A., Piontek, T., & Węglarz, J. (2014). Runtime power usage estimation of HPC servers for various classes of real-life applications. Future Generation Computer Systems, 36, 299-310.

Jarus, M., Varrette, S., Oleksiak, A., & Bouvry, P. (2013). Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD and ARM processors. In Energy Efficiency in Large Scale Distributed Systems (pp. 182-200). Springer Berlin Heidelberg.

Kirkpatrick, S., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680.

Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. Econometrica: Journal of the Econometric Society, 497-520.

Leung, J. Y. (Ed.). (2004). Handbook of scheduling: algorithms, models, and performance analysis. CRC Press.

Lindberg, P., Leingang, J., Lysaker, D., Bilal, K., Khan, S. U., Bouvry, P., ...& Li, J. (2011). Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids. Energy-Efficient Distributed Computing Systems, 189-214.

Liu, H., Abraham, A., & Hassanien, A. E. (2010). Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Generation Computer Systems, 26(8), 1336-1343.

Moraga, R., Whitehouse, G. & Depuy. G. (2013). Meta-raps: Un enfoque de solución eficaz para problemas combinatorios. Revista Ingeniería Industrial, 2(1).

Nesmachnow, S., Cancela, H., Alba, E. (2010). Heterogeneous computing scheduling with evolutionary algorithms. Soft Computing, 15(4), 685-701.

Nesmachnow, S., Cancela, H., Alba, E. (2012). A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. Applied Soft Computing, 12(2), 626-639.

Pinel, F., & Bouvry, P. (2011). A Model for Energy-efficient Task Mapping on Milliclusters. In Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering. Civil-Comp Press.

Pinel, F., Dorronsoro, B., Bouvry, P. (2010). A new parallel asynchronous cellular genetic algorithm for scheduling in grids. In Parallel Distributed Processing, Workshops and PhD Forum, 2010 IEEE International Symposium. pp. 1-8.

Pinel, F., Dorronsoro, B., Pecero, J. E., Bouvry, P., & Khan, S. U. (2013). A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids. Cluster computing, 16(3), 421-433.

Pinel, F., Pecero, J. E., Khan, S. U., & Bouvry, P. (2011, August). Energy-efficient scheduling on milliclusters with performance constraints. In Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications (pp. 44-49). IEEE Computer Society.

Plugaru, V., Varrette, S., & Bouvry, P. (2014). Performance Analysis of Cloud Environments on Top of Energy-Efficient Platforms Featuring Low Power Processors. In 6th International Conference on Cloud Computing Technology and

Science (CloudCom'14). IEEE Computer Society.

Ritchie, G., Levine, J. (2004). A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, pp. 178–183.

Stützle, T., & Hoos, H. (2000). MAX–MIN ant system. Future generation computer systems, 16(8), 889-914.

Valentini, G. L., Lassonde, W., Khan, S. U., Min-Allah, N., Madani, S. A., Li, J., ...& Bouvry, P. (2013). An overview of energy efficiency techniques in cluster computing systems. Cluster Computing, 16(1), 3-15.

Varrette, S., Guzek, M., Plugaru, V., Besseron, X., & Bouvry, P. (2013, October). HPC Performance and Energy-Efficiency of Xen, KVM and VMware Hypervisors. In 25th International Symposium on Computer Architecture and High Performance Computing (pp. 89-96). IEEE Computer Society.

Varrette, S., Plugaru, V., Guzek, M., Besseron, X., & Bouvry, P. (2014). HPC Performance and Energy-Efficiency of the OpenStack Cloud Middleware. In 43rd International Conference on Parallel Processing (ICPP-2014). IEEE Computer Society.

Witkowski, M., Oleksiak, A., Piontek, T., & Węglarz, J. (2013). Practical power consumption estimation for real life HPC applications. Future Generation Computer Systems, 29(1), 208-217.

Xhafa F, Abraham A (2010) Computational models and heuristic methods for grid scheduling problems. Future generation computer systems, 26(4), 608–621.

Xhafa, F. (2007). A hybrid evolutionary heuristic for job scheduling on computational grids. In Hybrid Evolutionary Algorithms, pp. 269-311.

Xhafa, F., Alba, E., Dorronsoro, B., Duran, B., & Abraham, A. (2008). Efficient batch job scheduling in grids using cellular memetic algorithms. In Metaheuristics for Scheduling in Distributed Computing Environments (pp. 273-299). Springer Berlin Heidelberg.

Xhafa, F., Barolli, L., Durresi, A. (2007). Batch mode scheduling in grid systems. International Journal of Web and Grid Services, 3(1), 19-37.

Xhafa, F., Carretero, J., Alba, E., Dorronsoro, B. (2008). Design and evaluation of tabu search method for job scheduling in distributed environments. In Proceedings of the 22th International Parallel and Distributed Processing Symposium, pp. 1–8.

Xhafa, F., Carretero, J., Dorronsoro, B., & Alba, E. (2009). A tabu search algorithm for scheduling independent jobs in computational grids. Computing and Informatics, 28, 237-250.

Zomaya, A. Y., & Teh, Y. H. (2001). Observations on using genetic algorithms for dynamic load-balancing. IEEE Transactions on Parallel and Distributed Systems, 12(9), 899-911.