UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

VMCA: UN SISTEMA PARA LA CONSOLIDACIÓN DE RECURSOS EN PLATAFORMAS DE VIRTUALIZACIÓN

Trabajo Fin de Master

Master Universitario en Computación Paralela y Distribuida

Autor

Carlos de Alfonso Laguna

Directores

Ignacio Blanquer Espert

Germán Moltó Martínez

Julio de 2013

Resumen

Los recursos proporcionados mediante la tecnología Cloud Computing se sirven desde los Centros de Procesos de Datos (CPD) de los proveedores de servicios, y suelen utilizar técnicas de compartimentación y virtualización para proporcionar dichos recursos. Estos CPDs suelen estar dimensionados para la carga que se espera, pero seguramente habrá situaciones de derroche energético porque haya máquinas reales encendidas sin estar alojando máquinas virtuales. Además del problema energético, nos podemos encontrar el problema adicional de los recursos de virtualización fragmentados, que puede hacer que no seamos capaces de atender una solicitud aún cuando haya recursos suficientes.

En este trabajo se describe el diseño de un sistema para la replanificación de máquinas virtuales en plataformas de virtualización que ataque a estos problemas. La solución óptima al problema de distribución de máquinas virtuales en nodos reales se suele modelar como un problema de ”empaquetado de objetos en compartimentos” (conocido en inglés como *bin packing*) y se suele tratar de resolver desde el punto de vista de la inteligencia artificial o utilizando métodos basados en heurísticas. Además, en el caso de los sistemas en producción nos encontramos con una situación inicial de distribución de las máquinas virtuales, y no se puede detener todo el sistema para poder hacer la redistribución. En el presente trabajo se ha desarrollado un algoritmo basado en heurísticas, en el que se parte de esta situación y se llega a una solución en la que se pueden ordenar las migraciones de máquinas virtuales de unos nodos a otros, para poder mejorar la situación de acuerdo a unos criterios establecidos.

Índice general

CAPITULO

1

Introducción

La necesidad de contener los consumos energéticos de un Centro de Proceso de Datos (CPD) ha pasado a un primer plano, debido principalmente al aumento de precio de la electricidad. De hecho, a nivel mundial, el consumo energético de los CPD tiene un impacto significativo (estimado en el 0.5 % en 2008) y está previsto que se cuadruplique en los próximos años (hasta 2020) [1]. Además hay que tener en cuenta que se estima que la eficiencia media de los CPD se encuentra en torno al 50 % [2], con lo que además del elevado consumo, gran parte de éste está siendo malgastado.

Por otro lado hay que tener en cuenta que el consumo energético no viene determinado únicamente por los servidores en sí, sino que hay todo un aparataje alrededor de ellos, que van desde los Sistemas de Alimentación Ininterrumpida (UPS, del inglés *Uninterruptible Power Supplies*) hasta las máquinas de aire acondicionado, dedicadas a contrarrestar el calor generado por la instalación. Pero además de lo que es el consumo energético de los equipos, la eficiencia de los componentes y el gasto asociado a la infraestructura de soporte, tenemos que tener en cuenta el gasto asociado al consumo del equipamiento ocioso. Llama la atención que se estima que la utilización media de los CPD se sitúa entre el 10 % y el 50 % de los picos de carga para los que han sido dimensionados [3]. Y esto nos lleva a la conclusión de

que podríamos reducir el consumo energético tan solo desactivando aquellas partes de los CPD que no se estén utilizando, y encenderlas cuando sean necesarias.

En este contexto surge la posibilidad de reducir el consumo energético. Las posibles acciones encaminadas al ahorro energético se pueden clasificar básicamente en las de Gestión Estática de Energía (SPM, del inglés *Static Power Management*) y las de Gestión Dinámica de Energía (DPM, del inglés *Dynamic Power Management*) [4]. Las primeras (SPM) están relacionadas con los componentes de los sistemas, el consumo de los mismos, su eficiencia, etc. y se centran en la utilización de componentes de alta eficiencia y bajo consumo, mientras que las segundas (DPM) están relacionadas con la adaptación de los sistemas a las circunstancias específicas de cada momento.

En el contexto del SPM, muchos fabricantes se están centrando en aumentar la eficiencia energética con el objetivo de reducir el impacto del consumo del equipamiento. Así cada vez se dispone de CPUs más eficientes, memorias de menor consumo o discos que eliminan las partes mecánicas para reducir la energía necesaria. En el caso de DPM básicamente existen dos aproximaciones: (1) regular el consumo de los componentes de los equipos en función de la utilización utilizando, por ejemplo, el módulo de Escalado Dinámico de Frecuencia y Voltaje (DVFS, del inglés *Dynamic Voltage and Frequency Scaling*) que permite regular la frecuencia y consumo del procesador; y (2) realizar una distribución inteligente de la carga de computación entre los distintos nodos de la plataforma con el objetivo de utilizar aquellos más eficientes y dejar otros en estado ocioso, para que puedan ser apagados o puestos en un modo de bajo consumo (ahorro de energía, hibernación, etc.).

Las técnicas de ahorro energético han sido estudiadas para el caso de ejecución de trabajos en el contexto de la computación científica, y recientemente ha tomado relevancia su estudio para el caso de las plataformas de virtualización que dan soporte al Cloud Computing.

El Cloud Computing introdujo un nuevo paradigma en la distribución y consumo de recursos y aplicaciones en Internet. Esta tecnología supone la posibilidad de utilizar recursos de computación, almacenamiento, aplicaciones, etc. bajo demanda y de forma dinámica, a través de Internet [5]. Ello se traduce en la posibilidad de disponer de, por ejemplo, espacio de almacenamiento, máquinas plenamente

funcionales, aplicaciones de funcionalidades diversas, etc. de acuerdo a las necesidades específicas del momento y pagando únicamente por el consumo que se haga. Así, desde el punto de vista de los usuarios finales supone la posibilidad de acceso a cualquier tipo de recursos de forma ágil y sencilla, mientras que desde el punto de vista de los proveedores, es un nuevo mecanismo para la prestación de servicios a través de Internet.

Estos servicios suelen estar soportados por los CPD de los proveedores de servicios, y suelen utilizar técnicas de compartimentación y virtualización para proporcionar todos estos servicios. Por ejemplo, cuando se proporciona espacio de almacenamiento, no se suele proporcionar un disco para cada usuario, sino un espacio de almacenamiento virtual que suele coexistir con el de otros usuarios en uno o varios dispositivos físicos. El caso de las máquinas es similar, ya que no se suele proporcionar una máquina física a los usuarios (esto sería la forma en que se solía hacer de forma clásica), sino que los proveedores se suelen apoyar en técnicas de virtualización para ofrecer a los usuarios finales las Máquinas Virtuales (MV) que coexisten en una misma máquina real.

El caso particular de las MVs resulta de especial interés, dado que se ha popularizado en gran medida gracias a los hipervisores, que están disponibles para el público en general en forma de productos (por ejemplo, Xen[1], KVM[2], VMWare[3], Virtual Box[4], etc.). Con esta tecnología se consigue, a partir del hardware de una máquina real y utilizando un hipervisor, proporcionar al usuario un sistema virtual capaz de comportarse como si de uno real se tratase. En la figura 1.1 se muestra el esquema básico de un sistema de virtualización de máquinas. La cuestión es que este mismo tipo de producto se utiliza en los CPDs de los proveedores para proporcionar el servicio que ofrecen, pero a la escala correspondiente del CPD.

Si bien para un usuario final puede resultar práctica la creación manual de una MV en su propio equipo, en el momento en que la necesite, cuando pasamos a la escala de CPD, en el que nos podemos encontrar decenas o cientos de máquinas

[1] http://www.xenproject.org
[2] http://www.linux-kvm.org
[3] http://www.vmware.com
[4] https://www.virtualbox.org

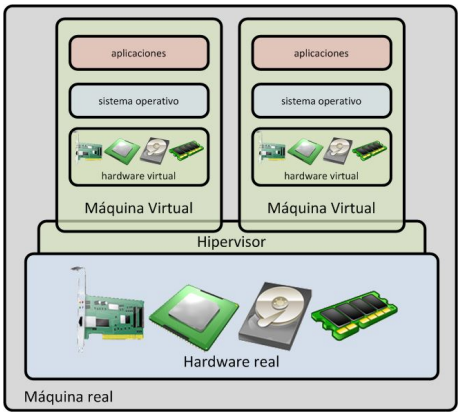

Figura 1.1: Arquitectura de virtualización de máquinas.

reales, resulta necesario automatizar el proceso de creación de éstas para que la situación sea manejable. Para ello, se suelen utilizar plataformas de gestión de MVs que, básicamente, se encargan de controlar todas las máquinas reales en las que se van a ejecutar MVs y dar una visión unificada al administrador del estado de la plataforma de virtualización y de las posibles acciones sobre ella (creación de máquinas virtuales, eliminación, migración, etc.). Muchos proveedores como Google, Amazon o Microsoft utilizan sus propios desarrollos para gestionar sus plataformas de virtualización, pero existen productos como VMWare vCenter, o alternativas gratuitas y otras de código abierto, como Eucalyptus[5], OpenNebula[6] (ONE) u OpenStack[7] que permiten crear plataformas de virtualización con las máquinas propias de la organización (también conocidas como plataformas *on-premises*).

La cuestión es que, al final de todo, los CPD que albergan las plataformas de virtualización están compuestos por sistemas similares a los centros de cálculo científicos, y probablemente estarán dimensionados de acuerdo a la demanda esperada. Así, proveedores como Google, Amazon o Microsoft tendrán decenas de miles de máquinas reales en múltiples CPD, mientras que la plataforma de virtualización de una empresa mediana contará con, apenas, unos cuantos servidores.

A la hora de operar con la plataforma de virtualización, resulta inmediato pensar que, a medida que los usuarios soliciten la puesta en funcionamiento de MVs,

[5]http://www.eucalyptus.com
[6]http://opennebula.org
[7]http://www.openstack.org

el middleware de gestión de la plataforma las irá distribuyendo entre las distintas máquinas reales, aplicando distintas políticas. Algunos ejemplos son la distribución homogénea de MVs entre máquinas reales para que las virtuales dispongan de los máximos recursos reales posibles, la compactación de MVs en máquinas reales para utilizar el mínimo número de equipos físicos posible, etc. Sin embargo, con una alta probabilidad, nos encontraremos con situaciones de derroche energético bien porque (1) haya máquinas reales encendidas sin que estén alojando MVs, o bien porque (2) las MVs que hay distribuidas entre las máquinas reales podrían estar en funcionamiento utilizando un número menor de recursos físicos. En ambos casos podríamos haber aplicado técnicas de ahorro energético tipo DPM para distribuir las MVs entre las máquinas reales de una forma más eficiente, y apagar aquellos nodos que no están alojando ninguna MV. En este sentido, podemos encontrar trabajos como [6] o [7] en los que se trata de actuar en este sentido: dejando libres algunos nodos reales para que puedan ser apagados.

Sin embargo, en la mayoría de los casos, las MVs se irán creando a medida que sean necesarias y se destruirán cuando ya no sean necesarias. Esto hace que, probablemente, la planificación que se había hecho en un momento dado, en base a una situación específica, haya dejado de ser la más eficiente. Esta situación se irá corrigiendo en cierta medida, según se vayan creando nuevas MVs puesto que se realizarán una nueva planificación de alojamiento de las nuevas, en base a la situación en la que se encuentre el CPD en el momento de su creación. Lo que ocurre es que podemos encontrarnos en casos en los que no se estén creando MVs y que suponga una situación en la que se estén utilizando los recursos de forma ineficiente, que sería interesante corregir. Un caso muy evidente es el que se muestra en la figura 1.2, en el que las circunstancias han llevado a que haya una MV ”pequeña” alojada en cada nodo físico, imposibilitando así el apagado de los mismos.

Estas situaciones estacionarias pueden parecer poco frecuentes en casos de grandes empresas como los Google, Amazon o Microsoft que tienen un servicio continuado a nivel mundial. Puede parecer raro que en estos proveedores, en un momento dado, no haya alguien poniendo en funcionamiento una MV. Pero en el caso de una empresa que realiza un despliegue cloud *on-premises* para consumo propio (servidores web de proyectos, servidores de aplicaciones, etc.), estas situaciones pueden darse al final de cada jornada de trabajo, y pueden llegar a mante-

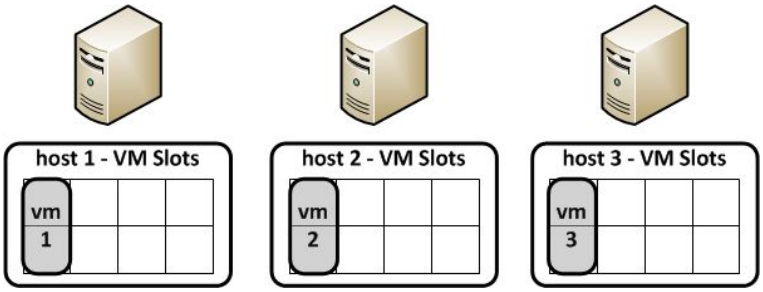

Figura 1.2: Distribución poco eficiente de máquinas virtuales en nodos reales.

nerse a lo largo del fin de semana o periodos vacacionales, dilatando así el impacto de esta incidencia (y con ello el derroche energético).

Por otro lado, la dinámica de creación y eliminación de MVs nos puede llevar a otra situación inconveniente que no está únicamente relacionada con el ahorro energético, sino también con la eficiencia de uso de los recursos de la plataforma de virtualización: la fragmentación de los recursos de virtualización.

Esta situación se dará cuando las MVs hayan quedado distribuidas en los nodos reales, de forma que se reserve sólo una parte de los recursos reales para dichas MVs y quede una parte de ellos libre, provocando así que se reduzca la eficiencia del uso del total de los recursos. El efecto es que el número de cores, memoria y discos físicos en funcionamiento y sin utilizar puede ser elevado. La consecuencia es que, independientemente del derroche energético, se puede dar que, en un momento dado, los criterios de calidad de servicio (QoS, del inglés *Quality of Service*) que se hayan establecido para la plataforma no permitan poner en funcionamiento una MV debido a que no haya una cantidad de recursos contiguos suficientes para alojarla (de forma intuitiva, porque ”no quepa”). Sin embargo, es posible que en realidad sí que se pudiera ubicar la máquina en caso de que los recursos no estuviesen fragmentados. Esto se puede ver de forma más intuitiva en la figura 1.3.

Si en la figura entendemos los slots de virtualización como Gb. de memoria, en el ejemplo estaríamos tratando de alojar una máquina que necesita 6 Gb., pero en el caso de arriba sólo tendríamos la posibilidad de proporcionar slots de entre 1 y 4 Gb. En caso de que los recursos de la plataforma no estuvieran fragmentados, dispondríamos de slots de entre 1 y 8 Gb., pudiendo así proporcionar el servicio.

Como vemos, una mala situación ya no sólo afecta al consumo sino que puede

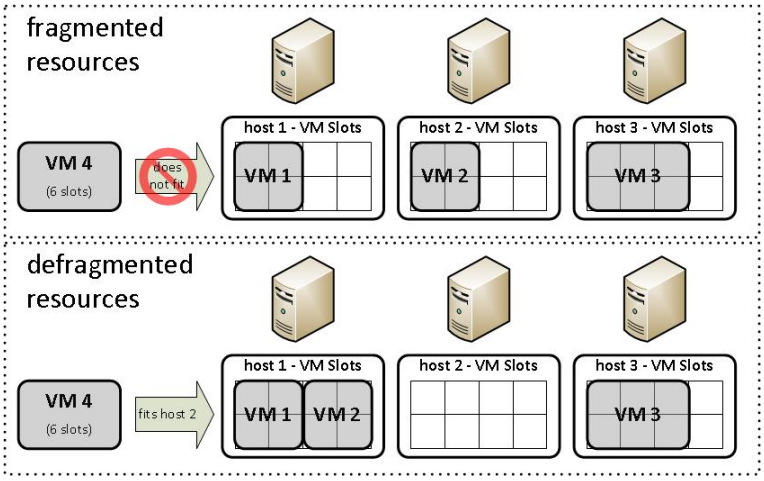

Figura 1.3: Los recursos de virtualización fragmentados pueden afectar a la QoS.

interferir en la capacidad de ofrecer un servicio y, por tanto, suponga un impacto en el negocio del proveedor. De nuevo, si nos fijamos en los CPD de los grandes proveedores como Google, Amazon o Microsoft, la incidencia de no poder entregar una VM a un usuario sería mínima porque se diluiría fácilmente entre la demanda de servicio. Pero en el caso de un CPD de menor entidad, esto se puede traducir en un problema de reputación que afecte al negocio.

Para tratar de solucionar esta situación en cierta forma deberíamos incluir una replanificación de la ubicación de las MV. Sin embargo, esto no es una tarea que se pueda afrontar de una forma tan inmediata como la planificación convencional, sino que se deben tener en cuenta una serie de factores que afectan de forma directa al rebalanceo de la carga. En este caso se deben tener en cuenta aspectos como el coste de la migración, puesto que mover una máquina supone un coste temporal y, eventualmente, un tiempo dilatado durante el que la máquina puede no ser accesible (en caso de que se haga una migración offline, consistente en apagar la máquina, copiar el disco y ponerla en funcionamiento de nuevo). Esto afecta a la replanificación, en el sentido de que no podemos simplemente suponer que vamos a partir de una situación actual, identificar una situación óptima y colocar las MVs de acuerdo a ello, ya que el proceso de movimiento de las MVs tiene un coste temporal. Además, no basta con identificar situaciones finales, sino que también deberemos de estar seguros de que haya una combinación de movimientos de máquinas que

llegan a esas situaciones, y averiguarlos. Esto es porque se pueden dar casos de ubicación de las máquinas a las que es imposible llegar desde la situación de partida, como se puede ver en la figura 1.4. Aquí vemos que, a pesar de tener una situación óptima con la que podríamos apagar un nodo, no es posible llegar a ella de no ser que contáramos con un soporte adicional donde ubicar transitoriamente las máquinas (y la posibilidad de desactivarlas durante el tiempo de reubicación de las mismas).

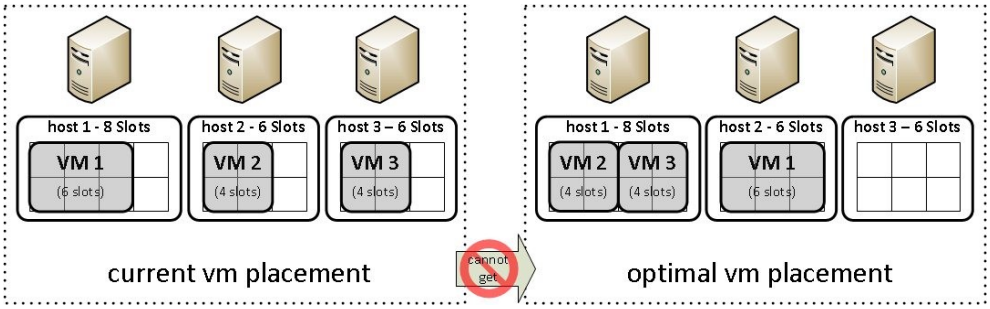

Figura 1.4: Ubicación óptima no alcanzable de máquinas virtuales.

Además de todo esto surge el problema de elegir la máquina que se va a migrar y de valorar si compensa migrar máquinas con respecto a la mejora que se va a obtener. Todo esto se debe compatibilizar con la incorporación de los criterios de planificación que equilibrarían la carga y que mantendrían los niveles de servicio de la plataforma (por ejemplo, cores reales por CPUs virtuales o memoria virtual por memoria real).

Para afrontar este problema se suelen utilizar dos tipos de aproximaciones. Por un lado, se aplican técnicas de inteligencia artificial [7][8] mientras que por otro, se hace una caracterización del problema como de *bin packing* multidimensional [9][10], con el inconveniente añadido de que partimos de una situación de ubicación inicial y queremos que el número de movimientos a realizar sea mínimo. La cuestión es que es bien conocido que este último problema es NP-completo y por tanto solo podremos aspirar a obtener una solución óptima para problemas de tamaño pequeño, utilizando técnicas como la fuerza bruta y evaluando todas las combinaciones de movimientos. En la práctica, estos problemas se suelen resolver tratando de aplicar heurísticas para llegar a una buena solución aproximada.

En este trabajo se describe el diseño de un sistema para la replanificación de MVs en las plataformas de virtualización que ataque a los problemas planteados. Los objetivos son tanto que se liberen nodos de virtualización como que haya recursos libres contínuos que permitan al proveedor de servicio el atender determinadas solicitudes que, en circunstancias normales, podrían no llegar a ser satisfechas.

Además, tendremos en cuenta el caso de los sistemas en producción, en los que nos encontramos con el problema adicional de que existe una situación inicial de distribución de las MVs, y no se puede suponer que podemos detener todo el sistema para poder hacer la redistribución y luego ponerlo en funcionamiento de nuevo. Aquí se muestra el desarrollado un algoritmo basado en heurísticas, en el que se parte de esta situación y se llega a una solución en la que se pueden ordenar las migraciones de MVs de unos nodos a otros, para poder mejorar la situación de acuerdo a unos criterios establecidos.

El trabajo no sólo se queda en el propio diseño, sino que se acaba plasmando en un producto llamado VMCA (del inglés *Virtual Machine Consolidation Agent*), que se distribuye en forma de código fuente y que puede ser utilizado para trabajar con plataformas de virtualización de código abierto populares, como es el caso concreto de ONE que se ha tomado como referencia para el presente desarrollo.

El resto del documento está estructurado de la siguiente forma: en primer lugar se presentan trabajos relacionados que se han encontrado en la literatura científica, y se revisan las distintas aproximaciones que se siguen para afrontar el problema. En el capítulo 3 se detallan los objetivos de diseño del sistema, así como las características esperadas del mismo. En el siguiente capítulo se describe en detalle el diseño teórico de los algoritmos propuestos, y la arquitectura del agente de consolidación de MVs. A continuación se explican los distintos tipos de heurísticas que se manejan en el sistema, y se proponen algunos criterios concretos que se pueden utilizar para el buen funcionamiento del algoritmo. En el capítulo 6 se detallan las adaptaciones que ha sido necesario realizar para poner en producción el sistema, de forma que pueda interactuar con una plataforma de virtualización basada en ONE. En este mismo capítulo se muestran los resultados obtenidos por el sistema, poniendo en práctica las distintas heurísticas planteadas, y se trata de averiguar cuales de ellas proporcionan mejores resultados. En el siguiente capítulo se incluye un pequeño un resumen del trabajo realizado y se proporcionan unas conclusiones acerca

de los resultados obtenidos. Así mismo se explican distintas líneas que se pueden seguir para mejorar y continuar con el trabajo desarrollado. En el último capítulo se da una visión de conjunto del producto desarrollado, y se sitúa dentro del marco de investigación del autor.

CAPITULO

2

Trabajos relacionados

De acuerdo a [11], las técnicas de consolidación de servidores se pueden dividir entre (a) estáticas, en las que simplemente se hace una planificación de la carga de acuerdo a la situación actual del despliegue, (b) semi-estáticas, en las que se replanifica la carga tras largos periodos de tiempo (días, semanas, etc.), y (c) dinámicas que tratan de reaccionar en tiempo real (o en poco tiempo) de acuerdo a la variación de la carga.

En el contexto de las técnicas estáticas podemos ubicar las que se utilizan en los planificadores que acompañan a los *middlewares* de virtualización. Estos planificadores están encargados recibir solicitudes de alojamiento de MVs y decidir en qué servidor real se van ubicar de forma efectiva. En este contexto existen numerosas técnicas, que van desde las más simples, como (i) el rellenado de servidores (que intenta colocar las MV de forma que se maximice el uso de los recursos disponibles), (ii) la distribución de la carga en los servidores (para reducir la compartición de los recursos reales entre los virtuales y, con ello, aumentar la eficiencia de las MV), (iii) técnicas como la ubicación aleatoria, o (iv) las políticas basadas en turnos que tratan de repartir la carga de forma homogénea entre los servidores (por ejemplo, round robin).

Podríamos considerar que el problema del ahorro de energía estaría ya resuelto mediante la aplicación de las técnicas estáticas, pero debemos pensar que, eventualmente, podrían fallar servidores (situación que se podría solucionar eliminando la MV correspondiente y lanzándola de nuevo) o aparecer la necesidad de crear otros nuevos (por ejemplo, para el desarrollo de un nuevo proyecto). En estas situaciones podríamos aplicar técnicas de consolidación de la carga que se encargasen de reubicar las MV para reducir el consumo de energía y desfragmentar los recursos, al tiempo que se mantiene la QoS.

La solución de este problema consiste en tratar de averiguar, en qué máquinas reales deberían quedar alojadas las MVs desplegadas en la actualidad, de acuerdo a diversos criterios (número de MVs por nodo real, MVs que no deben cambiar de ubicación, minimización del tiempo en el que una máquina no se encuentra disponible, número de CPUs virtuales menor o igual que el número de cores reales, etc.).

En caso de que el tamaño del despliegue fuese lo suficientemente reducido, podría ser abordada de forma simple mediante técnicas de Fuerza Bruta (analizando todas las posibilidades) o aplicando técnicas de programación dinámica, como la Ramificación y Poda (que trataría de explorar el mayor número de posibilidades, descartando aquellas que se puede detectar de forma temprana mediante heurísticas que no van a conducir a una solución mejor). El enfoque podría ser el de tratar de comprobar todas las posibilidades de ubicación de todas las MVs en los nodos reales, y realizar los movimientos necesarios para llegar a dicha situación.

En este trabajo se pretende abordar la problemática dentro de un contexto altamente dinámico, como podría ser el de una empresa que actuase como proveedor de recursos cloud o el de un entorno científico. En ambos casos nos podremos encontrar que tanto la solicitud como eliminación de MVs puede tener un alto grado de dinamismo, que los recursos resultarán muy ajustados para la potencial demanda (y querremos tenerlos desfragmentados para poder atender peticiones con características específicas), y que nos interesará reducir el número de servidores necesarios para apagar o poner en modo de bajo consumo aquellos que no estén alojando MVs.

En la actualidad, hay dos corrientes principales en cuanto a la consolidación dinámica de MVs en CPDs para ahorrar energía: la aplicación de técnicas de in-

teligencia artificial, o la caracterización del problema como uno de *bin packing* y tratar de resolverlo utilizando las aproximaciones habituales para ello, con las adaptaciones necesarias para el contexto de las MVs.

2.1 Aplicación de técnicas de inteligencia artificial

Una línea de trabajo trata de utilizar técnicas de inteligencia artificial para la gestión de CPDs y, dentro de ellos, contemplar también las plataformas de virtualización. Dentro de esta corriente podemos encontrar el trabajo de [7] que utiliza una aproximación de *aprendizaje automático* para tratar de hacer una consolidación de las aplicaciones de un CPD utilizando VMs. Una de las conclusiones de este trabajo es la orientación de éste hacia técnicas de *reinforcement learning* para una gestión más globalizada.

En este contexto se ha identificado un trabajo muy activo en los últimos años: el proyecto europeo GAMES (Green Active Management of Energy in IT Service Centers) [12], cuyos objetivos principales son los de desarrollar metodologías innovadoras, métricas, servicios y herramientas para la gestión activa de la eficiencia energética en CPDs. En particular, en este proyecto se utiliza el modelo MAPE (Monitorización, Análisis, Planificación y Ejecución) para la gestión de los CPDs. Concretamente se dispone de una serie de sensores (software y sensores ambientales) que se monitorizan de forma continuada y, se procesan de acuerdo a técnicas de modelado del contexto (del inglés *Context Modelling*) en base a una ontología específica. A partir de ahí, se analizan los datos de acuerdo a unos algoritmos de razonamiento (concretamente *reinforcement learning*) y en función de los resultados, se realizan las acciones correctivas destinadas a corregir una situación detectada.

Los objetivos de eficiencia se establecen de acuerdo a unas políticas que se deben mantener. Es decir, se indican unos parámetros que se deben mantener (temperatura de la sala, humedad, carga máximas y mínimas de CPU cada servidor, etc.) y se identifican acciones correctivas encaminadas a corregir dichas situaciones. Los criterios de intervención se fijan de acuerdo a una entropía calculada como combinación polinómica de los distintos sistemas de monitorización, y una tolerancia de los mismos, que determinan de forma objetiva, si se debe iniciar el proceso de corrección de la situación.

El inconveniente principal de la aproximación seguida en este proyecto es que el enfoque está orientado al CPD de forma global y no a la gestión de despliegues informáticos en particular. Esto tiene el problema de que se deben mezclar métricas sin relación aparente (por ejemplo, temperatura con carga de CPU) para establecer los criterios de acción. Además requiere unos ejercicios de abstracción muy complejos para poder llegar a conseguir resultados simples como la liberación de los nodos que nos planteamos en este trabajo. Esto es debido a que aspectos como los criterios de reequilibrado de carga no podrían ser modelados de forma directa de acuerdo a la ontología de GAMES puesto que no obedecen a criterios de eficiencia energética sino a parámetros de calidad de servicio.

Aún en el caso de que se puedan utilizar los resultados de este proyecto para gestionar los recursos de un despliegue cloud, como tratan de mostrar en [13] (si bien tenemos que pensar que es un ejercicio teórico puesto que no se detalla cómo se correlacionan las unidades para obtener los indicadores, ni se proporcionan detalles acerca de los experimentos), nos encontramos con que los algoritmos de razonamiento resultan muy costosos. De acuerdo a resultados presentados [8], los procesos de análisis de la situación pueden llevar decenas de segundos aún para despliegues pequeños (4 servidores). Esto es debido a que utilizan una aproximación de fuerza bruta, considerando todas las posibles acciones que puedan mejorar una situación. Para tratar de reducir este coste se utiliza una tabla en la que se almacenan las acciones realizadas ante una situación, a modo de caché. Dada la variabilidad en la casuística de tipologías de MVs, el número de servidores de que podría constar un despliegue cloud de tamaño significativo, y los posibles movimientos de MVs entre servidores, el coste de almacenamiento de esta tabla podría resultar un problema en sí mismo. Los propios autores del proyecto GAMES comentan que su aproximación se podría utilizar para reequilibrado de carga, pero en ningún momento detallan cómo o si lo han conseguido, puesto que se centran más en aspectos genéricos como los sensores, la actuación sobre los mismos, y las acciones genéricas como el apagado de nodos sin utilizar.

Ahondando en la arquitectura de este tipo de iniciativas con aproximaciones basadas en la inteligencia artificial, podemos ver que un resultado como el que nosotros nos planteamos podría integrarse de forma coherente. En esta línea se manejan conceptos como *sensores* y *acciones correctivas*, dado que están muy en

consonancia con las SLA (Service Level Agreement). Para poder realizar dicha integración se podría considerar el despliegue cloud que tratamos de gestionar como un ente abstracto. A partir de este supuesto, se podrían definir *sensores* y *acciones correctivas* genéricas sobre el mismo, y así poder introducir la plataforma de virtualización dentro del control global siguiendo los mecanismos que éste maneja. Un ejemplo de *sensor* podría proporcionar datos que se derivasen a partir de la situación instantánea del despliegue, para ser utilizados en la monitorización realizada por el control global. Estos datos podrían consistir en el número de nodos reales que quedarían en funcionamiento aplicando cada una de las políticas de consolidación disponibles en la plataforma. De forma complementaria se debería definir una *acción correctiva* que podría consistir en el establecimiento de la política de consolidación de MVs aplicada en el despliegue (mejora de eficiencia, mejora de calidad de servicio, etc.). Esta acción correctiva sería indicada por el control global, como resultado de los algoritmos de razonamiento.

Como resumen, las técnicas revisadas, basadas en inteligencia artificial tienen una vocación más ambiciosa que la que se plantea en este trabajo, ya que se considera el problema del ahorro energético desde un punto de vista global (dispositivos lógicos, equipos de enfriamiento, iluminación, etc.). En estos casos, la reubicación de las MVs no es uno de los objetivos del sistema, sino una consecuencia de la aplicación de las medidas correctivas aplicadas por los algoritmos de razonamiento. Además, esta reubicación de MVs se hace utilizando exclusivamente criterios de ahorro energético, sin que se contemplen otros aspectos como el balanceo de la carga o la disponibilidad de los recursos.

2.2 El problema del Bin Packing

Otra línea de trabajo muy activa en la consolidación dinámica de MV se basa en la ubicación de las mismas aplicando algoritmos de *bin packing*. Este tipo de problemas consiste, básicamente, en la colocación de una serie de objetos (con unas dimensiones) en una serie de contenedores (de tamaño limitado) de forma que el número de contenedores utilizado sea mínimo. Dado que es bien conocido que este problema es NP-completo, se suelen aplicar técnicas elaboradas para garantizar la finalización de los análisis en un tiempo razonable.

Un ejemplo de estas técnicas es la presentada en [14], que considera el problema como un *bin packing multidimensional* y, en lugar de utilizar las técnicas convencionales, desarrolla unos algoritmos sofisticados inspirados en los movimientos sociales de las hormigas. Otro ejemplo de consolidación de MVs se muestra en [15], donde se presenta un algoritmo basado en las costumbres migratorias de los gansos y el tipo de formaciones que adoptan. El inconveniente principal que presentan estas técnicas es el del coste temporal, ya que sus propios resultados muestran que se necesitan horas para resolver el problema en casos de uso de tamaño relativamente pequeño. Para tratar de obtener mejoras de rendimiento, se proponen sistemas tipo *peer to peer* como en [16], donde se aplican las técnicas de los movimientos de las hormigas anteriormente comentadas, u organizaciones como [17] donde se establece una consolidación de MV mediante negociación entre nodos de virtualización y sin una planificación global. El inconveniente en este caso es que resulta muy invasivo con la plataforma puesto que necesita instalar componentes de software adicionales en cada uno de los nodos de virtualización.

Estos casos en que los algoritmos tienen un elevado coste temporal resultan poco aplicables a entornos en los que se requiera un alto nivel de interactividad. Estos serían aplicables a despliegues que fuesen muy estáticos en el tiempo, como podría ser el sistema de producción de una empresa en la que suelen desplegarse servidores muy estables como el de dominio, el de correo, el servidor web, etc. que no suelen ser candidatos a ser encendidos y apagados de forma frecuente.

En cualquier caso estos sistemas de consolidación serían válidos para cargas con un comportamiento bastante estático. Para ello es necesario que la probabilidad de que cambie la situación de la plataforma (que se lance o elimine alguna MV) durante los periodos de análisis y migración de MVs sea muy baja. En ese caso tanto el coste del análisis como el de la consolidación de los servidores podrían quedar diluidos en el tiempo de vida de las propias máquinas.

La aproximación más habitual para la consolidación de MV entendido como un problema de *bin packing* suelen estar basada en técnicas de *First Fit* (FF) o *Best Fit* (BF). Estas soluciones consisten básicamente en utilizar una heurística para seleccionar un elemento para colocar un contenedor, y situarlo en él de acuerdo a distintos criterios: colocarlo en el ”primero que quepa” en el caso de FF, o colocarlo en ”el que mejor quepa” para el caso de BF. Estos algoritmos tienen la variante

Decreasing, que consiste en ordenar de forma decreciente, en función de la capacidad. De esta forma, además de FF y BF tendremos *First Fit Decreasing* (FFD) y *Best Fit Decreasing* (BFD).

En [9] proponen la utilización de este tipo de algoritmos para migrar un CPD físico a uno virtualizado, utilizando el menor número de nodos físicos posible, y para ello diseñan un algoritmo de BFD. En este caso no se contemplaba la reubicación dinámica de la carga, sino que se quedaban en la ubicación inicial.

En el caso de [10] ya se plantea un sistema de control de CPD en el que se contemplan acciones correctivas (como la utilización de DVFS o el apagado de nodos reales), y la aplicación de técnicas de consolidación. Para este tipo de técnicas se utiliza una aproximación basada en FFD. El inconveniente de este método es que está exclusivamente centrado en las prestaciones de las aplicaciones en cuanto a la CPU y no considera el problema desde un punto de vista multidimensional (por ejemplo, teniendo también en cuenta la memoria y el disco).

Otro de los trabajos identificados que está muy en la línea de lo que estamos planteando es el mostrado en [18], [6]. Lo que se plantea aquí es muy similar al trabajo anterior, con la salvedad de que considera que el problema desde un punto de vista multidimensional. Aquí se plantea que la consolidación dinámica de las MVs de un CPD se puede dividir en dos partes: por un lado, la ubicación adecuada de las nuevas MVs, y por otro, la reubicación de las MV que ya se encuentran desplegadas.

Para decidir dónde alojar las MVs que se van creando en el sistema utilizan una versión modificada de un algoritmo de BFD, intentando ubicar las máquinas en los nodos donde se estima que producirá un menor incremento de consumo energético. La optimización de la ubicación de las MVs que ya están desplegadas también la dividen en dos fases: por un lado la selección de las MVs que se pueden mover, y por otro, la selección del nodo donde van a quedar ubicadas. Esta segunda parte la resuelven mediante la aplicación del mismo algoritmo de ubicación de máquinas, pero ordenando las candidatas a ser migradas de acuerdo a la utilización de CPU.

A partir de ese algoritmo, lo que hacen es centrar el esfuerzo en establecer una serie de heurísticas para seleccionar las MVs que deben ser migradas. El mecanismo para ello consiste en establecer unos márgenes de utilización de los recursos de virtualización y tratar de quitar MVs de los nodos en función de dichos márgenes.

Por ejemplo, en caso de que se sobrepase un porcentaje de utilización (o reserva) de CPU en un nodo, se deberán migrar algunas de sus MV para volverlo a situar dentro de los límites.

A pesar de que la propuesta de este trabajo es bastante acertada, tiene el inconveniente de que supone que no hay coste de migración. Además, los estudios realizados se hacen desde un punto de vista teórico (mediante un simulador) y no tienen en cuenta algunos aspectos relacionados con la puesta en producción como puede ser la introducción de restricciones. Algunos ejemplos son que no sea posible migrar una máquina entre recursos de virtualización (por temas de incompatibilidad de hipervisores) o porque exista una dependencia de la MV con el recurso donde se ha desplegado.

Como hemos visto, dentro de este tipo de aproximación encontramos técnicas sofisticadas, pero la mayoría consisten en la aplicación de algoritmos tipo FF y sus variantes. Las principales diferencias en estos casos pueden ser las heurísticas que se utilicen, la diversidad de dimensiones (tipos de recursos) que se manejen o si se tiene en cuenta aspectos más ligados a la puesta en producción, como puede ser el coste de migración. Sin embargo, en sólo en algunos casos se tiene en cuenta el hecho de que en realidad se parte de una situación inicial de distribución de MVs en nodos de virtualización, mientras que en el resto se centran en averiguar la mejor colocación de las MVs sin tratar de averiguar la secuencia de migraciones que conduce a ella.

CAPITULO

3

Trabajo planteado

El trabajo que se plantea en este documento consiste en la creación de un sistema autónomo de consolidación de MVs, encargado de corregir la ubicación de las MV en los recursos reales que, como consecuencia de la creación y destrucción de las mismas, hayan quedado en una situación en la que dichos nodos de virtualización estén utilizados de forma poco eficiente. Esto se traduce en tratar de averiguar cómo deben recolocarse las MVs que se encuentran ya en funcionamiento en los recursos reales de un CPD, de forma que se *mejore la situación*.

Por tanto, uno de los objetivos de este sistema será el de conseguir liberar recursos reales, con el fin de poder apagarlos o ponerlos en modo de bajo consumo (el apagado efectivo de los nodos quedará fuera del ámbito del trabajo que aquí se plantea, porque existen productos como CLUES[1] encargados de ello). Pero al mismo tiempo se aplicarán una serie de criterios de alojamiento de MVs que permitirán mantener los niveles de QoS definidos para la plataforma. Es decir, a la hora de decidir qué MVs van a quedar alojadas en cada recurso real se tendrán en cuenta aspectos como el número de *cores* virtuales por *cores* reales, la cantidad de memoria virtual asignada por cantidad memoria real, etc.

[1] http://www.grycap.upv.es/clues

El trabajo se va a situar dentro de un ámbito de interactividad altamente dinámico, como podría ser el caso de un proveedor de servicio o en un entorno científico, en los que hay una actividad continua de creación y eliminación de MVs. Por lo tanto, para realizar la replanificación de ubicación de MVs se tendrán en cuenta aspectos como el coste de la migración de las máquinas y el tiempo que las máquinas van a estar fuera de servicio, así como aquellos criterios que se considere que pueden afectar a la experiencia del usuario final de las MVs.

Una de las consecuencias deseables del sistema es que los recursos de los nodos de virtualización se utilicen de forma más eficiente. Es decir, que haya la menor cantidad de memoria, CPUs, cores, etc. reales no utilizados distribuidos en nodos que alojen MVs. Al dejar nodos vacíos se pretende no sólo poder ahorrar energía (mediante el apagado de los equipos no utilizados), sino también consolidar el ”espacio libre” destinado a albergar MVs, con el objetivo de poder atender a peticiones que, en caso de tener los recursos fragmentados, no podrían ser satisfechas. Para medir los recursos se deberá contar con un sistema de monitorización que obtendrá los datos de la plataforma de virtualización. Por otro lado, para disponer de criterios objetivos que nos permitan determinar cómo se encuentran distribuidos, utilizaremos indicadores estadísticos, como la varianza.

En el diseño del sistema tendremos en cuenta las siguientes características:

- Deberá ser **independiente de la plataforma de gestión** de MVs subyacente. Siguiendo este principio trataremos de abarcar la mayor cantidad de middlewares de gestión posibles.

- Será **respetuoso con la interacción del usuario** con las MVs y, para ello, tratará de evitar la migración demasiado frecuente de las MVs. En otro caso, se podría dar lugar a una pérdida de prestaciones o accesibilidad con las MVs.

- **No se supondrá ninguna tecnología de migración en particular**, con la intención de que sea aplicable tanto a sistemas con migración ”en vivo” como ”offline”.

- Se tendrá en cuenta el hecho de que **la migración de una MV tiene un coste**. Este coste será principalmente temporal, pero también se deberá poder modelar un coste de cualquier otra naturaleza.

- **Será un sistema extensible**, para no imponer unos criterios concretos y que el usuario pueda incorporar nuevas heurísticas más adaptables a las circunstancias de su plataforma o a las de los usuarios de la misma.

- Estará **orientado a ser un producto real** y no sólo un ejercicio teórico, de forma que la implementación podrá ser desplegada en producción.

De forma intuitiva, la solución que se plantea analizará los nodos de la plataforma en busca de aquellos sobre los que se podrá actuar. Comprobará a qué otros nodos se pueden migrar sus MVs y si las migraciones reportarán un beneficio sobre la distribución de los recursos. En caso de que así sea, seleccionará aquellos movimientos de máquinas que aporten un mayor valor y supongan un menor coste (o esfuerzo) y, finalmente ordenará la migración de las MVs que cumplan con estos requisitos.

Esta solución se orientará a modo de ”agente” que monitorizará y analizará periódicamente el sistema. Dado que vamos a trabajar con un entorno altamente dinámico, será imprescindible que la solución tenga un coste temporal reducido, y que el tiempo que suponga la reubicación de las MVs tampoco sea muy largo, para no interferir con el funcionamiento normal de la plataforma.

De las propuestas vistas en los trabajos previos vamos a seguir una aproximación basada en el modelado del problema como uno de *bin packing* y, en nuestro caso, resolveremos el sistema mediante técnicas de BF de las máquinas alojadas en nodos con recursos fragmentados. Así, se va seguir una aproximación similar a la presentada en [6], introduciendo algunas variaciones. En primer lugar, se introducirá el concepto del coste de la migración y se podrá utilizar como factor de decisión a la hora de decidir el movimiento de las MVs. También se introducirá el concepto de ”estabilidad de una MV”, con la intención de evitar que haya MVs que estén moviéndose frecuentemente entre nodos, mientras que hay otras que casi nunca (o nunca) se muevan. Además se introducirá la posibilidad de incorporar criterios de selección de nodos de destino de las MVs migradas que vayan más allá de la estimación del incremento de consumo energético para, de esta forma, poder incorporar la posibilidad de mejora en la eficiencia en la utilización de los recursos disponibles en los nodos. Finalmente se orientará a la creación de un agente real

que pueda ser puesto en producción y que, por tanto, tenga en cuenta las posibles circunstancias de los despliegues reales.

CAPITULO

4

El Agente de Consolidación de Máquinas Virtuales

En este trabajo se describe el diseño y desarollo de un Agente de Consolidación de Máquinas Virtuales (VMCA, del inglés *Virtual Machine Consolidation Agent*), que se encargará de comprobar el estado de una plataforma de virtualización, analizar la situación, planificar y realizar los movimientos de MVs entre los recursos reales para conducir a una *situación mejor*. En el contexto de este agente se entenderá como una ”situación mejor” aquella en la que se haya conseguido reducir el número de máquinas reales que estén ocupadas por alguna MV o aquella en la que se haya conseguido una mayor cantidad de ”recursos contiguos”, manteniendo las políticas de QoS del despliegue. Se consideran recursos contiguos aquellos que se encuentran dentro de la misma máquina y que, por tanto pueden ser utilizados para atender a la solicitud de alojamiento de una MV.

Para tratar de abarcar la mayor cantidad de plataformas de virtualización posible, se ha tratado en todo momento de desacoplar VMCA del middleware concreto que gobernará el sistema. De esta forma se ha intentado que VMCA se pueda utilizar tanto en la mayoría de las plataformas actuales (ONE, OpenStack, Eucalyptus, etc.) como en otras que pudieran aparecer en un futuro y que soporten una serie de

operaciones básicas como pueden ser la obtención de información de la plataforma o la migración de VMs entre nodos físicos. Para ello se utilizará un sistema de *conectores* que permita desarrollar los componentes necesarios para conectar con el *middleware* con el que se vaya a interactuar. En cualquier caso, en este trabajo vamos a centrarnos en dos tipos de plataformas concretas: por un lado, un despliegue ficticio de *test* que nos servirá para modelar casos de uso y estudiar posibles problemas, y por otro lado una plataforma en producción a la que se tiene acceso, y que está basada en ONE.

De forma intuitiva, VMCA tendrá un comportamiento en *pipeline* relativamente sencillo y que constará de 4 fases similares a las del modelo MAPE propuesto en [8]. Estas fases podrán ser ejecutadas de forma secuencial y los resultados de cada una serán los datos de entrada de la siguiente:

1. **Obtención de la información de la plataforma**, que recuperará los datos relativos a los recursos disponibles en los nodos reales, las MVs alojadas en ellos, el estado de las mismas, etc.

2. **Análisis de la situación**, que se encargará de comprobar y evaluar la situación actual de la plataforma de virtualización, en función de los datos obtenidos en la fase anterior.

3. **Planificación de las mejoras**, que será un proceso donde se identificarán las posibles acciones que se van a poder realizar para conducir a la *situación mejor*. Durante esta fase se calculará el coste de realización de las posibles acciones y se estimarán las posibles ventajas que se puedan obtener en cada caso.

4. **Ejecución de las acciones correctivas**, que será una fase en la que se seleccionarán las operaciones de migración de MV que se van a realizar, en función de los cálculos de la fase anterior y las características concretas del despliegue.

Para implementar este sistema se ha diseñado un sistema que tiene una arquitectura bastante sencilla, y que se puede ver en la figura 4.1. En ella se muestra que VMCA está formado por cuatro componentes básicos: (i) un conector con el

middleware de gestión de la plataforma, (ii) el sistema de monitorización de la plataforma, (iii) el analizador de la situación y (iv) los criterios específicos para realizar la planificación de la migración de MV.

La arquitectura trata de reflejar el desacoplamiento de VMCA con la plataforma sobre la que se van a defragmentar los recursos, mediante la utilización de los conectores. Además muestra la vocación de extensibilidad y variabilidad en cuanto a la aplicación de posibles algoritmos de reordenación de MV. Para ello, se ha creado un sistema de análisis de la situación genérico, que contempla la posibilidad de utilizar distintos algoritmos o criterios de selección de MV a mover, formas de ocupación de los nodos, etc.

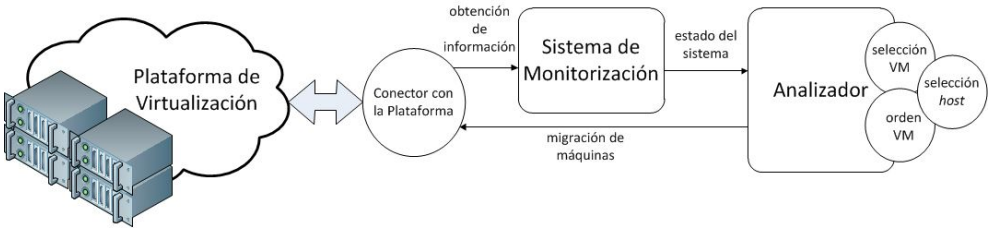

Figura 4.1: Arquitectura básica de VMCA.

En las siguientes secciones veremos más en detalle los distintos componentes y los aspectos más relevantes de su diseño.

4.1 Conexión con la plataforma de virtualización

De acuerdo al planteamiento anterior, VMCA tendrá dos puntos de conexión con la plataforma de virtualización: las fases 1 y 4. Esta interacción se puede dividir en dos apartados: (i) la monitorización de la plataforma y (ii) la ejecución de los movimientos de MV. Así que, para poder interactuar con el sistema, se ha definido un conjunto reducido de operaciones y un modelo de datos simple que permiten recoger la expresividad de una plataforma Cloud en cuanto a las MV que aloja, su monitorización y su manipulación. Esta abstracción nos permitirá tratar con el middleware a modo de *plug-in*, simplemente desarrollando los conectores específicos de la plataforma con la que se desee interactuar.

El modelo de datos que se maneja en VMCA está formado simplemente por una clase que modela los nodos que alojan las máquinas y otra clase que modela las MV en sí.

```
class Host(Identified):
  def __init__(self, id, memory = 0, cpu = 0, disk = 0):
    self.total_memory = memory
    self.total_cpu = cpu
    self.total_disk = disk
    self.id = id
```

Listing 4.1: Modelo de datos para los recursos de virtualización.

La información que VMCA necesita acerca de los recursos de virtualización se puede ver en el listado 4.1, y básicamente consiste en la cantidad de memoria, CPU y disco que se comparte en el nodo. Estos valores estarán expresados en una escala dependiente de la plataforma, y deberá ser la misma que la utilizada para indicar los datos de las MVs. De forma convencional se utilizarán *megas* para la memoria, *cores* para la CPU y *Gb.* para el disco.

```
class VM(Identified):
  ST_ERROR = -1
  ST_UNKNOWN = 0
  ST_RUNNING = 1
  ST_STOP = 2
  ST_OTHER = 3

  def __init__(self, id, memory = 0, cpu = 0, disk = 0):
    self.requested_memory = memory
    self.requested_cpu = cpu
    self.requested_disk = disk
    self.host_id = -1
    self.state = VM.ST_UNKNOWN
    self.id = id
```

Listing 4.2: Modelo de datos para las máquinas virtuales.

En cuanto a las MV, los datos que VMCA necesita acerca de la plataforma se pueden ver en el listado 4.2, y básicamente son la cantidad de memoria, CPU y disco que va a consumir del nodo de virtualización, además de una referencia al nodo en el que esta alojada, De igual forma, para establecer el enlace entre el modelo de datos de VMCA y la plataforma de virtualización, se utilizará un identificador que será dependiente de la propia plataforma.

Además de estos datos, VMCA necesitará poder identificar el nodo en el que se encuentra alojada una MV, y el estado de funcionamiento de la misma. En VMCA se contemplan los siguientes casos:

- **ST_RUNNING:** Este estado indica que la MV se encuentra en funcionamiento normal y que se encuentra en un estado en el que puede ser migrada a otro nodo.
- **ST_STOP:** En esta situación, la MV ha sido puesta en funcionamiento pero actualmente se encuentra parada. En este estado también puede ser migrada a otro nodo.
- **ST_ERROR:** Este es un estado en el que ha ocurrido un error de funcionamiento en la MV y, siguiendo una actitud conservadora para poder depurar el posible error, no puede ser migrada a otro nodo.
- **ST_UNKNOWN:** En este caso el conector de la plataforma no ha podido averiguar el estado de funcionamiento de la MV y por lo tanto no puede ser considerada dentro de la planificación.
- **ST_OTHER:** Este estado recoge todos aquellos estados adicionales que pueda considerar la plataforma de virtualización y que no puedan hacerse corresponder con ninguno de los anteriores. En esta situación la MV no será considerada para ser migrada.

Estos estados son dependientes de la plataforma de virtualización y, por lo tanto, no se puede establecer un diagrama de transición entre los mismos. Dicha transición viene definida por el propio *middleware* y VMCA únicamente puede contemplarlos de forma instantánea.

```
class Deployment:
  def get_info(self)
  def get_possible_destinations(self, vm_id, host_origin_id)
  def can_migrate_vm(self, vm_id, host_origin_id, host_destination_id)
  def migration_cost(self, vm_id, host_origin, host_dest)
  def migrate_vm(self, vm_id, host_origin, host_dest)
```

Listing 4.3: Clase para la interacción con la plataforma de virtualización.

De forma concreta, la interacción de VMCA con el middleware de gestión de la plataforma se hace mediante una clase que implementa las funciones mostradas en el listado 4.3, cuya funcionalidad se comenta a continuación:

- **get_info** se utiliza para obtener la información del despliegue en cuanto a los recursos de virtualización y las MV alojadas en los mismos, utilizando el modelo de datos de VMCA.
- **get_possible_destinations** es una función que se utiliza para averiguar a qué nodos se puede migrar una MV en particular. Este mecanismo se puede utilizar para modelar las restricciones de la plataforma en cuanto a movimiento de MV entre nodos de virtualización.
- **can_migrate_vm** se utiliza para comprobar si es posible migrar una máquina desde el nodo en que se encuentra a un destino en particular. Este es otro mecanismo que se puede utilizar para implementar restricciones en cuanto a la migración de MV entre recursos de virtualización.
- **migration_cost** es una función que se utiliza para obtener un valor numérico del coste de realizar la migración de una MV de un nodo a otro. Este valor estará en una escala definida por el usuario que implementa el conector, y estará expresado en unas unidades dependientes de la plataforma de virtualización.
- **migrate_vm** se utiliza para realizar la migración efectiva de una MV de un nodo de virtualización a otro.

Como se puede comprobar, la funcionalidad exigida a la plataforma es muy simple y su funcionalidad podrá ser obtenida, probablemente, utilizando las APIs o interfaces de cualquier middleware de gestión de MVs. A pesar de su sencillez, también permite expresar una casuística muy diversa en cuanto a restricciones en cuanto a la posible migración de MV, al tiempo que incluye el factor del *coste de migración* para que pueda ser tenido en cuenta por el sistema de análisis.

Un aspecto a destacar es que se ha hecho énfasis en el modelado de las restricciones de migración de MV. Esto es debido a que en un despliegue en concreto se pueden dar algunas de las siguientes situaciones:

- **No todas las MV pueden estar alojadas en cualquier nodo**, por ejemplo porque la plataforma incluya distintos hipervisores y una MV en particular no es compatible con el del nodo de destino, o que una MV necesite acceder a un dispositivo físico que únicamente se encuentra en el nodo en el que se encuentra alojada actualmente (por ejemplo, porque necesite montar un disco físico concreto).

- **A veces una MV alojada en un nodo no puede moverse a otro en particular**, por ejemplo porque esté implementada la *migración en vivo* y se requiera un disco compartido entre el nodo de origen y destino, y el par de recursos de virtualización no cuente con esta característica.

- **Hay MVs que por sus circunstancias especiales no deban moverse**. Por ejemplo, si un usuario ha lanzado un cluster compuesto por MVs que están trabajando en paralelo, será imprescindible evitar que alguna de las MVs se detenga para ser migrada con el objetivo de no interferir en los resultados que se vayan a obtener.

4.2 El sistema de monitorización

VMCA maneja un sistema de monitorización que se encarga de gestionar la lógica del agente en cuanto a recursos de virtualización y MVs. Este monitor se basa en la existencia de un conector con la plataforma mediante el que obtiene la información de la misma, utilizando el modelo de datos visto en la sección 4.1 para posteriormente tratarla y presentarla al sistema de análisis. Para obtener la información de la plataforma se utilizarán las herramientas propias del *middleware*, teniendo en cuenta que esta operación se debe hacer de forma eficiente y que ha reflejar la situación de forma fidedigna. En otro caso, la monitorización podría interferir en las prestaciones de VMCA o provocar la planificación de migraciones erróneas.

La información obtenida mediante el conector con la plataforma la podemos considerar como una instantánea del estado de la misma. Sin embargo, para implementar la funcionalidad de VMCA es imprescindible conocer la evolución en el

tiempo del estado del sistema, y el monitor será el encargado de mostrar esta continuidad. Esto quiere decir que éste será el componente que nos permitirá identificar las máquinas que se encuentran en un estado estable (porque lleven tiempo en el mismo nodo y en el mismo estado) y de llevar un histórico de los movimientos de las mismas entre los nodos. Este histórico servirá, principalmente, para tratar de evitar efectos que puedan interferir en el funcionamiento de las MV, como que haya una que esté continuamente migrando entre nodos, o que ocurra un efecto ”ping-pong”[1].

La forma de contemplar la continuidad temporal en el monitor consiste en asociar al estado de cada MV una marca temporal (*timestamp*), que indicará el instante de tiempo en el que llegó a dicho estado. Así, la tupla formada por (*identificador*, *estado*, *nodo*, *marca de tiempo*) se utilizará como referencia para determinar si una MV se encuentra en un estado ”estable” y por tanto puede ser considerada para ser movida a otro nodo. A partir de ello se podrán implementar el resto de funcionalidades esperadas para el sistema de monitorización.

Además de este procesamiento básico, relativo a la continuidad de la información a lo largo del tiempo, este componente se encarga de realizar unas acciones adicionales con el objetivo de facilitar el uso de los datos disponibles durante la fase de análisis:

- **Creación de estructuras de datos adicionales**, entre otras, preparación de listas de las MV asociadas a cada nodo para que sea más fácil su ubicación.

- **Normalización de los recursos en los nodos** ya que, en principio éstos podrán ser heterogéneos y necesitaremos disponer de una escala común para poder compararlos. Se realiza una normalización independiente de la memoria, la CPU y el disco a partir del valor máximo de cada recursos, para ponerlos todos ellos en el intervalo *[0..1]*.

- **Cálculo de recursos disponibles** en los recursos de virtualización. Se ha observado que algunos middlewares (en particular, ONE) proporcionan información instantánea acerca de los recursos libres basada en la monitorización

[1] El efecto ”ping-pong” consiste en que haya una MV que migre de un nodo a otro y que en la siguiente planificación vuelva al nodo original, y que esto ocurra de repetitiva.

de los recursos físicos con las herramientas del sistema (memoria libre, CPU libre, etc.). Estos datos no podemos utilizarlos tal cual para los objetivos de QoS porque no reflejan la reserva de dichos recursos sino la utilización instantánea y por lo tanto es posible que en el instante de observación no se encuentren utilizados de forma efectiva pero que con el uso de las MVs sí que sean necesarios. Por ejemplo, el hecho de que una CPU virtual se encuentra sin uso hará que las herramientas del sistema detecten una utilización reducida o nula del recuro real, pero es necesario que estos recursos se encuentren reservados cuando la CPU virtual vuelva a ser utilizada. En este trabajo vamos a asumir que cuando una MV está asignada a un nodo, en realidad estamos *reservando* los recursos que solicita. Por lo tanto, el sistema de monitorización hace un cálculo de los recursos disponibles mediante la sustracción de los recursos reservados de los totales. Estos valores se utilizarán posteriormente para verificar si una MV ”cabe” o no en un nodo.

Por otro lado, en lugar de limitar la funcionalidad del monitor a la gestión y organización de la información de la plataforma, también se le ha dotado de capacidades adicionales para que pueda ser utilizado para hacer un tratamiento de ella a nivel lógico, sin necesidad de realizar las operaciones de forma efectiva sobre ella. Así, por ejemplo, deberemos distinguir dos tipos de migraciones: la ”migración efectiva” y la ”migración lógica”. La primera de ellas se hará a nivel de plataforma (mediante el conector correspondiente) y moverá una MV de un nodo a otro. De esta forma, al consultar el estado del sistema posteriormente veremos reflejado el movimiento. En cambio, la ”migración lógica” se podrá hacer a nivel de monitor y consistirá en la simulación del movimiento de una MV de un nodo a otro, sin necesidad de hacerla.

Para realizar esta simulación, se calculará el estado en que se quedaría el sistema en caso de monitorizar el sistema tras haber realizado la migración y se actualizarán las estructuras de datos implicadas en el mismo, sin afectar a la plataforma. Esto nos permitirá hacer, durante la fase de análisis, las simulaciones de las acciones que realizaríamos sobre la plataforma para poder comprobar los efectos que éstas tendrían sobre ella.

Como se puede ver en el listado 4.4, el sistema de monitorización dispone de una serie de funciones propias relacionadas con el disparo de la monitorización de la plataforma y la obtención de información ya procesada, y un conjunto de funciones equivalentes a las de la gestión de la plataforma mediante las que podremos hacer las simulaciones.

```
class Monitor:
  # funciones de monitorizacion
  def monitor(self)
  def detect_stalled_vms(self, stall_threshold)
  def detect_stalled_hosts(self, stall_threshold)
  def get_stalled_hosts(self, stall_threshold)
  def get_stalled_vms(self, stall_threshold)
  def duplicate(self)

  # funciones de simulacion
  def get_info(self)
  def get_possible_destinations(self, vm_id, host_origin_id)
  def can_migrate_vm(self, vm_id, host_origin_id, host_destination_id)
  def migration_cost(self, vm_id, host_origin, host_dest)
  def migrate_vm(self, vm_id, host_origin, host_dest)
```

Listing 4.4: Clase del sistema de monitorización.

Las funciones de simulación del monitor tienen una componente lógica en cuanto al cálculo de las situaciones y mantenimiento de las estructuras de datos. Por ejemplo, para determinar si una máquina puede moverse de un nodo a otro, comprueba las estructuras de datos internas para ver si se encuentra allí. Pero estas funciones deben estar sincronizadas con la plataforma, en especial, en cuanto a la implementación de restricciones de movimiento de MV. Así, para hacer una simulación adecuada, el monitor se apoya en las funciones de los conectores con la plataforma.

Este mecanismo impone una serie de restricciones en la implementación de los conectores, ya que condiciona la semántica de algunas de sus funciones. De este modo, a nivel del conector hay que interpretar las funciones de la siguiente forma:

- **can_migrate_vm**: si la maquina *vm_id* estuviera en el nodo con identificador *host_origin_id*, ¿podría migrar al nodo con identificador *host_destination_id*?
- **migration_cost**: si la maquina *vm_id* estuviera en el nodo con identificador *host_origin_id*, ¿qué coste tendría la migración al nodo con identificador *host_destination_id*?

- **get_possible_destinations**: ¿a qué nodos podría migrar la maquina con identificador *vm_id* si estuviera en el nodo con identificador *host_origin_id*?

Por su parte, el monitor incluirá una serie de restricciones a nivel lógico que incluyen el hecho de que una MV no se pueda migrar de un nodo a otro, porque no se encuentre alojada por el nodo indicado o porque el nodo de destino no disponga de los recursos suficientes. Pero también se contemplarán aquí las restricciones relativas al histórico de movimientos, como que una MV haya cambiado de nodo muchas veces en un tiempo determinado.

4.3 Análisis de los recursos

La tarea del componente de análisis consiste en, a partir de los datos del monitor, comprobar si se podría *mejorar* la situación en cuanto a la distribución de MV en los nodos de virtualización, averiguar dicha situación y proporcionar la secuencia de movimientos que llevaría a dicha ”situación mejor”.

Si contemplamos la defragmentación de los recursos de la plataforma desde un punto de vista global, se podría dar el caso de que fuera necesario encender nodos de virtualización para corregir determinadas situaciones, como es el caso de la figura 4.2. En esta figura se puede ver, de forma intuitiva, que podríamos conseguir una situación mejor encendiendo el nodo *host1*, y moviendo a él las MV de los nodos *host2* y *host3* para vaciarlos y poder apagarlos. Esto tendría más impacto aún si contemplamos una situación en la que los nodos *host2* y *host3* fuesen equipos antiguos y poco eficientes, y el nodo *host1* fuese un equipo mucho más nuevo y de bajo consumo.

En este trabajo vamos a concentrar los esfuerzos en la liberación de nodos de virtualización, para dejarlos en un estado en que puedan ser apagados. Por lo tanto, no vamos a tener en cuenta aquellos recursos que no se encuentren activos en un momento dado. Sin embargo, en caso de que se diera la situación de la figura 4.2 y el nodo *host1* se encontrase encendido, VMCA debería planificar las migraciones para tender a la solución intuitiva.

Para tener en cuenta estos recursos de virtualización apagados tendríamos que (a) plantear un sistema con un alcance mayor, que se encargase también de decidir

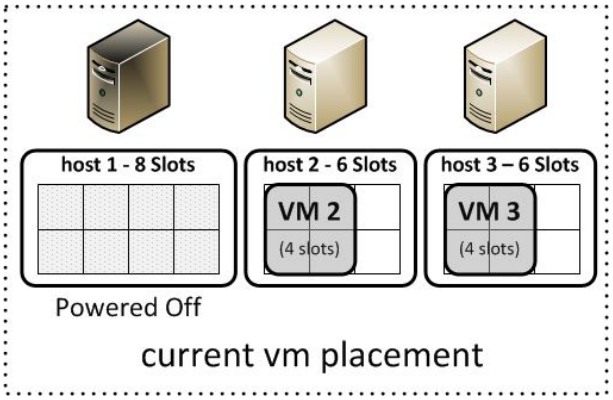

Figura 4.2: Necesidad de encender nodos para mejorar la distribución de MV.

cuando se deben activar los nodos, o (b) introducir un acoplamiento con un sistema de este tipo, como podría ser CLUES [19]. En cualquier caso, este tipo de actuaciones quedará fuera del alcance de este trabajo.

4.3.1 El algoritmo principal de análisis

En apartados anteriores hemos visto que este problema se puede caracterizar como de *bin packaging multidimensional*, y que es un problema *NP-completo*. La obtención de una solución razonable pasa por utilizar algún algoritmo que, mediante la aplicación de una serie de heurísticas, nos lleve a una solución cercana a la óptima. La solución propuesta en este trabajo consiste, básicamente, en seleccionar un nodo cuyas MVs se encuentren en un estado estable y, por tanto, sean candidatas a ser movidas y aplicar un algoritmo de BF para tratar de colocarlas en el resto de los recursos de virtualización.

En caso de que aplicando los movimientos propuestos el nodo quedara totalmente liberado de MVs, dichos movimientos se ejecutarían de forma efectiva sobre la plataforma. En otro caso, no se moverá ninguna de ellas para no *empeorar* la situación. Desde el punto de vista del rebalanceo de la carga en los recursos, esta técnica no proporcionará los mejores resultados, pero en la versión inicial de VM-CA se ha optado por focalizarnos en la liberación de los recursos de virtualización y no en la redistribución de los mismos. El algoritmo utilizado se puede ver en el listado 4.5.

```
nodos = detectar_nodos_estables(monitor)

hacer
  n = extraer_nodo_a_vaciar(nodos)
  migraciones = analizar_migraciones(n, monitor)
  si migraciones_vacian_nodo(migraciones, n) entonces
    aplicar_migraciones(migraciones)
mientras no_se_ha_vaciado_un_nodo y hay_nodos_estables_no_analizados(nodos)
```

Listing 4.5: Algoritmo básico de recolocación de máquinas virtuales

Como podemos ver, estamos aplicando una técnica tipo FF para seleccionar los nodos a vaciar, en lugar de hacer un análisis exhaustivo de las posibles alternativas. De esta forma conseguimos que la cota superior del coste del algoritmo sea $n \cdot \max(m_i)$, donde n es el numero de nodos estables y m_i el número de MVs en el nodo i, para los nodos estables. De forma habitual m_i será del orden de M/n, siendo M el número total de MVs, y $n <= N$, siendo N el número total de nodos. Por lo tanto tendremos que la cota superior del coste será $N \cdot M$.

Una mejora de este algoritmo puede ser el tratar de analizar todos los nodos para decidir en qué orden se deberían tratar de vaciar. El algoritmo mejorado se muestra en el listado 4.6.

```
mientras hay_nodos_estables_no_analizados(nodos)
  nodos = detectar_nodos_estables(monitor)

  posibles_secuencias_migraciones = []
  para_cada nodo de nodos:
    migraciones = analizar_migraciones(n, monitor)
    si migraciones_vacian_nodo(migraciones, n) entonces
      posibles_secuencias_migraciones.append(migraciones)

  migraciones = seleccionar_migraciones(posibles_secuencias_migraciones)
  aplicar_migracion(migraciones)
```

Listing 4.6: Algoritmo mejorado de recolocación de máquinas virtuales

Esto supondría, en cierto modo, una vuelta atrás puesto que estamos introduciendo un método de resolución por fuerza bruta. Su coste es del orden de $n^2 \cdot m$ y por tanto, una cota superior de $N^2 \cdot M$. Hasta el momento, hemos ido aplicando heurísticas para tratar de para facilitar la resolución del problema. Sin embargo, si el número de nodos no es excesivamente grande, y el coste de análisis de cada nodo es reducido, podríamos tratar de poner en producción este método.

En esta segunda versión del algoritmo tenemos una función *seleccionar_migraciones* que se encargará de identificar, de entre todas las posibles migraciones viables para los nodos, cual es la que se debe poner en práctica. Aquí se podrán poner en práctica distintas heurísticas, como pueden ser el vaciado del nodo con menor coste, el que produce mejor distribución de recursos, etc.

Un aspecto que puede resultar importante para la obtención de buenas soluciones con estos algoritmos es el de la heurística que determina el nodo que se va a tratar de vaciar en un momento dado. Aquí podremos encontrar diversas aproximaciones, que pueden ser la selección de aquel recurso que aloja menos MV, el que tiene más recursos libres o aspectos mucho más elaborados como hacer estimaciones del coste de migración de las MV y seleccionar las que menos costarían. Todas estas cosas quedarían recogidas por la función *extraer_nodo_a_vaciar*, y más adelante veremos algunas comparativas entre distintas propuestas.

4.3.2 Análisis de posibles migraciones

Una vez hemos seleccionado el nodo que vamos a tratar de vaciar, nos queda tratar de averiguar dónde podríamos reubicar sus MV. Esta funcionalidad se implementa en la función *analizar_migraciones* que aparece en los listados 4.5 y 4.6.

En este caso vamos a aplicar un algoritmo de BF para tratar de averiguar en qué nodos van a quedar mejor colocadas las MV del nodo que estamos tratando de vaciar. El concepto de *mejor colocación* de una MV será parte de una heurística que habrá que definir y de la que dependerá en gran medida el comportamiento y los buenos resultados del algoritmo. En el capítulo 5 discutiremos las distintas heurísticas que hemos considerado en este trabajo en más detalle. Algunas aproximaciones sencillas podrán ser colocar la MV en el nodo donde queden menos recursos libres, en el que tenga menos MV en funcionamiento, etc.

El algoritmo utilizado se muestra en el listado 4.7. En primer lugar, se hace un duplicado del objeto *monitor* para realizar la simulación de movimientos, porque la idea no es comprobar los movimientos de MVs de forma aislada. Hay que tener en cuenta que una vez movida una MV, ”ocupa” unos recursos y, por tanto, es necesario simular todos los movimientos de las MVs de un nodo en bloque para averiguar el progreso de reserva de los recursos. Podríamos pensar que bastaría con

obtener la lista de posibles movimientos evaluados una única vez y simplemente tratar de ordenarlos. Pero hay que tener en cuenta que cada movimiento de una MV cambia la situación en cuanto a la ocupación de los recursos y, por tanto, puede ser necesario reevaluarlos de nuevo (en función de cómo esté hecha la evaluación).

```
def analizar_migraciones(nodo_actual, monitor_real):
  monitor = monitor_real.duplicate()
  movimientos_hechos = []

  ordenar_maquinas(nodo_actual)
  para_cada mv de nodo_actual:
    destinos = monitor.get_possible_destinations(mv, nodo_actual)

    posibles_movimientos = []

    para_cada nodo_destino de destinos:
      evaluacion = evaluar_movimiento(mv, nodo_actual, nodo_destino)
      coste = monitor.migration_cost(mv, nodo_actual, nodo_destino)
      posibles_movimientos.append((mv, nodo_actual, nodo_destino, evaluacion,
          coste))

    si len(posibles_movimientos) > 0 entonces:
      movimientos = ordenar_movimientos(posibles_movimientos)
      movimiento = movimientos.pop()

      # sintaxis python para extraer campos de una tupla
      (mv, nodo_actual, nodo_destino, evaluacion, coste) = movimiento
      monitor.migrate_vm(mv, nodo_actual, nodo_destino)
      movimientos_hechos.append(movimiento)
```

Listing 4.7: Algoritmo básico de análisis de movimientos de MVs de un nodo.

Dado que el diseño del sistema está pensado para que sea extensible y adaptable a las políticas de la plataforma en particular, la mayoría de los aspectos subjetivos (y que, por tanto, requieren de una heurística) están expresados en forma de funciones. En concreto, podemos distinguir las siguientes personalizaciones:

- **ordenar_maquinas**: Se utiliza para establecer el orden en que se debe intentar mover las MVs del nodo.
- **evaluar_movimiento**: Se utiliza para establecer una evaluación numérica del movimiento de una MV a un nodo en concreto. Esto nos servirá para poder establecer una comparación objetiva entre los posibles movimientos de MVs y así decidir cuál tiene prioridad.

- **ordenar_movimientos**: Se utiliza para establecer una ordenación entre los movimientos para poder decidir cuál de ellos debe realizarse en primer lugar.

Hay que tener en cuenta que es muy importante hacer un diseño adecuado de las heurísticas para que trabajen de forma coordinada. Podemos ver un ejemplo de la influencia de distintas combinaciones de heurísticas en la figura 4.3. En ella podemos ver que, en el caso (1), si elegimos la *vm 2* para que sea movida en primer lugar y utilizamos un criterio de balanceo de recursos libres, nos quedaremos sin poder mover la *vm 1* porque no queda espacio y, por lo tanto, no podremos liberar el *nodo 1* de MVs. Sin embargo, en el caso (2) si cambiamos el orden de las MV y tratamos de mover la *vm 1* en primer lugar, no tendríamos este problema y podríamos recolocar las dos. También podríamos conseguir liberar el nodo si tomásemos el criterio de mover las MV donde menos recursos libres quedasen.

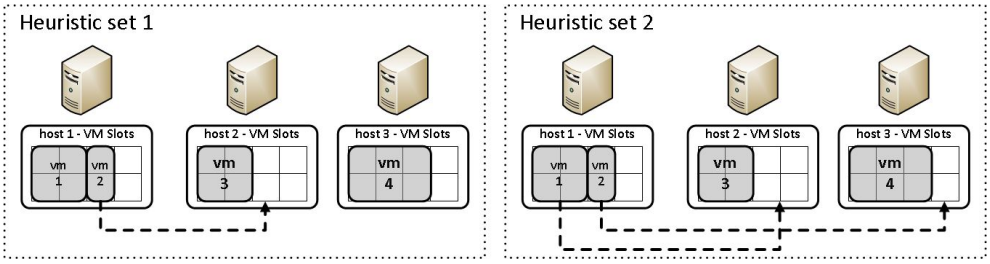

Figura 4.3: Influencia de distintos tipos de heurísticas.

CAPITULO

5

Heurísticas del sistema

En en capítulo 4 hemos visto la creación de un sistema para la liberación de nodos de virtualización de las MVs que están alojadas en ellos, mediante la migración de las mismas a otros nodos de la plataforma. Hemos visto un algoritmo que sigue una aproximación FF para la selección del nodo que se va a intentar vaciar, y posteriormente utiliza técnicas de BF para colocar las máquinas en el resto de nodos. Además hemos visto otro algoritmo que calcula el estado de la plataforma, tratando de vaciar todos los nodos (utilizando el mismo algoritmo de BF), y selecciona aquel que mejores resultados obtiene.

En todos los casos los algoritmos se basan en una serie de heurísticas, que es necesario definir en función de las características de la plataforma. En concreto, podemos distinguir tres tipologías de heurísticas: orden de vaciado de los nodos, evaluación del movimiento de las máquinas, orden de movimiento de las máquinas y selección de la lista de movimientos (en el caso del algoritmo mejorado).

Es importante tener en cuenta que el buen comportamiento del sistema dependerá de la combinación de heurísticas que se apliquen en conjunto. Es decir, un criterio de ordenación de nodos puede funcionar mejor con un sistema de evaluación de movimientos de MVs que con otro. Del mismo modo ocurre en el caso de la ordenación del movimiento de las MVs con respecto a la evaluación de los movimientos de éstas ya que, puede darse el extremo de que no se utilice en ningún

momento esta evaluación para determinar el orden en que se van a tratar de mover las máquinas.

Para muchos de los criterios relacionados con los nodos de virtualización vamos a basarnos en los recursos que estamos considerando en el sistema: memoria, CPU y disco. Para manejarlos de una forma más homogénea que como una tupla y compararlos de forma individualizada, a distintas escalas, vamos a utilizar la distancia euclídea de los valores normalizados con respecto al total de los nodos. Para ello, obtendremos para cada nodo i un valor r_i^{libres} de acuerdo a la ecuación 5.1. En dicha expresión los valores de $\max(memoria_j)$, $\max(cpu_j)$ y $\max(disco_j)$ se corresponden con la máxima cantidad de memoria, CPU y disco de entre el total de los nodos. En ese caso hemos utilizado los recursos libres, pero la expresión sería equivalente para el total de los recursos de cada nodo o para los recursos utilizados.

$$r_i^{libres} = \sqrt{m_i^2 + c_i^2 + d_i^2} \begin{cases} m_i = \dfrac{memoria_i^{libre}}{\max(memoria_j)} \\ c_i = \dfrac{cpu_i^{libre}}{\max(cpu_j)} \\ d_i = \dfrac{disco_i^{libre}}{\max(disco_j)} \end{cases} \quad j \in nodos \tag{5.1}$$

El espacio de los recursos que estamos manejando se puede adaptar a las características de la plataforma en particular. Así, podremos establecer unos pesos para los distintos recursos (α, β y κ) de forma que se de más importancia a unos que a otros. Es más, si no nos interesase utilizar alguno de los recursos dentro de la valoración conjunta, simplemente podríamos anularlo dándole un peso de 0. La propuesta generalizada queda reflejada en la expresión 5.2. En el resto del texto vamos a seguir utilizando el término *distancia euclídea* para hacer referencia a esta *distancia euclídea ponderada y generalizada*.

$$r_i^{libres} = \frac{\sqrt{\alpha \cdot m_i^2 + \beta \cdot c_i^2 + \kappa \cdot d_i^2}}{\alpha + \beta + \kappa} \begin{cases} m_i = \dfrac{memoria_i^{libre}}{\max(memoria_j)} \\ c_i = \dfrac{cpu_i^{libre}}{\max(cpu_j)} \\ d_i = \dfrac{disco_i^{libre}}{\max(disco_j)} \end{cases} \quad j \in nodos \tag{5.2}$$

Para el caso de los recursos solicitados por las MVs, podemos utilizar una expresión similar a las anteriores, normalizando con respecto al resto de máquinas del nodo. El cálculo de los recursos de la máquina (r_i) se haría utilizando la expresión 5.3. En caso de que eventualmente necesitásemos realizar comparaciones entre las MVs que se encontrasen en distintos nodos, podríamos utilizar una expresión equivalente, pero considerando el total de MVs del sistema.

$$r_i = \frac{\sqrt{\alpha \cdot m_i^2 + \beta \cdot c_i^2 + \kappa \cdot d_i^2}}{\alpha + \beta + \kappa} \begin{cases} m_i = \dfrac{memoria_i}{\text{máx}(memoria_j)} \\ c_i = \dfrac{cpu_i}{\text{máx}(cpu_j)} \\ d_i = \dfrac{disco_i}{\text{máx}(disco_j)} \end{cases} \quad j \in maquinas_{nodo} \tag{5.3}$$

En las siguientes secciones vamos a comentar algunas de las alternativas que se han tenido en cuenta en el desarrollo de este trabajo, para cada uno de los distintos apartados.

5.1 Selección del orden de vaciado de nodos

El orden de vaciado de nodos va a determinar cual de ellos se va a tratar de vaciar en primer lugar. Hay que tener en cuenta que, en función de este orden y el resto de heurísticas, podremos vaciar más o menos nodos y, además, variar los posibles agrupamientos de MVs que vamos a obtener. En el desarrollo de este trabajo se han tenido en cuenta las siguientes heurísticas:

- **Primero el nodo con menor número de MV.** En este caso simplemente se selecciona para vaciar en primer lugar aquel nodo que cuenta con un menor número de MV. Esta es una heurística que busca la simplicidad y se basa en el principio de que ”hay que intentar mover el menor número de MVs posible”, sin tener en cuenta las características de las mismas. Esto puede ser adecuado para reducir el número de MV que se pueden ver afectadas, si bien el coste total de la migración puede aumentar ya que no se consideran las características de las mismas (en particular, el tamaño de los discos, que

suelen definir el coste temporal de la migración debido a la copia de los ficheros de que constan).

- **Primero el nodo que tenga peor distribuidos los recursos en valor absoluto.** Esta es una heurística más elaborada, que trata de establecer un criterio para medir la calidad de la distribución de los recursos libres de un nodo. Para ello se calcula la distancia euclídea en el espacio de los recursos normalizados, para cada nodo i, de acuerdo a la fórmula 5.1. Una vez tenemos este valor para todos los nodos, podremos escoger el mayor de ellos, que será el que suponga una cantidad de recursos mayor desaprovechada.

  En el caso de que las MVs sean muy homogéneas y los nodos también lo sean, este criterio funcionará igual que el anterior. Sin embargo, si trabajamos en un entorno altamente heterogéneo, esta heurística nos podrá proporcionar unos resultados mejores o, al menos, equivalentes a si trabajásemos en un entorno homogéneo.

- **El que tenga las máquinas más pequeñas.** Este criterio es una combinación de los anteriores, y trata de establecer una relación entre los recursos utilizados y las MVs alojadas. Lo que se hace es calcular la distancia euclídea para calcular los recursos utilizados ($r_i^{utilizados}$), con la fórmula 5.1, y dividirla por el número de MVs del nodo. De esta forma, obtendremos la media de recursos utilizados por MV. A partir de ese valor, escogeremos el que tenga un valor menor.

- **El que tenga un porcentaje de recursos libre mayor.** Esta heurística se basa en el cálculo de la utilización relativa de los recursos, en forma de porcentaje. Así, lo que se hace es utilizar la fórmula 5.1 para calcular los valores correspondientes a los recursos totales (r_i^{total}) y a los recursos utilizados (r_i^{libres}), y calcular la relación entre ellos como la fracción r_i^{libres}/r_i^{total}. Posteriormente se elegirá en primer lugar aquél que tenga una proporción de recursos libres mayor.

- **Orden aleatorio**. Esta es una heurística muy simple, que se basa en que no se considera demasiado importante el orden en que van a ser elegidos los

nodos. Se podría aplicar cuando la carga del sistema no es muy homogénea o no es muy predecible, o simplemente a modo de test.

5.2 Evaluación del movimiento de las máquinas

La evaluación del movimiento de una MV de un nodo a otro trata de establecer una escala numérica de lo buena que resulta la operación. Lo que se pretende es establecer un criterio para que, ante un conjunto de posibles movimientos, podamos elegir uno frente a otro.

El resultado de la evaluación del movimiento de una MV se reduce a un objeto que, típicamente será un valor numérico expresado en una escala definida por el usuario. Sin embargo, en caso de que el usuario quiera, podrá utilizar sistemas más complejos (como objetos, tuplas, etc.). Esta evaluación se deberá utilizar de forma coordinada con la heurística de ”selección del orden de movimiento de las máquinas”, dado que se utiliza de forma exclusiva para establecer una posible ordenación.

Un aspecto a tener en cuenta para evaluar los MVs y conseguir una heurística conjunta coherente es el de la ordenación de liberación de nodos. De hecho, las evaluaciones de MVs van en la línea de las heurísticas de ordenación ya utilizadas en la sección 5.1.

Algunas de las heurísticas que se han considerado en este trabajo son las siguientes[1]:

- **Mejor cuantas menos máquinas queden en el nodo de destino.** Este criterio es interesante para tratar de distribuir de forma homogénea las MVs en los nodos, sin tener en cuenta más que el hecho de que se puedan alojar las máquinas en ellos. Es una heurística simple que probablemente sea aplicable en el caso de que la plataforma sea muy homogénea en cuanto a los nodos de virtualización y los tamaños de las MVs. La implementación consiste en utilizar el número de MVs como escala y, para cada movimiento, devolver como valor de evaluación el número de MVs en el nodo tras el movimiento.

[1]Hay que tener en cuenta que todas las evaluaciones se podrán utilizar también con el criterio inverso, bien adecuando la función de ordenación o bien devolviendo el valor calculado en negativo

Este mecanismo de asignación de MVs a nodos suele estar disponible en los middlewares de gestión de plataformas de virtualización habituales, como es el caso de ONE.

- **Mejor cuanto más se utilicen los recursos.** Esta heurística trata de estimar cómo quedarán distribuidos los recursos del nodo de destino, una vez la máquina haya sido movida. De esta forma, cuantos más recursos queden ocupados, se considerará que el movimiento es mejor. Como ocurría con la selección de nodos, este criterio se comportará de forma equivalente al anterior en el caso de que el despliegue sea homogéneo y los tamaños de MVs similares.

  La implementación consiste en calcular los recursos libres del nodo tras el movimiento $r_{destino}^{libres}$ utilizando la expresión 5.1 y devolverlos en valor negativo. De este modo, se primará el rellenado de los nodos.

- **Mejor cuanto mejor se utilicen los recursos.** Este criterio es similar al anterior, pero utilizando una escala relativa a la cantidad de recursos del nodo. En este caso, se calcula el ratio de ocupación de los recursos con respecto a la cantidad total como $r_i^{ocupado}/r_i^{total}$, utilizando la expresión 5.1, y se utiliza como valor de evaluación. De esta forma se estarán primando los movimientos de MVs que vayan rellenando los nodos de forma más eficiente.

- **Mejor cuanto mejor queden distribuidos los recursos utilizados entre las MVs.** Esta heurística simplemente calcula la cantidad de recursos dedicados a alojar MVs en el nodo, tras el movimiento, y lo divide entre el número de máquinas. También podríamos utilizar el valor de porcentaje de utilización de recursos, pero produciría resultados equivalentes.

- **Mejor mover la máquina más "grande" en primer lugar.** Este criterio es muy sencillo y tratará de mover las máquinas que más recursos soliciten en primer lugar. Dado que la evaluación de movimientos de MVs se realiza en el contexto de un único nodo, normalizaremos los recursos que solicita la MV que estamos probando con respecto al resto de MVs del nodo y calcularemos la distancia euclídea, utilizando la expresión 5.3.

Otra opción de evaluación en este caso sería la de comparar un recurso en concreto, como podría ser la cantidad de disco solicitada por las máquinas, o una combinación de ellos.

- **Mejor cuanto mejor distribuidos queden los recursos.** Esta heurística se basa en la utilización de la varianza de los recursos que quedan libres en los nodos, tras la realización del movimiento. De esta forma, con valores menores de la varianza, tendremos que los recursos libres están mejor distribuidos, puesto que están próximos a la media. El cálculo de la varianza se muestra en la expresión 5.4, donde r_i^{libres} indica la cantidad de recursos libres del nodo i y $\overline{r^{libres}}$ la media de recursos libres del total de los nodos considerados.

$$S^2 = \frac{1}{nodos} \sum_{i=1}^{nodos} (r_i^{libres} - \overline{r^{libres}})^2 \tag{5.4}$$

5.3 Selección del orden de movimiento de máquinas

Para establecer el orden de movimientos de MVs contaremos tanto con los datos de las MVs, como con la evaluación del movimiento que se pretende realizar y el coste (según los datos del monitor) que supondría dicho movimiento. De esta forma, podremos utilizar todos estos datos como parte de la heurística. Algunos ejemplos de heurísticas serán los siguientes:

Además, es importante tener en cuenta que se pueden establecer criterios principales de ordenación y criterios secundarios, que se podrán utilizar en caso de equivalencia con respecto al primero.

- **Primero los movimientos más valiosos.** Este criterio deja todo el trabajo en la heurística de ”evaluación de movimiento de MV” y simplemente utilizará el valor calculado para establecer una ordenación creciente de los movimientos a realizar, dando más preferencia a los más valiosos.

  Esta heurística es muy versátil, porque podemos adaptar la evaluación de los movimientos a los datos de la propia máquina. Por ejemplo, si queremos

mover primero las más pequeñas, podemos utilizar una valoración de movimientos relacionada con el tamaño de la máquina (como hemos visto en la sección 5.2) y establecer una ordenación decreciente.

- **Primero los movimientos de menor coste.** Este es otro ejemplo de ordenación que puede utilizarse. Si asumimos que el coste de una migración es principalmente temporal, con este criterio se trataría de realizar en primer lugar aquellas migraciones más rápidas. De este modo estamos intentando prever la situación de que ocurra alguna asignación de MV al nodo que se está tratando de vaciar. Esto invalidaría el esfuerzo realizado con las migraciones (puesto que el nodo pasaría a estar inestable), y al haber hecho primero aquellos movimientos menos costosos, desperdiciaríamos el menor trabajo posible.

  Resulta evidente que, en este caso, no estaríamos utilizando para nada la evaluación de los movimientos de MVs, con lo que lo más recomendable sería no realizar ningún tipo de valoración. Una alternativa consistiría en utilizar la evaluación del movimiento de la MV como criterio secundario en la ordenación, para casos en los que el coste de la migración sea el mismo.

- **Primero los que mejor ratio beneficio por coste obtengan.** En este caso se pretende combinar los criterios anteriores, calculando la cantidad de beneficio que proporciona un movimiento por unidad de coste. Un ejemplo intuitivo sería el de disponer del tamaño de la máquina a mover como valor de evaluación y el coste en segundos. Al utilizar esta heurística estaríamos calculando la cantidad de recursos movidos por unidad de tiempo.

- **Primero los de las máquinas que están más tiempo estables.** En este caso lo que pretendemos es evitar, en la medida de lo posible, el movimiento continuado de las MVs. Para ello, elegiremos en primer lugar aquel movimiento que, independientemente del coste y de la evaluación, corresponda a la máquina que más tiempo lleva en un estado estable (en el mismo nodo, funcionando). En realidad este mecanismo se podría implementar fácilmente mediante la evaluación del movimiento, pero se ha diferenciado para ilustrar que es posible utilizar otro tipo de criterios menos convencionales.

5.4 Selección de la lista de movimientos

La selección de la lista de movimientos a realizar se necesita únicamente en el caso del algoritmo mejorado, mostrado en el listado 4.6. En ese caso, dispondremos de una lista de listas de movimientos, y para cada movimiento tendremos tanto los datos de la MV y los nodos involucrados, como la evaluación y el coste del mismo.

Al igual que ocurre en el caso de la sección 5.3, será conveniente tomar un criterio principal y otro secundario para resolver en caso de empate.

A partir de estos datos, podemos utilizar diferentes heurísticas. Algunos ejemplos son los siguientes:

- **Primero la más corta.** Esta heurística es muy sencilla y simplemente se basa en elegir en primer lugar la secuencia de movimientos que cuenta con un menor número de migraciones. Con esto lo que estamos haciendo es tratar de reducir el número de operaciones de migración en total.

- **Primero la más barata.** Este mecanismo trata de elegir la secuencia de movimiento en función del coste de los mismos. Si consideramos que el coste es proporcional al tiempo que cuesta migrar la máquina, aplicando este criterio primaremos las soluciones más rápidas. Esto puede ser adecuado para minimizar las posibilidades de interferencia en el análisis (por ejemplo, la posible asignación de nuevas máquinas al nodo seleccionado).

- **Primero la más valiosa.** Esta es una heurística equivalente a la anterior, pero considerando el valor de los movimientos. En este caso, lo que tratamos de hacer es primar aquellos movimientos que reportan un mayor beneficio a la plataforma.

- **Primero la que mejor ratio valor/coste produce.** Este criterio no es más que la combinación de los dos anteriores, estableciendo un ratio entre el valor aportado por el movimiento y el coste que supone para la plataforma. Así se trata de calcular el beneficio por unidad de coste que reporta la secuencia de movimientos y, con ello, primar la secuencia económicamente (o temporalmente) más eficiente.

- **Otras heurísticas.** Además de los ejemplos detallados, podríamos implementar infinidad de criterios, que pueden ir desde seleccionar la lista de las máquinas más pequeñas a la lista de máquinas que deja los recursos libres de los nodos de una forma mejor distribuida (utilizando la varianza de los recursos libres vista en la seccion 5.2), pasando por la selección de forma aleatoria.

CAPITULO

6

Puesta en producción y pruebas de funcionamiento

En los capítulos anteriores hemos visto el diseño de un agente de consolidación de MVs, que hemos denominado VMCA. Sin embargo, hasta el momento lo hemos visto desde un punto de vista eminentemente teórico. En este trabajo se propuso el objetivo de realizar una implementación práctica, en forma de producto que recogiese el trabajo de investigación y que pudiese ponerse en funcionamiento.

VMCA se ha diseñado como un agente que se ejecuta cada cierto tiempo para llevar a cabo la monitorización y análisis de la plataforma y, eventualmente, ordenar la migración de las MVs. De forma general, la monitorización del sistema recaerá en la propia plataforma, y VMCA extraerá la información de ella mediante los conectores, que serán los que se encargarán de transformarla al formato del agente.

En este capítulo vamos a ver distintas adaptaciones y asunciones que se han tenido que hacer en distintos aspectos, para conseguir el prototipo planteado. Además dedicaremos una sección para mostrar algunos de los resultados en cuanto a tiempos de ejecución, prestaciones y comparativas de heurísticas, que se han obtenido en las pruebas de funcionamiento.

6.1 Adaptaciones para la puesta en producción

En primer lugar vamos a revisar tanto las adaptaciones que se han tenido que realizar para conseguir un prototipo que pudiera ser utilizado en un sistema en producción, como algunas de las decisiones que se han tomado para poder conseguir una implementación del modelo teórico.

6.1.1 El algoritmo principal

Una de las consecuencias del funcionamiento de VMCA es la migración de MVs de un nodo real a otro. La migración que se considera en la actualidad puede ser de dos tipos:

- **Migración offline (fuera de línea).** Este tipo de migración es la más sencilla y consiste básicamente en (1) detener la emulación de la MV, (2) realizar una instantánea del disco y un volcado de la memoria a disco, (3) copiar los ficheros de la MV (principalmente disco e instantánea de memoria) al nodo de destino, y (4) utilizar los ficheros copiados para reanudar el funcionamiento de la MV en el estado exacto en que se encontraba.

- **Migración online (en línea).** El caso de la migración *online* es mucho más complejo, y requiere (en la práctica totalidad de los casos) de un sistema de ficheros compartido entre los nodos origen y destino, y que los discos de la MV se encuentren alojados en dicho sistema de ficheros. A partir de este requisito, lo que ocurre (conceptualmente) es que los hipervisores de origen y destino intercambian la memoria y el estado de la MV que se encuentra en funcionamiento, hasta que se llega un momento en que la información es idéntica en ambos nodos. En ese momento, se desactiva la MV en el nodo origen y se queda activada en el nodo de destino, y la migración se da por completada.

La migración *online* se llama así porque la MV se encuentra funcionando en todo momento y en la práctica se consigue que esté dando respuesta en todo momento al usuario y, por lo tanto, el cambio de nodo real es transparente para el usuario. Sin embargo, en la migración *offline* la MV queda detenida por el tiempo

en que se realiza la copia (por red) de los ficheros que la definen y, en particular, el disco duro. Dado que los discos duros tienen un tamaño apreciable (del orden de Gb.), esta copia por red puede ser lenta aún en sistemas de alta velocidad.

En la sección 4.3 hemos visto que el resultado de dicho algoritmo es una lista de MVs que deben moverse de un nodo a otros. En realidad esto supone un problema con respecto a la migración *offline* ya que el tiempo de copia de los ficheros de las MVs será del orden de minutos. Descartando el tiempo que estará detenida cada MV durante su migración (esto sería inevitable en el caso *offline*), podemos advertir que durante todo ese tiempo de copia podrían haber ocurrido creaciones y eliminaciones de MVs que invalidasen la lista de movimientos que teníamos previstos.

Podríamos pensar que en el caso *online* no tendríamos ese problema, pero los hipervisores tienen un soporte limitado en cuanto al número de migraciones simultáneas de múltiples VMs que compartan nodo de origen o destino[1]. Dado que en nuestro algoritmo todas las MVs parten del mismo nodo, nos encontraríamos con estas posibles limitaciones. Además nos encontraríamos con el problema de congestión de la red a traves de la que deberían copiarse los ficheros y transmitirse las páginas de memoria, que podría llegar a hacer que se tardase incluso más que en el caso de realizar las migraciones de forma secuencial. Suponiendo que pudiéramos indicar al hipervisor la realización de todas las migraciones éstas ocurrirían durante un tiempo casi equivalente al que tardaría de forma secuencial, y en el periodo de tiempo durante el que se realizan la situación de la plataforma podría cambiar.

La alternativa que hemos propuesto en VMCA consiste en forzar a que el resultado de la ejecución del algoritmo sea una única migración de MV. Con ello conseguiremos que, la siguiente vez que se ejecute el algoritmo de análisis, se vuelva a analizar el estado de la plataforma y, con ello, se contemplarían las posibles variaciones que hubieran ocurrido. Además, ante dos ejecuciones consecutivas del algoritmo, si no se han creado o eliminado MVs en la plataforma, el resultado de

[1]La última versión de Hyper-V (2012) indica que no tiene límite de número simultaneo, VMWare vSphere 5 indica que tiene soporte para un máximo de 8 migraciones concurrentes, KVM no especifica esta circunstancia.

la segunda deberá ser ”la continuación” de la secuencia de migraciones proporcionada en la primera. Esto se puede ver de forma intuitiva en la figura 6.1.

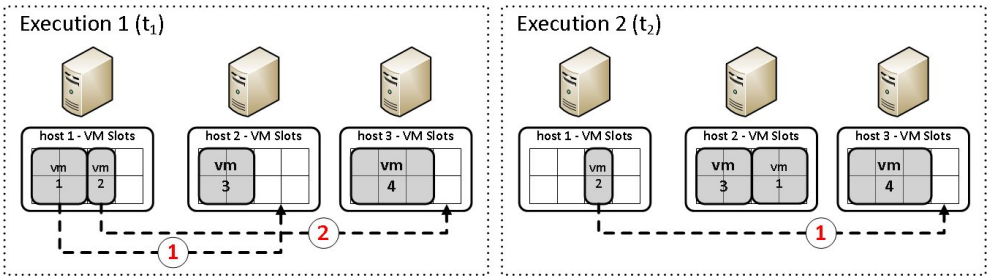

Figura 6.1: Dos ejecuciones consecutivas de VMCA deben continuar la secuencia de migración.

En la figura se muestra que en la primera ejecución (instante t_1), la secuencia de migración es (*vm 1* de *host 1* a *host 2*, *vm 2* de *host 1* a *host 3*) y, como consecuencia de este resultado, VMCA ordenaría el movimiento de la *vm 1*. De acuerdo al principio de determinismo planteado, y dado que no ha ocurrido ninguna creación o eliminación de MV, en el instante t_2, el resultado del algoritmo debería ser la secuencia (*vm 2* de *host 1* a *host 3*).

Al utilizar esta aproximación, el diseño de las heurísticas cobra una mayor relevancia, ya que es necesario que éstas tengan un comportamiento determinista. En la implementación realizada en este trabajo estos aspectos se han tenido en cuenta, pero a pesar de que es una característica deseable, VMCA no impondrá ninguna restricción, ni realizará ningún control sobre la situación. Respetar o no estas características será una decisión que deberán tomar los usuarios que estén interesados en enriquecer las heurísticas de VMCA.

Este mecanismo, sin embargo, introduce un problema que hasta ahora no habíamos tenido y consiste en que, al generar un único movimiento de MV, el vaciado de un nodo por completo puede requerir varias iteraciones. Por ejemplo, en el caso de la figura 6.1, liberar el *host 1* supondría dos iteraciones aún cuando no hubiera ocurrido ninguna creación o eliminación de MV. Como trabajo de continuación para el producto en este aspecto quedaría la mejora de este comportamiento y evitar así reanálisis innecesarios. Para ello bastaría con introducir un sistema de ”encolado” de migraciones que permitiese continuar con la secuencia de migración obtenida, en caso de que el estado del nodo origen y los nodos de destino continuasen

siendo los mismos, y observar el sistema para que se ordenasen las siguientes tan pronto como terminasen las migraciones en curso.

6.1.2 Adaptaciones para usabilidad

Una de las características esenciales para que una aplicación pueda ponerse en producción es que el usuario pueda adaptarla a sus necesidades. En nuestro caso, se ha tenido acceso a un despliegue a un despliegue de plataforma de virtualización que se encuentra en producción y, por tanto, se han podido captar algunas de las peculiaridades que se tienen que tener en cuenta en VMCA:

- **Márgenes de recursos de los nodos y mantenimiento de QoS.** Desde VMCA se ha asumido en todo momento que una MV que está alojada por un nodo, en realidad está haciendo una *reserva* de los recursos que solicita. En caso de que un nodo tuviera reservados todos sus *cores* por MVs y éstas estuviesen trabajando al 100 %, podrían hacer que la máquina real dejara de responder. Ocurriría algo similar en el caso de la memoria RAM reservada, o incluso más grave en el caso del disco (porque podría provocar que hubiese pérdida de información).

  Para estos casos, se han introducido unas variables de configuración que permiten considerar unos márgenes de holgura para los recursos de los nodos de virtualización. En concreto, se considera unos valores de memoria, cores y disco que deberán quedar sin reservar tras mover una MV al nodo. En el desarrollo, estos valores se denominan *SPARE_MEMORY*, *SPARE_CPU* y *SPARE_DISK*, y deberán estar expresados en las mismas unidades que utiliza el conector de la plataforma para el sistema de información.

- **Número de MVs máximo por nodo.** Este es un caso particular de margen de recursos y es relativo al número de MVs alojadas en un nodo. En ocasiones puede ser interesante establecer la densidad máxima de VMs por máquina real, por ejemplo, para limitar los problemas derivados del fallo del propio nodo de virtualización. En el desarrollo actual se puede establecer el número máximo de VMs por nodo mediante la variable de configuración *VM_MAX*.

- **Nodos deshabilitados.** Por distintos motivos puede ser interesante no mover las máquinas que se encuentran en algunos nodos. Un ejemplo puede ser porque hay un equipo que nunca vaya a ser apagado aún cuando esté libre. En ese caso, puede interesar que el nodo sea destino de movimientos de MVs, pero que no sea necesario vaciarlo aún cuando las circunstancias hagan que sea candidato para ello. En el desarrollo se ha creado una variable de configuración con el nombre *DISABLED_HOSTS* que contiene una lista con los nombres de los nodos deshabilitados.

- **Frecuencia de análisis.** Es necesario establecer la frecuencia con que se debe ejecutar el análisis ya que determinará la carga del sistema. Además, tal como ha quedado el algoritmo (un único movimiento por ejecución), el tiempo entre análisis será crucial para el tiempo de vaciado por completo de un nodo. Para este valor se ha utilizado la variable con nombre *SECONDS_BETWEEN_CHECKS*.

- **Tiempo de estabilidad de una MV.** Durante el texto anterior hemos establecido que para considerar que una MV puede ser movida, debe estar en el mismo nodo, en el mismo estado, durante un tiempo establecido. El tiempo para que una MV sea considerada estable se podrá controlar mediante la variable de configuración *STALL_THRESHOLD*.

- **Control de exclusión de MVs.** Además de los nodos en general, puede ser interesante evitar que ciertas MVs se muevan de los nodos en los que están siendo alojadas. Esto puede ser interesante por distintos motivos, como que haya una dependencia de un dispositivo real que se encuentra en el nodo en que ha sido desplegada[2]. Para controlar qué MVs no van a ser consideradas para ser movidas, se utiliza una aproximación simple y poco intrusiva, que consiste en que las que se encuentren en esta circunstancia nunca sean consideradas estables. Para que el usuario pueda establecer las MVs que se encuentran en esta circunstancia se han introducido dos mecanismos: (1) un

[2]En teoría no es deseable que una MV tenga una dependencia de un dispositivo real, pero en la práctica los hipervisores permiten a las MVs el acceso a dispositivos reales de forma exclusiva (por ejemplo, un dispositivo USB o un disco físico).

control de exclusión por identificador de la MV y (2) un control por *palabras clave*.

El primero de los mecanismos es muy intuitivo, y consiste en que las MVs que tengan determinado identificado en la plataforma no pueden ser movidas. Esta lista se controla mediante la variable de configuración *EXCLUDED_VM_IDS*, que consiste en una lista de identificadores dependiente de la plataforma. Para el segundo caso, se ha enriquecido la implementación del sistema de información relativo a las MVs, mediante el campo "KEYWORDS". En este campo se pueden, para cada MV, incluir una lista de palabras clave que especifiquen algunas características más allá de los recursos que utiliza, y que son dependientes de la plataforma. Posteriormente, la lista de exclusión se controla mediante la variable de configuración *EXCLUDED_KEYWORDS* y la semántica consiste en que si *alguna* de las palabras clave de esta lista aparece en el campo *KEYWORDS* de la MV, no se podrá mover. Algunos ejemplos de *keywords* para el caso de ONE pueden ser el nombre de la plantilla (*template*) en que está basada, o el nombre del usuario que la ha puesto en funcionamiento.

6.1.3 Conexión con OpenNebula

La conexión con ONE se realiza mediante una clase que implementa los métodos indicados en la sección 4.1. Para interactuar con el servidor que gestiona la plataforma, se utiliza el API para XML-RPC que ofrece este sistema, y por lo tanto, este conector incorpora sus propias variables de configuración (*ONE_XML_RPC* y *ONE_AUTH_FILENAME*), que serán necesarias para establecer la conexión.

ONE dispone de un sistema de monitorización de la plataforma, que puede ser consultado mediante el interfaz XML-RPC por un usuario con los privilegios adecuados (cuyas credenciales estarán establecidas en el fichero indicado en la variable ONE_AUTH_FILENAME). Además de otros aspectos de la plataforma, ONE proporciona información acerca tanto de los nodos de virtualización como de las MVs. La información que se puede obtener es más que suficiente para poder crear las estructuras de datos que necesita VMCA de forma adecuada. Sin embargo nos encontramos con una serie de peculiaridades:

- **ONE proporciona los datos en XML** y, por tanto, ha sido necesario crear una serie de clases, junto con un *parser* para interpretar los datos y extraer los necesarios para VMCA.

- **ONE utiliza los conceptos de CPU y VCPU para las MVs.** En VMCA se ha supuesto que la asignación de una MV a un nodo se acaba plasmando en una ”reserva” de recursos. En realidad son conceptos distintos puesto que CPU en ONE es el porcentaje de CPU real que se va a destinar a la MV, mientras que VCPU son los *cores* con que contará. El problema viene porque en realidad el valor de CPU no siempre aparece en la configuración, debido a que ONE solo proporciona la información con la que fue creada la MV y no con la que fue instanciada. Aquí se ha optado por, en los casos en que no aparezca, utilizar el valor VCPU a efectos de reserva de recursos, si bien sería conveniente realizar pruebas más exhaustivas para averiguar la relación entre CPU y VCPU en ONE en caso de ausencia de alguno de estos datos.

- **ONE dispone de plantillas (*templates*) para definir las MVs** y en ellas se indican aspectos específicos de las MVs. En particular, a nuestros efectos nos interesa el hecho de que ahí se indican condiciones de despliegue de las máquinas y, en particular, si se deben poner en algún nodo en concreto o si tienen restricciones en cuanto a características de los mismos. Esto podría condicionar el hecho de la eventual migración de la MV y se ha optado por, para evitar tener que desarrollar sistemas mucho más complejos como podría ser el tener que interpretar dichas restricciones, indicar el nombre de la plantilla como palabra clave (*keyword*) de la MV. De esta forma, en caso necesario, podremos evitar que se cambie de nodo mediante los mecanismos al efecto, descritos anteriormente.

- **ONE hace una asignación de las MVs a usuarios** y, por un motivo análogo al anterior, se ha optado por incluir el nombre de usuario del propietario de la MV dentro de las palabras clave de la estructura de datos. Con ello podremos utilizar el mismo mecanismo de evitar las migraciones (si hiciese falta), basado en las palabras clave.

- **ONE maneja numerosos estados para las MVs**, mientras que VMCA únicamente contempla un conjunto reducido. Lo que se ha hecho es hacer corresponder todos los estados en que ONE acepta que una MV pueda ser migrada a otro nodo, con el estado ”RUNNING” del modelo de datos de VMCA, mientras que el resto – salvo ”STOP” y ”UNKNOWN” de ONE, que se han hecho coincidir con los homólogos de VMCA – se han asemejado a ”OTHER” del modelo VMCA para evitar que puedan ser movidas.

- **La monitorización de los nodos en ONE es inestable** y en ocasiones puede indicar que están en un estado de fallo cuando en realidad no es así. Para poder corregir esta situación lo que se ha hecho es, a nivel ONE, mantener un histórico simple del estado de los nodos de forma que únicamente reporte a VMCA un fallo del mismo después de dos monitorizaciones en las que el nodo esté fuera de línea. En realidad, el hecho de que se intente migrar una MV a un nodo que se encuentre *offline* no supone un problema puesto que, a efectos prácticos, a pesar de que se le ordene la migración, ONE no realizará el movimiento de la máquina y en la siguiente monitorización seguirá apareciendo en la máquina real correspondiente.

6.1.4 Histórico de la plataforma

Para implementar el sistema de histórico de la plataforma se ha asociado a cada monitorización del sistema una marca de tiempo (*timestamp*) correspondiente al instante en que fue obtenida la información. Posteriormente, los datos obtenidos son cotejados con los que ya estaban disponibles en el sistema y, en caso de que los datos de una MV no hubieran cambiado (en cuanto a estado de funcionamiento de la MV y el nodo de virtualización asignado), se tomará la menor marca temporal entre la anterior y la de la última monitorización. Actuando así conseguimos implementar el concepto de *marca temporal* para el estado de la MV. Una vez se han procesado los datos de monitorización, se utiliza un sistema persistente (concretamente *sqlite*[3]) para almacenar la nueva información.

[3]http://www.sqlite.org

6.1.5 Otros aspectos

El código ha sido desarrollado en lenguaje *Python* y se han tratado de aprovechar las características del mismo para conseguir un código muy legible, a la par que eficiente. En este sentido, se han creado clases básicas para las heurísticas, que se pueden componer (mediante el mecanismo de *mix-ins*, a modo de interfaces) con las clases correspondientes al análisis de la plataforma para conseguir así el comportamiento esperado.

En aras de la eficiencia (con el propósito de minimizar el tiempo de análisis en despliegues grandes), también se ha hecho una serie de simplificaciones en el código. Por ejemplo, hemos utilizado el concepto del *espacio de los recursos* para tener una forma sencilla para comparar los recursos libres. Para ello se utiliza la fórmula 5.2, pero a efectos de implementación, dado que no se necesita el valor exacto (puesto que sólo se utiliza para comparar), se ha utilizado una versión simplificada con pesos decimales y sin utilizar la raíz cuadrada.

Otro ejemplo de simplificación es el caso de que en el algoritmo principal (visto en 4.3.1) y en el análisis de los posibles movimientos (visto en 4.3.2) se ordenan los movimientos de las MVs con el simple propósito de obtener el elemento colocado en mejor posición (*ordenar_movimientos(posibles_movimientos)*). En lugar de utilizar un algoritmo más costoso, se ha seguido la aproximación de averiguar el valor máximo (o mínimo, según el caso), sin tener que llegar a realizar la ordenación completa (ya que queda descartada).

6.2 Pruebas de funcionamiento

En este apartado se muestran los resultados obtenidos de algunas pruebas de funcionamiento del sistema. Para ello, se ha desarrollado una *plataforma ficticia* que nos permite hacer simulaciones de cómo se comportaría un despliegue real sin tener que acudir a él. Además, nos permite generar casos de uso y resolverlos utilizando distintas heurísticas para poder comprobar las diferencias de resultados que obtenemos entre ellas.

El análisis de la aplicación de las posibles heurísticas para distintos casos de uso es complejo, debido a que existen numerosas combinaciones. Si combinamos

las heurísticas propuestas en el capítulo 5, podemos obtener 72 posibles heurísticas compuestas. Además, cada caso de uso tiene características singulares, con lo que necesitamos realizar múltiples pruebas para cada posible combinación.

VMCA tiene dos facetas muy distintas: por un lado la cualidad de liberación de nodos de MVs (para poder apagarlos), y por otro la redistribución de las MVs entre los nodos, para defragmentar los recursos de virtualización disponibles y poder atender a peticiones específicas. Para el primer caso resulta evidente que un número elevado de nodos apagados sería bueno para considerar que VMCA ha hecho un buen trabajo. Sin embargo, para el caso de la defragmentación de los recursos, podríamos querer evaluar lo bien distribuidos que estan (buscando una varianza de los recursos baja). Pero también podríamos, ya que tenemos nodos encendidos, querer tener los recursos distribuidos de forma desigual para atender a diversos tipos de solicitudes de alojamiento de MVs.

Este trabajo lo hemos orientado hacia la vertiente de ahorro energético más que en la defragmentación de recursos. Así, nos hemos centrado en utilizar como elemento de comparación principal el *número de nodos que se consiguen liberar* como resultado de la aplicación de cada combinación de heurísticas. Para ello, hemos generado múltiples casos de uso y, suponiendo que todos los nodos se encontraban en una situación estable, hemos aplicado las distintas combinaciones de heurísticas hasta que no se han podido liberar más nodos. De esta forma, hemos obtenido, para cada sistema de heurísticas (elección de nodo a vaciar en primer lugar, valoración de movimiento y elección de nodo de destino), el número de máquinas reales que se podrían llegar a apagar.

Cada caso de uso tiene características particulares y ello supone que, en función de las MVs y de los nodos reales, será posible apagar un número de nodos como máximo (suponiendo que todas las MVs quedan perfectamente alojadas en los nodos). Por lo tanto, no podemos comparar los valores en valor absoluto.

En su lugar, hemos tratado de averiguar el valor máximo teórico de nodos que podríamos apagar y, para ello, hemos utilizado la propuesta 6.1 hecha en [20], que permite calcular mediante la expresión 6.2 una cota del número óptimo de nodos ($OPT(L)$) con el que podríamos conseguir alojar las MVs, utilizando un algoritmo de FFD.

$$FFD(L) \leq \frac{11}{9} \cdot OPT(L) + 1 \tag{6.1}$$

$$FFD(L) - 1 \cdot \frac{9}{11} \leq OPT(L) \tag{6.2}$$

Así que hemos implementado el algoritmo FFD, ordenando de forma decreciente los nodos, en función de la utilización de sus recursos, y hemos ido colocando las MVs. Este algoritmo supone que no existe una distribución inicial y, por tanto, puede alojar cada MV en el nodo que quiera de la plataforma. Sin embargo, de esta forma podemos obtener una estimación el número máximo de nodos que podríamos conseguir liberar ($\hat{N}_i^{libres}$), para cada caso de uso i.

Una vez disponemos de esta cota superior, para cada caso i y cada heurística h, hemos calculado para cada número de nodos que han quedado libres (n_i^h) tras la aplicación del algoritmo. A partir de estos valores, hemos calculado la relación entre el resultado obtenido y el óptimo de acuerdo a la expresión 6.3.

$$B_i^h = \frac{\hat{N}_i^{libres}}{n_i^h} \tag{6.3}$$

De esta forma hemos conseguido obtener un indicador de lo buena que es una solución (de acuerdo al criterio de apagado de nodos), que podemos comprar entre los distintos casos de uso.

Una vez tenemos este indicador, hemos generado una serie de plataformas sintéticas, de forma aleatoria y una batería de casos de uso (también sintéticos y aleatorios) para resolver. Para las pruebas de este trabajo hemos supuesto una cantidad de disco holgada para albergar las MVs. Por lo tanto, hemos prescindido del recurso de almacenamiento en disco, y nos vamos a centrar únicamente en la cantidad de memoria y CPU que solicitan las MVs.

Las plataformas de pruebas van a estar formadas por una serie de nodos de virtualización heterogéneos, con las siguientes caraterísticas de memoria y CPU:

- **Grande:** 16 Gb. de RAM, 8 Cores.
- **Mediano:** 8 Gb. de RAM, 4 Cores.
- **Pequeño:** 4 Gb. de RAM, 2 Cores.

Los casos de uso se van a complementar con una serie de MVs alojadas en la plataforma, a partir de las siguientes tipologías:

- **Muy Grande:** 8 Gb. de RAM, 4 Cores, 100 Gb. HD.
- **Grande:** 4 Gb. de RAM, 2 Cores, 50 Gb. HD.
- **Mediana:** 2 Gb. de RAM, 2 Cores, 20 Gb. HD.
- **Pequeña:** 1 Gb. de RAM, 1 Core, 10 Gb. HD.
- **Diminuta:** 512 Mb. de RAM, 1 Core, 10 Gb. HD.

Para las pruebas hemos creado una serie de plataformas sintéticas, de 10 y 50 nodos, y para cada una de ellas hemos creado algunos tipos de casos de uso que constan de distintas cantidades de MVs. Para cada tipo de caso de uso hemos generado 100 casos concretos, que constan de MVs al azar de entre los tipos indicados. La matriz de pruebas que hemos creado se muestra en la tabla 6.1.

	número de MVs									
nodos plataforma	5	10	20	30	40	50	100	150	200	250
10	•	•	•	•	•					
50						•	•	•	•	•

Tabla 6.1: Matriz que de pruebas que se han realizado.

Cada punto de la matriz indica que se han lanzado 100 casos de test distintos que constan del número de MVs indicado en la cabecera de la columna, en una plataforma con el número de nodos indicados a la izquierda. Por ejemplo, se han lanzado 100 casos de prueba de 5 MVs en una plataforma de 10 nodos. Para cada caso de prueba hemos obtenido el indicador de calidad de la solución, calculando el valor B_i^h. Durante el desarrollo del trabajo se han realizado muchas más pruebas que constaban de distintos número de MVs. Sin embargo, aquí se muestran los casos de uso cuyos tamaños han proporcionados resultados más significativos puesto que el hecho de llegar a saturar la plataforma de MVs provoca que no sea posible liberar nodos y que, por tanto, el análisis pierda relevancia.

Para cada combinación de plataforma y número de MVs contamos con 72 posibles heurísticas (resultado de la combinación de las heurísticas propuestas en el

capítulo 5 y se ha resuelto cada caso de uso con todas ellas. De forma que disponemos de 7200 indicadores por escenario.

Dado que tenemos una cantidad elevada de heurísticas, lo que se ha hecho es analizar la influencia de cada tipo de heurísticas (selección del nodo a vaciar, valoración del movimiento y elección del nodo de destino) en la calidad de la solución final obtenida. El indicador de calidad es un porcentaje que representa el grado de alcance de la solución óptima para cada caso de uso. Para tratar de ver lo bueno que es el resultado final, vamos a construir una gráfica que nos muestre la concentración de la calidad de la solución obtenida para los casos de uso.

Para ello hemos dividido el espectro del indicador de calidad de la solución en bloques de tamaño 20 % (0 % a 20 %, 20 % a 40 %, etc.) y hemos contabilizado el número de casos de uso cuyos resultados se encuentran dentro de los rangos de cada bloque. Una vez contabilizados todos los casos, hemos obtenido el porcentaje que supone cada bloque con respecto al total de casos resueltos. De esta forma podemos ver el porcentaje de casos que han conseguido una calidad de solución entre el 0 % y el 20 % del valor óptimo, el que ha conseguido entre el 20 % y el 40 %, y así sucesivamente. En la gráfica 6.2 se muestran los porcentajes de casos de uso obtenidos para cada nivel de calidad de la solución, una vez VMCA ha realizado las migraciones que ha considerado. La fila de arriba se corresponde con una plataforma de 10 nodos, en la que se alojaban, de izquierda a derecha, 5, 10, 20, 30 y 40 MVs. La fila de abajo refleja los resultados en una plataforma de 50 nodos en la que se alojaban, de izquierda a derecha, 50, 100, 150, 200 y 250 MVs.

De las gráficas se desprende que cuando la plataforma está casi vacía o casi llena (extremos izquierdo y derecho, respectivamente), VMCA obtiene casi los mismos resultados que se obtendrían en el caso óptimo. Por ejemplo, podemos ver que si tomamos la primera gráfica de la plataforma de 10 nodos, podemos ver que el nivel de calidad 80 % o más se ha alcanzado en el 83 % de los casos, mientras que el nivel de calidad entre el 60 % y el 80 % se ha obtenido para el 17 % de los casos. En los casos intermedios no se obtienen soluciones tan próximas a la óptima, pero aún así se alcanza una concentración razonablemente alta de buenas soluciones (por encima del 50 % de la solución óptima).

Para estudiar la influencia de las distintas heurísticas, hemos seguido la misma metodología, pero en esta ocasión hemos discriminado los valores agrupándolos en

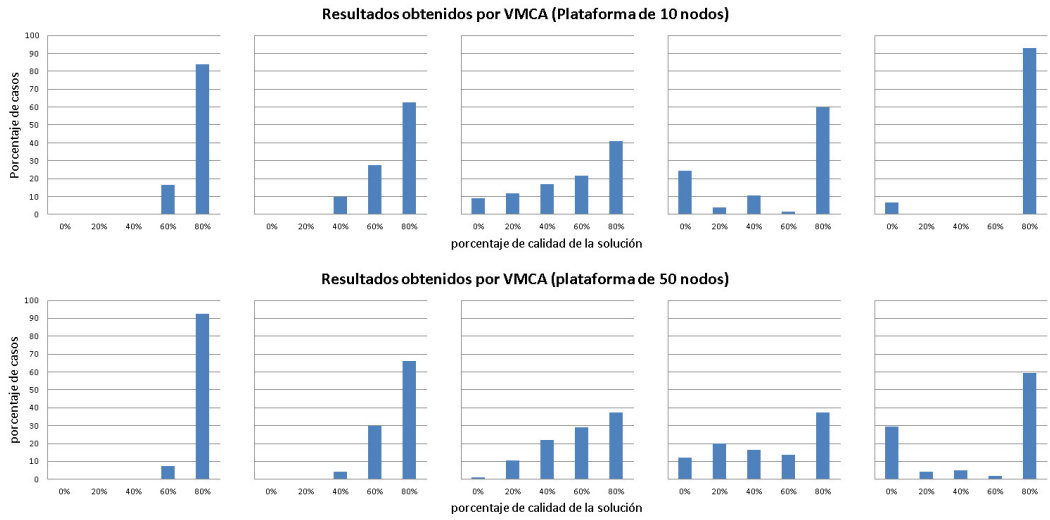

Figura 6.2: Dispersión de bondad de solución para plataforma de 10 y 50 nodos. En la fila de arriba, de izquierda a derecha, casos de 5, 10, 20, 30 y 40 MVs. En la fila de abajo, de izquierda a derecha, casos de 50, 100, 150, 200 y 250 MVs.

función de los criterios concretos de cada heurística. De este modo, conseguimos averiguar el porcentaje de casos que han alcanzado un determinado grado de bondad, en los que se aplicaba determinado criterio. Para poder comparar los criterios entre sí, lo que hemos hecho ha sido representar los resultados obtenidos agrupados por heurísticas. En la figura 6.3 se muestran los resultados para cada una de las heurísticas propuestas, en el caso concreto de la plataforma de 10 nodos.

En realidad, la figura muestra los resultados vistos en la figura 6.2, pero desglosados por cada uno de los criterios correspondientes a cada heurística. Cada fila de gráficas se corresponde con una de las heurísticas que se han propuesto en el capítulo 5, y en cada gráfica en particular se muestran superpuestos los porcentaje de casos de uso (eje vertical) que han obtenido un porcentaje determinado de calidad (eje horizontal), para cada uno de los criterios implementados (series de datos). De esta forma podemos ver la influencia que tienen en la solución las diferentes reglas de cada heurística.

En cada gráfica se puede ver que, cuanto más concentración de casos haya a la derecha, el criterio está proporcionando resultados más cercanos al óptimo y, por tanto, es mejor. Así, podemos apreciar que en todas las filas, los casos extremos

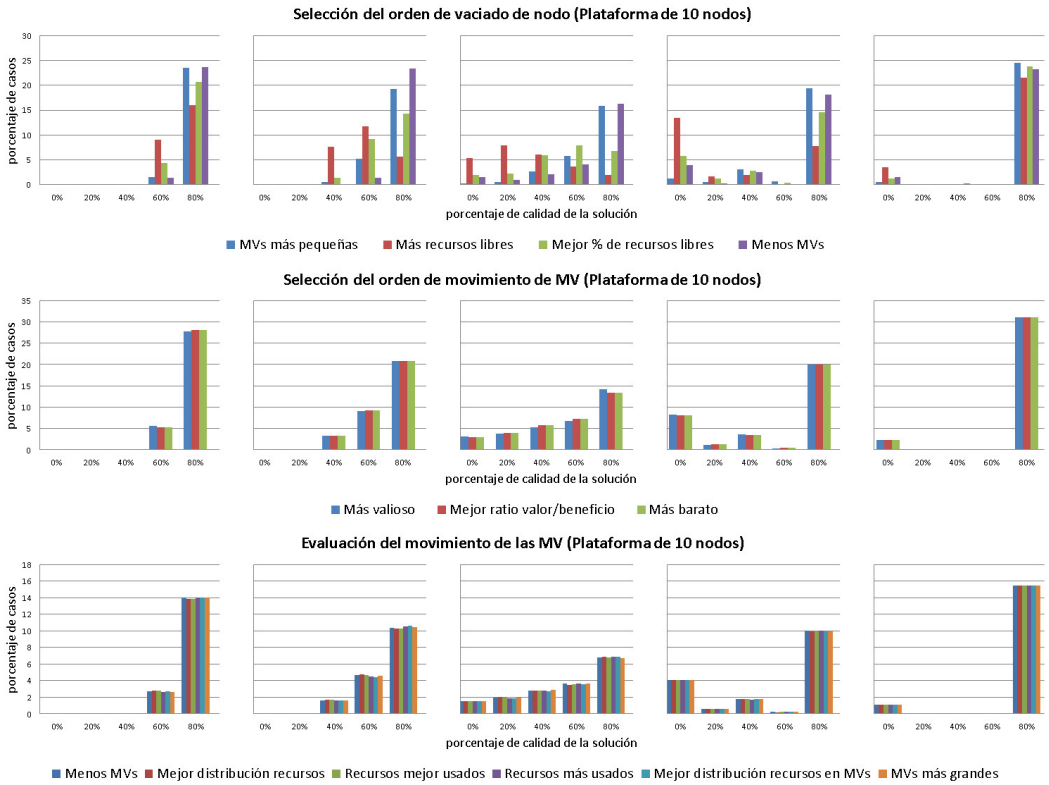

Figura 6.3: Dispersión de bondad de solución para una plataforma de 10 nodos, en función de los tipos de heurísticas. De izquierda a derecha, casos de 5, 10, 20, 30 y 40 MVs

(más a la izquierda y más a la derecha) en los que la plataforma está casi vacía y saturada, respectivamente, las heurísticas consiguen resultados muy similares entre ellas. Cuando la plataforma está casi vacía, casi siempre se consigue vaciar el máximo número de nodos, mientras que cuando está llena, la solución siempre es igual que la óptima (no se consigue vaciar ningún nodo).

En la figura 6.3 podemos ver que, para la plataforma de 10 nodos, tanto para los casos de ”selección del orden de movimiento de MVs” como el de ”evaluación del movimiento de las MVs” (las dos filas inferiores de gráficas) apenas hay diferencias en cuanto a las soluciones conseguidas. Esto es debido a que los distintos criterios obtienen, para cada nivel de calidad de resultados, prácticamente el mismo número de casos. Sin embargo, sí que se aprecian variaciones en el caso de ”selección del orden de vaciado de nodos”. En concreto, podemos ver que cuando la plataforma

dispone de nodos entre los que mover las MVs, los criterios de ”selección del nodo con menos MVs” y ”selección del nodo con MVs más pequeñas” obtienen un mayor número de casos con porcentajes de cercanía a la solución óptima mayores. Sin embargo, los otros criterios tienden a obtener un mayor número de casos de uso resueltos con porcentaje de calidad peor.

Para el caso de la plataforma de 50 nodos se han obtenido unos resultados prácticamente idénticos, y las heurísticas tienen el mismo comportamiento observado para el caso de 10 nodos. Durante la elaboración de este trabajo también se han realizado pruebas con plataformas sintéticas de 100 nodos, pero los resultados también muestran las mismas pautas. Esta similitud entre resultados nos confirma que las heurísticas implementadas tienen un comportamiento muy determinista, tal como pretendíamos.

A la vista de estos resultados podemos ver que los tipos de heurísticas ”selección del orden de movimiento de MVs” y ”evaluación del movimiento de las MVs” apenas tienen influencia por sí mismas en la aplicación del algoritmo diseñado. Sin embargo, tenemos que ver si, dentro de los criterios que sí hemos visto diferencias aportan alguna variabilidad. Para ello hemos fijado la heurística de ”selección del orden de vaciado de nodos” a los criterios a aquellos que habían proporcionado mejores resultados (”selección del nodo con menos MVs” y ”selección del nodo con MVs más pequeñas”) y hemos obtenido las calidades de las soluciones para los casos de uso en los que intervenían de forma simultánea alguno de estos criterios y las heurísticas de ”selección del orden de movimiento de MVs” y ”evaluación del movimiento de las MVs”. De nuevo hemos seguido la misma metodología para analizar los resultados, podemos respresentar las series de datos correspondientes a cada uno de los criterios específicos. Los resultados, para las plataforma de 50 nodos, se muestra en la figura 6.4.

En las dos series de gráficas de la parte superior se muestra la dispersión de la bondad de las heurísticas que no parecían influir por sí mismas, fijando el criterio de ”selección del orden de vaciado de nodos” a ”Menos MVs”, mientras que las dos de la parte inferior se corresponden con el de ”MVs más pequeñas”. Las gráficas nos muestran que estas heurísticas tampoco parecen tener apenas influencia en los resultados obtenidos.

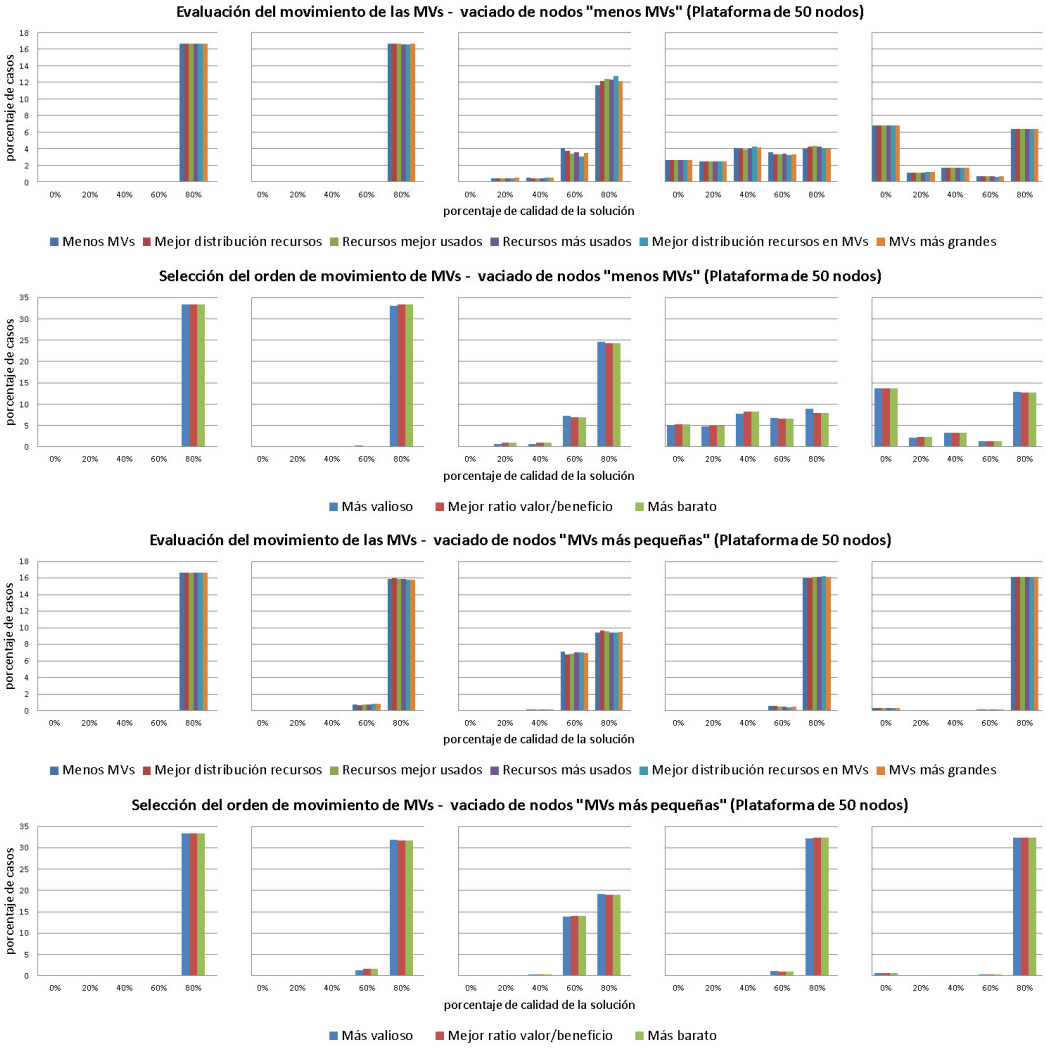

Figura 6.4: Dispersión de bondad de solución para una plataforma de 50 nodos, en función de los tipos de heurísticas, con el orden de vaciado de nodos fijado. De izquierda a derecha, casos de 50, 100, 150, 200 y 250 MVs

Esto puede ser porque se han realizado implementaciones de heurísticas generalistas y además se han adaptado para una plataforma sintética. En esta plataforma de pruebas, se han asociado los costes al tamaño de disco de las MVs, ya que no se ha estimado que en una plataforma real el tiempo de transferencia de los ficheros que componen dichos discos tendrá gran peso sobre el coste de migración. En caso de que tuviéramos modelos de costes temporales de transferencia más fidedignos

o que pudieran introducir la variabilidad de una plataforma real, es posible que la ”selección del orden de movimientos de MVs” tuviera más peso en el resultado final, ya que las heurísticas propuestas están asociadas dichos costes.

Finalmente nos queda por comparar el caso del algoritmo mejorado, que introduce la posibilidad de disponer de distintas posibles secuencias de migraciones de MVs, y elegir entre ellas la que se estime que es la más adecuada. Para ello vamos a seguir la misma metodología, intentando tratar de ver la influencia que tiene la selección de la secuencia en la solución final. La introducción de esta heurística incorpora una variabilidad mayor, puesto que se han propuesto 3 criterios al respecto. Por lo tanto, esto supone 216 posibles combinaciones de heurísticas y 21600 indicadores en total. Los resultados obtenidos por VMCA para la plataforma de 10 nodos se pueden ver en la gráfica de la figura 6.5.

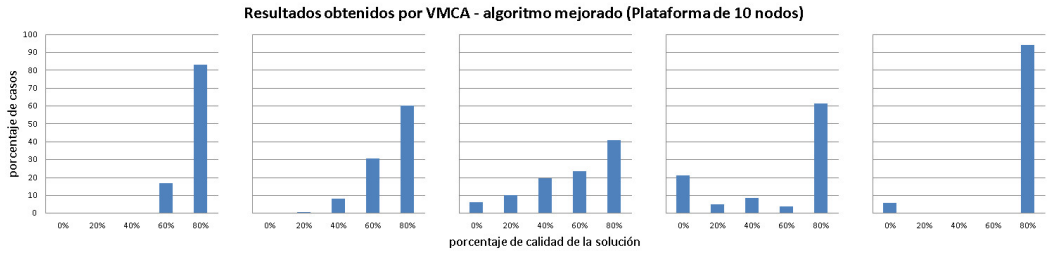

Figura 6.5: Dispersión de bondad de solución para plataforma de 10 nodos, utilizando el algoritmo mejorado. De izquierda a derecha, casos de 5, 10, 20, 30 y 40 MVs.

En este caso también hemos hecho un análisis de la influencia de las distintas heurísticas en la actividad de VMCA. En las gráficas de la figura 6.6 podemos ver los resultados obtenidos para una plataforma de 10 nodos, utilizando el algoritmo mejorado y siguiendo la misma metodología de análisis que en los casos anteriores. Al igual que el caso sencillo, para el caso de la plataforma de 50 nodos obtenemos unos datos similares, tanto para el caso de la actividad de VMCA en sí como en el análisis de influencia de las heurísticas. Por ello no se ha considerado necesario mostrar las gráficas obtenidas.

Como se puede observar, la heurística de ”selección de la lista de movimientos”, específica en el algoritmo de análisis más completo, no parece tener mucha influencia en los resultados finales obtenidos. Esto puede ser debido a que, en cierto modo, los principios de estos criterios ya se encuentran contemplados dentro de

Figura 6.6: Dispersión de bondad de solución para una plataforma de 10 nodos, en función de los tipos de heurísticas, utilizando el algoritmo mejorado. De izquierda a derecha, casos de 5, 10, 20, 30 y 40 MVs

la heurística de ”selección del orden de movimiento de las MVs” y, como hemos visto anteriormente, esta última no contaba con mucho peso en el resultado final.

También se ha comprobado si la influencia de las heurísticas estudiadas anteriormente varía al utilizar este nuevo algoritmo. En las tres filas inferiores de la figura 6.6 se puede ver que los resultados obtenidos son muy similares al caso del algoritmo simple, aún siendo casos de uso completamente distintos. Esto parece

indicar que aún introduciendo este nuevo criterio, la heurística de mayor peso continua siendo la de ”selección del orden de vaciado de los nodos”.

En este apartado hemos hecho pruebas de funcionamiento de VMCA sobre plataformas sintéticas formas por 10 y por 50 nodos. En estas pruebas hemos tratado de averiguar la influencia que tienen heurísticas propuestas en el capítulo 5 en los resultados de los algoritmos teóricos planteados en 4.3. Los resultados obtenidos consisten en una distribución de MVs en nodos reales que es consecuencia de los movimientos de MVs que resultan del proceso de VMCA. Para poder comparar los resultados, nos hemos fijado en los nodos que quedarían libres de MVs y, por tanto, listos para ser apagados o puestos en modo de ahorro de energía. De los resultados obtenidos podemos concluir que la heurística de ”selección del orden de vaciado de nodos” es la que tiene un peso más determinante para conseguir el mayor número de nodos reales apagados, mientras que las otras apenas tienen incidencia puesto que proporcionan resultados equivalentes. En cualquier caso, hay que tener en cuenta que esto no quiere decir que las heurísticas no sean útiles, sino que los criterios que se han comparado proporcionan resultados prácticamente iguales. Además hemos podido comprobar que esto es así tanto en el caso del algoritmo simple, como en el caso del algoritmo más complejo.

CAPITULO

7

Conclusiones y trabajos futuros

En este trabajo hemos diseñado un sistema para la defragmentación de los recursos disponibles en plataformas de virtualización, así como para la consolidación de MVs con el objetivo de liberar nodos reales para que puedan ser apagados, con la intención de ahorrar energía y poder disponer de espacio para atender peticiones de alojamiento de MVs que en otro caso podrían no haber sido satisfechas.

El problema ha sido estudiado en otros trabajos, siguiendo distintas aproximaciones, pero aquí nos hemos decantado por utilizar una solución basada en la solución al problema bien conocido de *bin packing*. Este problema tiene una solución relativamente sencilla de implementar, basada en un algoritmo de FFD, pero no es directamente aplicable porque si bien dicho algoritmo podría proporcionar una buena solución, en la práctica supondría tener que parar todas las MVs, reorganizarlas y, posteriormente, ponerlas de nuevo en funcionamiento.

El diseño aquí planteado tiene en cuenta que la plataforma se encuentra en una situación concreta, y asume que tiene que llegar a la mejor solución posible, pero realizando movimientos de MVs. Esto supone una complejidad adicional porque en un momento dado, aunque existiese una solución mejor, es posible que no pudiera

llegarse a conseguir porque no se pudiera averiguar la lista de movimientos que conducen a ella.

Se han propuesto dos algoritmos: uno más sencillo y otro un poco más costoso, pero que realiza un análisis más exhaustivo. Sin embargo, hemos visto que con el algoritmo menos costoso podemos llegar a soluciones similares a las del método de más complejidad.

El sistema propuesto se rige por el hecho de que se tienen que tomar decisiones que conduzcan a la solución correcta, sin saber con certeza si se se conseguirá. Aquí se han propuesto una serie de heurísticas que permiten tomar dichas decisiones manteniendo criterios basados en la experiencia y en las características de la plataforma sobre la que se va a actuar.

A lo largo del trabajo se han descrito estas heurísticas y se ha comprobado la influencia de ellas sobre los resultados que se consiguen tras la aplicación del sistema, comparándolo con los resultados que se obtendrían si se pudiera utilizar el algoritmo FFD que nos llegaría a dar la mejor solución. Hemos visto que algunas de estas heurísticas si que determinan completamente la calidad de la solución, mientras que otras apenas tienen influencia sobre la misma.

Las baterías de pruebas realizadas se han hecho sobre plataformas sintéticas, que no tienen en cuenta aspectos específicos como puede ser el hecho de que haya MVs que no puedan ser movidas de un nodo a otro, que existan restricciones de migraciones entre nodos concretos, etc. Por lo tanto, aún habiendo hecho un estudio bastante elaborado, habrá que tener en cuenta las características concretas de las plataformas en que se utilicen para decidir las heurísticas más adecuadas en cada caso. Además, también sería importante estudiar la influencia de las características de las MVs que se vayan a desplegar (tamaño de memoria, de disco y número CPUs) y la incidencia que tendrían las combinaciones de criterios para la migración en conjunción con ellas. En el presente estudio se han utilizado MVs con características genéricas, pero responden al caso de uso de la plataforma ONE a la que se tenía acceso y no a otros casos, como podrían ser los tipos de MV disponibles en la plataforma del usuario concreto, o incluso en proveedores comerciales como Amazon EC2 o en Windows Azure.

Pero no sólo nos hemos quedado en el diseño del algoritmo y del sistema en sí, sino que también se ha desarrollado el producto VMCA, que se puede poner

en producción para actuar sobre una plataforma de virtualización basada en ONE. Este sistema se puede descargar desde la página web creada al efecto[1], o se puede acceder a las últimas versiones contactando con el autor.

La distribución del producto se hace en formato código abierto, y se ha hecho un esfuerzo bastante grande para que el usuario pueda implementar sus propias heurísticas o adaptar las existentes a la plataforma en concreto. También se ha hecho hincapié en favorecer el alcance del producto y, en principio, se podría adaptar a plataforma existentes y a otras que pudieran surgir, siempre que se pudiese acceder al sistema de información y crear las estructuras necesarias para VMCA, y que existiesen mecanismos para ordenar la migración de las MVs entre los nodos.

En la versión actual del sistema nos hemos centrado en la liberación de nodos para permitir su apagado posterior por falta de uso. Esto nos permitirá contar con una mayor cantidad de recursos continuos, para la atención de peticiones de alojamiento de MVs. Sin embargo, no se ha trabajado de forma específica el rebalanceo de las MVs con el único propósito de compactar los recursos. Este es un problema que aún debe ser desarrollado en trabajo futuros y que permitirá mantener los nodos de virtualización encendidos, pero siendo capaces de albergar MVs con características específicas. Para ello habría que tener en cuenta no solo los equipos que están encendidos, sino también aquellos que se encuentran apagados y cuyo encendido podría aportar valor a la distribución de los recursos libres.

Por otro lado, la posibilidad de encender nodos de virtualización también nos daría pie a la implementación de heurísticas más elaboradas, relacionadas con el consumo de energía en su conjunto y no sólo con la suposición de que apagando nodos bajará dicho consumo. Por ejemplo, podríamos implementar heurísticas que tuviesen en cuenta las características energéticas de todos los equipos reales de la plataforma y que tratase de concentrar la carga de MVs en aquellos más eficientes.

[1] http://www.grycap.upv.es/clues/en/addons.php

CAPITULO

8

Marco de trabajo y resultados científicos

El autor de este trabajo forma parte del Grupo de Grid y Computación de Altas Prestaciones (GRyCAP)[1] desde el año 2000. Desde entonces ha participado en distintos proyectos nacionales, europeos e internacionales, así como en contratos de investigación aplicada en los campos de Grid Computing y de Cloud Computing. Durante el transcurso de estos proyectos ha generado numerosas publicaciones en congresos y revistas científicas del área. En los últimos años ha centrado su actividad en las áreas del Cloud Computing y del ahorro energético (del inglés, *Green Computing*), campos en los que se encuentra el presente trabajo.

VMCA es una iniciativa que se enmarca dentro del ecosistema del proyecto *Cluster Energy Saving* (CLUES)[2], como complemento del mismo. CLUES es un sistema de gestión de energía para clusters de altas prestaciones e infraestructuras cloud, cuya función principal es la de apagar los nodos internos del cluster cuando no están siendo utilizados y, de forma recíproca, encenderlos de nuevo cuando son necesarios. El sistema CLUES se integra con el middleware de gestión del cluster, como puede ser un gestor de colas o un sistema de gestión de infraestructura cloud,

[1] http://www.grycap.upv.es
[2] http://www.grycap.upv.es/clues

mediante el uso de una serie de conectores. Por lo tanto, VMCA deja del lado de CLUES la planificación del apagado efectivo de las máquinas reales que forman la plataforma de virtualización, de acuerdo a los criterios establecidos.

El ecosistema de CLUES está formado por una serie de proyectos que abarcan distintas facetas tanto del ahorro de energía como de aspectos relativos al cloud. Por ejemplo, en la actualidad se encuentra en desarrollo el proyecto *Elastic Cloud Computing Cluster* (EC3) [21], cuyo objetivo es el de desplegar clusters elásticos en una plataforma cloud como Amazon EC2. El resultado de la utilización de EC3 es un cluster completamente instalado, gobernado por un sistema de colas como PBS o SGE, en el que tanto el nodo principal como los internos son MVs. El cluster resultante tiene características elásticas, de modo que inicialmente no cuenta con ningún nodo en funcionamiento y se van poniendo en funcionamiento MVs a medida que es necesario (hasta un máximo indicado por el usuario). Una vez los nodos internos han realizado su trabajo, las MVs que no están siendo utilizadas se apagan utilizando CLUES.

EC3 está diseñado para trabajar tanto con Amazon EC2 como con despliegues basados en ONE. En este último caso, se consigue particionar el cluster físico, entregando a cada usuario uno que tiene instaladas las aplicaciones y librerías que él necesita (al utilizar MVs específicas). Dado que las MVs irán siendo creadas y eliminadas de acuerdo a su uso, VMCA será un complemento ideal para consolidar las MVs que se encuentran en funcionamiento y poder apagar los nodos reales.

Otro proyecto adicional es *Cluster as a Service* (CaaS), que básicamente es un interfaz gráfico para EC3. En este caso, el usuario puede indicar qué tipo de máquinas necesita, el tamaño del cluster, las credenciales de usuario de la plataforma donde se va a desplegar, el número máximo de MVs a lanzar, las políticas de apagado, etc. y el servidor CaaS le proporcionará la dirección del nodo principal y las credenciales de acceso al cluster.

Tanto CLUES como los proyectos adyacentes están generando resultados científicos, de los que el autor de este trabajo es partícipe. Se han generado tres artículos en revistas indexadas [22], [23] y [21], uno en un congreso indexado [19] y otro en un congreso no indexado [24]. Se espera publicar los resultados específicos de VMCA en un congreso indexado, durante el próximo año.

Además de los resultados científicos, CLUES y su ecosistema también han tenido reseñas divulgativas tanto en blogs tecnológicos como en prensa escrita y en el canal de televisión de la Universidad Politécnica de Valencia. Se puede encontrar información al reespecto en la página web del proyecto. En esta misma página se publican las aplicaciones asociadas al proyecto (entre ellas, VMCA) en forma *Open Source*. En la actualidad cuentan con numerosas descargas desde distintas partes del mundo y contactos de usuarios a la dirección de correo del proyecto.

Bibliografía

[1] W. Forrest. How to cut data centre carbon emissions?, 2008.

[2] Lakshmi Ganesh. *Data Center Energy Management*. PhD thesis, Faculty of the Graduate School of Cornell University, 2012.

[3] M.M. Rafique, N. Ravi, S. Cadambi, A.R. Butt, and S. Chakradhar. Power management for heterogeneous clusters: An experimental study. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, 2011.

[4] GiorgioLuigi Valentini, Walter Lassonde, SameeUllah Khan, Nasro Min-Allah, SajjadA. Madani, Juan Li, Limin Zhang, Lizhe Wang, Nasir Ghani, Joanna Kolodziej, Hongxiang Li, AlbertY. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Fredric Pinel, JohnatanE. Pecero, Dzmitry Kliazovich, and Pascal Bouvry. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1):3–15, 2013.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[6] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, 28(5):755–768, May 2012.

[7] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 215–224, New York, NY, USA, 2010. ACM.

[8] Ioan Salomie, Tudor Cioara, Ionut Anghel, Daniel Moldovan, Georgiana Copil, and Pierluigi Plebani. An energy aware context model for green it service centers. In *Proceedings of the 2010 international conference on Service-oriented computing*, ICSOC'10, pages 169–180, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] Yasuhiro Ajiro and Atsuhiro Tanaka. Improving packing algorithms for server consolidation. In *Int. CMG Conference*, pages 399–406. Computer Measurement Group, 2007.

[10] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[11] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 28–28, Berkeley, CA, USA, 2009. USENIX Association.

[12] The games research project, 2013.

[13] T. Cioara, I. Anghel, I. Salomie, G. Copil, D. Moldovan, and A. Kipp. Energy aware dynamic resource consolidation algorithm for virtualized service centers based on reinforcement learning. In *Parallel and Distributed Computing (ISPDC), 2011 10th International Symposium on*, pages 163–169, 2011.

[14] E. Feller, L. Rilling, and C. Morin. Energy-aware ant colony based workload placement in clouds. In *Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on*, pages 26–33, 2011.

[15] Cristina Bianca Pop, Ionut Anghel, Tudor Cioara, Ioan Salomie, and Iulia Vartic. A swarm-inspired data center consolidation methodology. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, WIMS '12, pages 41:1–41:7, New York, NY, USA, 2012. ACM.

[16] E. Feller, C. Morin, and A. Esnault. A case for fully decentralized dynamic vm consolidation in clouds. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 26–33, 2012.

[17] Moreno Marzolla, Ozalp Babaoglu, and Fabio Panzieri. Server consolidation in clouds through gossiping. In *Proceedings of the 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, WOWMOM '11, pages 1–6, Washington, DC, USA, 2011. IEEE Computer Society.

[18] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 826–831, Washington, DC, USA, 2010. IEEE Computer Society.

[19] F. Alvarruiz, C. de Alfonso, M. Caballer, and V. Hernñdez. An energy manager for high performance computer clusters. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 231–238, 2012.

[20] Minyi Yue. A simple proof of the inequality ffd (l) $\leq$ 11/9 opt (l) + 1, $\forall$l for the ffd bin-packing algorithm. *Acta Mathematicae Applicatae Sinica*, 7(4):321–331, 1991.

[21] Miguel Caballer, Carlos de Alfonso, Fernando Alvarruiz, and Germán Moltó. Ec3: Elastic cloud computing cluster. *Journal of Computer and System Sciences*, (0):–, 2013.

[22] Carlos De Alfonso, Miguel Caballer, Fernando Alvarruiz, and GermáN Moltó. An economic and energy-aware analysis of the viability of outsourcing cluster computing to a cloud. *Future Gener. Comput. Syst.*, 29(3):704–712, March 2013.

[23] Carlos de Alfonso, Miguel Caballer, Fernando Alvarruiz, and Vicente Hernández. An energy management system for cluster infrastructures. *Computers & Electrical Engineering*, (0):–, 2013.

[24] C De Alfonso, M Caballer, and V Hernández. Efficient power management in high performance computer clusters. In *Proceedings of the International conference on green computing*, volume 2010, 2010.

Lista de Acrónimos

BF	Best Fit (Donde mejor ”quepa”).
BFD	BF Decreasing (Donde mejor ”quepa”, con orden inverso).
CPD	Centro de Proceso de Datos.
DPM	Dynamic Power Management (Gestión Energética Dinámica).
DVFS	Dynamic Voltage and Frequency Scaling (Escalado dinámico de Frecuencia y Voltaje).
FF	First Fit (Donde primero ”quepa”).
FFD	FF Decreasing (Donde primero ”quepa”, con orden inverso).
MAPE	Monitorización, Análisis, Planificación y Ejecución.
MV	Máquina Virtual.
ONE	OpenNebula.
QoS	Quality of Service (Calidad de Servicio).
SLA	Service Level Agreement (Acuerdo de Nivel de Servicio).
SPM	Static Power Management (Gestión Energética Estática).

UPS	Uninterruptible Power Supplies (Sistemas de Alimentación Ininterrumpida).
VMCA	Virtual Machine Consolidation Agent (Agente de Consolidación de Máquinas Virtuales).