Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias en Ciencias de la Computación

Consolidación heterogénea para calendarización consciente de la energía de tareas con contención de recursos

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

Presenta:

Luis Angel Galaviz Alejos

Ensenada, Baja California, México
2017

Tesis defendida por

Luis Angel Galaviz Alejos

y aprobada por el siguiente Comité

Dr. Andrey Chernykh
Director de tesis

Miembros del comité
Dr. Adán Hirales Carbajal

Dr. Gustavo Olague Caballero

Dr. Raúl Rivera Rodríguez

Dr. Jesús Favela Vara
Coordinador del Posgrado Ciencias de la Computación

Dra. Rufina Hernández Martínez
Directora de Estudios de Posgrado

Resumen de la tesis que presenta **Luis Angel Galaviz Alejos** como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la computación.

Consolidación heterogénea para calendarización consciente de la energía de tareas con contención de recursos

Resumen aprobado por:

_____

Dr. Andrey Chernykh
Director de tesis

La nube es un paradigma de cómputo distribuido con gran popularidad. El gran consumo energético generado por la infraestructura de cómputo de las nubes tiene grandes repercusiones económicas y ambientales. Uno de los servicios principales de la nube es la renta de equipo de cómputo a través de la red conocido como IaaS. En una nube IaaS los proveedores de servicio son los encargados de administrar y aprovisionar los recursos de cómputo al usuario. La virtualización de los recursos de cómputo de la nube permite que varias máquinas virtuales se ejecuten en el mismo recurso físico, haciendo posible la consolidación. Las técnicas de consolidación tratan asignar todas las VMs a la mínima cantidad posible de máquinas físicas, esto genera una mejor administración de recursos y como resultado un ahorro energético. Una de las desventajas de la consolidación es que varias VMs compiten y comparten los recursos de hardware, lo que puede llevar a cuellos de botella en el uso del hardware. Nosotros proponemos clasificar las VMs de acuerdo al tipo de aplicaciones que estas ejecutan, en específico CPU y Memoria Intensivas. Esta clasificación ayuda a asignar las VMs a los recursos físicos de forma que se eviten conflictos en el uso de CPU y Memoria. Estudiamos la consolidación determinista el caso mono-objetivo de minimización de energía, y el bi-objetivo con violaciones al SLA y energía. Utilizamos un modelo construido con mediciones reales en el servidor Express x3650 M4. Utilizamos el algoritmo MOCell, el cual hace un muestreo del espacio de búsqueda ubicando a los individuos del algoritmo genético en un grid toroidal. Además, comparamos el algoritmo MOCell con el algoritmo clásico NSGA-II. Comparamos los frentes obtenidos mediante híper volumen y cubrimiento de conjuntos. Evaluamos nuestros algoritmos con cargas de sistemas reales del archivo de cargas de trabajo paralelas. Propusimos un operador de cruzamiento llamado uniforme SLA, el cual obtiene los mejores resultados pero tiene la desventaja de ser muy lento. Además, MOCell obtiene mejores frentes no dominados que NSGA-II. Para el caso mono-objetivo nuestra heurística CFFit obtiene los mejores resultados.

Palabras clave: Nube, Consolidación, Energía, Contención, SLA

Abstract of the thesis presented **by Luis Angel Galaviz Alejos** as a partial requirement to obtain the Master of Science degree in Computer Science

Heterogeneous consolidation for energy-aware scheduling of jobs with resource contention

Abstract approved by:

_____

Dr. Andrey Chernykh
Thesis Director

Cloud computing is popular a distributed computing paradigm. The high energy consumption generated by the cloud computing infrastructure has great economic and environmental repercussions. One of the main services offered by the cloud is the rental of computer equipment through the network known as IaaS. In an IaaS, cloud service providers are in charge of managing and provisioning the computing resources to the user. Virtualization of cloud computing resources allows multiple virtual machines to run on the same physical resource, making consolidation possible. The consolidation techniques try to assign all virtual machines to the minimum possible number of physical machines, this allows a better management of resources and, as a result an energy saving. One of the main disadvantages of consolidation is that multiple VMs compete and share hardware resources, which can lead to bottlenecks in the use of hardware. We propose to classify the VMs according to the type of applications it runs, specifically, CPU and Memory intensive. This classification helps to assign the VMs to the physical resources in a way that conflicts in the use of CPU and Memory could be avoided. We study deterministic consolidation in the case of single objective case for energy minimization, and SLA violations and energy minimization for bi-objective. We use a workload-aware model build from real energy measurements on server Express x3650 M4. We attack the problem using genetic algorithms. We use MOCell algorithm, it does a sampling of the search space by placing the individuals of the genetic algorithm in a toroidal grid. We compared the MOCell algorithm with the classic NSGA-II algorithm. We compare the fronts obtained by hyper volume and set coverage. We evaluate algorithms with workload traces from Parallel Workloads and Grid Workload Archives. We propose a crossover named uniform SLA, which has the best results but has the disadvantage of being very slow. In addition, MOCell gets better non-dominated fronts than NSGA-II. For the single objective case our heuristic CFFit obtains better results.

Keywords: Cloud, Consolidation, Energy, Contention, SLA

Agradecimientos

Este trabajo de tesis no hubiese sido posible sin el apoyo económico de CONACYT, mediante su programa de becas de posgrado. Además, agradezco a CICESE por ser la institución que me recibió para cursar mi posgrado, su personal me brindó apoyo tanto en mi desarrollo académico y personal.

Hago una especial mención a mi asesor de tesis, el doctor Andrey Chernykh. En lo académico su perfeccionismo sacó lo mejor de mí y me enseñó cómo hacer trabajo académico de calidad, además gracias a él pude conocer y hacer amistad con personas muy competentes dentro del mundo del super cómputo de las cuales aprendí muchísimo. En el ámbito personal agradezco que siempre promovió la amistad y convivencia dentro del equipo de trabajo del laboratorio de cómputo paralelo y en más de una ocasión fue el anfitrión de pequeñas reuniones siempre con un ambiente muy familiar y agradable.

Agradezco a mi comité conformado por los doctores Adan Hirales Carbajal, Gustavo Olague Caballero, Raul Rivera Rodriguez merecen una mención especial ya que sus observaciones fueron imprescindibles para el desarrollo de mi tesis.

Doy gracias a todos mis amigos dentro y fuera del posgrado que hicieron de mis estudios una experiencia amena, llevadera y divertida. Especialmente agradezco a mis amigos del laboratorio de visión por computadora y cómputo paralelo de CICESE, ya que las discusiones con ellos fueron muy útiles para aprender cosas nuevas y mejorar los productos de mi trabajo en CICESE.

Por último, agradezco a mis padres por su apoyo en el proceso de estudiar mi posgrado. Ya que ellos me motivaron a estudiar un posgrado, me ayudaron a reubicarme a la ciudad de Ensenada, y muchas otras cosas que hicieron esto posible.

Tabla de contenido

Lista de figuras

Lista de tablas

Capítulo 1. Introducción

Facebook, Amazon, Apple, Microsoft, Google y Yahoo y otras compañías están transformando rápidamente la manera en como trabajamos, comunicamos, vemos películas o televisión, escuchamos música y compartimos fotos, todo esto a través de la nube. El crecimiento de la nube es realmente impresionante, se estima que la cantidad de información digital aumente 50 veces para el 2020, todo para satisfacer nuestro deseo de tener acceso instantáneo a información infinita desde nuestras computadoras, teléfonos y dispositivos móviles (Cook, 2012).

1.1 Antecedentes

1.1.1 Cómputo en la nube

El cómputo en la nube es un paradigma de cómputo distribuido ampliamente aceptado por organizaciones públicas y privadas. Éste se basa en proveer tres tipos principales de servicios a través de internet asegurando la calidad de servicio (QoS) a sus clientes: Software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS). Este trabajo se centra en las nubes IaaS, en las cuales los recursos de cómputo (como procesadores, memoria, etc.) son provistos como servicios.

Software como servicio (SaaS) es un modelo de distribución de software donde un proveedor ofrece una aplicación a través de internet en vez de vender un paquete de software que el usuario debe comprar e instalar. Algunos ejemplos son Gmail de Google, Hotmail de Microsoft, Microsoft Office 365, etc.

Infraestructura como servicio (IaaS) proveen recursos al usuario, el cual tiene que manejarlos por sí mismo para ejecutar tareas. El usuario es responsable de manejar sus recursos para satisfacer sus necesidades de cómputo y requisitos de QoS. Los proveedores de IaaS ofrecen recursos de hardware como capacidad de almacenamiento, capacidad de CPU y memoria a través de internet. De esta forma los usuarios sólo rentan los recursos necesarios para sus necesidades en vez de comprar todo el equipo. Uno de los principales proveedores de este servicio es Amazon Web Services.

Plataforma como servicio (PaaS) ofrece una plataforma de desarrollo en la nube. Ésta puede ser vista como una capa intermedia entre el SaaS y el IaaS. En este modelo, el usuario adquiere una serie de herramientas soportadas por el proveedor. El desarrollador puede escribir aplicaciones para una plataforma particular

sin preocuparse por la infraestructura involucrada. El usuario sube aplicaciones y configuraciones para las mismas. Un Ejemplo es Google App Engine, el cual permite correr aplicaciones en la infraestructura de Google.

Los proveedores de nubes mantienen su calidad mediante acuerdos de nivel de servicio (SLA) el cual representa un contrato que especifica las obligaciones mínimas del proveedor hacia sus clientes o lo que esperan recibir los clientes a cambio del precio pagado. Desde la perspectiva del proveedor no es posible cumplir todas las expectativas del cliente y por lo es necesario hacer un balance, es decir, llegar a un acuerdo mediante un proceso de negociación. A este acuerdo se le conoce como SLA.

1.1.2 Arquitectura de la nube

La Figura 1 muestra la arquitectura típica de un sistema de asignación de una nube. Una nube es un centro de datos donde en las máquinas físicas (MF) están ejecutando un software llamado monitor de máquinas virtuales (VMM), el cual se encarga de ejecutar una o más máquinas virtuales (VM), y a su vez cada VM puede correr una o varias aplicaciones.

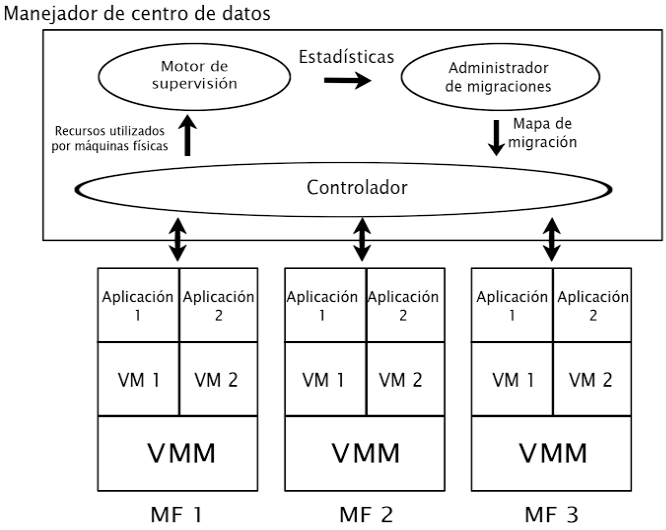


Figura 1: Arquitectura de una nube. (Varasteh and Goudarzi, 2015)

El manejador de un centro de datos de nubes se compone de (Varasteh et al., 2015): controlador, motor de monitoreo y manejador de migraciones. El motor de monitoreo lee constantemente información del

procesador, interfaz de red, uso de memoria, y otros datos de las maquinas a través del controlador. Este entonces procesa los datos y los pasa al manejador de migraciones. El manejador de migraciones usa toda la información y el algoritmo de consolidación para determinar las migraciones que se deben llevar a cabo. Por último, aplica los cambios a través del controlador.

1.1.3 Consumo energético en la nube

De acuerdo con el trabajo de (Priya et al., 2013), la nube es una tecnología verde por su eficiencia energética y eficiencia de recursos. Las razones para considerar a la nube una tecnología verde son: primeramente la virtualización, la cual permite crear distintas máquinas virtuales (VM) en una sola máquina física (MF), es decir, un único servidor corre la imagen de múltiples sistemas operativos concurrentemente, esto reduce la cantidad de servidores físicos con un beneficio verde; La segunda es la consolidación de carga de trabajo, ésta consiste en ejecutar la carga en la menor cantidad de servidores posibles, para que se utilicen mejor, ahorrando así energía.

La tercera es el software de automatización, éste ayuda a optimizar la utilización de los recursos y la última es el pago por uso y autoservicio, esto se refiere a que los usuarios sólo utilizan los recursos cuando los necesita.

Uno de los principales desafíos en el cómputo en la nube es el del manejo eficiente de la energía. De acuerdo con (Cook, 2012) en el 2007 el consumo energético mundial del internet/nube (centros de datos y redes de telecomunicaciones) fue aproximadamente 623 mil millones kWh, si la nube fuera un país, tendría el quinto lugar de demanda eléctrica del mundo. Las compañías de tecnologías de la información deben cumplir las metas globales y nacionales de reducción de marca de carbono, y deben compensar los incrementos en gastos energéticos; por lo tanto, cualquier tecnología emergente de la comunicación e información requiere de métricas de ahorro energético. En este contexto, los proveedores de nubes IaaS evalúan distintas estrategias para aumentar el ahorro energético en sus centros de datos, incluyendo el equipo de cómputo, enfriamiento y fuente de alimentación.

1.1.4 Consolidación

En (Varasteh et al., 2015) se definen las técnicas de consolidación como aquellas que asignan una serie de VM en la menor cantidad posible de MFs para optimizar la utilización de recursos y reducir el consumo de energía haciendo que las maquinas corran en un estado donde la energía es utilizada más eficientemente. Utilizando migración en vivo de VM, es posible transferir una VM de una MF a otra sin un tiempo de caída de servicio considerable.

La contención de recursos es el conflicto en el acceso a distintos recursos compartidos de hardware, tales como memoria, disco duro, red, etc.

Los administradores de los centros de datos desean consolidar las VM en servidores compartidos, incrementado así los niveles de utilización, lo que lleva a la reducción de costos de hardware y energía. Por otro lado, el colocar VMs en una misma maquina tiene como resultado que las VM compartan y compitan por los recursos de cómputo, ya que cada VM tiene acceso restringido a los recursos de hardware. La contención de recursos puede degradar significativamente el rendimiento de máquinas virtuales y consecuentemente, las aplicaciones alojadas en ellas (Fox et al., 2012).

En este trabajo se propone un algoritmo de consolidación que busca minimizar el consumo energético mientras se evita la degradación del desempeño de las VMs, esto se logra suponiendo que cada VM ejecuta un tipo de tarea específico (CPU Intensiva, memoria intensiva, disco duro intensiva y red intensiva) y que, por lo tanto, cada una hace uso de distintos recursos de hardware.

1.1.5 Trabajo Previo

En esta sección se muestran varios artículos sobre algoritmos de consolidación de VMs tomando en cuenta distintos factores para la toma de decisiones, como el uso de CPU, uso de memoria, costos de migración, saturación de la red de comunicaciones en el centro de datos, la degradación del desempeño, temperatura, etc. Además, existen trabajos que se centran en la asignación de VM, migración de VM o ambas. En la Tabla 1 se muestran un resumen de los trabajos previos.

Aunque existen varios trabajos sobre consolidación, según nuestro conocimiento muy pocos consideran el efecto de contención generado por el tipo de carga de trabajo ejecutada por las VMs. Nuestro trabajo

se diferencia del resto en dos factores: primero, no existe ningún trabajo que considere el porcentaje de concentración de VM para la selección de máquinas para consolidación; segundo, no existe un modelo de energía que considere la contribución al consumo energético debido a la ejecución de cada tipo de aplicación por separado.

MHCPESJ: Política de concentración heterogénea para calendarización consciente de la energía con trabajos con contención de recursos. En (F. A. Armenta-Cano et al., 2017) se continua el trabajo (F. Armenta-Cano et al., 2015), en estos se aborda el problema de calendarización en línea de trabajos con contención de recursos. Este trabajo propone *Min_c*, el cual es un nuevo enfoque de asignación de tareas tomando en cuenta la concentración de distintos tipos de tareas heterogéneas que pueden incluir CPU Intensivas, disco duro intensivas, I/O intensivas, memoria intensivas, red intensivas y otras aplicaciones. La idea principal del enfoque está basada en el hecho de que distintas aplicaciones utilizan distintos recursos de hardware. Cuando las tareas de un tipo son asignadas al mismo recurso pueden crear contención en CPU, memoria, disco duro o red, lo cual puede resultar en degradación del rendimiento y un incremento en el consumo energético. Los resultados experimentales muestran que *Min_c* es una mejor estrategia comparada con algunas heurísticas clásicas como Random, First Fit, round Robin, entre otras.

PTVMAC: Asignación consciente de la temperatura y energía para centros de datos de nubes. En (Wang et al., 2015) se propone un algoritmo de consolidación en servidores heterogéneos tomando en cuenta la temperatura y consumo energético. El objetivo propuesto por este mecanismo es reducir el consumo de energía y el número de migraciones mientras se evita violar los acuerdos de nivel de servicio (SLA, por sus siglas en inglés) en las nubes. Los resultados muestran que el método propuesto tiene mejor rendimiento promedio que otras técnicas basadas en la energía.

NAVMC: Migración consciente de la red en centros de datos de nubes. En (Maziku and Shetty, 2014a) se evalúa a VMPatrol en un banco de pruebas GENI caracterizada por una red WAN y escenarios de tráfico realísticos. La migración en vivo puede consumir todo el ancho de banda para las aplicaciones memoria intensivas. VMPatrol es un modelo de costos de migración en una WAN, este tiene la habilidad de asignar el mínimo ancho de banda para permitir completar la migración de VM en un tiempo especificado por el usuario con tráfico de red realístico. Los resultados indican que el tiempo requerido para una migración depende en el tamaño de memoria de la VM, la razón de páginas sucias de VM y del ancho de banda.

HVCUIGG: Consolidación heterogénea de máquinas virtuales utilizando un algoritmo genético de agrupamiento. El trabajo (Wu and Ishikawa, 2015) se enfoca en limitar el gasto en costos de migración para ahorrar energía a través de la consolidación de VM en una nube con servidores heterogéneos. Los factores tomados en cuenta para la migración son el ancho de banda disponible de la red, el tamaño de memoria de la VM, la razón de páginas sucias y utilización de CPU. Los resultados mostraron que su algoritmo siempre alcanza la mejor compensación entre costos de migración y ahorro energético.

PACMan: Consolidación consciente del rendimiento. En (Nath, 2013) se presenta a PACMan, el cual es un sistema que asigna VMs tal que la degradación del rendimiento está dentro de un umbral ajustable mientras minimiza los recursos utilizados. Además, PACMan cuenta con un modo de operación que prioriza la eficiencia de recursos, llamado Eco Mode, en el número de servidores usados es fijo y las VM son asignadas de forma ajustada mientras se minimiza la degradación de rendimiento. PACMan opera entre el 10% de la solución óptima y ahorra alrededor de 30% de la energía comparado con otros esquemas que no toman en cuenta la interferencia. Eco Mode alcanza hasta un 52% de reducción en degradación del rendimiento comparado con algoritmos que no toman en cuenta la degradación.

NASVWIM: Calendarización consciente de la red para VMs con modelos de interferencia. En (Verboven et al., 2015) se presenta un modelo de interferencia del rendimiento basado en las características de cargas de trabajo virtuales. Este modelo es utilizado en un calendarizador. Mediante el uso de datos de entrenamiento se predice la degradación del rendimiento usando técnicas de modelado existentes y una combinación de las mismas. Su modelo toma en cuenta el CPU, cache, métricas de disco y red. Los experimentos muestran que detectar el tipo de carga de una aplicación y la predicción del tiempo de ejecución es útil en la calendarización en centros de datos.

DENS: Calendarización energéticamente eficiente consciente de la red. En (Kliazovich et al., 2010) se propone una metodología llamada DENS, la cual balancea el consumo energético de un centro de datos, el rendimiento individual de las tareas y las demandas de tráfico. Este enfoque optimiza la compensación entre la consolidación de tareas y los patrones de tráfico. Contrario a las soluciones tradicionales de calendarización que consideran los centros de datos como un conjunto de servidores, DENS desarrolla un modelo jerárquico consistente con las topologías de los centros de datos del estado del arte. DENS mantiene la carga de un servidor en operación alrededor de 0.9 y la carga promedio del switch de enlace ascendente en 0.95. Pero, DENS tiene un leve incremento en el número de servidores utilizados. En promedio, el calendarizador DENS dejó 956 servidores ociosos mientras el calendarizador Green dejó 1016.

ESWCT: Esquema de calendarización consciente de la energía usando un esquema de consolidación tomando en cuenta la carga de trabajo en nubes. En (Hongyou et al., 2013) se proponen dos algoritmos: Energy-aware Scheduling algorithm using Workload-aware Consolidation Technique (ESWCT) y Energy-aware Live Migration algorithm using Workload-aware Consolidation Technique (ELMWCT). Ambos algoritmos se basan en el hecho de que en los centros de datos de nubes varios usuarios comparten múltiples recursos y que distintas cargas de trabajo tienen características de consumo diferentes. Estos tratan de ejecutar todas las VMs en la mínima cantidad de MFs y apagar los servidores sin uso para reducir el consumo energético. Este artículo trabaja con máquinas heterogéneas y con VM con distintos requerimientos de CPU, memoria y red. Las simulaciones muestran que ambos algoritmos utilizan eficientemente los recursos del centro de datos, y que los distintos recursos tienen una utilización balanceada, lo que de demuestra su capacidad de ahorrar energía.

WCVCA: Algoritmo de consolidación de VMs consciente de las características de la carga de trabajo. En (Yang et al., 2012) se proponen varios algoritmos de asignación, uno de aproximación y dos de programación dinámica, los cuales toman en cuenta las características de una VM para la consolidación. En este artículo se trabaja con cargas datos intensivas y CPU intensivas. Además, se toma en cuenta el hecho de que los sistemas de la nube copian una imagen prototipo en las máquinas antes de iniciar una VM. Se sabe que las VM que ejecutan aplicaciones intensivas en datos y corren en una MF sin su imagen se ven afectadas en su rendimiento, en cambio éste problema no afecta a las VM con carga CPU Intensiva. Los resultados muestran que el algoritmo de aproximación consigue soluciones buenas en poco tiempo comparado con los algoritmos de programación dinámica.

UACSCV: Using Ant Colony System to Consolidate VMs for Green Cloud Computing. En (Farahnakian et al., 2015) se propone ACS-VMC el cual es un algoritmo que utiliza la meta heurística *ant colony system*, con el objetivo de reducir el consumo energético mediante migraciones en un ambiente distribuido mientras se mantiene la calidad de servicio deseada. Éste modelo toma en cuenta el CPU, memoria y red. Los experimentos en cargas realísticas muestran que ACS-VMC reduce el consumo energético mientras mantiene los requisitos de rendimiento en un centro de datos de nube.

RVCEPV: Consolidación robusta de VMs para energía y rendimiento en centros de datos virtualizados. En (Takouna et al., 2014) se propone un enfoque que consta de tres algoritmos: detección de servidores sobrecargados, selección de VM y asignación de VM. Éstos están basados en el análisis estadístico de los datos históricos de utilización de CPU. Los algoritmos son probados en CloudSim utilizando cargas de trabajo realísticas de PlanetLab. Los resultados muestran que éste enfoque reduce la energía consumida

por los servidores, el número de migraciones y los cambios de estado energético. Además, reduce el consumo energético de la red debido a las migraciones de VM.

TNAVMECM: Hacia una migración consciente de la red: evaluando los costos de migración en los centros de datos de nubes. En (Maziku and Shetty, 2014b) se evalúa a Remedy en un banco de pruebas GENI caracterizado por una WAN y escenarios de tráfico realísticos. Remedy es un modelo de estimación de costos para calcular el tráfico generado debido a la migración de VMs. Éste modelo toma en cuenta el tamaño de memoria de la VM, la razón de páginas sucias y el ancho de banda de conexión para calcular el costo de migración. Los experimentos muestran que Remedy trabaja mejor con anchos de banda de 1 Gbps o superiores y razones de páginas sucias bajo 3000 páginas/s.

Tabla 1: Algoritmos de Consolidación

Artículo	Parámetros Considerados							Mecanismo	
	CPU	Memoria	Red	Temperatura	Tipo de carga de trabajo	Costos de migración	Contención	Migración	Asignación
MHCPESJ					●		●		●
PTVMAC				●		●		●	
NAVMC		●	●					●	
HVCUIGG	●	●	●			●		●	
PACMan							●		●
NASVWIM	●		●		●		●		●
DENS			●						●
ESWCT	●	●	●		●		●	●	●
WCVCA					●				●
UACSCV	●	●	●					●	
RVCEPV	●					●		●	
TNAVMECM		●	●					●	

1.2 Justificación

La demanda de los servicios en la nube crece día con día. Las ventajas que este ofrece, como el acceso ubicuo, la eliminación de costos de mantenimiento de equipos de cómputo, la capacidad de acceder al poder de cómputo de acuerdo las necesidades específicas del cliente, etc. no hace más que aumentar su popularidad y, como consecuencia, el interés de los académicos y del sector privado. Uno de los problemas principales en la nube es la reducción de costos de operación y emisiones al medio ambiente.

En este trabajo pretendemos contribuir al estudio de las técnicas de ahorro de energía en la nube a través del diseño de políticas de consolidación en la nube para el ahorro de consumo energético. Uno de los defectos de la consolidación es la contención de recursos, lo cual puede llevar a la disminución de la calidad de servicio del cliente. Exploramos las oportunidades de ahorro energético mediante la implementación de un modelo de energía que clasifica las VMs de acuerdo al tipo de aplicaciones que esta ejecuta. Las aplicaciones se clasifican en CPU intensivas y memoria intensivas de acuerdo a cuál es el recurso de hardware más demandado por estas. De esta manera atacamos simultáneamente la problemática del consumo energético y de la contención de recursos en la consolidación.

1.3 Hipótesis

- Utilizar perfiles de consumo energético de cada tipo de aplicación por separado, puede ayudar a evaluar mejor el consumo eléctrico en un servidor.
- Utilizar perfiles de consumo energético de la combinación de distintos tipos de aplicaciones en un procesador, puede ayudar a evaluar mejor el consumo eléctrico en un servidor.
- La consolidación de máquinas virtuales tomando en cuenta el tipo de tareas reduce la energía consumida en las nubes

1.4 Objetivos

1.4.1 Objetivo general

Diseño e implementación de un algoritmo bi-objetivo para la consolidación de VM en procesadores homogéneos, el cual tome en cuenta las distintas cargas de trabajo ejecutadas por las VMs, con el objetivo de ahorrar energía y minimizar las violaciones la calidad de servicio.

1.4.2. Objetivos específicos

- Diseñar e implementar el algoritmo de consolidación que tome en cuenta el tipo de tareas ejecutadas por las VMs (CPU y memoria intensiva)

- Diseñar e implementar un algoritmo genético e identificar que operadores permiten obtener mejores resultados
- Comparar la calidad de las soluciones obtenidas contra heurísticas clásicas (first fit, round robin, random)
- Hacer un análisis bi-objetivo de los frentes objetivos mediante híper volumen y cubrimiento de conjuntos para determinar cuál es el mejor algoritmo.

Capítulo 2. Definición del problema

En este trabajo se aborda el problema de consolidación de VMs en un entorno fuera de línea, esto último implica que todas las VMs están disponibles para ser ejecutadas desde el inicio del algoritmo. Dado un conjunto de VMs que ejecutan distintos tipos de carga, las cuales deben ser asignadas a un conjunto de MFs con velocidad homogénea. Asignaremos las VMs a las MF con dos objetivos: reducir el consumo energético y minimizar las violaciones a la calidad de servicio. Este problema puede ser representado en la notación de tres campos como:

$$P \mid p_j, tipo_j, v_j \mid E^{op}, SLA \tag{1}$$

Donde:

P: Es un conjunto de procesadores idénticos, es decir, tienen la misma velocidad (En MIPS) y eficiencia energética (En MIPS por watt). En este trabajo se utiliza el término MF o servidor de manera indistinta.

$tipo_j$: Indica el tipo de VM de acuerdo a la carga que ésta ejecuta (CPU intensivas, disco duro intensivas, Memoria intensivas y Red intensivas). Cada VM puede ejecutar únicamente tareas de un solo tipo.

v_j: Es la velocidad de la VM en MIPS.

E^{op}: Primera función objetivo, reducir la energía total consumida en kWh.

SLA: Segunda función objetivo, minimizar la cantidad de violaciones al SLA.

En este trabajo las VM se clasifican en tipos según el tipo de tareas que ejecutan en: CPU intensivas, disco duro intensivas, memoria intensivas, red intensivas, etc. Cada VM puede ejecutar tareas de un solo tipo.

Estas clases se definen como:

CPU intensivas (CI): Son aquellas VM que ejecutan tareas que requieren de una gran cantidad de operaciones del CPU para ser completadas. Por ejemplo: operaciones con matrices, cálculo de números primos y de transformada de Fourier. Existen pruebas que contienen diferentes programas que ejecutan las operaciones antes mencionadas. Algunos ejemplos son Sysbench y Linpack.

Disco duro intensivas (DI): Son aquellas VM que ejecutan tareas que requieren de una gran de lecturas y escrituras al disco duro para ser completadas. Sysbench cuenta con una prueba de éste tipo, la cual consiste en crear archivos grandes en el disco duro y posteriormente hacer una gran cantidad de lecturas y escrituras sobre éstos.

Memoria intensiva (MI): Son aquellas VM que ejecutan tareas tal que su tiempo de ejecución está determinado principalmente por la cantidad de memoria para alojar sus datos. Las pruebas Sysbench y Stream cuentan con cuentan con código de este tipo, éstas consisten en medir el ancho de banda de memoria para el acceso a vectores grandes.

Red intensiva (RI): Son aquellas VM que ejecutan tareas que requieren de una gran cantidad de ancho de banda para ser completadas.

Suponemos que una existe una combinación de los distintos tipos de VM en un procesador puede ayudar a evitar la contención en el hardware, para así lograr a mantener la calidad de servicio y reducir el consumo energético.

Consideraremos un conjunto $h_1, h_2, \dots, h_m$ de procesadores heterogéneos con distinta velocidad y eficiencia energética. Cada procesador h_i es descrito por una tupla $\{s_i, eff_i\}$, donde s_i es su velocidad (En MIPS) y eff_i (En MIPS por watt) es su eficiencia energética.

Consideraremos un conjunto $VM_1, VM_2, \dots, VM_n$ de VMs que ejecutan tareas independientes, un conjunto $D = \{CI, DI, MI, RI\}$ de tipos de VM y un entero $k = |D|$ el cual indica la cantidad de distintos tipos de VM. Cada VM_j está descrita por la tupla $VM_j = (p_j, tipo_j, v_j)$, donde p_j es el tiempo de ejecución de la VM; v_j es la velocidad de cómputo la VM en GHz y $tipo_j \in D$ e indica el tipo de VM.

2.1 Modelo energético

Un modelo de energía es una función que calcula la energía consumida por un procesador haciendo uso de información colectada a nivel servidor, componente o aplicación. Es posible crear distintos modelos energéticos considerando distintos factores como utilización de CPU, disco duro, red, RAM, etc. Los modelos de energía clásicos (Priya et al., 2013) emplean sólo la utilización total para estimar el consumo energético. Por el contrario, nosotros utilizamos un modelo propuesto en (F. A. Armenta-Cano et al., n.d.)

el cual está validado experimentalmente, este considera tanto la utilización de cada tipo de VM por separado como el efecto que tiene la consolidación de distintos tipos de VMs en el mismo servidor (mezcla).

Para desarrollar el modelo se ejecutaron distintos tipos de pruebas (simulando así el tipo de aplicación) y se hicieron mediciones del consumo energético del servidor. A continuación se da una explicación sobre el proceso de generación del modelo energético, seguido por una descripción del mismo.

2.1.1 Medición de energía

La medición de energía se puede hacer a varios niveles, es decir, a nivel de centro de datos, rack, servidor o componente. Nosotros estamos interesados en la medición del consumo a nivel de servidor o inferior (también conocido como monitoreo de grano fino). En el artículo (Tang et al., 2015) se mencionan los dos enfoques existentes para la medición a grano fino:

- *Medición energética por hardware*: Consiste en utilizar un equipo ajeno al sistema de cómputo para medir. Un ejemplo de este enfoque de medición es el IBM *Power Executive*, el cual utiliza sensores embebidos para medir el consumo energético. Este método de medición es muy exacto, pero tiene la desventaja de que si el equipo no cuenta con instrumentos de medición integrados (Ej.: DELL PowerEdge M100e y la serie IBM BladeCenter H), entonces se requiere de hardware extra que suele ser costoso debido que es difícil de instalar, esto último pasa en servidores altamente compactos como los servidores Blade.

- *Modelado energético por software:* Es el cálculo del consumo de energía mediante un modelo, usando información colectada desde a nivel servidor, componente o aplicación (Utilización de CPU, memoria, etc.). Existe información del sistema disponible a través del software (asistentes y herramientas del sistema operativo) y del hardware (Hardware performance counters o contadores de rendimiento). Los modelos de energía son entrenados basado en la correlación entre un estado (o utilización de un recurso) de un componente de hardware y el consumo energético del mismo. Entre los modelos de energía, los modelos de regresión lineal son los más utilizados, dada su simplicidad y precisión. En resumen, el problema de modelado de consumo de energía es: dado una serie de valores que representan distintos estados del equipo y su respectivo consumo de energía bajo estos mismos, encontrar una función que relacione los estados con el consumo de energético.

Intel introdujo la característica conocida como *Running Average Power Limit* (RAPL) como parte de la arquitectura Sandy Bridge. El RALP provee una serie de sensores que permiten la medición del consumo energético de los componentes a nivel CPU enlistados en la Tabla 2. Es posible acceder y configurar los sensores del RAPL leyendo los registros específicos de la máquina (MSRs) (Hähnel et al., 2012).

Tabla 2: Lista de sensores RAPL disponibles

RAPL_PKG	Todo el paquete de CPU
RAPL_PP0	Sólo los cores del procesador
RAPL_PP1	Un dispositivo específico en el uncore
RAPL_DRAM	Controlador de memoria

En (Treibig et al., 2010) se describe a LIKWID ("Like I knew What I'm Doing"), el cual es un conjunto de herramientas de línea de comandos para apoyar en la optimización de código. Está orientado a la programación orientada al desempeño en un ambiente de Linux, no requiere parchado del kernel y es adecuado para arquitecturas de procesadores Intel y AMD.

Algunas de las herramientas de LIKWID son:

Likwid-topoly: Sondea los hilos de hardware y topología de caché en sistemas multicore.

Likwid-perfeCtr: Mide las métricas de los *performance counters* sobre el tiempo completo de ejecución de una aplicación o, con la ayuda de una simple API, entre dos puntos arbitrarios de código.

Likwid-powermeter: Lee los valores de consumo energético del RAPL.

La medición de consumo energético se puede hacer a través de las herramientas *Likwid-powermeter* o *Likwid-perfeCtr*.

El procedimiento empírico para generar el modelo de (F. A. Armenta-Cano et al., n.d.) está constituido de tres partes. En la primera, se midió y comparó el consumo energético de cada aplicación bajo diferentes niveles de utilización. En la segunda, se midió el consumo energético de la combinación de distintos tipos. Para la medición del consumo energético se utilizó la herramienta *Likwid-powermeter* la cual permite acceder al RAPL para consultar el consumo energético. Para la medición a nivel servidor se utiliza una

unidad de distribución de energía (PDU): VMR-8HD20-1 con las salidas medidas, en las cuales únicamente el servidor está conectado. Parte de este trabajo de tesis consistió en la creación de un script para la recopilación de datos desde el PDU (Ver Anexo 1).

Para las medidas, se asignó una aplicación por core y los lógicos no fueron utilizados. Las pruebas consisten en la asignación de una aplicación de benchmark a un core específico y obtener la energía medida durante la ejecución, entonces agregar una aplicación más hasta alcanzar la utilización máxima (utilizar todos los cores). El consumo ocioso para cada core es 9 watts en promedio.

Los experimentos de medición se hicieron en un Express x3650 M4 con dos procesadores Xeon IvyBridge E5-2650v2 95W a 2.6 Ghz. Cada core tiene dos hilos (16 con hyperthreading), 32 Kb de cache nivel 1, 256 Kb de cache nivel 2, 20 Mb de cache nivel 3 y un total de 64 Gb de RAM.

2.1.2 Descripción del modelo energético

La contribución de utilización $u_{i,j}(t)$ de la VM_j a h_i en un tiempo t está definido como $u_{i,j}(t) = \frac{v_j}{s_i}$. Sea l un tipo de VM ejecutada h_i, la utilización agregada $U_{i,l}(t)$ en un tiempo t se define como $U_{i,l}(t) = \sum_{j=1}^{p} u_{i,j}(t)$ la cual es la suma de la contribución de utilización de las *p* VM de tipo *l* ejecutándose en h_i.

Consideramos el consumo de energía contribuido por cada tipo de VM en el procesador por separado, por lo tanto, contamos con una función independiente por cada tipo de aplicación $f_1\left(U_{i,1}(t)\right), f_2\left(U_{i,2}(t)\right), \dots, f_k\left(U_{i,k}(t)\right)$. Las funciones f_l utilizadas son CI PDU y MI 107 PDU (Figura 2). Donde $f_l\left(U_{i,l}(t)\right)$ representa la fracción consumo de energía de energía independiente de una VMs de tipo l en h_i, calculamos $F_{i,T}(t)$ como la suma de todas las fracciones de consumo energético de cada tipo de VM:

$$\boldsymbol{F_{i,T}(t) = \sum_{i=1}^{k} f_l\left(U_{i,l}(t)\right),\ 0 \le F_{i,T}(t) \le 1} \quad (2)$$

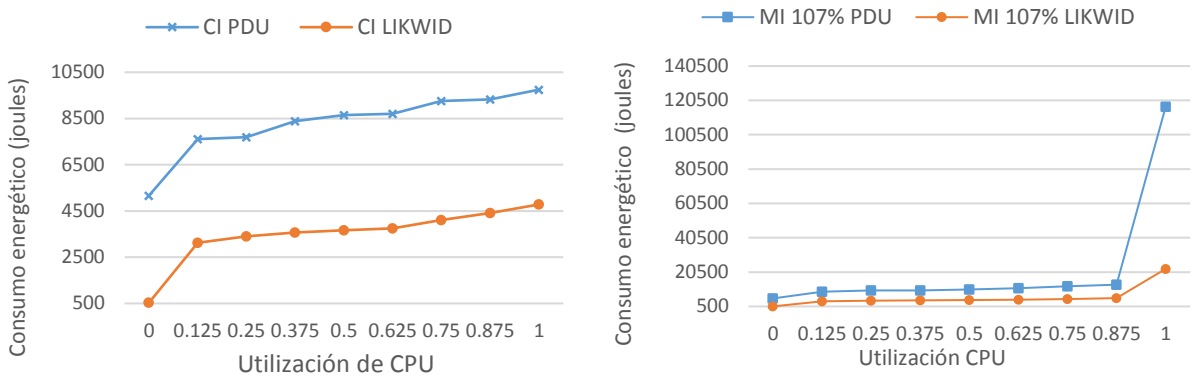


Figura 2: Gráficas de consumo por separado f_l. Según los experimentos de (F. A. Armenta-Cano et al., n.d.) cuando existen conflictos de memoria puede haber un ahorro significativo de energía balanceado los tipos de VMs consolidadas en un servidor. Se muestran dos gráficas una construida con LIKWID y una con PDU para cada tipo de aplicación. CI corresponde al perfil de consumo de las VMs de tipo CI, MI 107% es el consumo de VMs MI cuando existen conflictos de memoria.

El consumo de energía de h_i en un tiempo t consiste de dos componentes de consumo: el correspondiente a cuando está encendido pero no está siendo utilizado $e^{proc}_{ocioso_i}$, y a cuando el procesador está en uso $e^{proc}_{uso_i}(t)$. El consumo total e^{proc}_i está dado por $e^{proc}_i(t) = o_i(t)(e^{proc}_{ocioso_i} + e^{proc}_{uso_i}(t))$, donde $o_i(t) = 1$, si el procesador está encendido en el tiempo t y $o_i(t) = 0$ en caso contrario.

Si consideramos el consumo total como la suma de los consumos independientes producidos por cada tipo de VM, entonces el consumo en un tiempo t del procesador en uso viene dado por:

$$\boldsymbol{e^{proc}_{uso_i}(t) = \left(e^{proc}_{max_i} - e^{proc}_{ocioso_i}\right) * F_{i,T}(t)} \quad (3)$$

Donde $e^{proc}_{max_i}$ es el consumo máximo cuando el procesador está siendo completamente utilizado.

Tomando en cuenta los tipos de tareas (F. A. Armenta-Cano et al., 2017), la asignación de distintos tipos de aplicaciones en el mismo procesador podría causar una reducción en el consumo de energía, aún menor a la suma de los consumos independientes de cada una. Además tiene un impacto en el rendimiento evitando los cuellos de botella (en el CPU, disco duro, etc.) los cuales podrían resultar en una reducción del rendimiento del sistema y un aumento en su consumo energético. Por esto, consideramos el efecto que tiene la ejecución de una combinación distintos tipos de VM en un solo procesador.

La utilización total de h_i está dada por $U_{i,T}(t) = \sum_{l=1}^{k} U_{i,l}(t)$. Definimos $a_{i,l}(t) = \frac{U_{i,l}(t)}{U_{i,T}(t)}$ como la concentración de tipos de VM de tipo l en h_i en un tiempo t. Los elementos del conjunto $\mathrm{A}_i(t) = \{\alpha_{i,1}(t), \alpha_{i,2}(t), \ldots, \alpha_{i,k}(t)\}$ indican la concentración de cada tipo de VM en h_i.

Debido a que existe una gran cantidad de combinaciones de concentraciones, simplificamos el modelo considerando la concentración de un sólo tipo de aplicación contra todas las demás. Por ejemplo, si sólo tomamos en cuenta la concentración de CI, no importa cuál sea el valor de concentración de los demás tipos.

Consideramos k funciones $g_1\left(a_{i,1}(t)\right), g_2\left(a_{i,2}(t)\right), \ldots, g_k\left(a_{i,k}(t)\right)$. Donde el valor de $g_l\left(a_{i,l}(t)\right)$ es un coeficiente ($0 \leq g_l\left(a_{i,l}(t)\right) \leq 1$), el cual simboliza como se ve afectado el consumo energético al combinar distintos tipos de VM en un procesador, tomando en cuenta únicamente la concentración de las VM de tipo l. La gráfica g_l utilizada para este trabajo es PDU 107% (Figura 3).

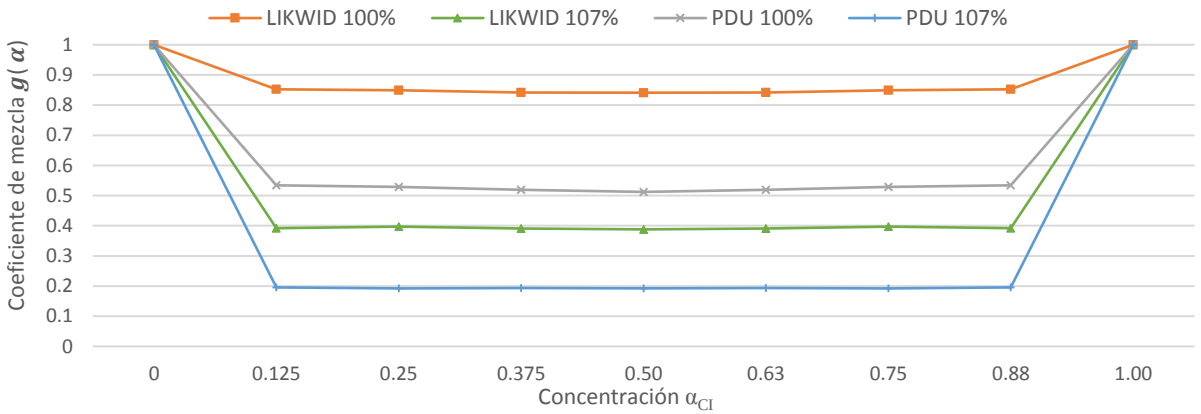


Figura 3: Funciones de mezcla cuando el CPU tiene 100% de utilización g_l. Ambas gráficas fueron validadas experimentalmente en el trabajo (F. A. Armenta-Cano et al., n.d.). La línea amarilla es el coeficiente de consumo por mezcla cuando se asigna el 100% de la memoria del servidor; la línea azul es la fracción de consumo cuando se asigna más del 100% de la memoria del procesador (generando conflictos), es en este caso cuando considerar el tipo de VM ayuda a tener un mayor ahorro de energía.

Si tomamos en cuenta las combinaciones de tipos de VM, el consumo del procesado en uso en un tiempo t está dado por:

$$\boldsymbol{e}_{\boldsymbol{uso}_i}^{\boldsymbol{proc}}(\boldsymbol{t}) = \left(\boldsymbol{e}_{\boldsymbol{max}_i}^{\boldsymbol{proc}} - \boldsymbol{e}_{\boldsymbol{ocioso}_i}^{\boldsymbol{proc}}\right) * \boldsymbol{F}_{\boldsymbol{i,T}}(\boldsymbol{t}) * \boldsymbol{g}_{\boldsymbol{l}}(\boldsymbol{a}_{\boldsymbol{i,l}}(\boldsymbol{t})) \quad (4)$$

El considerar la concentración de un único tipo de VM en el modelo trae consigo un problema: seleccionar cuál de las concentraciones considerar para el cálculo de $e_{uso_i}^{proc}$. Para resolverlo nosotros proponemos dos alternativas: seleccionar a l como el primer elemento de $\mathrm{A}_i(t)$ cuyo valor sea distinto de cero o seleccionar a l como $\max\{\mathrm{A}_i(t)\}$.

El consumo de energía total es la integral de la potencia consumida durante el tiempo de operación C_{max}:

$$\boldsymbol{E^{op} = \int_{t=1}^{Cmax} E^{op}(t)dt \ \ con \ E^{op}(t) = \sum_{i=1}^{m} e_i^{proc}(t)} \quad (5)$$

Capítulo 3. Algoritmo celular multi-objetivo (MOCell)

Debido que nuestro problema a tratar pertenece a la clase NP Difícil (Varasteh et al., 2015), no se conoce un algoritmo que pueda solucionarlo a optimalidad en tiempo polinomial, es por esto que se requiere acudir a los algoritmos heurísticos o meta-heurísticos. Los algoritmos heurísticos son aquellos que explotan el conocimiento que se tiene de un problema para encontrar soluciones suficientemente buenas en tiempo polinomial; por otro lado, los algoritmos meta-heurísticos hacen una búsqueda inteligente sobre las posibles soluciones al problema (espacio de búsqueda). Un algoritmo genético (AG) es un algoritmo meta-heurístico que toma varias ideas del proceso de evolución de las especies para la solución de problemas algorítmicos. Para nuestro problema utilizamos algoritmos genéticos para minimizar la cantidad de violaciones al SLA y la energía consumida. En esta sección se muestran los conceptos principales de algoritmos genéticos mono objetivo y multi-objetivo.

3.1 Algoritmo genético mono objetivo

La idea básica de un algoritmo genético es emular una población de individuos en un medio con recursos limitados, los cuales compiten por los recursos causando la selección natural. Esto causa un aumento en la aptitud de la población. Dada una función objetivo a ser maximizada, creamos una población de soluciones candidatas (Elementos del dominio de la función). Entonces aplicamos la función objetivo a estas soluciones como una medida de aptitud (entre más grande mejor). En base a la aptitud, algunos de los mejores candidatos son seleccionados para generar la próxima generación, esto se hace aplicando recombinación o mutación en ellos. Recombinación es un operador que se aplica a dos candidatos (padres), produciendo más candidatos (hijos). La mutación se aplica a un candidato y su resultado es un nuevo candidato. Los nuevos individuos, compiten en la población por un lugar en la próxima generación. Este proceso se repite hasta que se alcance un criterio computacional (Eiben and Smith, 2008).En la Figura 4 se puede ver el pseudocódigo de un AG clásico.

Existen dos fuerzas principales que forman la base de los sistemas evolutivos:

- Operadores de variación (Cruce y recombinación) crean la diversidad necesaria dentro de la población
- Selección actúa como una fuerza incrementando la calidad media de las soluciones en la población.

La combinación de estas fuerzas produce la mejora de la función objetivo (adaptación) conforme pasan las generaciones.

```
AlgoritmoGenetico()
1  Inicializar población con soluciones candidatas aleatorias
2  Evaluación de cada solución
3  While !CondicionTerminacion
4     Selección de padres
5     Recombinación de pares de padres
6     Mutación de los hijos resultantes
7     Evaluación de nuevos candidatos
8     Selección de individuos para la próxima generación
9  EndWhile
```

Figura 4: Pseudocódigo de algoritmo genético clásico.

3.2 Algoritmos Evolutivos Multi-Objetivo (MOEAs)

En los problemas con múltiples funciones objetivo en conflicto, no existe una única solución óptima, por el contrario, existe un grupo de ellas a las cuales se les conoce como soluciones Pareto óptimas. No es posible aseverar que alguna de estas soluciones es mejor que la otra, es por eso que se busca encontrar tantas de estas como sea posible (Deb et al., 2002). A continuación se dan algunos conceptos básicos sobre optimización multi-objetivo sobre funciones de minimización (los mismos conceptos aplican a maximización con las modificaciones correspondientes) (Nebro et al., 2006):

Problema Multi-Objetivo (MOP): Encontrar un vector $\boldsymbol{x}^* = [x_1^*, x_2^*, \dots, x_n^*]$ que satisfaga las m restricciones de desigualdad $g_i(\boldsymbol{x}) \geq 0, i = 1, 2, \dots, m,$ las p restricciones de igualdad $h_i(\boldsymbol{x}) \geq 0, i = 1, 2, \dots, p,$ y minimice la función vectorial $\boldsymbol{f}(\boldsymbol{x}) = [f_1(x), f_2(x), \dots, f_k(x)]^T$, donde $\boldsymbol{x} = [x_1, x_2, \dots, x_n]^T$ es el vector de variables de decisión. El conjunto de todos los valores que satisfacen las restricciones definen una *región factible* Ω y cualquier punto $x \in \Omega$ es una solución factible.

Punto Pareto Óptimo: Un punto $\boldsymbol{x}^* \in \Omega$ es Pareto óptimo si para cada $\boldsymbol{x} \in \Omega$ y cada $I = \{1, 2, \dots, k\}$ entonces $\forall_{i \in I}(f_i(\boldsymbol{x}) = f_i(\boldsymbol{x}^*))$ o existe al menos una $i \in$ I tal que $f_i(\boldsymbol{x}) > f_i(\boldsymbol{x}^*))$.

El frente $\boldsymbol{x}^*$ es Pareto óptimo si no existen vectores factibles x que puedan aumentar algún objetivo causando simultáneamente que al menos uno de ellos empeore.

Dominancia de Pareto: Un vector $\boldsymbol{u} = (u_1, \dots, u_k)$ domina a un vector $\boldsymbol{v} = (v_1, \dots, v_k)$ (denotado como $\boldsymbol{u} \preccurlyeq \boldsymbol{v}$) si y sólo si $\boldsymbol{u}$ es parcialmente menor que $\boldsymbol{v}$, es decir, $\forall i \in \{1, \dots, k\}$, $u_i \leq v_i \ \wedge \ \exists i \in \{1, 2, \dots, k\} : u_i < v_i$.

Conjunto Óptimo de Pareto: Para un MOP $\boldsymbol{f}(\boldsymbol{x})$, el conjunto óptimo de Pareto está definido como $P^* = \{x \in \Omega \mid \neg\exists x' \in \Omega, \ \boldsymbol{f}(\boldsymbol{x}) \preccurlyeq \boldsymbol{f}(\boldsymbol{x}')\}$.

Frente de Pareto: Para un MOP $\boldsymbol{f}(\boldsymbol{x})$ y su conjunto óptimo de Pareto P^*, el frente de Pareto $PF^* = \{\boldsymbol{f}(\boldsymbol{x}), \ \boldsymbol{x} \in P^*\}$.

La meta de un MOP es encontrar su frente de Pareto. Dado que un frente de Pareto puede contener muchos puntos, una buena solución debe contener un número limitado de ellos que estén tan cerca como sea posible al frente de Pareto y además estén uniformemente distribuidos entre sí.

El algoritmo de optimización conocido como Algoritmo genético de ordenamiento elitista no dominado II (NSGA II) (Deb et al., 2002) utiliza dos mecanismos para dirigir la búsqueda de mejores soluciones. El primero de ellos ordena la población en un conjunto de frentes no dominados, siendo las soluciones del frente número uno, aquellas soluciones que no son dominadas por ninguna otra solución hasta el momento, entre más pequeño el valor es mejor, con este valor es posible a tomar decisiones al generar o reemplazar nuevas soluciones por el algoritmo; El segundo, es un mecanismo que calcula la distancia promedio entre un punto del frente y sus dos puntos más cercanos, a este se le conoce como distancia de amontonamiento, se prefieren aquellos puntos que tengan una mayor distancia, pues indica que estos están más esparcidos sobre el frente no dominado.

3.3 MOCell

Los algoritmos genéticos clásicos (AG) tienen una sola población de individuos y aplican los operadores a estos como un todo. Por el contrario existen los AG estructurados, en los cuales la población está descentralizada de alguna forma. Entre los AGs estructurados, están los modelos distribuidos y celulares, los cuales son bastante populares. En muchos casos, los algoritmos descentralizados proveen de un mejor muestreo del espacio de búsqueda, resultando en un mejor comportamiento numérico comparado con los algoritmos de una sola población (Nebro et al., 2006) (Figura 5).

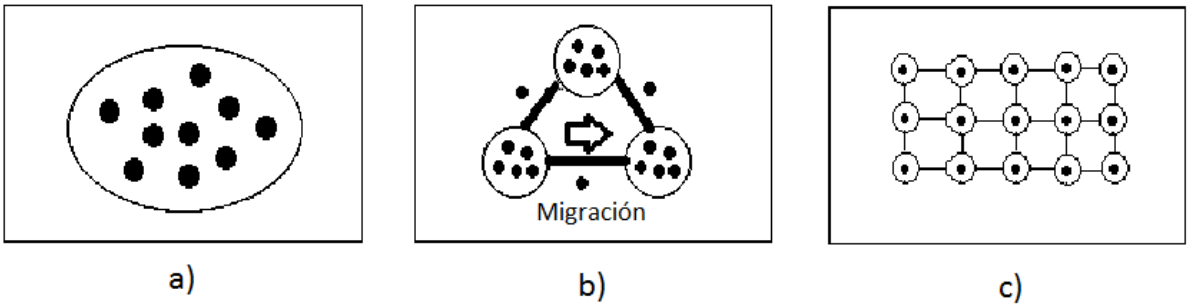


Figura 5: Tipos de AGs. En a) Un AG con una sola población. En b) Un AG distribuido, en el cual la población se divide en varias subpoblaciones y existe un intercambio entre ellas mediante un mecanismo llamado migración. En c) Un AG celular, donde cada individuo tiene asignado un único espacio en un grid (Nebro et al., 2006).

El traslape de los pequeños vecindarios del AG celular ayudan a explorar el espacio de búsqueda porque la difusión de soluciones es lenta a través de la población promoviendo la exploración (diversificación), mientras la exploración (intensificación) se lleva a cabo dentro de los vecindarios dentro por los operadores genéticos.

En un algoritmo genético celular la población está estructurada en un grid toroidal de d dimensiones ($d = 1,2,3$) y un vecindario es definido sobre él. El algoritmo opera iterativamente sobre cada individuo del grid, cada individuo actual puede interactuar únicamente con aquellos individuos en su vecindario, así que los padres se seleccionan de su vecindario mediante un criterio dado. En la Figura 6 se muestra un ejemplo de un grid de MOCell.

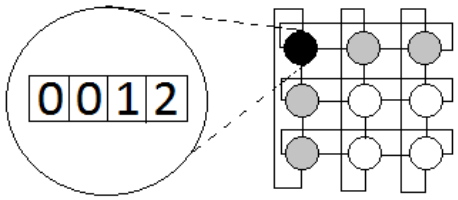


Figura 6: Ejemplo de Grid de MOCell. Cada individuo de la población se coloca en un grid toroidal. Los operadores genéticos se aplican solo entre un individuo (punto negro) y sus vecinos (puntos grises).

Para nuestro problema utilizamos un AG celular multi-objetivo llamado MOCell propuesto en (Nebro et al., 2006), su pseudocódigo se muestra en la Figura 7. MOCell comienza creando un frente de Pareto en blanco (línea 2). Los individuos se acomodan en un grid toroidal de 2 dimensiones y los operadores genéticos se aplican sucesivamente sobre ellos (líneas 7 y 8) hasta que se alcance la condición de terminación (línea 3). Por lo tanto, para cada individuo, el algoritmo consiste en seleccionar dos padres de

su vecindario, recombinándolos para obtener los hijos, mutándolos, evaluando el individuo resultante, e insertándolo en la población auxiliar (si este no es dominado por el individuo actual) y en el frente de Pareto. Finalmente, tras cada generación, la población vieja es reemplazada por la auxiliar, y se ejecuta un procedimiento de retroalimentación (línea 14) el cual remplaza un número fijo de individuos de la población por soluciones del mejor frente encontrado.

MOCell()

```
1   Configurar_parámetros()
2   Frente_Pareto = Crear_Frente() //Crea frente en blanco
3   While !CondiciónTerminación()
4       for individuo = 1 to mocell.tamPoblacion do
5           v_list = Leer_Vecinos(mocell.posición(individuo))
6           padres = Selección(v_list)
7           hijos = Recombinación(mocell.Pc, padres)
8           hijos = Mutacion(mocell.Pm, hijos)
9           Evaluar_Aptitud(hijos)
10          Insert(posición(individuo), hijos, aux_pob)
11          Insertar_Pareto(individuo)
12      endfor
13      mocell.pob = aux_pob
14      mocell.pob = FeedBack(mocell, Frente_Pareto)
15  endwhile
```

Figura 7: Pseudocódigo de MOCell

3.4 Representación de solución

Cada servidor es representado mediante un ID único. El cromosoma es representado por un arreglo que tiene una longitud igual a la cantidad de VMs. Cada VM tiene un ID único que corresponde con un dato en el cromosoma, el cromosoma guarda un ID del servidor, indicando a cuál servidor fue asignada la VM. En la Figura 8 se muestra un ejemplo de esta representación.

1	2	2	1	2

Figura 8: Ejemplo de representación de solución. Un ejemplo con 5 VMs y 2 servidores, en esta solución la VM 1 está asignada al servidor 1, la VM 2 está asignada al servidor 2, y así sucesivamente.

3.5 Operadores de cruzamiento

Para determinar cuál es el operador de cruzamiento más adecuado para nuestro problema se implementaron los algoritmos de cruzamiento clásicos: cruzamiento en un punto y uniforme. Además se propuso el operador SLA Uniforme. A continuación se describe cada uno de ellos.

3.5.1 Cruzamiento uniforme

Este funciona tratando cada gen independientemente y haciendo una selección aleatoria sobre de cual padre debe heredar el gen. Esto se implementa generando una cadena de l variables aleatorias de una distribución uniforme sobre $[0, 1]$. En cada posición, si el valor está por debajo de un parámetro p, (en este trabajo usamos 0.5), el gen es heredado del primer padre y en caso contrario del segundo. El segundo hijo se crea usando el mapeo inverso. Este operador cuenta con la propiedad de que existe una tendencia de transmitir el 50% de los genes de cada padre y está en contra de transmitir un gran número de genes co-adaptados de un padre, este efecto es conocido como sesgo distribucional (Eiben et al., 2008). En la Figura 9 se puede observar el pseudocódigo de este operador seguido de un ejemplo de este en Figura 10.

Cruzamiento_Uniforme(*padres, umbral*)

```
1   hijo1 = padres. elemento(0)
2   hijo2 = padres. elemento(1)
3   lista_serv = leer_Vms()
4   lista_vms = leer_Servidores()
5   for i = 0 to tamaño(lista_vms):
6       real_aleatorio = Aleatorio_Real(0, 1)
7       If real _aleatorio ≤ umbral:
8           hijo1.asignaVm(lista_vms.elemento(i)), servidor2)
9           hijo2.asignaVm(lista_vms.elemento(i)), servidor1)
10      endif
11  endfor
12  return nueva_Lista(hijo1, hijo2)
```

Figura 9: Pseudocódigo de cruzamiento uniforme.

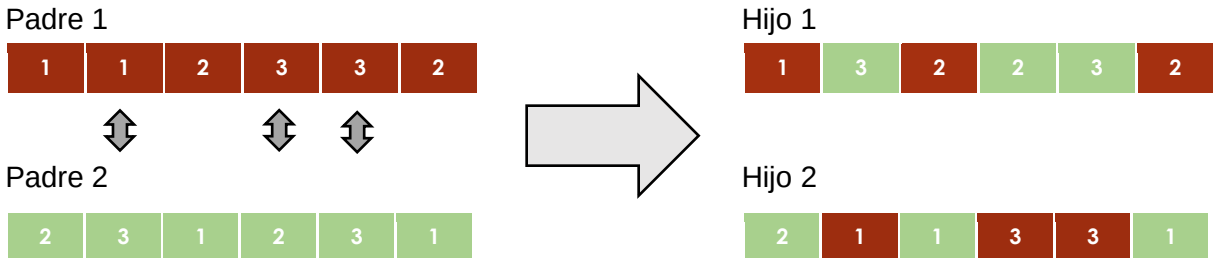


Figura 10: Ejemplo de cruzamiento uniforme sobre nuestra representación. A la izquierda se encuentran el padre 1 (en rojo) y el padre 2 (en verde), a la derecha se muestran ambos hijos, las flechas indican cuál de los genes fueron seleccionados para intercambiarse, en este caso, los alelos 2, 4 y 5 se intercambian entre el padre 1 y el padre 2.

3.5.2 Cruzamiento uniforme SLA

Este cruzamiento trata cada gen de forma independientemente (Figura 11 y Figura 12), genera un número aleatorio con el cual se decide si se hará un intercambio de los valores de dicho gen en cada padre, posteriormente verifica si al hacer dicho intercambio se empeoran las violaciones al SLA (Líneas 16, 20 y 22) y se hacen sólo aquellos intercambios que no aumenten el número de violaciones a SLA, por lo que ninguno de los hijos podrá tener un valor de SLA peor que sus padres.

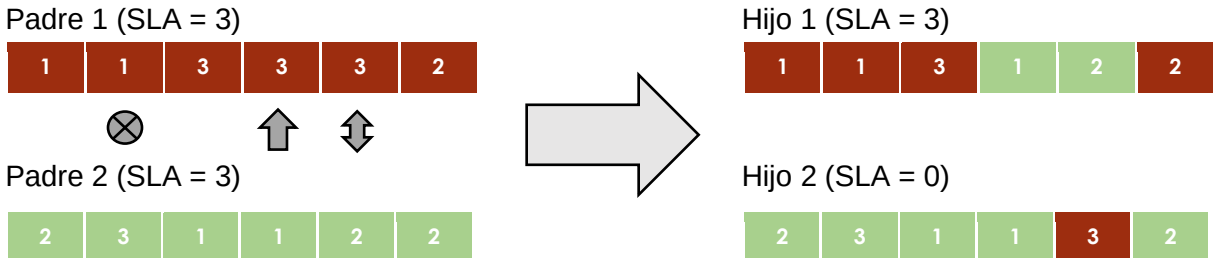


Figura 11: Ejemplo de cruzamiento uniforme SLA. A la izquierda se encuentran el padre 1 (en rojo) y el padre 2 (en verde), a la derecha se muestran ambos hijos, los símbolos entre ambos padres en los alelos 2, 4 y 5 indican que estos genes fueron considerados para intercambiar el valor de los alelos (En ese orden). En el alelo 2, no se hizo ningún intercambio, pues de hacerlo se aumentaría el SLA de ambos padres. En el alelo 4, se copia el contenido del alelo del padre 2 al padre 1, lo contrario no se hace pues aumentaría el SLA. En el alelo 5, se intercambia el valor de ambos alelos, pues este intercambio no aumenta el SLA. El Hijo 1 tiene el mismo SLA que su padre, sin embargo, el hijo 2 tiene un SLA más pequeño que el del padre 2.

```
Cruzamiento_SLAUniforme(padres)
1   hijo1 = padres. elemento(0)
2   hijo2 = padres. elemento(1)
3   lista_serv = leer_Servidores()
4   lista_vms = leer_Vms()
5   for i = 0 to tamaño(lista_vms):
6       copia_hijo1 = hijo1
7       copia_hijo2 = hijo2
8       servidor1 = hijo1.servidorAsignado(lista_vms.elemento(i))
9       servidor2 = hijo2.servidorAsignado(lista_vms.elemento(i))
10      copia_hijo1.asignaVm(lista_vms.elemento(i)), servidor2)
11      copia_hijo2.asignaVm(lista_vms.elemento(i)), servidor1)
12      Evaluar_Aptitud(copia_hijo1)
13      Evaluar_Aptitud(copia_hijo2)
14      entero_aleatorio = Aleatorio(0, 1)
15      If entero_aleatorio = 0:
16          If hijo1.SLA ≥ copia_hijo1.SLA:
17              hijo1.asignaVm(lista_vms.elemento(i)), servidor2)
20          else if hijo2.SLA ≥ copia_hijo2.SLA:
21              hijo2.asignaVm(lista_vms.elemento(i)), servidor1)
24          endif
25      endif
26  endfor
27  return nueva_Lista(hijo1, hijo2)
```

Figura 12: Pseudocódigo de cruzamiento SLA Uniforme.

3.5.3 Cruzamiento en un punto

Este cruzamiento funciona generando un número aleatorio en el rango $[1, l - 1]$ (donde l es la longitud de la representación) y partiendo ambos padres en dicho punto, creando dos hijos mediante el intercambio de las colas (Eiben et al., 2008), en la Figura 13 se muestra el pseudocódigo del cruzamiento en un punto seguido de un ejemplo en Figura 14.

```
CruzamientoUnPunto(padres)
1  padre1 = padres.elemento(0)
2  padre2 = padres.elemento(0)
3  entero_aleatorio = Aleatorio(0, tamaño(padre1))
4  partesPadre1 = Dividir(padre1, entero_aleatorio)
5  partesPadre2 = Dividir(padre2, entero_aleatorio)
6  hijo1 = Concatenar(partesPadre1.elemento(0), partesPadre2.elemento(1))
7  hijo2 = Concatenar(partesPadre2.elemento(0), partesPadre1.elemento(1))
8  return nueva_Lista(hijo1, hijo2)
```

Figura 13: Pseudocódigo cruzamiento un punto

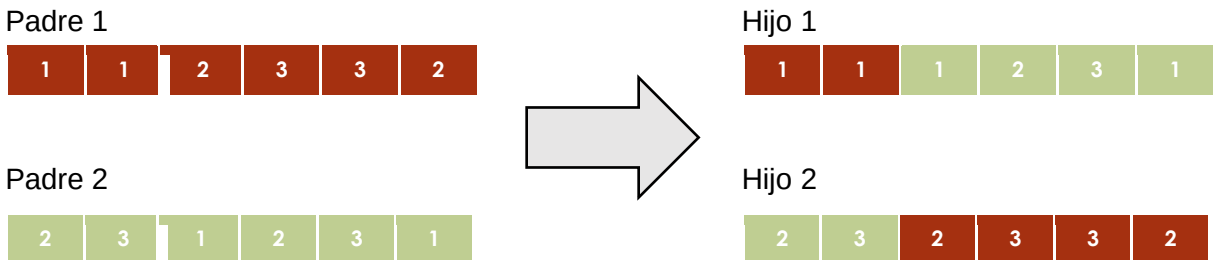


Figura 14: Ejemplo de cruzamiento en un punto. A la izquierda se encuentran el padre 1 (en rojo) y el padre 2 (en verde), en la parte derecha se muestran ambos hijos. El punto de cruzamiento se muestra en los padres mediante una franja blanca gruesa, ambos hijos se concatenan tomando el punto de cruce como referencia.

3.6 Operadores de mutación

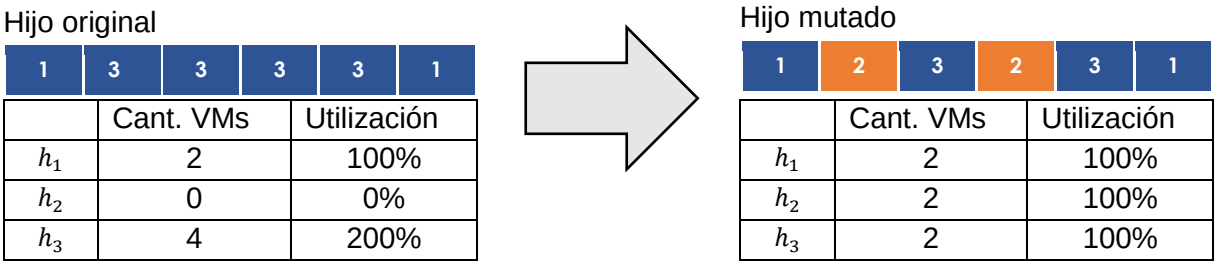
Para determinar cuál es el operador de mutación más adecuado para nuestro problema se implementaron los algoritmos de mutación clásicos: Cambio de bit e intercambio. Además se propuso el operador Local Search SLA, balanceo de concentración e intercambio VM. A continuación se describe cada uno de ellos.

3.6.1 Búsqueda Local SLA

El operador de mutación búsqueda local SLA busca los servidores que tienen más carga, y mueve una todas de sus VMs una a una los servidores con menos carga hasta que estos servidores con dejen de superar su capacidad. Este operador está especialmente diseñado para disminuir la cantidad de violaciones a SLA. En la Figura 15 se muestra el pseudocódigo del operador seguido de un ejemplo en Figura 16.

```
LocalSearchSLA(solución, lista_serv, lista_vms)
1   lista_maxServ = max_util(lista_serv, lista_vms)  // Busca el servidores con mayor utilización
2   lista_asignadas = asignadas(solucion, lista_maxServ)  // Crea una lista con las VMs asignadas a los
    servidores lista_maxServ
3   vms_a_mover = tamaño(lista_asignadas)
4   for i to vms_a_mover:
5       vm_mov = lista_asignadas.pop() // Selecciona aleatoriamente VM a mover
6       serv_inicio =  solución.servidorAsignado(vm_mov)
7       lista_minserv = min_util(lista_serv, lista_vms) // Busca servidores con menos utilización
8       serv_dest  =  lista_minserv.indice(entero_aleatorio)  // Selecciona aleatoriamente servidor
    destino
9       asigna_VM(solución, serv_dest, vm_mov) // Mueve la VM
    endfor
10  return solución
```

Figura 15: Pseudocódigo mutación Búsqueda local SLA



	Cant. VMs	Utilización
h_1	2	100%
h_2	0	0%
h_3	4	200%

	Cant. VMs	Utilización
h_1	2	100%
h_2	2	100%
h_3	2	100%

Figura 16: Ejemplo de ejecución de operador Búsqueda Local SLA con servidores de capacidad 2. El servidor 3 es el que tiene mayor carga, el servidor 2 tienen la menor carga. Las VMs en color naranja fueron las que se reasignaron después de la ejecución del operador.

3.6.2 Balanceo de concentración

El operador Balanceo de concentración selecciona dos servidores aleatoriamente y balancea la concentración de VMs entre ellos, es decir, toma todas las VMs de ambos tipos CPU Intensivas y Memoria intensivas y asigna la mitad de cada tipo a cada uno de los servidores, además, este operador asigna las VMs a cada servidor de acuerdo a su duración para de esta forma reducir el tiempo de ejecución en cada servidor. En la Figura 18 se muestra el pseudocódigo del operador seguido de un ejemplo en la Figura 17.

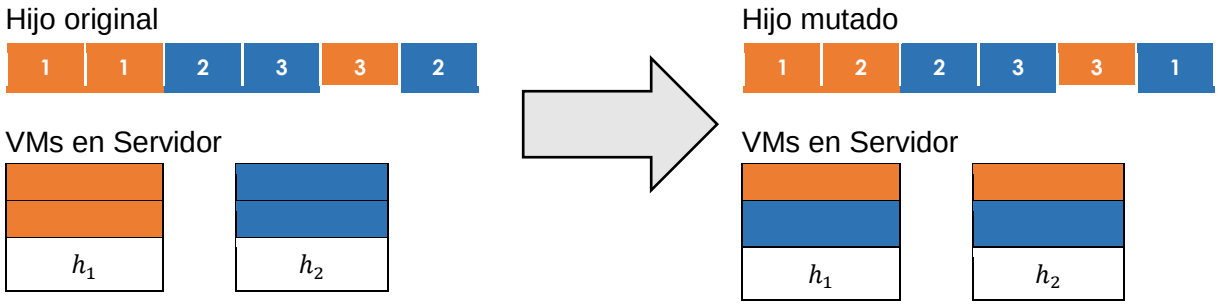


Figura 17: Ejemplo de mutación Balanceo de concentración. El color naranja representa las VMs memoria intensivas, el color azul las CPU intensivas. En este caso, se seleccionaron el servidor $\boldsymbol{h_1}$ y $\boldsymbol{h_2}$ y balancea la concentración entre ellos.

BalanceoConcentración(*solución, lista_serv*)

```
1   entero_aleatorio = Aleatorio(0, tamaño(lista_serv)
2   servidor1 = lista_serv.pop(entero_aleatorio)
3   resultado  = nueva_solucion(solución)
4   entero_aleatorio = Aleatorio(0, tamaño(lista_serv)
5   servidor2 = lista_serv.pop(entero_aleatorio)
6   lista_asignadas_serv1 = asignadas(solución, servidor1)
7   lista_asignadas_serv2 = asignadas(solución, servidor2)
8   lista_vms_asignadas = nueva_Lista(lista_asignadas_serv1, lista_asignadas_serv2)
9   lista_CI = lista_vms_asignadas.filtrar(CPU_Intensive)
10  lista_MI = lista_vms_asignadas.filtrar(MI_Intensive)
11  ordenar_tiempos_decremental(lista_CI)
12  ordenar_tiempos_decremental(lista_MI)
13  bandera = verdadero
14  lista_vmsOrdenadas = concatenar(lista_CI, lista_MI)
15  while  tamaño(lista_vmsOrdenadas) > 0:
16      vmActual = lista_vmsOrdenadas.pop()
17      if bandera = true:
18          asigna_VM(resultado, servidor1, vmActual)
19      else:
20          asigna_VM(resultado, servidor2, vmActual)
21      endif
22  bandera != bandera  // Invierte valor lógico de la bandera
23  endwhile
24  return resultado
```

Figura 18: Pseudocódigo condición de Balanceo de concentración.

3.6.3 Cambio de bit

Cambio de bit, por su traducción al español BitFlip (BF). Este operador recorre cada uno de los alelos de la solución y posteriormente genera un número aleatorio entre $[0, 1]$ y si este es mayor que un valor umbral se reemplaza el valor de este alelo por un valor aleatorio generado entre un rango de valores válidos (Eiben et al., 2008). En el contexto de nuestro problema, este toma una VM al azar y la asigna a un host al azar. En Figura 19 se muestra el pseudocódigo de este operador, en la Figura 20 se muestra un ejemplo del funcionamiento del operador.

```
CambioDeBit (solución, probabilidad, lista_serv)
1   for vm  in solución:
2       real_aleatorio = Aleatorio_Real(0, 1)
3       if real_aleatorio ≤ probabilidad:
4           servidor_aleatorio = Aleatorio(0, tamaño(lista_serv)
5           asigna_VM(solución ,vm, servidor_aleatorio)
6       endif
7   endfor
8   return solución
```

Figura 19: Pseudocódigo de mutación cambio de bit.

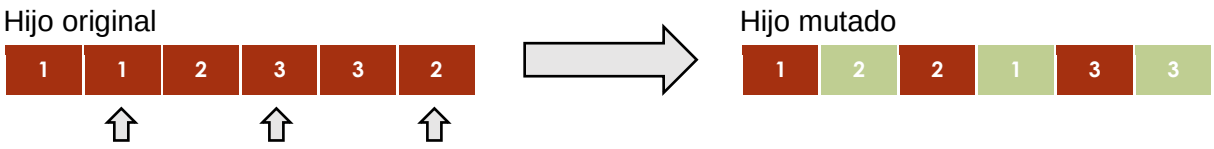


Figura 20: Ejemplo de mutación cambio de bit. A la izquierda se encuentran el hijo antes de la mutación, a la derecha se encuentra el hijo después de mutar. Los alelos 2, 4 y 6 cambiaron su valor aleatoriamente dentro de una serie de valores admisibles.

3.6.4 Intercambio

Por su traducción del inglés, Swap (SW). Este operador selecciona dos máquinas virtuales aleatoriamente e intercambia los hosts a los que estas fueron asignadas. En la Figura 21 se muestra el pseudocódigo de este operador. En la Figura 22 se muestra el pseudocódigo.

Intercambio(*solución, lista_vms*)	
1	*entero_aleatorio* = Aleatorio(0, tamaño(*lista_vms*))
2	*resultado* = nueva_solucion(*solución*)
3	*vm1* = lista_vms.elemento(*entero_aleatorio*)
4	*entero_aleatorio* = Aleatorio(0, tamaño(*lista_vms*))
5	*vm2* = lista_vms.elemento(*entero_aleatorio*)
6	*servidor1* = servidor_asignado(*solución*, *vm1*)
7	*servidor2* = servidor_asignado(*solución*, *vm2*)
8	asignar(*resultado*, *servidor1*, *vm2*)
9	asignar(*resultado*, *servidor2*, *vm1*)
10	**return** *resultado*

Figura 21: Pseudocódigo de mutación Intercambio.

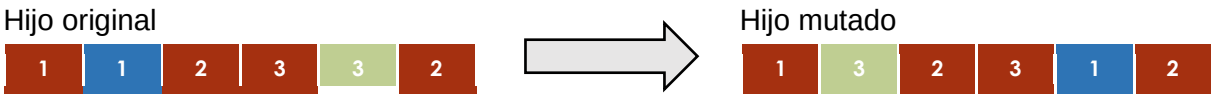


Figura 22: Ejemplo de mutación Intercambio. A la izquierda se encuentran el hijo antes de la mutación, a la derecha se encuentra el hijo después de mutar. La VM 1 (en azul) es reasignada al servidor 3, la VM 5 (en verde) es reasignada al servidor 1.

3.6.5 Intercambio VM

El operador Intercambio VM funciona identificando cuales son los hosts más cargados, si todos los host tienen la misma utilización (línea 4-5) se hace un intercambio normal. En caso contrario (líneas 7 a 12), significa que existen servidores con distinta utilización, en este caso, se selecciona aleatoriamente una de las VM asignada a los servidores más utilizados (línea 9) y un host de los menos utilizados (línea 11) y se asigna esta VM este servidor (línea 12), es decir, se mueve una VM asignada en uno de los servidores más utilizados a uno de los servidores menos utilizados. En la Figura 23 se muestra el pseudocódigo seguido de un ejemplo en la Figura 24.

IntercambioVM(*solución*)

```
1   lista_max_serv = max_util(solución)  // Busca los servidores con mayor utilización
2   lista_minserv = min_util(solución) // Busca servidores con menor utilización
3   resultado = nueva_solucion(solución) // Crea copia de la solución
4   if tamaño(lista_max_serv) = 1:
5       resultado = Intercambio(solución)
6   else:
7       lista_vms_asignadas = asignadas(solución, lista_max_serv)  // Crea una lista con las VMs
          asignadas a los servidores más utilizados
8       entero_aleatorio = aleatorio(0, tamaño(lista_vms_asignadas))
9       vm = lista_vms_asignadas.elemento(entero_aleatorio)
10      entero_aleatorio = aleatorio(0, tamaño(lista_minserv))
11      serv_destino = lista_ minserv.elemento(entero_aleatorio)
12      asigna_VM(resultado, serv_destino, vm)
13  endif
14  return resultado
```

Figura 23: Pseudocódigo de mutación intercambio VM.

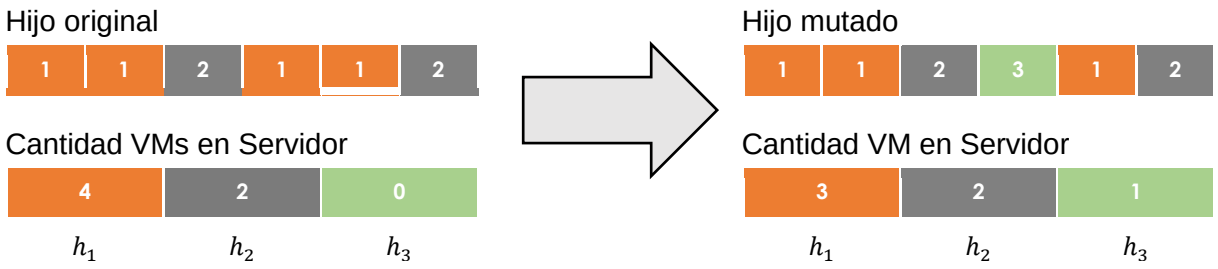


Figura 24: Ejemplo de mutación Intercambio VM. A la izquierda se encuentran el hijo antes de la mutación, a la derecha se encuentra el hijo después de mutar, en la parte inferior se muestra la cantidad de VMs en cada servidor antes y después de la mutación. El color naranja, representa en la VM está asignada a uno de los servidores más cargados, este color en el host representa que el servidor es uno de los que tiene mayor carga. El color verde en la VM representa que esta fue asignada a uno de los host con menor carga o a un servidor con poca carga. El color gris representa que la VM o servidor no pertenece a los más utilizados ni a los menos utilizados. En este ejemplo, la VM 4 fue reasignada del servidor más cargado $\boldsymbol{h_1}$ al servidor con menos carga $\boldsymbol{h_3}$.

Capítulo 4. Heurísticas Clásicas

Para tener un punto de comparación entre los algoritmos clásicos de consolidación y los nuestros, implementamos algunas de las heurísticas clásicas de calendarización. Estas heurísticas generan soluciones sin violaciones a la calidad de servicio, las cuales son comparadas con nuestros algoritmos. Además, propusimos la heurística concentración FFit la cual hace una asignación de VMs consciente del tipo de aplicación de las mismas.

4.1 Round Robin

La heurística round robin reparte las VMs a los servidores uno a uno. Es decir, Round Robin (RR) asigna la primera VM al primer servidor, segunda VM a segundo servidor, etc. una vez que llega al último servidor (Línea 7 y 8) repite el proceso anterior, hasta que no haya más VMs (Figura 25).

RoundRobin (*Servidores*, *lista_vms*, *lista_servidores*)

```
1   índiceServ = 1
2   solución = nueva_solución()
3   While tamaño(lista_Vms) > 0:
4       vmActual = lista_Vms.pop()
5       asigna_VM(solución, vmActual, índiceServ)
6       If (índiceServ > tamaño(lista_servidores)):
7           índiceServ = 1
8       Else:
9           índiceServ = índiceServ + 1
10      EndIf
11  EndWhile
12  return solución
```

Figura 25: Pseudocódigo de heurística RoundRobin.

4.2 Primer ajuste

Por su traducción del inglés, first fit (FFit). La heurística FFit toma el primer servidor disponible y le asigna VMs una a una, hasta que no se puedan asignar más VMs sin violar el SLA, en este momento se selecciona el siguiente servidor y se repite el proceso hasta que no haya más VMs (Figura 26).

PrimerAjuste(*lista_Vms, lista_servidores*)

```
1   solución = nueva_solución()
2   While tamaño(Servidores) > 0:
3       servidorActual = lista_servidores.pop()
4       While (!Lleno(lista_servidores)):  // Mientras la utilización del servidor sea menor a 100%
5           vmActual = lista_Vms.pop()
6           If (Cabe(servidorActual, vmActual)):  // Verifica si al asignar la VM supera la capacidad
7               asigna_VM(solución, vmActual, servidorActual)
8           else:
9               break;  // La Vm no cabe en el servidor, seleccionamos el siguiente
10      EndWhile
11  EndWhile
12  return solución
```

Figura 26: Pseudocódigo de heurística primer ajuste.

4.3 Aleatorio

La heurística aleatoria (Rand) asigna las VMs aleatoriamente a cada uno de los servidores. Es decir, selecciona aleatoriamente a que servidor asignar la VM de una lista de servidores admisibles. Un servidor admisible es aquel que puede aceptar la asignación de la VM sin violar el SLA (Figura 27).

```
Aleatorio(lista_servidores, lista_Vms)
1   índiceVm = 0
2   solución = nueva_solucion()
3   While tamaño(lista_servidores) > 0:
4       vmActual = lista_Vms.pop()
5       serv_aleatorio = Aleatorio(1, tamaño(Servidores))
6       asigna_VM(solución, vmActual, serv_aleatorio)
7   EndWhile
8   return solución
```

Figura 27: Pseudocódigo de heurística aleatorio.

4.4 Concentración FFit

La heurística concentración FFit (CFFit) es consciente del tipo de aplicación. Esta trata de emparejar la cantidad de VMs CPU intensivas y memoria intensivas en un servidor. Los servidores que tienen una mayor diferencia entre la cantidad de VMs CI y MI son desbalanceados y, en caso contrario, se les llama balanceados. Esta heurística trata de asignar las VMs con tiempos de ejecución más largos en servidores balanceados y asignar las VMs más cortas a servidores desbalanceados. Esta trata de minimizar el tiempo necesario para ejecutar los servidores desbalanceados minimizando como consecuencia el consumo energético.

ConcentraciónFFit(*lista_servidores*, *lista_Vms*)

```
1   lista_VmCi = lista_Vms.filtrar(CPUIntensive)
2   lista_VmMi = lista_Vms.filtrar(MemoryIntensive)
3   ordenar_tiempos_decremental(lista_VmCi)
4   ordenar_tiempos_decremental(lista_VmMi)
5   solución = nueva_solucion()
6   While tamaño(lista_VmCi) > 0 or tamaño(lista_VmMi) > 0:
7       servidorActual = lista_servidores.pop()
8       alterna = true // Bandera para alternar la asignación de VMs CPI y MI en el servidor actual
9       While (!Lleno(servidorActual)):  // Mientras la utilización del servidor sea menor a 100%
10          if alterna = true:
11              vmActual = lista_VmCi.pop()
12          else:
13              vmActual = lista_VmMi.pop()
14          endif
15          alterna != alterna // Invierte el valor lógico de la bandera
16          If (Cabe(servidorActual, vmActual)):  // Verifica si al asignar la VM supera la capacidad
17              asigna_VM(solución, servidorActual, vmActual)
18          else:
19              break;  // La Vm no cabe en el servidor, seleccionamos el siguiente
20      endif
21  EndWhile
22  return solución
```

Figura 28: Pseudocódigo de heurística Concentración FFit.

Capítulo 5. Metodología Experimental

En esta sección se muestra las herramientas usadas para llevar a cabo nuestro experimento. Utilizamos una librería llamada JMetal la cual es un conjunto de algoritmos metaheurísticos para optimización multi-objetivo. Además se muestra la carga de trabajo utilizada y la metodología de análisis de resultados.

5.1 JMetal Optimization Framework

Para nuestros experimentos utilizamos JMetal 4.5.2, este es un framework desarrollado para la optimización multi-objetivo utilizando metaheurísticas (Durillo et al., 2010)(Durillo and Nebro, 2011). Este se desarrolló con el objetivo de ser fácil de usar, portable, flexible y extensible. Su licencia es del tipo GNU Lesser General Public License y puede ser obtenido de forma gratuita en su página oficial.

Entre las características de JMetal están:

- Implementación de varios algoritmos de optimización multiobjetivo como: NSGA-II, SPEA2, PAES, PESA-II,OMOPSO, MOCell, AbYSS, MOEA/D, Densea, CellDE, GDE3, FastPGA, IBEA, SMPSO, MOCHC y SMS-EMOA.
- Un conjunto de problemas de prueba: Zitzler-Deb-Thiele, Kursawe, Srinivas, Tanaka, etc.
- Implementación de indicadores de calidad: Híper volumen, propagación, distancia generacional, distancia generacional invertida, épsilon.
- Diferentes representaciones de variables: binarias, reales, enteros, permutaciones.

JMetal está hecho siguiendo una arquitectura orientada a objetos que facilitan la inclusión de nuevos componentes y el uso de los existentes. Las principales clases de JMetal son las siguientes:

- *Algorithm*: Representa el padre de todos los optimizadores, todas las metaheuristcas en JMetal deben heredar de ella. Una declaración de algoritmo puede requerir de parámetros los cuales pueden ser accedidos mediante los métodos *addParameter()* y *getParameter()*. Además, existen métodos análogos para la adición de operadores (*addOperator()*) y consulta de operadores (*getOperator())*. Su método principal es *excecute()* el cual comienza la ejecución del algoritmo.
- *SolutionSet:* representa un conjunto de soluciones. Cada objeto *Solution* tiene un objeto asociado del tiempo (*SolutionType*) y está compuesto de un arreglo de objetos de tipo *Variable*. El esquema de JMetal

es muy flexible porque un objeto *Solution* no está restringido a contener variables de la misma representación; en cambio, puede estar compuesto de un arreglo de distintos tipos de variables.

- *Problem:* Contiene dos métodos básicos: *evaluate()* y *evaluateConstraints()*. Ambos métodos reciben un objeto *Solution* el cual representa una solución candidata al problema; el primero la evalúa, y el segundo determina las violaciones totales a las restricciones de la solución.
- *Operator*: Es una superclase que representa los operadores genéricos utilizados por distintos algoritmos (Ej.: Mutación, cruce y selección son operadores de algoritmos genéticos). Esta contiene los métodos *getParameter()* y *setParameter()* los cuales son usados para añadir valores específicos al operador (Por ejemplo probabilidad de mutación de un algoritmo genético).

5.2 Carga de trabajo

Para que una simulación entregue resultados útiles es necesario utilizar datos de cargas de trabajo que se acerquen lo más posible a la de los sistemas reales de cómputo. Utilizaremos bitácoras del sitio *Parallel Workloads Archive* el cual es bastante citado y popular entre los investigadores, éste se encarga de proveer gratuitamente datos de calidad (sin errores, consistentes, completos, etc) de cargas de trabajo de sistemas reales, en (Feitelson et al., 2014) se discute sobre la calidad de los datos de éste sitio. Consideraremos cada tarea en la bitácora como una VM que ejecuta un tipo de carga específico (CI, MI, DI y RI).

Debido a que las bitácoras de *Parallel Workloads Archive* no cuentan con utilización de CPU ni tipo de tarea, es necesario agregar estos parámetros mediante algún mecanismo. Suponemos que cada usuario ejecuta tareas de un solo tipo, por lo tanto, con el objetivo de agregar el tipo de VM, asignaremos de forma equiprobable a cada ID del usuario un único tipo de VM.

Nuestro log consiste en 28 días de tareas, éstas contienen nueve rastros de: DAS2—Universidad de Amsterdam, DAS2—universidad de tecnología Delft, DAS2—universidad Utrecht, DAS2—universidad Leiden, KTH, DAS2—Vrije universidad Amsterdam, HPC2N, CTC, y LANL. En la Figura 29 se puede observar la distribución de tareas de cada uno de los días.

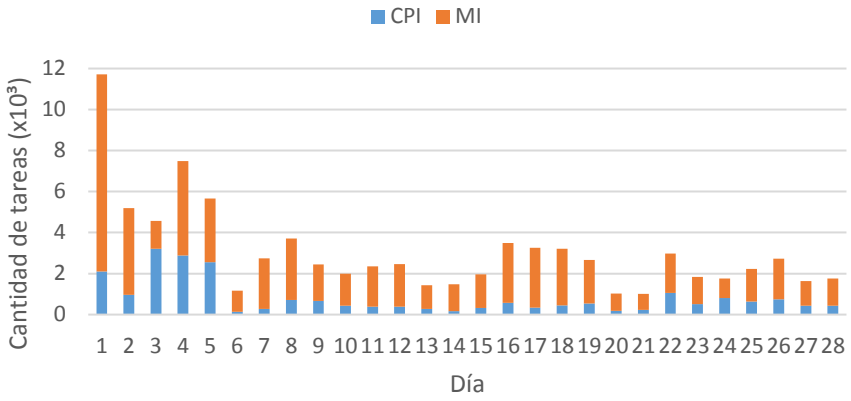


Figura 29: Cantidad de VMs de carga de trabajo por día.

5.3 Cota inferior de consumo de energía

Con el objetivo de crear un punto de referencia robusto con el cuál medir la efectividad de nuestros algoritmos creamos un frente teórico, el cual consiste de dos puntos: una solución con 0 violaciones al SLA y otra con una solución con una cantidad de violaciones al SLA igual a la cantidad de VMs (máximo posible). En la Figura 30 se encuentra el pseudocódigo de este procedimiento.

Para la solución con ninguna violación esta cota trata de asignar las VMs con tiempos de ejecución más largos en servidores con concentración balanceada y asignar las VMs más cortas a servidores con concentración desbalanceada. Además, toma el menor tiempo de ejecución de todas las VMs asignadas al servidor (línea 23) y reemplaza el tiempo de ejecución de todas las VMs por este valor (Línea 24).

Para la solución con SLA igual a la cantidad de VMs, se asignan todos las VMs al primer servidor (línea 18) y acota el tiempo al mínimo de todas las VMs (línea 28 y 29).

CotaInferior(*lista_servidores*, *lista_Vms*)

```
1   frente = nuevo_frente()
2   lista_VmCi = lista_Vms.filtrar(CPUIntensive)
3   lista_VmMi = lista_Vms.filtrar(MemoryIntensive)
4   ordenar_tiempos_decremental(lista_VmCi)
5   ordenar_tiempos_decremental(lista_VmMi)
6   NoSLAsolución = nueva_solución()
7   SLAsolución = nueva_solución()
8   While tamaño(lista_VmCi) > 0 or tamaño(lista_VmMi) > 0:
9       servidorActual = lista_servidores.pop()
10      alterna = true // Bandera para alternar la asignación de VMs CPI y MI en el servidor actual
11      While (!Lleno(servidorActual)):  // Mientras la utilización del servidor sea menor a 100%
12          if alterna = true:
13              vmActual = lista_VmCi.pop()
14          else:
15              vmActual = lista_VmMi.pop()
16          endif
17          alterna != alterna // Invierte el valor lógico de la bandera
18          asinga_VM(SLAsolución, lista_servidores.elemento(0), vmActual)
19          If (Cabe(servidorActual, vmActual)):  // Verifica si al asignar la VM supera la capacidad
20              asigna_VM(NoSLAsolución, servidorActual, vmActual)
21          else:
22             vmsAsignadas = vmAsignadas(NoSLAsolución, servidorActual)
23             tiempo  = min_tiempo(vmsAsignadas)
24              acotar_tiempo(NoSLAsolución, vmsAsignadas, tiempo)
25              break;  // La Vm no cabe en el servidor, seleccionamos el siguiente
26         endif
27  EndWhile
28  tiempo  = min_tiempo(lista_Vms)
29  acotar_tiempo(SLAsolución, lista_Vms, tiempo)
30  frente.agregarSolución(NoSLAsolución, SLAsolución)
31  return frente
```

Figura 30: Pseudocódigo de Cota Inferior.

5.4 Degradación del desempeño

Con el propósito de escoger la mejor estrategia, utilizaremos el método del perfil de desempeño, el cual es propuesto en (Dolan and Moré, 2002), a continuación se da una breve descripción de su funcionamiento.

Suponga que se resuelven un conjunto de problemas (experimentos) P mediante un conjunto de algoritmos S , se generan resultados y se guarda la información de interés de la corrida de cada algoritmo, por ejemplo: el tiempo de ejecución, número de evaluaciones de función objetivo, etc.

La cantidad de problemas está representada por n_p, la cantidad de algoritmos representada por n_s. Si por ejemplo, estuviéramos interesados en medir el rendimiento del tiempo de cómputo (aunque esta idea aplica para cualquier medida), para cada problema p y algoritmo s, definimos:

$$t_{p,s} = Tiempo\ de\ computo\ para\ resolver\ el\ problema\ p\ por\ el\ algoritmo\ s$$

Para comparar el rendimiento del algoritmo s en el problema p, dividimos el rendimiento de $t_{p,s}$ entre el mejor rendimiento encontrado para el problema; es decir, usamos la razón de desempeño:

$$r_{p,s} = \frac{t_{p,s}}{\min\{\, t_{p,s} : \ s \in S \,\}}$$

Aunque el rendimiento de un algoritmo s en un problema p puede ser de interés, nosotros buscamos obtener una conclusión general del desempeño del algoritmo. Por lo tanto utilizamos el perfil de desempeño $p_s(\tau)$, el cual se define como:

$$p_s(\tau) = \frac{1}{n_p} tamaño\{\, p \in P : \ r_{p,s} \leq \tau \,\}$$

El perfil de desempeño $p_s(\tau)$ es la probabilidad de que la razón de desempeño del algoritmo $s \in S$ se encuentre a un factor τ de la mejor razón posible. Una propiedad importante de los perfiles de desempeño es que son inmunes a los resultados extremos en un pequeño número de problemas, por lo que evita que resultados muy variables en problemas aislados puedan alterar el análisis.

5.5 Indicadores de calidad de frentes no dominados

El resultado de correr un algoritmo multi-objetivo es una colección de vectores no dominados, esto complica el análisis del desempeño de estos ya que para compararlos es necesario comparar los conjuntos no dominados que producen.

No existe una única manera de medir la calidad de los frentes no dominado, por ejemplo, si llamamos Z^* al frente óptimo de Pareto, supongamos que tenemos una solución de un único punto $a \in Z^*$ y otra que consiste de un conjunto amplio de puntos el cual no es dominado por a ni se encuentra en Z^*. Comparando ambas soluciones, la primera tiene un solo punto del frente óptimo de Pareto, lo que significa que no existe otro punto que la domine; Por el contrario, aunque los puntos del segundo frente no pertenecen al frente Pareto óptimo tiene una representación más amplia del mismo, y sus puntos podrían no encontrarse tan lejos del él (Knowles and Corne, 2002). Existen varias medidas de calidad para comparación de frentes no dominados, como el híper volumen, distancia generacional, error máximo de frente de Pareto, cubrimiento de conjuntos, etc. En este trabajo utilizamos el híper volumen y cubrimiento de conjuntos para medir la calidad de nuestros frentes.

5.5.1 Híper volumen

El indicador de calidad del híper volumen calcula el volumen de la región multidimensional encerrada entre un punto de referencia y los puntos del frente, por lo que calcula el área de la región que dicho frente domina (Figura 31). Entre las ventajas de este indicador están el que es independiente (es una propiedad del frente en sí mismo), es independiente al escalamiento, su concepto es intuitivo, y por último, permite ordenar los frentes de acuerdo a este valor.

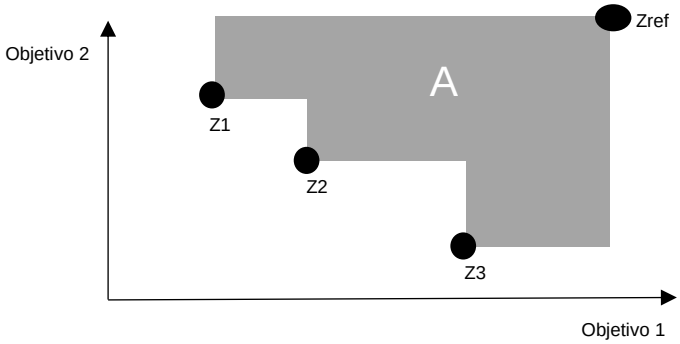


Figura 31: Ejemplo de híper volumen en un problema bi-objetivo de minimización. Los puntos Z1, Z2, Z3 pertenecen al frente no dominado, Zref es el punto de referencia para el cálculo del volumen, A es el híper volumen del frente.

5.5.2 Cubrimiento de conjuntos

Este indicador de calidad se propuso en (Zitzler and Thiele, 1998). Sean $\boldsymbol{A}, \boldsymbol{B} \subseteq \boldsymbol{X}$ dos conjuntos de vectores de decisión. La función $C(\boldsymbol{A}, \boldsymbol{B})$ mapea el par ordenado $(\boldsymbol{A}, \boldsymbol{B})$ al intervalo $[0, 1]$:

$$\boldsymbol{C(A,B)} \coloneqq \frac{|\{\, \boldsymbol{b \in B} \mid \exists \boldsymbol{a} \in \boldsymbol{A}\colon \boldsymbol{a \succcurlyeq b} \,\}|}{|\boldsymbol{B}|} \tag{6}$$

El valor $C(\boldsymbol{A}, \boldsymbol{B}) = 1$ significa que todos los vectores de decisión en $\boldsymbol{B}$ están débilmente dominados por $\boldsymbol{A}$. Lo opuesto, $C(\boldsymbol{A}, \boldsymbol{B}) = 0$ representa la situación donde ninguno de los puntos de $\boldsymbol{B}$ es débilmente dominado por $\boldsymbol{A}$. Observe que siempre se deben considerar ambas direcciones, ya que $C(\boldsymbol{A}, \boldsymbol{B})$ no es necesariamente igual a $1 - C(\boldsymbol{A}, \boldsymbol{B})$. En la Figura 32 se ve un ejemplo de cubrimiento de conjuntos.

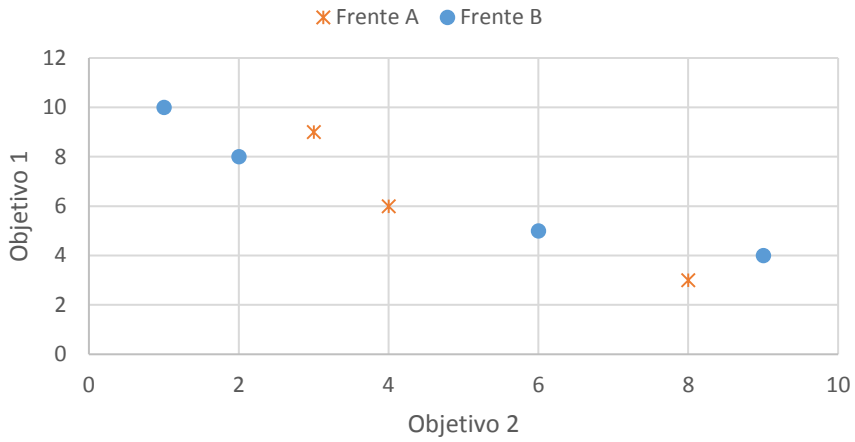


Figura 32: Cubrimiento de conjuntos de los conjuntos A y B para un problema de minimización. El valor de $\boldsymbol{C(A,B) = 0.25}$ y $\boldsymbol{C(B,A) = 0.33}$.

5.6 Metodología experimental

Para investigar las propiedades de nuestro problema hicimos dos tipos distintos de experimentos distintos. En primero de ellos se estudió del problema de bi-objetivo de minimización donde los objetivos son la minimización de las violaciones y consumo de energía. Se probaron distintos algoritmos metaheurísticos y se midió su tiempo de ejecución, híper volumen y cubrimiento de conjuntos para determinar cuál es la estrategia que mejor se ajusta a nuestro problema. En el segundo experimento se estudia la versión mono objetivo de nuestro problema, es decir, buscamos la solución que gaste la mínima cantidad de energía posible y no produzca violaciones al SLA. Se compararon la calidad de las soluciones obtenidas por varios algoritmos híbridos y heurísticos mediante la degradación y perfil de desempeño de su consumo

energético. Los algoritmos metaheurísticos e híbridos utilizados en nuestros experimentos comparten los parámetros de $evaluaciones = 20000$, $P_c = 0.9$ y $P_m = \frac{1}{n}$.

Con el objetivo de entender mejor el comportamiento de nuestro problema ambos experimentos se hicieron con dos infraestructuras distintas. En la primera de ellas la velocidad de las VMs es de 1 y la de los servidores es de 2. En la segunda, la velocidad de las VMs es de 1 y la de los servidores es de 8. Es decir, en el primero de ello tenemos 2 VMs por servidor, en el segundo 8 VMs por servidor. La cantidad de hosts en cada día es la suficiente para poder ejecutar las VMs sin violaciones al SLA, es decir de $m = \left\lceil \frac{n}{velocidad} \right\rceil$, donde n es la cantidad de VMs en un día de carga.

5.6.1 Experimento bi-objetivo: SLA y Energía

Con el objetivo de investigar cuales son los mejores operadores para este problema se probaron catorce algoritmos. Cada uno consiste en una combinación distinta de operadores de mutación y cruzamiento (En la Tabla 3 se muestran las configuraciones utilizadas). Cada algoritmo genera un frente por cada día de carga. Se normalizaron los frentes obtenidos (por el mínimo) de cada día por separado y se tomaron las soluciones no dominadas. De esta forma obtenemos un único frente por cada uno de los algoritmos. Posteriormente se comparó la calidad de los frentes de cada algoritmo mediante los indicadores de híper volumen y cubrimiento de conjuntos. Además, se hizo un análisis de degradación y perfil de desempeño del tiempo de ejecución de estos algoritmos.

Tabla 3: Distintas configuraciones utilizadas para el algoritmo.

Configuración (Abreviatura)	Algoritmo	Operadores	
		Cruzamiento	Mutación
MC-SP-BC	MOCell	Un Punto	Balanceo de concentración
MC-SP-BF	MOCell	Un Punto	Cambio de bit
MC-SP-SW	MOCell	Un Punto	Intercambio
MC-SP-VMSW	MOCell	Un Punto	Intercambio VM
MC-UN-BC	MOCell	Uniforme	Balanceo de concentración
MC-UN-BF	MOCell	Uniforme	Cambio de bit
MC-UN-SW	MOCell	Uniforme	Intercambio
MC-UN-VMSW	MOCell	Uniforme	Intercambio VM
MC-UN-LS	MOCell	Uniforme	Búsqueda Local SLA
MC-UNSLA-BC	MOCell	Uniforme SLA	Balanceo de concentración
MC-UNSLA-BF	MOCell	Uniforme SLA	Cambio de bit
MC-UNSLA-SW	MOCell	Uniforme SLA	Intercambio
MC-UNSLA-VMSW	MOCell	Uniforme SLA	Intercambio VM
NS-UN-BF	NSGA II	Uniforme	Cambio de bit

5.6.2 Experimento mono-objetivo: Energía

Para medir la efectividad de MOCell y NSGA-II para minimizar energía para las soluciones con 0 SLA, comparamos 4 configuraciones de algoritmos metaheurísticos híbridos y los comparamos contra las heurísticas clásicas RR, FFit y Rand. Las configuraciones de los algoritmos híbridos consideras fueron MO-UN, MO-SP, NS-UN y NS-SP. Estos algoritmos híbridos generan soluciones por un mecanismo de cruzamiento y posteriormente se le aplica búsqueda local para lograr alcanzar el valor SLA 0 (Por lo que no cuentan con mutación). Estos algoritmos convierten el problema bi-objetivo en mono objetivo de minimización de energía a través de su mecanismo de búsqueda. Además, proponemos una heurística llamada CFFit la cual trata de balancear la concentración y el tiempo de ejecución de las VMs. Para hacer la comparación tomamos los valores de consumo energético de las soluciones sin violaciones al SLA de los distintos algoritmos y medimos su degradación del desempeño y perfil de desempeño respecto a la cota inferior de nuestro problema.

Capítulo 6. Resultados

En este capítulo se muestran los resultados obtenidos por nuestros experimentos mono-objetivo de minimización de energía y bi-objetivo de minimización de energía y violaciones al SLA. Ambos experimentos se hicieron para dos infraestructuras distintas para así contar con una mejor idea del comportamiento del problema. En la primera la velocidad de los servidores es el doble de la de las VMs. En la segunda, la velocidad de los servidores es ocho veces la de las VMs. Ambos experimentos se hicieron una velocidad de servidores y VMs homogénea.

6.1 Resultados bi-objetivo: SLA y Energía

A continuación, se muestran los resultados del experimento bi-objetivo para cada una de las infraestructuras. Cada uno de los casos estudiados cuenta con un análisis de híper volumen, cubrimiento de conjuntos y tiempo de ejecución. Para el híper volumen se consideró la degradación respecto a la cota inferior teórica de nuestro problema (absoluta) y la degradación entre las estrategias sin la cota inferior (degradación relativa). Para poder seleccionar una única estrategia, utilizamos el promedio de las métricas de híper volumen y cubrimiento de conjuntos.

6.1.1 2 VMs por Servidor

Las Figura 33 y Figura 34 se muestran los frentes no dominados obtenidos para 2 VMs por servidor de los catorce algoritmos. En esta gráfica podemos observar que los algoritmos MO-UN-SW, MO-UN-LS, MO-UN-BC, MO-UN-VMSW, MO-SP-SW, MO-SP-BC y MO-SP-VMSW se encuentran en la región superior derecha de la gráfica por lo que tienen los peores valores de violaciones a SLA (mayores a 400) y consumo energético (Mayor a $4000x10^3$). En la esquina inferior izquierda de la gráfica se encuentran los algoritmos MO-UNSLA-BF, MO-UNSLA-BC, MO-UNSLA-SW y MO-UNSLA-VMSW los cuales encuentran las mejores soluciones en SLA (entre 0 a 450 violaciones). En la parte inferior de la gráfica entre los valores de 200 y 500 violaciones al SLA, se encuentran los algoritmos MO-UN-BF, MO-SP-BF, NS-UN-BF. Estos algoritmos encuentran soluciones entre $3000x10^3$ y $5000x10^3$ de energía normalizada, convirtiéndolos en las mejores estrategias para minimizar energía.

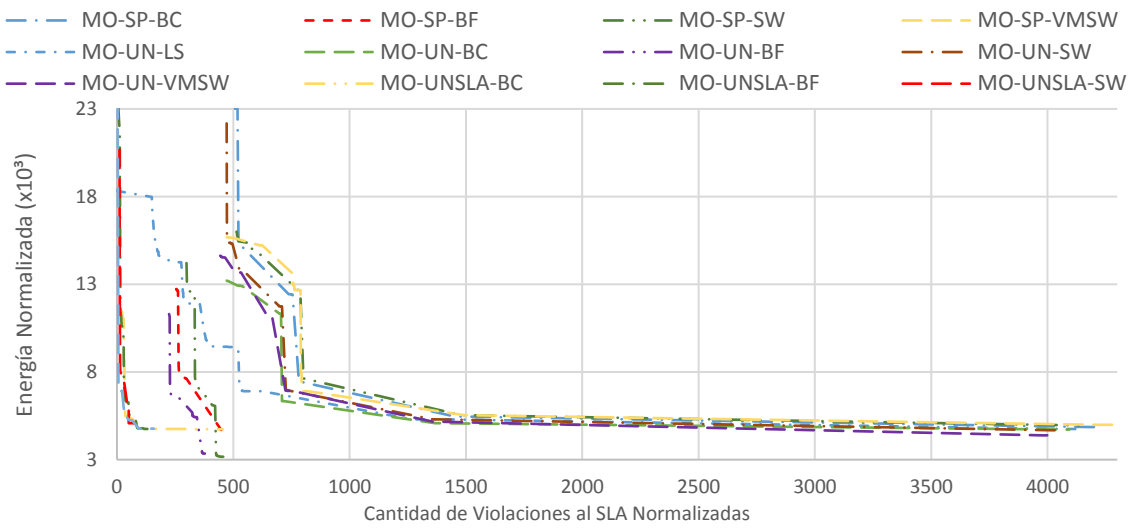


Figura 33: Frentes no dominados de las estrategias para 2 VMs por servidor (Escala grande). En esta gráfica se muestran los frentes dominados normalizados generados por las catorce estrategias.

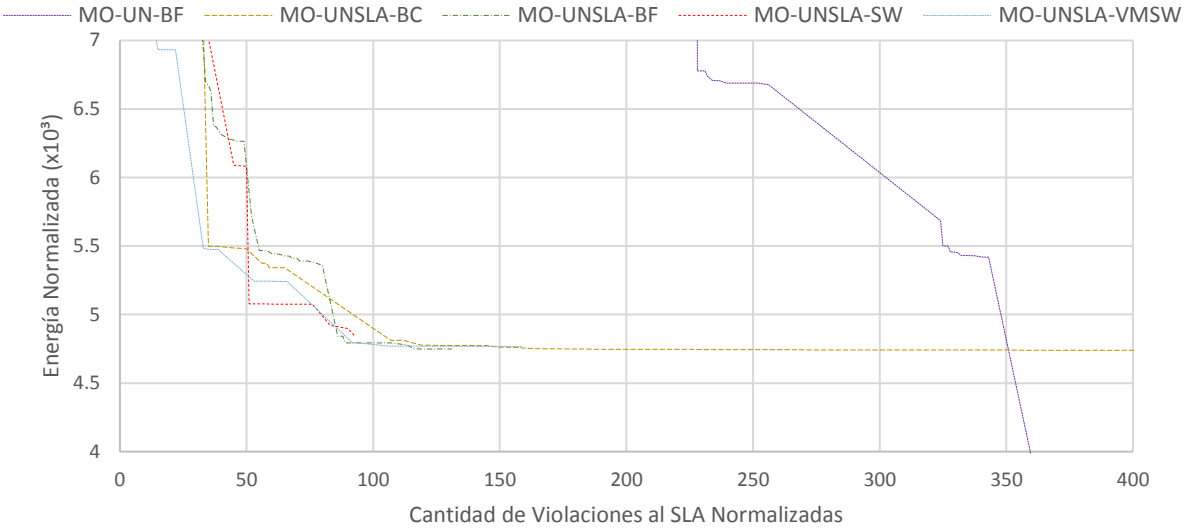


Figura 34: Frentes no dominados de las estrategias para 2 VMs por servidor (Escala fina). En esta gráfica se muestran los frentes dominados normalizados generados en una escala pequeña para así apreciar mejor sus características.

La Tabla 4 muestra los resultados de cubrimiento de conjuntos. De acuerdo a cubrimiento de conjuntos los peores algoritmos son MO-SP-BC, MO-SP-SW, MO-SP-VMSW y MO-UN-SW con un rango de dominancia y dominado superior a 11. Por otro lado, los algoritmos con mejor cubrimiento de conjuntos son MO-UNSLA-VMSW con un rango de dominancia 1 y dominado 1, MO-UNSLA-SW con 4 y 2, MO-UNSLA-BC con 3 y 3, y por último MO-UNSLA-BF con 2 y 4.

Tabla 4: Cubrimiento de conjuntos para 2 VMs por servidor. En esta tabla cada renglón se encuentra el conjunto $\boldsymbol{A}$ y en las columnas al conjunto $\boldsymbol{B}$. Cada celda contiene el valor de $\boldsymbol{C}(\boldsymbol{A},\boldsymbol{B})$, domina promedio se refiere al promedio de dominancia de la estrategia dada (un valor arte es mejor), dominado promedio se refiere a que tan dominada es la estrategia dada por otras (un valor pequeño es mejor).

	MO-SP-BC	MO-SP-BF	MO-SP-SW	MO-SP-VMSW	MO-UN-BC	MO-UN-BF	MO-UN-LS	MO-UN-SW	MO-UN-VMSW	MO-UNSLA-BC	MO-UNSLA-BF	MO-UNSLA-SW	MO-UNSLA-VMSW	NS-UN-BF	Domina Promedio	Rango
MO-SP-BC	1.00	0.00	0.44	0.74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.16	**12**
MO-SP-BF	1.00	1.00	1.00	1.00	1.00	0.00	0.89	1.00	0.83	0.00	0.00	0.00	0.00	0.27	0.57	**7**
MO-SP-SW	0.07	0.00	1.00	0.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.12	**13**
MO-SP-VMSW	0.22	0.00	0.11	1.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.10	**14**
MO-UN-BC	1.00	0.00	1.00	1.00	1.00	0.00	0.23	0.69	0.44	0.00	0.00	0.00	0.00	0.00	0.38	**10**
MO-UN-BF	1.00	1.00	1.00	1.00	1.00	1.00	0.91	1.00	1.00	0.13	0.00	0.00	0.00	0.73	0.70	**5**
MO-UN-LS	1.00	0.00	1.00	0.90	0.52	0.00	1.00	0.76	0.76	0.00	0.05	0.04	0.06	0.24	0.45	**8**
MO-UN-SW	0.96	0.00	1.00	0.86	0.28	0.00	0.25	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.31	**11**
MO-UN-VMSW	1.00	0.00	1.00	0.87	0.43	0.00	0.25	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.40	**9**
MO-UNSLA-BC	1.00	0.74	1.00	1.00	0.72	0.81	1.00	0.90	0.83	1.00	0.36	0.16	0.21	0.73	0.75	**3**
MO-UNSLA-BF	1.00	0.74	1.00	1.00	0.72	0.81	1.00	0.90	0.83	0.33	1.00	0.16	0.24	0.73	0.75	**2**
MO-UNSLA-SW	1.00	0.74	1.00	1.00	0.72	0.81	0.86	0.90	0.83	0.22	0.45	1.00	0.18	0.73	0.75	**4**
MO-UNSLA-VMSW	1.00	0.74	1.00	1.00	0.72	0.81	0.86	0.90	0.83	0.44	0.79	0.60	1.00	0.73	0.82	**1**
NS-UN-BF	1.00	0.26	1.00	1.00	1.00	0.00	0.76	1.00	1.00	0.03	0.00	0.00	0.00	1.00	0.58	**6**
Dominado Promedio	0.88	0.37	0.90	0.92	0.58	0.30	0.57	0.72	0.60	0.15	0.19	0.14	0.12	0.37		
Rango	**12**	**7**	**13**	**14**	**9**	**5**	**8**	**11**	**10**	**3**	**4**	**2**	**1**	**6**		

En la

Tabla 5 se muestra la degradación del desempeño del tiempo de ejecución de los catorce algoritmos. Podemos observar que excepto por los algoritmos con cruzamiento UNSLA la diferencia de degradación media y mediana entre ellos es muy pequeña (Menor que 1.41% y 1.15% respectivamente). Los algoritmos que utilizan el operador UNSLA son los más lentos, con una degradación media mayor al 200%. En la Figura 35 y Figura 36 se muestra el perfil de desempeño de los mejores (entre 0 y 0.08) y peores algoritmos (entre

1.75 y 2.3) respectivamente. De acuerdo con la Figura 35 los mejores algoritmos son MO-SP-SW con una probabilidad de 92% de tener una degradación de 0.022, segudio de MO-SP-VMSW con una probabilidad de 89% para el mismo valor de degradación.

Tabla 5: Media, mediana y desviación estándar de degradación de desempeño de tiempo de ejecución para 2 VMs por sevidor

Algoritmo	Tiempo de ejecución (%)		
	Media	Mediana	Desv. Est.
MO-SP-VMSW	0.92	0.77	0.94
MO-SP-SW	0.99	0.91	0.89
MO-SP-BF	1.29	1.09	1.12
MO-SP-BC	1.47	1.03	1.41
MO-UN-VMSW	1.69	1.56	1.20
MO-UN-BC	1.80	1.59	1.12
MO-UN-SW	2.02	1.62	1.68
MO-UN-BF	2.25	2.20	1.30
NS-UN-BF	2.33	2.34	0.98
MO-UN-LS	2.81	2.00	2.09
MO-UNSLA-BF	200.91	198.82	10.21
MO-UNSLA-SW	203.05	201.91	8.14
MO-UNSLA-BC	203.68	202.91	7.87
MO-UNSLA-VMSW	204.27	203.83	7.92

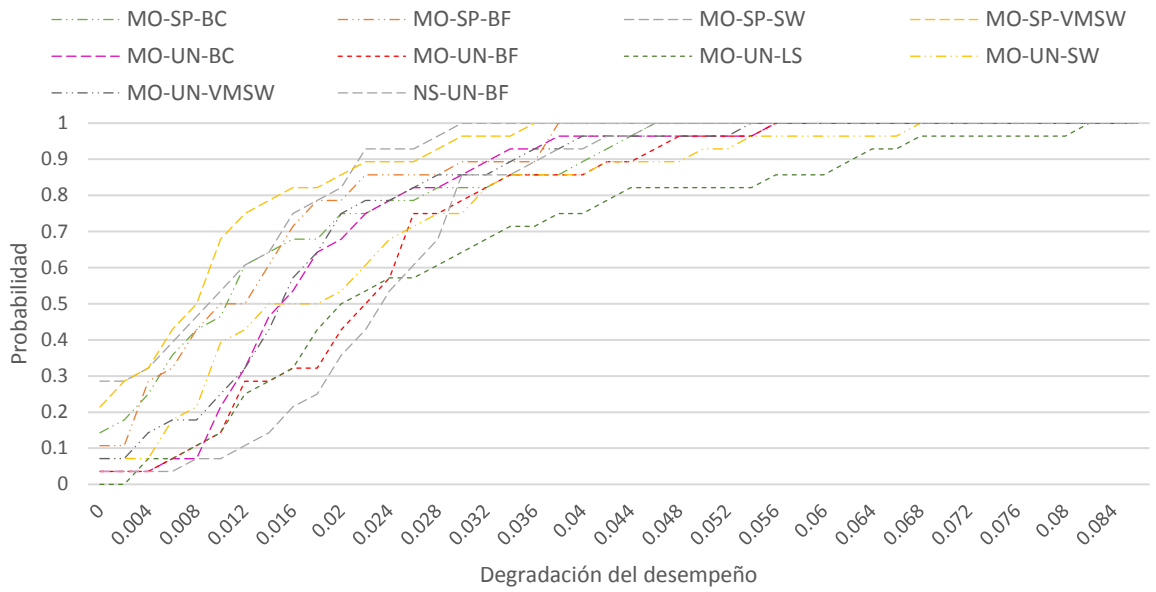


Figura 35: Perfil de desempeño de algoritmos con mejor tiempo de ejecución para 2 VMs por servidor

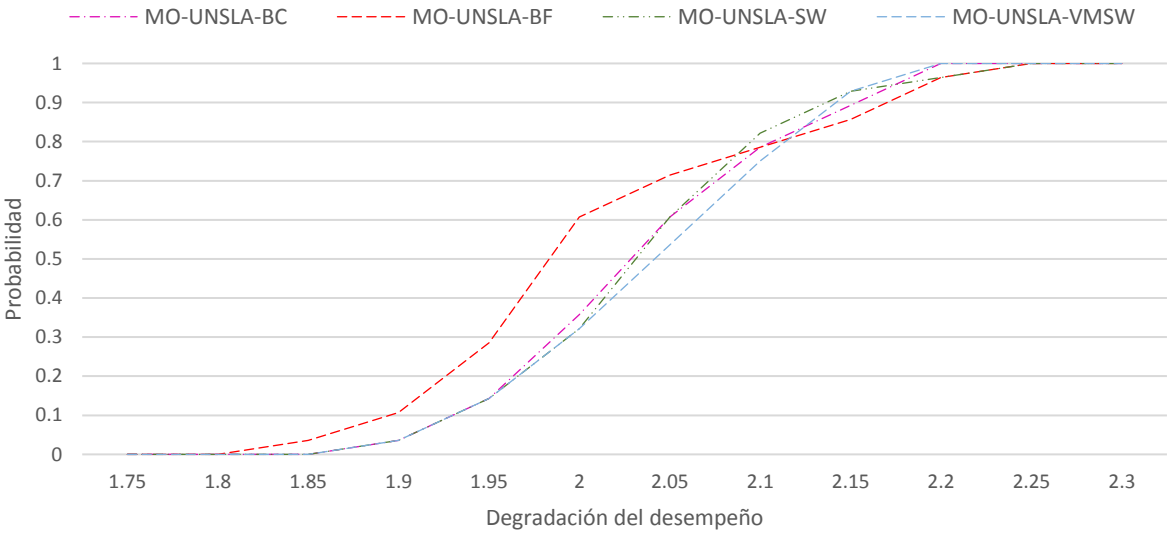


Figura 36: Perfil de desempeño de algoritmos con peor tiempo de ejecución para 2 VMs por servidor

Los valores de híper volumen junto a su rango de cubrimiento de conjuntos se encuentran en la Tabla 6. De acuerdo con esta métrica los mejores algoritmos son MO-UN-BF, MO-UNSLA-VMSW, MO-UNSLA-BF y MO-UNSLA-BC, estas difieren muy poco en su híper volumen porque tienen una degradación relativa menor a 1.2%. Los peores son MO-SP-SW, MO-SP-BC, MO-SP-VMSW y MO-UN-SW con una degradación relativa mayor al 20%.

Analizando estos indicadores en conjunto podemos observar que MO-UN-BF obtiene el mejores híper volumen y obtiene un quinto lugar respecto a cubrimiento de conjuntos. Este algoritmo obtiene un frente más amplio pero la mayoría de sus soluciones son dominadas por aquellos algoritmos que usan el operador de cruzamiento UNSLA, sin embargo, obtiene mejores valores para minimización de energía. MO-UN-BF obtiene mejores valores que NS-UN-BF en ambas métricas, en híper volumen lo supera por 1.5%, en cubrimiento de conjuntos lo supera con un rango 5 para dominancia y dominado contra un 6 de rango NS-UN-BF para ambos valores.

El algoritmo MO-UNSLA-VMSW es el mejor para minimización de las violaciones al SLA. Este obtiene el primer lugar en cubrimiento de conjuntos y tiene una degradación relativa de híper volumen de 0.64%. Sin embargo, los algoritmos con cruzamiento UNSLA cuentan con la desventaja de ser 200% más lentos que el resto de los algoritmos.

Tabla 6: Resultados métricas para 2 VMs por servidor. En esta tabla se muestra el híper volumen de cada una de las estrategias acompañada de su degradación del desempeño y el rango del cubrimiento de conjuntos que domina (estrategia que domina más en promedio a las otras) y por la que es dominado dominado (estrategia que es menos dominada en promedio).

Algoritmo	Híper Volumen			Rango Cubrimiento Conjuntos
	Volumen ($x\ 10^7$)	Degradación Absoluta (%)	Degradación relativa (%)	
Cota Inferior	11.5	0.00	-	-
MO-UN-BF	9.49	20.79	0.00	5
MO-UNSLA-VMSW	9.43	21.56	0.64	1
MO-UNSLA-BF	9.418	21.74	0.78	2
MO-UNSLA-BC	9.417	21.75	0.79	3
MO-UNSLA-SW	9.39	22.14	1.12	4
NS-UN-BF	9.36	22.44	1.37	6
MO-SP-BF	8.96	27.95	5.93	7
MO-UN-LS	8.68	32.11	9.38	8
MO-UN-BC	8.03	42.71	18.15	10
MO-UN-VMSW	8.01	43.22	18.57	9
MO-UN-SW	7.91	44.96	20.01	11
MO-SP-VMSW	7.72	48.49	22.94	14
MO-SP-BC	7.68	49.25	23.56	12
MO-SP-SW	7.63	50.24	24.39	13

6.1.2 8 VMs por Servidor

Las Figura 37 y Figura 38 se muestran los frentes no dominados obtenidos para 8 VMs por servidor de los catorce algoritmos. Los algoritmos MO-SP-BC, MO-SP-SW, MO-SP-SW, MO-SP-VMSW, MO-UN-BC, MO-UN-SW, MO-UN-VMSW y MO-UN-LS se encuentran en la región superior derecha de la gráfica, por lo que obtienen los peores valores de SLA (mayores a 300 violaciones) y energía (mayores a 13,000 de energía normalizada). Por otro lado, los algoritmos son MO-UN-BF, MO-SP-BF, NS-UN-BF, obtienen soluciones con una energía normalizada menor a 10,000. Estas son las mejores estrategias para minimizar energía. MO-UNSLA-BC, MO-UNSLA-BF, MO-UNSLA-SW y MO-UNSLA-VMSW obtienen los mejores valores de SLA, entre 0 y 120 violaciones.

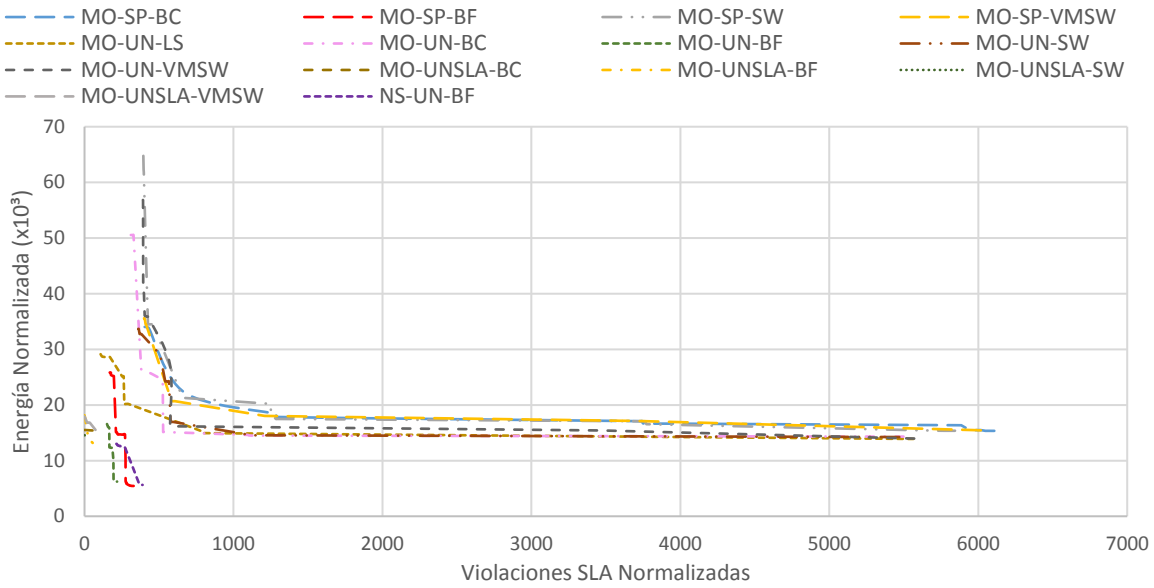


Figura 37: Frentes no dominados de las estrategias para 8 VMs por servidor (Escala grande). En esta gráfica se muestran los frentes dominados normalizados generados por las catorce estrategias.

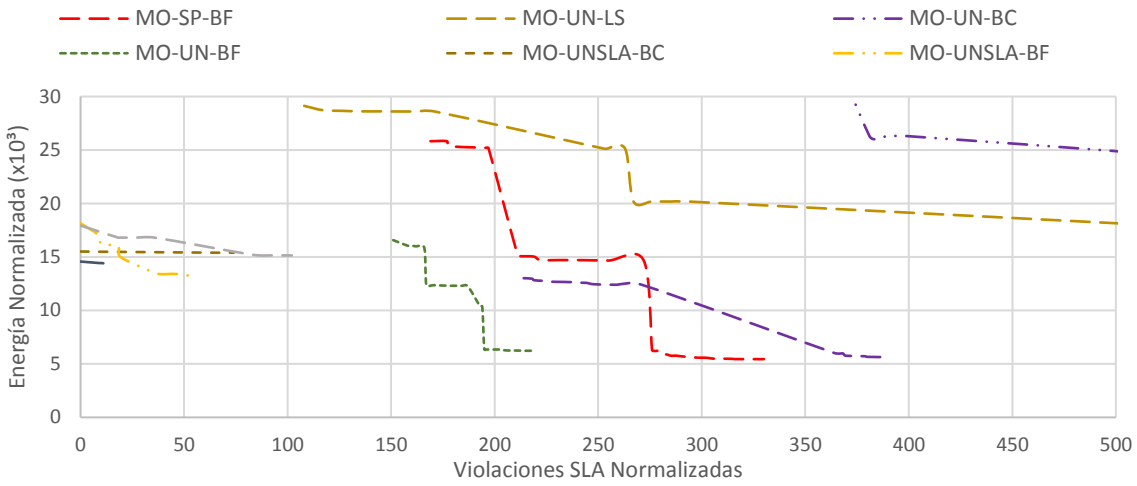


Figura 38: Frentes no dominados de las estrategias para 8 VMs por servidor (Escala fina). En esta gráfica se muestran los frentes dominados normalizados generados en una escala pequeña para así apreciar mejor sus características.

En la Tabla 7 se muestran los resultados de los algoritmos de acuerdo a cubrimiento de conjuntos. De acuerdo a esta métrica los peores algoritmos son MO-SP-BC, MO-SP-SW, MO-SP-VMSW y MO-UN-SW con un rango de dominancia superior a 11. El mejor algoritmo es MO-UNSLA-SW con el primer lugar en dominado y dominancia.

Tabla 7: Cubrimiento de conjuntos para 8 VMs por servidor. En esta tabla cada renglón se encuentra el conjunto $\boldsymbol{A}$ y en las columnas al conjunto $\boldsymbol{B}$. Cada celda contiene el valor de $\boldsymbol{C}(\boldsymbol{A},\boldsymbol{B})$, domina promedio se refiere al promedio de dominancia de la estrategia dada (un valor arte es mejor), dominado promedio se refiere a que tan dominada es la estrategia dada por otras (un valor pequeño es mejor).

	MO-SP-BC	MO-SP-BF	MO-SP-SW	MO-SP-VMSW	MO-UN-BC	MO-UN-BF	MO-UN-LS	MO-UN-SW	MO-UN-VMSW	MO-UNSLA-BC	MO-UNSLA-BF	MO-UNSLA-SW	MO-UNSLA-VMSW	NS-UN-BF	Domina Promedio	Rango
MO-SP-BC	1.00	0.00	0.15	0.18	0.00	0.00	0.00	0.00	0.16	0.00	0.00	0.00	0.00	0.00	0.11	**14**
MO-SP-BF	1.00	1.00	1.00	1.00	1.00	0.00	0.71	1.00	1.00	0.00	0.00	0.00	0.00	0.38	0.58	**4**
MO-SP-SW	0.49	0.00	1.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.12	**12**
MO-SP-VMSW	0.14	0.00	0.31	1.00	0.00	0.00	0.00	0.00	0.16	0.00	0.00	0.00	0.00	0.00	0.12	**13**
MO-UN-BC	1.00	0.00	1.00	1.00	1.00	0.00	0.04	0.62	0.79	0.00	0.00	0.00	0.00	0.00	0.39	**9**
MO-UN-BF	1.00	0.37	1.00	1.00	1.00	1.00	0.79	1.00	1.00	0.00	0.00	0.00	0.00	0.62	0.63	**3**
MO-UN-LS	1.00	0.00	1.00	1.00	0.37	0.00	1.00	0.44	0.76	0.00	0.00	0.00	0.00	0.00	0.40	**8**
MO-UN-SW	1.00	0.00	1.00	1.00	0.20	0.00	0.00	1.00	0.55	0.00	0.00	0.00	0.00	0.00	0.34	**10**
MO-UN-VMSW	0.89	0.00	0.88	0.73	0.00	0.00	0.00	0.13	1.00	0.00	0.00	0.00	0.00	0.00	0.26	**11**

MO-UNSLA-BC	0.77	0.15	0.65	1.00	0.37	0.21	0.63	0.56	0.79	1.00	0.40	0.00	0.50	0.00	0.50	6
MO-UNSLA-BF	1.00	0.34	1.00	1.00	1.00	0.21	1.00	1.00	1.00	0.50	1.00	0.00	0.90	0.00	0.71	2
MO-UNSLA-SW	1.00	0.34	1.00	1.00	0.80	0.21	0.75	0.78	0.79	1.00	0.50	1.00	1.00	0.00	0.73	1
MO-UNSLA-VMSW	1.00	0.17	1.00	1.00	0.37	0.21	0.67	0.56	0.79	0.00	0.10	0.00	1.00	0.00	0.49	7
NS-UN-BF	1.00	0.15	1.00	1.00	1.00	0.00	0.67	1.00	1.00	0.00	0.00	0.00	0.00	1.00	0.56	5
Dominado Promedio	0.88	0.18	0.86	0.87	0.51	0.13	0.45	0.58	0.70	0.18	0.14	0.07	0.24	0.14		
Rango	14	6	12	13	9	2	8	10	11	5	4	1	7	3		

En la Tabla 8 se encuentra la degradación del desempeño del tiempo de ejecución. El mejor algoritmo respecto al tiempo es MO-UN-VMSW con degradación media del tiempo de ejecución de 0%, seguido por MO-UN-BF 12.59%, seguido MO-SP-BF con 12.66% y por último MO-SP-BC con 14.16%. Los peores en tiempo de ejecución utilizan el operador de cruzamiento UNSLA y tienen una degradación media mayor al 210%. En las figuras Figura 39 y Figura **40** se encuentran las gráficas de perfil de desempeño de tiempo de ejecución para los mejores algoritmos (entre 0 y 0.3) y los peores (entre 1.4 y 3). De acuerdo con la Figura Figura 39 el mejor algoritmo es MO-SP-BF con una probabilidad de 90% de tener una degradación menor de 0.165, seguido de MO-UN-BF con un 85% de tener el mismo valor de degradación.

Tabla 8: Media, mediana y desviación estándar de degradación de desempeño de tiempo de ejecución para 8 VMs por sevidor

Algoritmo	Tiempo de ejecución (%)		
	Media	Mediana	Desv. Est.
MO-UN-VMSW	0.00	0.00	0.00
MO-UN-BF	12.59	13.17	3.70
MO-SP-BF	12.66	13.44	3.18
MO-SP-BC	14.16	13.70	2.95
MO-SP-VMSW	14.46	14.68	2.84
NS-UN-BF	15.04	14.94	3.04
MO-UN-LS	15.39	15.37	3.75
MO-UN-BC	15.44	15.54	3.63
MO-UN-SW	15.61	15.96	3.21
MO-SP-SW	17.29	17.07	3.00
MO-UNSLA-BC	218.95	215.07	15.63
MO-UNSLA-BF	219.58	219.20	19.56
MO-UNSLA-VMSW	223.68	222.74	12.53

MO-UNSLA-SW	223.78	223.01	11.14

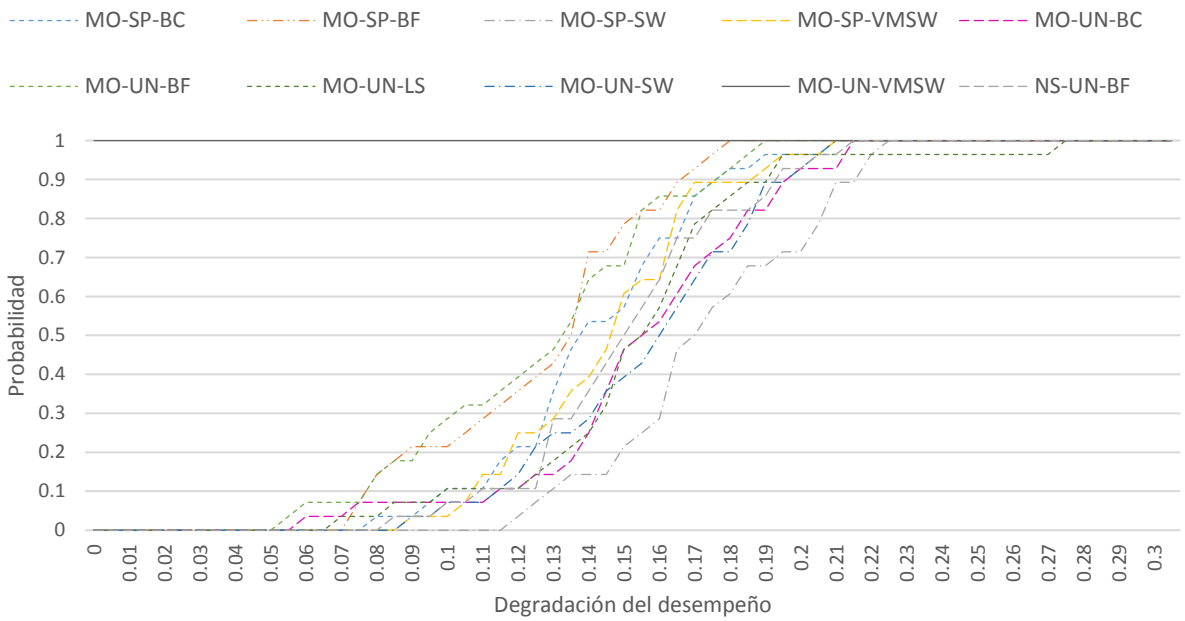


Figura 39: Perfil de desempeño de algoritmos con mejor tiempo de ejecución para 8 VMs por servidor

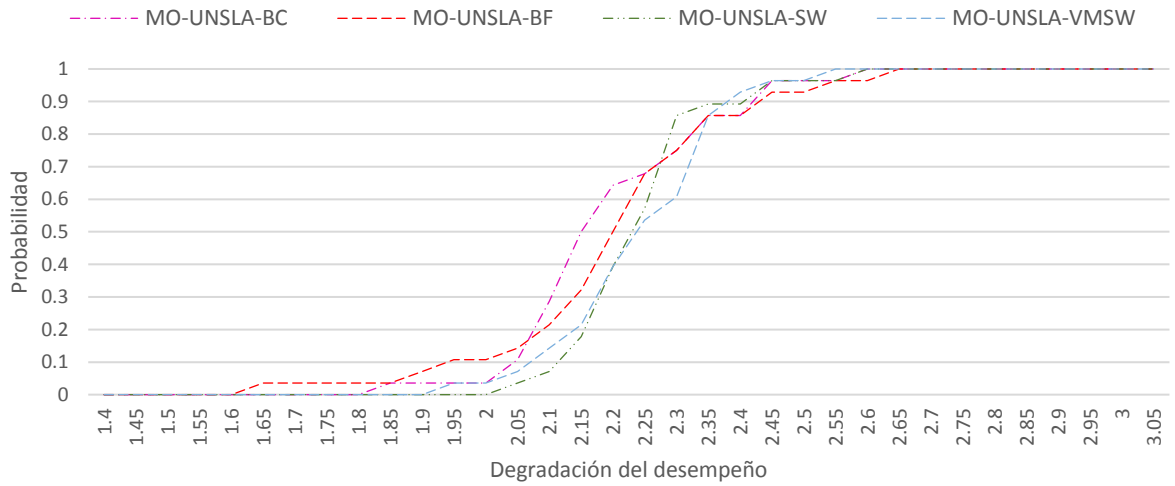


Figura 40: Perfil de desempeño de algoritmos con peor tiempo de ejecución para 8 VMs por servidor

Los valores de híper junto a su rango de cubrimiento de conjuntos se encuentran en la Tabla 9. Los algoritmos con mayor híper volumen son MO-SP-BF, MO-UN-BF, NS-UN-BF con una diferencia de la degradación relativa de híper volumen entre ellos menor al 1%. Los peores son MO-UN-SW, MO-SP-SW, MO-SP-VMSW y MO-SP-BC con una degradación relativa mayor al 26%.

Tabla 9: Resultados métricas 8 VMs por Servidor. En esta tabla se muestra el híper volumen de cada una de las estrategias acompañada de su degradación del desempeño y el rango del cubrimiento de conjuntos que domina (estrategia que domina más en promedio a las otras) y por la que es dominado (estrategia que es menos dominada en promedio).

Métrica	Híper Volumen			Cubrimiento Conjuntos
	Volumen $(\mathbf{x}\ 10^8)$	Degradación absoluta (%)	Degradación relativa (%)	
Cota Inferior	3.99	0	-	-
MO-SP-BF	3.54	12.64	0.00	4
MO-UN-BF	3.515	13.45	0.72	3
NS-UN-BF	3.506	13.73	0.96	5
MO-UNSLA-BF	3.18	25.48	11.40	2
MO-UNSLA-SW	3.10	28.52	14.10	1
MO-UNSLA-VMSW	3.06	30.50	15.86	7
MO-UNSLA-BC	3.04	31.11	16.39	6
MO-UN-LS	2.97	34.08	19.03	8
MO-UN-BC	2.90	37.68	22.23	9
MO-UN-SW	2.87	39.08	23.47	10
MO-UN-VMSW	2.80	42.63	26.62	11
MO-SP-SW	2.68	48.89	32.18	12
MO-SP-VMSW	2.67	49.26	32.50	13
MO-SP-BC	2.66	49.73	32.92	14

Analizando ambas métricas, MO-SP-BF, MO-UN-BF y NS-UN-BF obtienen los mejores resultados para minimización de energía. El algoritmo MO-SP-BF supera a MO-UN-BF en híper volumen por 0.7% de degradación relativa, esta es una diferencia poco significativa. MO-UN-BF tiene un rango de dominancia de 3 y un rango dominado 2, es mejor que MO-SP-BF el cual obtiene 4 y 6 en rango de dominancia y dominado respectivamente. MO-UN-BF obtiene mejores resultados que NS-UN-BF en ambas métricas, en híper volumen lo supera 0.96% de degradación relativa. En cubrimiento de conjuntos MO-UN-BF obtiene rango de dominancia y dominado 3 y 2 contra 5 y 6 de NSGA-II. MO-UN-BF es el mejor algoritmo para minimizar energía debido a que aunque no obtiene el mejor híper volumen, se distancia de él por muy poco. Además, si comparamos los comparamos con MO-SP-BF y NS-UN-BF, este obtiene los mejores valores de cubrimiento de conjuntos.

MO-UNSLA-BF y MO-UNSLA-SW son los mejores algoritmos para el objetivo de SLA. MO-UNSLA-BF sobrepasa a MO-UNSLA-SW en 2.7% de degradación relativa de híper volumen. En cubrimiento de conjuntos MO-UNSLA-BF pierde contra MO-UNSLA-SW, el primero tiene un rango dominancia de 2 y dominado de 4, el segundo es el mejor en ambos rangos. En otras palabras, MO-UNSLA-BF es mejor en híper volumen y MO-UNSLA-SW en cubrimiento de conjuntos. Ambos algoritmos cuentan con la desventaja de tener una degradación del desempeño del tiempo de ejecución mayor al 210%.

6.2 Resultados mono-objetivo: Energía

A continuación se muestran los resultados para la versión mono objetivo de cada una de nuestras infraestructuras. Los algoritmos se compararon mediante la degradación y perfil de desempeño de su consumo energético.

6.2.1 2 VMs por servidor

En la Figura 41 se muestra la degradación del desempeño de los 28 días de carga seguidos.

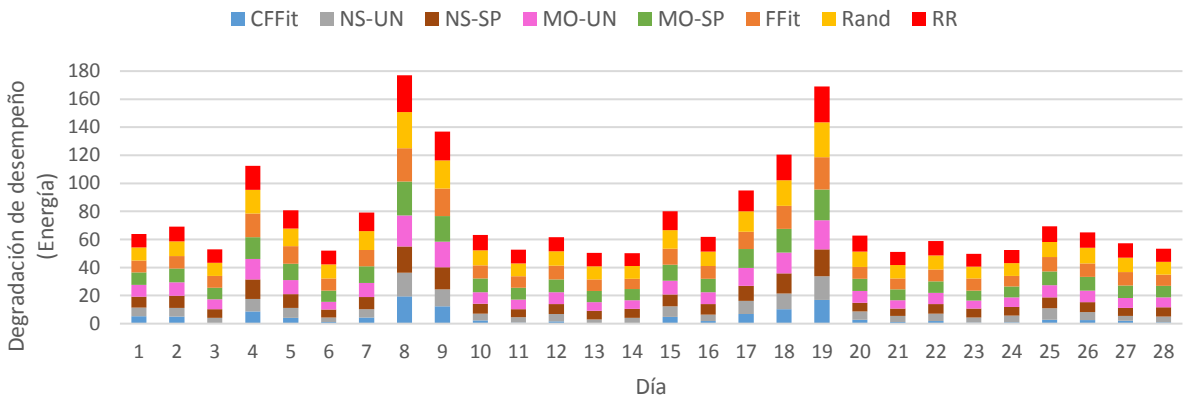


Figura 41: Degradación del desempeño de energía para 2VMs por servidor

La Tabla 10 la degradación media, mediana y desviación estándar de los valores de consumo energético obtenidos por cada algoritmo. De acuerdo con nuestros experimentos nuestra heurística CFFit es la mejor con una degradación de 441% de la cota inferior teórica. CFFit es mejor que los algoritmos híbridos por 202% de degradación media. Es interesante observar que los algoritmos con MOCell tiene peor rendimiento que NSGA-II, NS-UN supera a MO-UN con 319% de degradación media y NS-SP supera a MO-SP con 225%. Por otro lado, las peores estrategias son RR y Rand, con una degradación media mayor a 1200%.

Tabla 10: Degradación Media, Mediana y Desviación estándar de la energía para 2VMs por servidor.

Algoritmo	Degradación de energía		
	Media	Mediana	Desv. Est.
CFFit	4.418	2.466	4.946
NS-UN	6.600	5.489	3.732
NS-SP	8.751	7.075	3.987
MO-UN	9.797	8.422	4.422
MO-SP	11.042	9.328	4.425
FFit	11.144	9.345	4.551
Rand	12.401	10.424	4.673
RR	12.604	10.458	4.751

La Figura 42 muestra el perfil de desempeño de cada uno de los algoritmos. Si comparamos las mejores dos estrategias CFFit y NS-UN, la primera tiene la una probabilidad de 80% de tener una degradación del desempeño de 7. La segunda tiene con una probabilidad de 67% de tener el mismo valor de degradación. Esto muestra que CCFit es mejor que NS-UN.

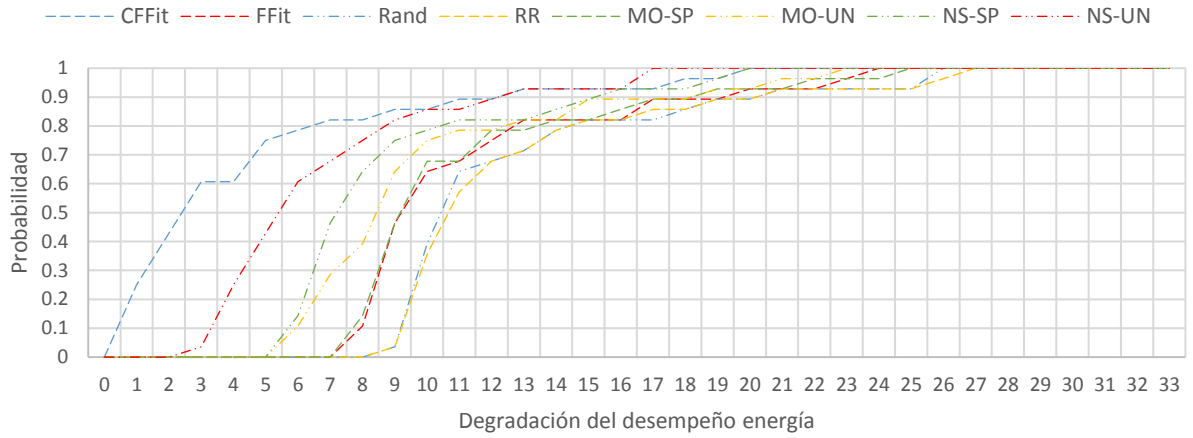


Figura 42: Perfil del desempeño de energía para 2VMs por servidor

6.2.2 8 VMs por servidor

En la Figura 43 se muestra la degradación del desempeño de los 28 días de carga seguidos.

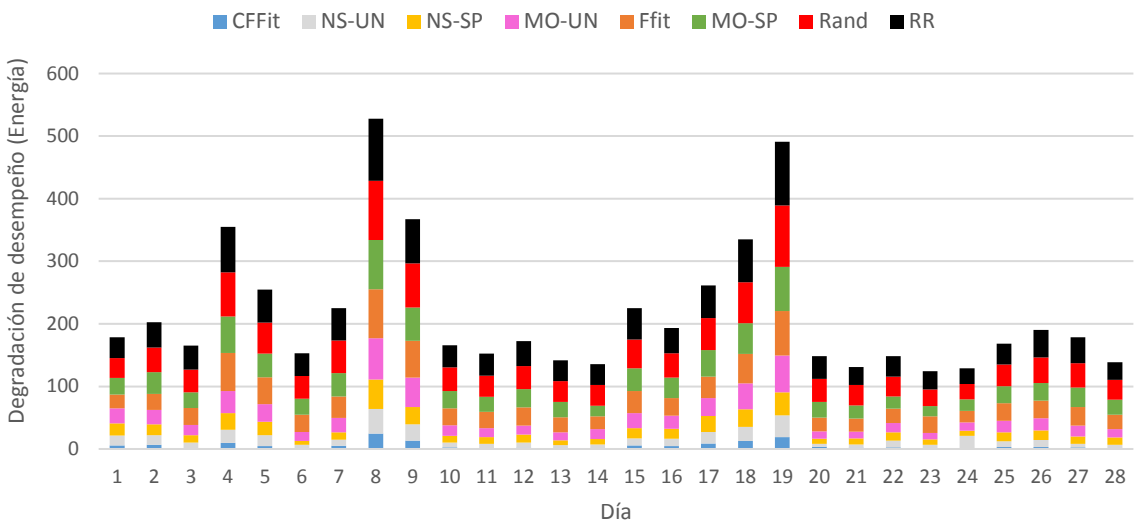


Figura 43: Degradación del desempeño de energía para 8 VMs por servidor

La Tabla 11 la degradación media, mediana y desviación estándar de los valores de consumo energético obtenidos por cada algoritmo. De acuerdo con nuestros experimentos nuestra heurística CFFit es la mejor con una degradación de 538% de la cota inferior teórica. CFFit supera a los algoritmos híbridos por 787% de degradación media. Es interesante observar que los algoritmos con MOCell tiene peor rendimiento que NSGA-II, NS-UN supera a MO-UN con más de un 1000% de degradación media y NS-SP sobrepasa a MO-UN con más de 1700%. Por otro lado, las peores estrategias son RR y Rand, con una degradación media mayor a 4400%.

Tabla 11: Degradación Media, Mediana y Desviación estándar de la energía para 8 VMs por servidor.

Algoritmo	Degradación de energía		
	Media	Mediana	Desv. Est.
CFFit	5.385	2.973	5.844
NS-UN	13.261	10.288	8.989
NS-SP	16.492	13.236	9.658
MO-UN	23.435	18.025	14.242
FFit	33.564	27.831	15.665
MO-SP	33.640	27.839	15.633
Rand	44.789	37.271	18.758
RR	45.850	39.363	19.938

La Figura 44 muestra el perfil de desempeño de cada uno de los algoritmos. Si comparamos las dos mejores estrategias CFFit y NS-UN, la primera tiene una probabilidad de 92% de tener una degradación del desempeño de 1600%, mientras NS-UN tiene una probabilidad de 71% para el mismo valor de degradación. Esto muestra que CCFit es mejor que NS-UN.

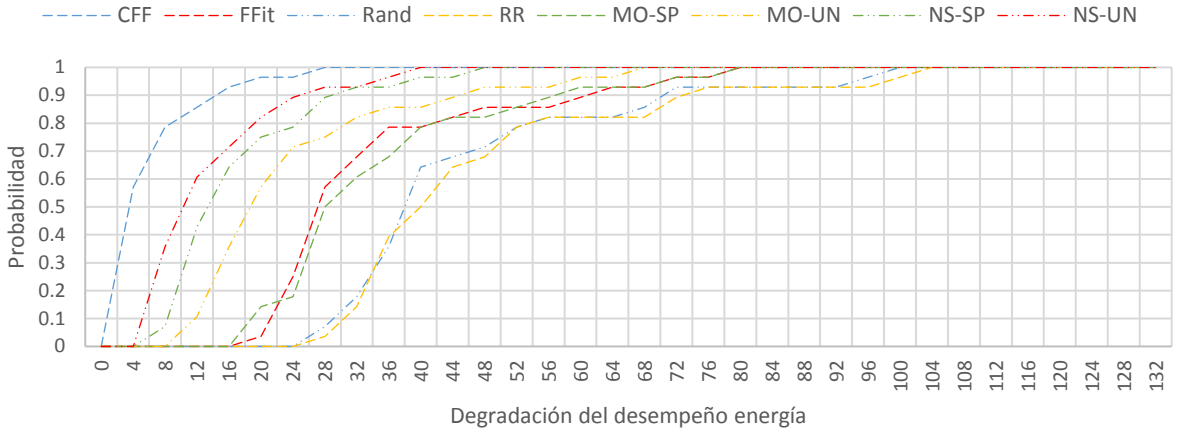


Figura 44: Perfil del desempeño de energía para 8 VMs por servidor

Capítulo 7. Discusión

Si quisiéramos aplicar este algoritmo en una nube real tendríamos muchas fuentes de incertidumbre. En un ambiente real el tipo de aplicación no es conocido e incluso puede variar respecto al tiempo. En un ambiente real es difícil conocer el tiempo de ejecución de las VMs tanto para el usuario como para el proveedor. La velocidad de las VMs es conocida y estática, sin embargo, en un ambiente real esta puede variar debido a la elasticidad de la nube. Por lo tanto, para llevar esta idea a un sistema real es necesario un método para la predicción del tiempo de ejecución, un sistema para la clasificación de VMs de acuerdo a su tipo de aplicación.

Nuestros algoritmos funcionan en un ambiente fuera de línea y todas las tareas están disponibles desde que inicia el sistema, sin embargo, en un ambiente real esto no pasa puesto que a lo largo del día distintos usuarios someten tareas a la nube. Para poder extender nuestro algoritmo a un ambiente en línea se requiere de ejecutar el algoritmo por lotes, es decir, asignar las VMs disponibles en ese momento, la dificultad en este caso radica en determinar cuándo se debe definir un lote y ejecutar el algoritmo. Incrementar o disminuir la cantidad de ejecuciones de un algoritmo de consolidación por lotes puede afectar el uso de recursos, consumo energético y/o el rendimiento del centro de datos. El ejecutarlo en periodos cortos hace que las asignaciones de VMs pasen más rápido, y como resultado, los servidores se prendan y apaguen más veces, disminuyendo el tiempo de vida del equipo de cómputo. Por otro lado, correr el algoritmo en largos periodos de tiempo también tiene efectos negativos en el rendimiento del sistema dado que este no reaccionaria lo suficientemente rápido a los cambios en la carga de trabajo y perder oportunidades de ahorro energía (Varasteh et al., 2015). Otra posible mejora a nuestro algoritmo es considerar la migración de VMs, es decir, considerar la posibilidad de mover una VM que ya inicio su ejecución de un servidor a otro abriendo la posibilidad a nuevas oportunidades de ahorro energético.

Nosotros estudiamos los casos donde todas las VMs y servidores tienen velocidad homogénea, así que los resultados de nuestro análisis pueden ser distintos en un ambiente con velocidades heterogéneas. En nuestro experimento comparamos catorce algoritmos utilizando la misma configuración, sin embargo, se desconoce cuáles son los mejores valores de probabilidad de cruce y mutación de cada uno de ellos.

Capítulo 8. Conclusiones

En este problema presentamos las conclusiones de esta tesis y trabajo futuro.

En años recientes la popularidad del cómputo en la nube ha ido en aumento. La nube requiere de una gran infraestructura, esto provoca un gran gasto energético, lo cual se ha convertido en un gran problema económico y ambiental. La consolidación es una técnica muy popular para el ahorro de energía en nubes de IaaS, pero cuenta con la desventaja de que, en algunos escenarios, puede generar contención de recursos. Una posible solución a la contención es considerar las necesidades de hardware de las aplicaciones ejecutada por las VMs. Este trabajo contribuye al desarrollo de estrategias algorítmicas para el ahorro energético mediante la exploración de las oportunidades de ahorro energético ofrecidas por la calendarización consciente del tipo aplicación de las VMs.

Existen pocos trabajos que abordan el problema de consolidación consciente del tipo de aplicación y, hasta donde sabemos, este es el primer trabajo que considera tanto el consumo energético de cada tipo de aplicación ejecutada por separado, como el efecto de la combinación de ejecutar una mezcla de estas en un servidor.

Estudiamos problema de consolidación determinista bi-objetivo para minimizar el consumo energético y las violaciones al SLA con el modelo de energía consciente del tipo aplicaciones ejecutadas por las VMs de (F. A. Armenta-Cano et al., n.d.) . Además, estudiamos la versión mono objetivo de nuestro problema con el objetivo de minimización de energía. En específico estudiamos el caso de máquinas homogéneas y VMs homogéneas. Para tener una mejor visión de las características del problema, estudiamos dos distintos tipos de infraestructura. En la primera la velocidad de los servidores es el doble de las VMs. En la segunda, la velocidad de los servidores es ocho veces mayor que la de las VMs.

Dado que el problema de la consolidación determinista es NP Difícil, no es posible encontrar la solución óptima al problema en un tiempo factible. Por lo anterior, optamos por las técnicas metaheurísticas para abordar este problema, en específico, algoritmos genéticos. Recientemente, el algoritmo conocido como MOCell ha tomado una gran popularidad, ya que este permite de un mejor muestreo del espacio de búsqueda mediante la ubicación de los individuos en un grid toroidal. Comparamos distintas combinaciones de operadores bajo una probabilidad de cruce, probabilidad de mutación, iteraciones y población fijas para así determinar cuál es la más adecuada para nuestro problema. Además, incluimos una configuración del algoritmo clásico NSGA-II para compararlo contra MOCell. Para el cruzamiento

consideramos los operadores clásicos de cruzamiento en un punto y uniforme, además de proponer uno llamado cruzamiento uniforme SLA. Para la mutación consideramos los operadores clásicos BF, SW y búsqueda local SLA, además de proponer los operadores balanceo de concentración y VMSW.

Para el caso bi-objetivo se probaron 13 combinaciones distintas operadores de MOCell y una de NSGA-II. Y se midió la calidad de los frentes obtenidos por los algoritmos mediante los indicadores de híper volumen y cubrimiento de conjuntos. Además, se analizó también el tiempo de ejecución de cada uno de ellos.

Los resultados para nuestro análisis bi-objetivo muestran que el mejor algoritmo para minimización de energía es MO-UN-BF, puesto a que se degrada muy poco del mejor híper volumen encontrado (menos del 1%) y además tiene el mejor cubrimiento de conjuntos entre los algoritmos para minimización de energía. Los mejores algoritmos respecto al SLA son MO-UNSLA-BC, MO-UNSLA-SW y MO-UNSLA-VMSW, sin embargo, se necesitan más experimentos para determinar cuál de estos es el mejor. Si bien los algoritmos con cruzamiento uniforme SLA obtienen muy buenos resultados y menores valores de SLA, cuentan con la desventaja de ser más de 200% más lentos que los demás.

Por último, los resultados muestran que nuestra configuración de MOCell supera a NSGA-II. MO-UN-BF es mejor que NS-UN-BF en híper volumen por menos de un 2%, además, además de siempre obtener los mejores valores en cubrimiento de conjuntos.

Para el caso mono objetivo, nuestra heurística CFFit obtiene mejores soluciones, en 2VMs por servidor es mejor por un 200% de degradación media, en 8 VMs por servidor por más de 700%. Es interesante notar que en el caso mono objetivo, nuestros algoritmos con NSGA-II superan a los MOCell con más de un 200% de degradación media de consumo energético.

Nuestro trabajo puede ser fácilmente extendido estudiando otros casos del problema como aquellos con cargas más de dos tipos de aplicación (Por ejemplo, red intensiva y disco duro intensiva), con infraestructura heterogénea y VMs con velocidad heterogénea. Además, es necesario hacer una sintonización de parámetros para cada uno de los catorce algoritmos aquí estudiados.

Lista de referencias

Armenta-Cano, F. A., Tchernykh, A, Miranda-Lopez, V. (n.d.). Energy Consumption Model for Scheduling Applications with Resource Contention (Under edition).

Armenta-Cano, F. A., Tchernykh, A., Cortes-Mendoza, J. M., Yahyapour, R., Drozdov, A. Y., Bouvry, P., … Nesmachnow, S. 2017. Min_c: Heterogeneous concentration policy for energy-aware scheduling of jobs with resource contention. *Programming and Computer Software*, *43*(3), 204–215. https://doi.org/10.1134/S0361768817030021

Armenta-Cano, F., Tchernykh, A., Cortés-Mendoza, J. M., Yahyapour, R., Drozdov, A. Y., Bouvry, P., … Avetisyan, A. 2015. Heterogeneous job consolidation for power aware scheduling with quality of service. *CEUR Workshop Proceedings*, *1482*(218), 687–697.

Cook, G. 2012. How clean is your cloud. *Catalysing an Energy Revolution*, (April), 52. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:How+Clean+is+Your+Cloud+?#0

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, *6*(2). Retrieved from https://www.iitk.ac.in/kangal/Deb_NSGA-II.pdf

Dolan, E. D., Moré, J. J. 2002. Benchmarking optimization software with performance profiles. *Mathematical Programming, Series B*, *91*(2), 201–213. https://doi.org/10.1007/s101070100263

Durillo, J. J., Nebro, A. J. 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, *42*(10), 760–771. https://doi.org/10.1016/j.advengsoft.2011.05.014

Durillo, J. J., Nebro, A. J., Alba, E. 2010. The jMetal Framework for Multi-Objective Optimization : Design and Architecture. *Evolutionary Computation*, *5467*, 18–23. https://doi.org/10.1109/CEC.2010.5586354

Eiben, A. E., Smith, J. E. 2008. *Introduction to Evolutionary Computing (Natural Computing Series)*. (G. Rozenberg, B. Th., K. J.N., and S. H.P., Eds.) (Second Edi). Springer. https://doi.org/10.1007/978-3-662-44974-8

Farahnakian, F., Ashraf, A., Pahikkala, T., Liljeberg, P., Plosila, J., Porres, I., Tenhunen, H. 2015. Using Ant Colony System to Consolidate VMs for Green Cloud Computing. *IEEE Transactions on Services Computing*, *8*(2), 187–198. https://doi.org/10.1109/TSC.2014.2382555

Feitelson, D. G., Tsafrir, D., Krakov, D. 2014. Experience with using the Parallel Workloads Archive. *Journal of Parallel and Distributed Computing*, 1–25. https://doi.org/10.1016/j.jpdc.2014.06.013

Fox, A., Turner, A., Kim, H. S. 2012. Resource contention-aware Virtual Machine management for enterprise applications. In *2012 IEEE Global Communications Conference (GLOBECOM)* (pp. 1641–1646). IEEE. https://doi.org/10.1109/GLOCOM.2012.6503349

Hähnel, M., Döbel, B., Völp, M., Härtig, H. 2012. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review*, *40*(3), 13. https://doi.org/10.1145/2425248.2425252

Hongyou, L., Jiangyong, W., Jian, P., Junfeng, W., Tang, L. 2013. Energy-aware scheduling scheme using workload-aware consolidation technique in cloud data centres. *China Communications*, *10*(12), 114–124. https://doi.org/10.1109/CC.2013.6723884

Iskren Ivov Chernev. (n.d.). Moment.js. Retrieved July 5, 2017, from https://momentjs.com/

Kliazovich, D., Bouvry, P., Khan, S. U. 2010. DENS: Data Center Energy-Efficient Network-Aware Scheduling. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing* (pp. 69–75). IEEE. https://doi.org/10.1109/GreenCom-CPSCom.2010.31

Knowles, J., Corne, D. 2002. On metrics for comparing nondominated sets. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)* (Vol. 1, pp. 711–716). IEEE. https://doi.org/10.1109/CEC.2002.1007013

Lieuallen, A., Boodman, A., Sundström, J. (n.d.). Greasemonkey download page. Retrieved July 4, 2017, from https://addons.mozilla.org/es/firefox/addon/greasemonkey/?src=cb-dl-mostpopular

Maziku, H., Shetty, S. 2014a. Network Aware VM Migration in Cloud Data Centers. *2014 Third GENI Research and Educational Experiment Workshop*, 25–28. https://doi.org/10.1109/GREE.2014.18

Maziku, H., Shetty, S. 2014b. Towards a network aware VM migration: Evaluating the cost of VM migration in cloud data centers. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)* (pp. 114–119). IEEE. https://doi.org/10.1109/CloudNet.2014.6968978

Nath, A. R. and A. K. and S. G. and J. L. and S. 2013. PACMan: Performance Aware Virtual Machine Consolidation. *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013*, 83–94. Retrieved from https://www.usenix.org/conference/icac13/technical-sessions/presentation/roytman

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., Alba, E. 2006. A Cellular Genetic Algorithm for Multiobjective Optimization. *Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO 2006)*, 25–36.

Priya, B., Pilli, E. S., Joshi, R. C. 2013. A survey on energy and power consumption models for Greener Cloud. In *2013 3rd IEEE International Advance Computing Conference (IACC)* (pp. 76–82). IEEE. https://doi.org/10.1109/IAdCC.2013.6514198

Spaanjaars, I. 2010. *Beginning ASP. NET 4: in C# and VB*. Wiley Publishing. Retrieved from http://books.google.com/books?hl=en&lr=&id=lCfuPZS4Fl8C&oi=fnd&pg=PT9&dq=Beginning+ASP.NET+in+C+%23+and+VB&ots=JlUW6Zgc1R&sig=d0h3R83dLbrCRfnHdkIPc0BkNaw

Takouna, I., Alzaghoul, E., Meinel, C. 2014. Robust virtual machine consolidation for efficient energy and performance in virtualized data centers. *Proceedings - 2014 IEEE International Conference on Internet of Things, iThings 2014, 2014 IEEE International Conference on Green Computing and Communications, GreenCom 2014 and 2014 IEEE International Conference on Cyber-Physical-Social Computing, CPS 20*, (iThings), 470–477. https://doi.org/10.1109/iThings.2014.84

Tang, G., Jiang, W., Xu, Z., Liu, F., Wu, K. 2015. Zero-Cost, Fine-Grained Power Monitoring of Datacenters Using Non-Intrusive Power Disaggregation. In *Proceedings of the 16th Annual Middleware Conference on - Middleware '15* (pp. 271–282). New York, New York, USA: ACM Press. https://doi.org/10.1145/2814576.2814728

Treibig, J., Hager, G., Wellein, G. 2010. LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. *Proceedings of the International Conference on Parallel Processing Workshops*, 207–216. https://doi.org/10.1109/ICPPW.2010.38

Varasteh, A., Goudarzi, M. 2015. Server Consolidation Techniques in Virtualized Data Centers: A Survey. *IEEE Systems Journal*, 1–12. https://doi.org/10.1109/JSYST.2015.2458273

Verboven, S., Vanmechelen, K., Broeckhove, J. 2015. Network Aware Scheduling for Virtual Machine Workloads with Interference Models. *IEEE Transactions on Services Computing*, *8*(4), 617–629. https://doi.org/10.1109/TSC.2014.2312912

VMR/NPS Series User's Guide. (n.d.). Retrieved July 3, 2017, from https://www.wti.com/guides/vmr+nps_usersguide.pdf

Wang, J. V., Cheng, C.-T., Tse, C. K. 2015. A power and thermal-aware virtual machine allocation mechanism for Cloud data centers. In *2015 IEEE International Conference on Communication Workshop (ICCW)* (pp. 2850–2855). IEEE. https://doi.org/10.1109/ICCW.2015.7247611

Wilton, P., Mcpeak, J. 2010. *Beginning JavaScript* (4th Editio). Wiley Publishing.

Wu, Q., Ishikawa, F. 2015. Heterogeneous Virtual Machine Consolidation Using an Improved Grouping Genetic Algorithm. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems* (pp. 397–404). IEEE. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.92

Yang, J. S., Liu, P., Wu, J. J. 2012. Workload characteristics-aware virtual machine consolidation algorithms. *CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, 42–49. https://doi.org/10.1109/CloudCom.2012.6427540

Zitzler, E., Thiele, L. 1998. Multiobjective optimization using evolutionary algorithms - A comparative case study. *Parallel Problem Solving from Nature - PPSN V*, 292–301. https://doi.org/10.1007/BFb0056872

Anexos

1 PDU Script

Para la creación del modelo energético de la Sección 2.2 se hicieron una serie de mediciones empíricas con un PDU VMR-8HD20-1 y se creó un modelo energético en base a estas. La interfaz web del PDU sólo cuenta con información por día, semana, mes y año, sin embargo, para poder crear un modelo lo más preciso posible se requerían de mediciones por segundo o intervalos menores. Esto creo la necesidad de crear un código javascript para acceder a esta información en una ventana de tiempo más pequeña. A continuación se describen las características del PDU, los conceptos básicos de funcionamiento de las páginas web y el script creado.

1.1 Descripción del dispositivo

La unidad de distribución de energía para la creación del modelo energético fue la VMR-8HD20-1. Esta provee funciones para monitorear el consumo de energía del equipo de cómputo. Pertenece a la serie VMR de la compañía WTI, este es un controlador para administración de energía. Este permite la administración remota y segura del consumo de eléctrico de un rack a través de SSL, SSH, SNMP, navegador web, telnet, modem externo o una terminal loca. Entre sus funciones cuenta con un monitoreo de notificar de notificar al administrador de la red o personal de mantenimiento si el consumo energético sobrepasa un valor umbral. Además lleva un registro del consumo energético en un archivo bitácora, que puede ser leído en ASCII, XML, CSV o en formato gráfico. Para más información acerca del dispositivo ver (“VMR/NPS Series User’s Guide,” n.d.).

Los dispositivos VMR ofrecen dos interfaces de configuración: la interfaz de navegador web y la interfaz de texto. La interfaz de navegador web está disponible sólo a través de la red, y la interfaz de texto a través de la red (SSH o Telnet), modem o PC local.

1.2 Funcionamiento de páginas web

Para poder entender el funcionamiento del script del PDU primero es necesario tener una idea general sobre el funcionamiento de las páginas web.

Cuando un usuario escribe una dirección web cualquiera en un navegador, este manda una petición a un servidor web en esa dirección. Esto se hace a través del protocolo HTTP, el Protocolo de transferencia de híper texto. Cuando un servidor está activo y la petición es válida, el servidor acepta la petición, la procesa, y entonces manda una respuesta al navegador del cliente. Este proceso se puede visualizar en la Figura 45. Un servidor huésped aloja el sitio web y este puede ser accedido por distintos clientes (Spaanjaars, 2010).

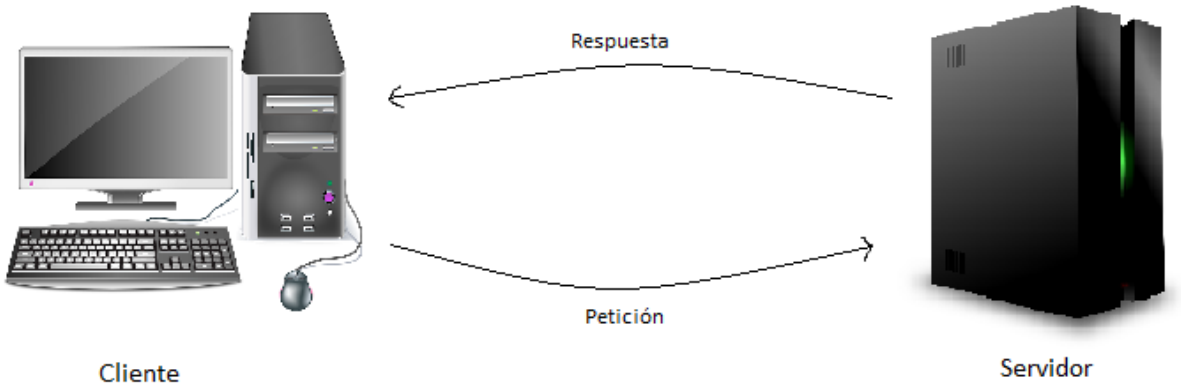


Figura 45: Sistema Cliente servidor

Los componentes principales de una página web son:

- *Código HTML:* Es el lenguaje estándar para crear páginas web y todos los navegadores actuales tienen la capacidad de interpretarlo. Los documentos HTML son archivos de texto que contienen marcas, texto, y datos adicionales que modifican el mismo. El código HTML usa texto rodeado por brackets para indicarle al navegador cómo mostrar el contenido (Spaanjaars, 2010).

- *Código CSS:* Sus siglas en inglés significan Cascade Style Sheet. Este es un lenguaje diseñado para darle formato a las páginas web. Por ejemplo: colores de fondo, bordes de los elementos HTML, propiedades de la fuente (tamaño, color, familia, etc.) (Spaanjaars, 2010).

- *Código Javascript:* Este es un lenguaje de programación que es interpretado por el navegador del cliente. Este permite mejorar la experiencia de usuario modificando el código HTML de manera dinámica en respuesta a sus acciones en la página web. Por ejemplo: cambiando el tamaño de los botones, desactivando/activando menús, etc. Jquery es una librería de javascript que hace más sencillo el desarrollo de páginas web. Para más información sobre javascript y jquery ver (Wilton and Mcpeak, 2010).

Para archivos estáticos como archivos HTML, CSS, javascript o imágenes, el servidor huésped sólo lee el archivo del disco local y lo manda al navegador del cliente. Por otro lado, existen archivos dinámicos como por ejemplo las páginas web ASPX (Tecnología producida por Microsoft). El servidor web entrega el archivo dinámico y este es procesado por un software que entonces genera el código HTML correspondiente y la manda al navegador del cliente (Spaanjaars, 2010).

1.3 Descripción del script

Este script consiste en un código javascript inyectado al navegador Firefox para así poder obtener la funcionalidad deseada de manera local en el navegador. Para inyectar el código se hizo uso del complemento conocido como Greasemonkey v3.11 (Lieuallen et al., n.d.). Este permite inyectar un archivo javascript creado por el usuario e inyectarlo en el navegador para así personalizar el funcionamiento de una determinada página web en el navegador del cliente.

Al ingresar la dirección IP del PDU podemos ingresar a la interfaz web del mismo. El menú “Metering/Power Metering/ Power History” en la interfaz web con versión de firmware 1.64 (Figura 46) no cuenta con alguna herramienta para obtener los datos de consumo por segundo, sin embargo, estos si son guardados por el PDU y son mostrados en la gráfica. Esto motivó la creación de un código javascript para poder extraer la información de consumo por segundo.

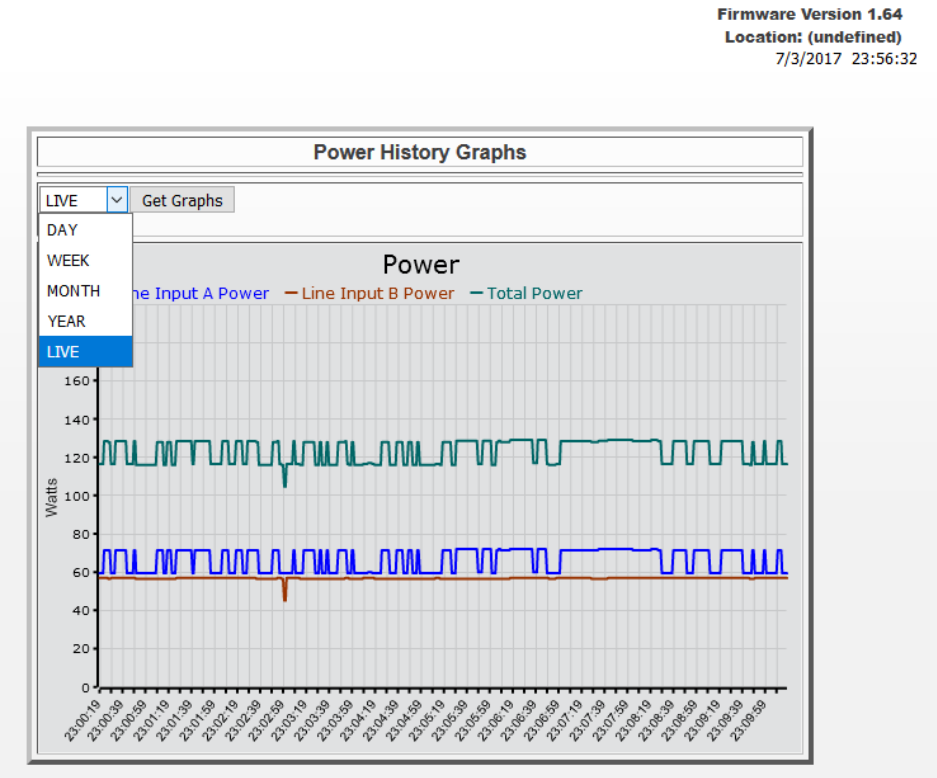


Figura 46: Captura de pantalla de interfaz Web

Al observar el código HTML de la gráfica “Power History Graphs”, descubrimos que esta cuenta con un parámetro “flashvars="data-file=/flash/power_history_local_xxxxx.json”. Este parámetro apunta a un archivo JSON el cual es la fuente de los datos mostrados en la gráfica, este formato es un estándar muy popular para el intercambio de datos en internet. Los valores “xxxxx” son un número que diferencia los archivos en dentro del PDU. El PDU hace mediciones por segundo, sólo que estos datos no son accesibles a través de la interfaz web. Por esto, se creó un script que accede al archivo JSON fuente de la gráfica y extrae las mediciones de la Línea A, Línea B y Línea Total ordenados por hora, minuto y segundo.

A continuación se explica el funcionamiento del script. De la línea 1 a 10 se encuentra la cabecera del código script (Figura 47), estos datos son utilizados por Greasemonkey para configurar algunos parámetros de este. Entre los más importantes está el nombre y descripción del script en la línea 2 y 4 respectivamente. En las líneas 5 se declara el URL dónde se inyectará el script, este corresponde a la dirección donde se aloja la interfaz web del PDU. Las líneas 7 y 8 indican que se están utilizando las librerías jquery y moment, moment (Iskren Ivov Chernev, n.d.) es una librería que facilita el tratamiento de datos fecha/hora en javascript.

```
1   // ==UserScript==
2   // @name         PDU Data
3   // @namespace    lgalaviz
4   // @description Extrae datos de PDU
5   // @include      http://158.97.9.87/cgi-bin/gethtml?/formWTIDisplayPHGraphs.html
6   // @version      1
7   // @grant        none
8   // @require https://code.jquery.com/jquery-3.2.1.min.js
9   // @require https://momentjs.com/downloads/moment-with-locales.js
10  // ==/UserScript==
```

Figura 47: Fragmento de Script – Cabecera

De la línea 68 a 76 se encuentra la función ready (Figura 48). Esta función forma parte de jquery, esta se activa una vez que todos los elementos del código HTML fueron cargados correctamente. Una vez se ejecuta la función ready, en la línea 70 se agrega un botón extra al código HTML con la leyenda “Download Data by Second”. En las líneas 72 a 74 se vincula el evento click del botón anterior con la función get Data.

En la Figura 49 se encuentra la función getData, la cual se encarga de leer los datos del PDU. En la línea 31 y 32 se obtiene el nombre del archivo donde se guardan las mediciones del PDU. En las líneas 34 a 65 se encuentra la función getJSON, esta lee el archivo JSON de origen y lo procesa. De las líneas 36 a 42 se guardan los datos de tiempos y consumos de Línea A, B y Total en algunas variables. En las líneas 44 a 62 se crea un texto en HTML para así ordenar los datos en una tabla y facilitar su uso y visualización. En la línea 64 se llama la función download (Figura 50) encargada de descargar este texto.

```
68  $(document).ready(function() {
69    // Agrega botón extra a la interfaz web
70    $("form[name=serial_form]").append("<input id='downloadbtn' type='button'
      value='Download Data by second'>");
71
72    $("#downloadbtn").click(function() {
73      // Descarga la información del PDU
74      getData();
75    });
76  });
```

Figura 48: Fragmento de script - Función Ready

```
// Lee los datos json del PDU
function getData(){
  var lineA, lineB, lineTotal, intervals;

  // Lee la dirección del archivo Json
  var jsonfile = $("param[name=flashvars]").val();
  jsonfile = jsonfile.split("=")[1];

  $.getJSON(jsonfile, function (data) {
    console.log("postback");
    // Obtiene los valores de tiempo
    intervals = data.x_axis.labels.labels;

    // Obtiene los valores de energía
    lineA = data.elements[0].values;
    lineB = data.elements[1].values;
    lineTotal = data.elements[2].values;

    // Genera una tabla HTML
    var resultText = "<html><head><title>PDU's Data</title></head><body> <table>
    <thead> <tr><td>Hour</td>" +
        "<td>Minute</td><td>Seconds</td> <td>Line A</td><td>Line B</td><td>Line
        Total</td></tr></thead><tbody>";

    var hour, tablerow;

    // Llena tabla HTML
    for(var i= 0; i < lineA.length; i++){
      // Crea un objeto "moment" para facilitar el procesado de datos tipo fecha
      hour = moment(intervals[i], "HH:mm:ss");

      // Agrega un renglón a la tabla
      tablerow = "<tr> <td>" + hour.hours() + "</td><td>" + hour.minutes() +
      "</td><td>" + hour.seconds() +
                "</td><td>" + lineA[i] + "</td><td>" + lineB[i] + "</td><td>" +
                lineTotal[i] +"</td></tr>";
      resultText+= tablerow;
    }

    // Cierra las llaves HTML de la tabla
    resultText+= "</tbody></body></html>";

    download("data.html", resultText);
  });
}
```

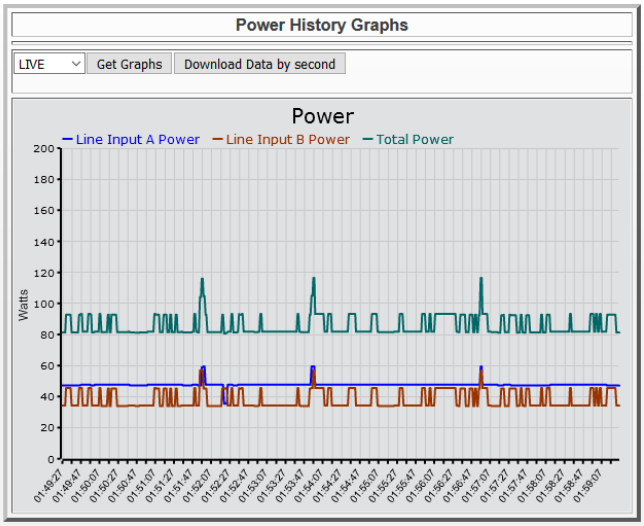
Figura 49: Fragmento Script - Función getData

```
// Descarga un archivo de texto con un nombre filename y un contenido text
function download(filename, text) {
  var element = document.createElement('a');
  element.setAttribute('href', 'data:text/plain;charset=utf-8,' +
  encodeURIComponent(text));
  element.setAttribute('download', filename);

  element.style.display = 'none';
  document.body.appendChild(element);

  element.click();

  document.body.removeChild(element);
}
```

Figura 50: Fragmento script - Función download

Una vez se inyecta el script aparece un botón que al clickearlo nos permite descargar los datos del PDU. Los datos descargados tienen formato HTML y se muestran en una tabla (

Figura 51).



Hour	Minute	Seconds	Line A	Line B	Line Total
1	49	27	47.2	34.2	81.4
1	49	28	47.2	34.2	81.4
1	49	29	47.2	34.2	81.4
1	49	30	47.2	34.2	81.4
1	49	31	47.2	45.6	92.8
1	49	32	47.2	45.6	92.8
1	49	33	47.2	45.6	92.8
1	49	34	47.2	45.6	92.8
1	49	35	47.2	45.6	92.8
1	49	36	47.2	45.6	92.8
1	49	37	47.2	34.2	81.4
1	49	38	47.2	34.2	81.4
1	49	39	47.2	34.2	81.4

Figura 51: Funcionamiento script