Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TRABAJO FINAL DE CARRERA

**TÍTULO DEL TFC : Gestor de recursos sobre sistemas virtualizados**

**TITULACIÓN: Ingenieria Técnica de Telecomunicación, especialidad Sistemas de Telecomunicación**

**AUTOR: Jesús Gavilán García**

**DIRECTOR: David González**

**SUPERVISOR: Toni Oller**

**FECHA: 15 de diciembre de 2011**

**Título :** Gestor de recursos sobre sistemas virtualizados

**Autor:** Jesús Gavilán García

**Director:** David González

**Supervisor:** Toni Oller

**Fecha:** 15 de diciembre de 2011

Resumen

Actualmente el modelo para proveer servicios de Tecnologías de la Información tiende a basarse en la implementación de nubes y virtualización. Debido principalmente a la flexibilidad y abstracción de los recursos de cara al usuario final como al aprovechamiento y optimización de recursos de cara al proveedor de servicios.
El principal objetivo de este trabajo es proponer e implementar un sistema de gestión de recursos para una arquitectura basada en un clúster con sistemas de ficheros distribuidos en red y con virtualización de sistemas operativos [1]. La arquitectura ha sido implementada por el MediaCAT de la Fundación i2CAT con el objetivo de proveer servicios de escritorios y servidores virtualizados.
La solución propuesta esta basada en un sistema con agentes monitorizadores de la utilización de los recursos del clúster. Los agentes permiten controlar el comportamiento del sistema. Los datos recogidos por los agentes son procesados por un gestor de recursos. Este gestor de recursos en base a una política de reasignación de recursos y consumo energético permite gestionar las cargas de trabajo y consolidación de los servidores que conforman el clúster.
Este trabajo estudia la modelización del problema que supone la reasignación de recursos bajo diversas limitaciones para encontrar posibles soluciones óptimas. Posteriormente el problema es modelado basándose en los problemas de tipo *Bin-Packing* y Suma de Subconjuntos, planteando algoritmos de aproximación y heurística para su resolución
Después de definir las claves para conseguir el objetivo planteado, el sistema será capaz de gestionar de forma automática ó semiautomática la gestión de recursos del sistema garantizando la consolidación, el balanceo de carga de los servidores y el ahorro de energía.

**Title :** Resource management for virtualized systems based on cloud environment

**Author:** Jesús Gavilán García

**Director:** David González

**Supervisor:** Toni Oller

**Date:** December 15, 2011

Overview

Nowadays, the model for providing IT services/resources is based on cloud computing an virtualization concepts. Both concepts allow to provide flexibility, resource abstraction, resource optimization and management to a server cluster architecture.
The overall aim of this project is to propose and implement a resource management system for an architecture already implemented. This architecture were implemented by i2CAT foundation to provide desktop and server virtualization services. The system is based on a cluster with network file system and operating system virtualization [1].
In order to achieve the aim, it is necessary to understand the problem of resource management with multiple variables to optimize and how it can be modeled to find solutions.
It is proposed a solution system based on agents. Agents allow monitorize the system behavior and resource utilization. Along to these agents is implemented a resource management module to manage the resource allocation, policy about power savings, minimize the number of active servers and workload balance.
Finally the system will be able to manage server consolidation, resource allocation and power savings in automatic or semi-automatic mode.

# ÍNDICE GENERAL

# ÍNDICE DE FIGURAS

# ÍNDICE DE CUADROS

# GLOSARIO

**BPP** Bin Packing Problem

**DHCP** *Dynamic Host Configuration Protocol*, protocolo utilizado para asignar automáticamente direcciones IP a las equipos de una red.

**HTTPS** *Hypertext Transfer Protocol Secure*, protocolo que emplea autentificación y encriptación en comunicaciones basadas en comunicaciones web.

**Libvirt** es el conjunto de método, librerías y herramientas que interactúan con las capacidades de virtualización. Brinda una capa de abstracción y un protocolo que permite administrar y migrar máquinas virtuales.

**LVM** Logical Volume Manager, herramienta utilizada en Linux para gestionar los volúmenes lógicos,

**MV** Máquina Virtual, es una implementación basada en software de un máquina física.

**NFS** *Network File System*, es un sistema de archivos de red que permite acceder,modificar y almacenar archivos de forma remota como si se tratara de archivos locales.

**PXE** *Pre-boot eXecution Environment*, es una combinación de protocolos DHCP y TFTP utilizado para arrancar máquinas mediante la interfaz de red.

**SSP** Problema de Suma de Subconjuntos (*Subset Sum Problem*)

**TFTP** *Trivial file transfer Protocol*, es un protocolo de transferencia de archivos que es usado para automatizar la transferencia de archivos de configuración o arranque entre dos máquinas.

**IA** Inteligencia Artificial.

**XML** *eXtensible Markup Language*, lenguaje desarrollado para permitir la descripción de la información contenida en internet a través de estándares y formatos comunes.

**XML-RPC** Es un protocolo de llamada de procedimiento remoto que utiliza XML para codificar datos y que emplea HTTP como protocolo de transmisión de mensajes.

# INTRODUCCIÓN

La gestión de recursos en los datacenters es un problema cada vez más importante debido a que repercute en el coste del servicio por el elevado consumo energético de los equipos. Esto es relevante pues se estima que el coste de consumo energético en todo el tiempo de uso del servidor llega a superar el propio coste del servidor en sí [2].

Por este motivo la virtualización surge como solución a los problemas de optimización de recursos y consumo energético que presentan los servidores basados en el modelo de equipo por servicio. Bajo este concepto, las arquitecturas de servicios han evolucionado hacia lo que actualmente se define como computación en la nube (*Cloud Computing*), que permite ofrecer un nuevo punto de vista para la construcción de servicios de computación sobre internet.

La virtualización además permite la consolidación de servidores, cuyo objetivo es conseguir un eficiente uso de los recursos para reducir el número total de servidores físicos y como consecuencia reducir costes.

En base a estos principios, el objetivo de este TFC es proporcionar un sistema de gestión de recursos a una arquitectura basada en sistemas virtualizados bajo el concepto de computación en la nube que brinda el servicio de escritorios y servidores virtualizados a los usuarios de la Fundación i2CAT [1].

El trabajo que se presenta, profundiza en la propuesta de optimización y monitorización de recursos proponiendo un sistema basado en agentes implementados en los diferentes nodos, y un módulo IA (inteligencia artificial) que mediante algoritmos y métodos heurísticos establecen los procedimientos para mejorar la optimización de los recursos globales del sistema.

El primer capítulo de este trabajo muestra los objetivos iniciales que se propone abarcar para la consecución de un sistema gestor de recursos.

El segundo capítulo sirve como introducción a la virtualización y da paso a definir el escenario en el cual se basa este TFC. El sistema sobre el cual se aplicará el gestor de recursos posee una arquitectura basada en el concepto IaaS, *Infraestructure as a Service*, implementando una nube privada. Una plataforma virtualizada, dinámica y distribuida.

El segundo capítulo explica el estado actual del arte de la gestión de recursos en plataformas virtualizadas. Se hace mención los métodos empleados por Xen y a los trabajos publicados recientemente que han ido localizando la solución al problema tanto de la consolidación de servidores como el ahorro de energía. Se estudia además las características que ofrecen las soluciones comerciales.

En los posteriores capítulos se estudia principalmente el problema que supone la reasignación de recursos en sistemas virtualizados. El módulo IA es el encargado de definir las políticas de gestión, y es donde está situado el problema. Para ello se modeliza del problema y la complejidad de este. Se explica el algoritmo y métodos empleados que se utilizan en la implementación del módulo IA.

Los siguientes capítulos de este trabajo muestran los resultados de la simulación del algoritmo. Aquí se definen los procedimientos que se require para simular el rendimiento de la solución.

Por último se realiza el estudio medioambiental. En este trabajo este aspecto es relevante, ya que uno de los objetivos es conseguir la minimización del consumo energético. Se hace un repaso a la importancia que actualmente tiene el Green IT sobre las nuevas tecnologías basadas en idea de la nube. Además de los retos y las tendencias que un futuro las tecnologías basadas en la idea de la computación en la nube puedan conseguir.

# CAPÍTULO 1. OBJETIVOS Y METAS DEL SISTEMA GESTOR DE RECURSOS

Los objetivos y metas que se han propuesto inicialmente para garantizar la gestión de los recursos son los siguientes:

1. **Estudio de las condiciones iniciales y estado actual del sistema**

   Antes de emprender el proyecto es necesario establecer los conceptos básicos en el cual se fundamenta la arquitectura del sistema diseñado por el MediaCAT y las condiciones iniciales para poder implementar el sistema gestor de recursos o IA.

2. **Migración de máquinas virtuales**

   Una de las características que ofrece la arquitectura es la habilidad de mover máquinas virtuales(MVs) de una máquina física a otra de manera transparente. Comprobar su correcto funcionamiento es un objetivo prioritario ya que los algoritmos que se aplicará para gestionar los recursos requieren este mecanismo.

3. **Tolerancia a fallos**

   Se pretende implementar un grado de tolerancia a fallos con una resistencia a caídas de un nodo simultáneamente. Permitiendo así que cuando un nodo falle, la plataforma, de manera dinámica, detecte la caída y se migren las máquinas virtuales a otro nodo.

4. **Balanceo**

   El balanceo de la carga de trabajo es el proceso de distribuir la demanda de servicios /recursos a través de multiples servidores. El objetivo es conseguir una utilización óptima de los recursos del clúster de servidores. Para ello se requiere un módulo que gestione las demandas de recursos de los usuarios. Este módulo ha de ser capaz de determinar cual es el servidor más adecuado para satisfacer la demanda de recursos entrantes. Es necesario establecer políticas de gestión de recursos en casos de saturación de servicios. Es por eso que se requiere un balanceo de carga de los servidores en función de la demanda de recursos, además de ejecutarse de una manera eficiente y rápida.

5. **Escalabilidad**

   El numero N de servidores esclavos no ha de tener cota, viene definido únicamente por las necesidades totales. Esta posibilidad dependerá principalmente de la complejidad del algoritmo utilizado para reasignar los recursos.

6. **Monitorización**

   Para poder gestionar los recursos del sistema se requiere implementar la capacidad de monitorizar los recursos tanto de CPU y RAM de los nodos secundarios en tiempo real y de forma centralizada.

   Este sistema es necesario para mantener no solo información actual de la utilización de los recursos sino que sea capaz de almacenar un histórico. Mediante el

procesado de esta información se puede conseguir datos sobre los picos de utilización, la periodicidad y las tendencia de uso de los recursos. El cumplimiento de este objetivo está a cargo del MediaCAT de la Fundación i2CAT.

7. **Minimizar información de control y carga de red**

   El sistema gestor y de monitorización de recursos ha de ser capaz de minimizar la información redundante y el tránsito de información de control.

   El gestor de recursos ha de preocuparse de minimizar el número de máquinas virtuales a migrar para poder reducir la carga de red.

8. **Transparencia**

   Los cambios de configuración han de ser transparentes al usuario final. Para ello es importante implementar una interfaz en la que permita al usuario final gestionar los recursos iniciales y características de sus MVs. Tanto el *live migration* como las políticas de gestión de recursos han de ser transparentes al usuario, además de garantizar el mínimo impacto posible sobre el rendimiento del sistema.

9. **Consolidación de servidores**

   La consolidación de servidores trata de definir un conjunto de estrategias para reducir costes asociados con el funcionamiento del clúster de servidores. Tiene como objetivo simplificar la gestión de recursos, reducir la complejidad eliminando el número de servidores innecesariamente activos. La consolidación de servidores está relacionado con la gestión de la carga de trabajo de los servidores. Existen diferentes tipos de consolidación de servidores en función del ámbito a optimizar:

   - **Consolidación Lógica:** Se refiere a implementar procesos comunes que implica centralizar servidores. Reduciendo el número de puntos de control. Reduciendo operaciones de control como backup, acciones de recuperación y mantenimiento busca reducir costes de mantenimiento.
   - **Consolidación Física:** Implica la reducción de las multiples localizaciones físicas donde se encuentran los servidores. Bajo este tipo se consigue eliminar la replicación de acciones que puedan producirse sobre servidores en localizaciones diferentes. Al centralizar la ubicación de los servidores se consigue reducir la complejidad de la red y hacerla más eficiente.
   - **Consolidación transparente:** Mediante este tipo de consolidación todos los elementos del sistema se convierten en entidades virtuales. Bajo este entorno se requiere implementar un sistema de gestión de recursos altamente automatizado para minimizar costes de administración.

   La consolidación transparente es la que se adecúa a la situación actual del sistema. Básicamente se debe a las características que presenta la arquitectura del sistema, ya que actualmente se cuenta con un único clúster centralizado. La arquitectura actual fue diseñada para soportar virtualización, por lo que se requiere un gestor de recursos que pueda controlar bajo una serie de políticas y algoritmos las cargas de trabajo de los nodos. Este es el objetivo final de este TFC.

10. **Implementación inicial en entorno real**

Para comprobar el funcionamiento del sistema gestor de recursos se probará el algoritmo mediante la realización de simulaciones. Posteriormente a medida que las condiciones lo permitan se considera la implementación inicial sobre el sistema real para comprobar el funcionamiento del algoritmo. Esta implementación estará supeditada a que los grupos de trabajo unifiquen el proyecto así como también que las características del hardware y software de la arquitectura implantada permitan las funcionabilidades establecidas por el gestor de recursos.

11. **Ahorro de energía**

    El consumo energético de los datacenters está aumentando a medida que aumentan los servicios de computación en la nube. Se estima que el porcentaje de utilización de un servidor de media está por debajo del 30 % [2]. Esto supone un importante derroche de energía y costes en proporción a las dimensiones que presente el datacenter. Debido a los costes que supone, el ahorro de energía es una de las prioridades en el diseño de arquitecturas clusterizadas.

    Los factores que contribuyen al exceso de consumo energético en los datacenters se deben principalmente a la infrautilización del hardware disponible. Especialmente es un factor crítico la correcta asignación de los recursos en función de la demanda.

    Por este motivo, el impacto que supone la virtualización sobre el consumo energético es muy significativo.

    Una arquitectura virtualizada por si sola supone una paso importante para conseguir minimizar el consumo energético, pero puede aportar mejores resultados implementar un sistema que gestione y monitorice la utilización de los recursos.

    Por lo que garantizando una mejor utilización de los recursos de los servidores se puede garantizar la minimización del consumo energético. Esto se consigue consolidando los servidores y gestionando dinámicamente las recursos disponibles.

# CAPÍTULO 2. ESCENARIO ACTUAL DEL SISTEMA

Este capítulo revisa los conceptos y definiciones necesarias para seguir correctamente la lectura de este documento, además se describen las condiciones iniciales de la arquitectura del sistema y las características que presenta.

## 2.1. Introducción a los sistemas virtualizados y la computación en la nube

Para entender la arquitectura sobre la que se basa este TFC es importante definir unos conceptos previos, tales como la virtualización y la computación en la nube.

### 2.1.1. Virtualización

La virtualización se define como el proceso de abstracción de recursos o servicios a partir de recursos físicos. Esta definición se aplica a todo tipo de infraestructura informática, como pueden ser recursos de red, plataformas hardware, sistemas operativos o dispositivos de almacenamientos [3]. Desde el punto de vista de la gestión de servidores, la virtualización es un método que permite ejecutar múltiples sistemas operativos (SO) en una única máquina física. La virtualización de servidores representa el modelo que más se está extendiendo debido a las deficiencias que presenta el concepto tradicional de uso de un servidor por servicio. El modelo tradicional equivale tener un SO por máquina física que implica un coste de infraestructura elevado y un alto grado de infrautilización de recursos. Aparte de los beneficios que la virtualización aporta en la optimización de recursos, también aporta beneficios que van desde la simplificación de la administración, la portabilidad e independencia del tipo de hardware, al incremento de la utilización del hardware y sobre todo respecto a la consolidación de servidores. Desde el punto de vista económico reduce la infraestructura física, reduce el coste operativo de los Datacenters relacionados con el consumo eléctrico y la refrigeración.


Figura 2.1: Virtualización

### 2.1.2. Computación en la nube

La computación en la nube es un conjunto de servicios brindados a través de la red. Estos servicios pueden ser optimizados y garantizar una calidad dinámica flexible. Estos servicios pueden dar acceso a hardware, recursos como aplicaciones, almacenamiento, ancho de banda, todos ellos integrados en una plataforma que es transparente al usuario final. Para conseguir estas características la computación en la nube utiliza la virtualización como clave para su desarrollo [3].

Las ventajas que aporta la computación en la nube son diversas, entre ellas: la elasticidad y escalabilidad, que permite variar la capacidad de los recursos brindados a los usuarios en función de la demanda. El autoprovisionamiento, permite a los usuarios obtener los recursos o servicios automáticamente evitando los retrasos que supone el modelo tradicional de datacenter, en la que se requiere constantemente aprobación de cada servicio o recurso requerido. La accesibilidad, pues permite el uso de los recursos independientemente del dispositivo que posee el usuario final.

En la figura 2.2 se puede observar las capas o niveles que soporta la computación en la nube. Estas capas están situadas por encima de la capa de virtualización que abstrae los recursos de la capa inferior.

Estos servicios se basan en 3 capas. La infraestructura hace referencia a los recursos como son los equipos, los dispositivos de red, almacenamiento y todos aquellos elementos que son necesarios para que el software pueda funcionar.

La plataforma representa al software básico necesario de cada aplicación, como pueden ser las bases de datos, web servers.

La aplicación es la última capa de cara al usuario final. Abstrae la información que requiere el usuario de los datos, por ejemplo permite el acceso de una base de datos desde un móvil, portátil o cualquier dispositivo, cuya localización y tecnología es transparente al usuario.

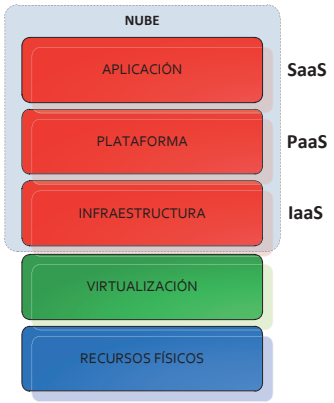


Figura 2.2: Estructura de una nube

#### *2.1.2.1. Modelos de Nube*

**Basados en el tipo de despliegue**

Las nubes se pueden clasificar en función del lugar donde estén implementadas.

- **Nube pública:** Infraestructura implementada por una organización externa( proveedor) que brinda servicios via internet. Este modelo se basa en diferentes niveles de SLA ó acuerdo del nivel de servicio para garantizar determinada calidad sobre el servicio que presta el proveedor.

  Existen varias compañías que prestan servicios de datacenters privados como por ejemplo Amazon EC2 , Google, Sun ó Microsoft.

- **Nube privada:** Infraestructura utilizada por una organización para su uso interno, de esta forma se obtiene el control sobre cada aspecto de la implementación. Existen diversas tecnologías que permiten implementar estos modelos; tales como Eucalyptus, Elastra, VMware, Azure Microsoft, Xen Server.

  La arquitectura del sistema que trabaja este TFC es claramente una nube privada.

**Basados en los servicio que soporta**

Una forma de clasificar la computación en la nube es a partir de los servicios que puede ser capaz de brindar.

- **Infrastructure as a Service(IaaS):** Bajo este concepto los recursos físicos, tales como los servidores, los dispositivos de red ó almacenamiento se ofrecen como servicio. También incluye el sistema operativo y la tecnología de virtualización para gestionar los recursos. Ejemplos de este modelo son el Amazon EC2, Mosso de Rackspace.

  La arquitectura del sistema trabajada en este TFC esta basado en este modelo.

- **Platform as a Service(PaaS):** Con este modelo se provee no solo la infraestructura sino también el conjunto de software básico necesario para que el usuario/desarrollador pueda realizar aplicaciones para la gestión de los recursos físicos. Es decir provee el soporte necesario al usuario para que desarrolle su propia solución.

  Ejemplos de ello son los servicios de web hosting, Google AppEngine,Microsoft Azure.

- **Software as a Service(SaaS):** Bajo este modelo el software es desplegado en Internet y disponible al usuario final cuando lo requiera. Ejemplos son las soluciones que brinda Google tales como Picasa, Documents, Google Music, etc.

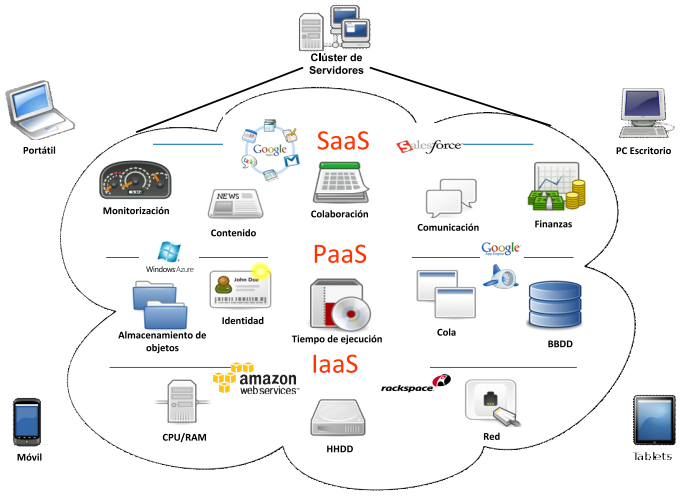


Figura 2.3: Tipos de nube

### 2.1.3. Plataforma de Virtualización

Las plataformas de virtualización aúnan los conceptos de virtualización y computación en la nube. Las plataformas de virtualización permiten virtualizar una plataforma hardware, que incluye recursos CPU, recursos de red y almacenamiento. En las plataformas de virtualización lo que hace posible la abstracción de los recursos físicos y que sean distribuidos y compartidos por distintos usuarios es el *Hypervisor*. Existen diversas plataformas de virtualización, los más utilizados son KVM,QEMU,Sun xVM Server, VMWare, y Xen que es la plataforma empleada en este TFC.

#### *2.1.3.1. Elementos de una Plataforma de Virtualización*

**Hypervisor**

El *Hypervisor* es el elemento que hace la función de capa subyacente que permite la abstracción de los recursos físicos sean compartidos y gestionados para los múltiples usuarios. Bajo el contexto de virtualización de servidores, el *Hypervisor* actúa como una capa que gestiona la plataforma física (hardware) virtualizando el hardware y haciendo que estos recursos físicos sean compartidos y usados de forma transparente por múltiples sistemas operativos.Las instancias de estos sistemas operativos son las denominadas máquinas virtuales (MV), estos son sistemas operativos huéspedes o clientes.

El *Hypervisor* posee un arquitectura que permite a los sistemas operativos clientes correr bajo un sistema operativo anfitrión. Este trabajo se centra en tipo Linux hypervisor, que integra el *Hypervisor* dentro del kernel de Linux, y que puede implementar *full virtualization*.

**Host (Anfitrión)**

Permite la creación de un entorno de simulación de la que dependerán las máquinas

virtuales. Es el sistema operativo principal a partir del cual se implementan las diferentes particiones o MVs mediante el *Hypervisor* y la asignación de recursos.

**Guest (Máquinas Virtuales)**

Es la representación de una maquina real utilizando software/aplicaciones que funcionan bajo un sistema operativo *Guest*. El uso de máquinas virtuales permite compartir el mismo hardware del servidor (*Host*) en diferentes instancias. Proporciona aislamiento total del sistema operativo del *Host* y el resto de MVs, además permite encapsular una MV entera de forma que facilita la migración de estas entre distintos *Hosts*, en tiempo real y sin tener que apagar las máquinas (en caliente).

#### *2.1.3.2. Técnicas utilizadas en las plataformas de virtualización*

**Full Virtualization (Virtualización Completa)**

Mediante la virtualización completa o nativa se implementa una capa subyacente que permite acceder a los recursos físicos del *Host*. De este modo permite al *SO Guest* o Sistema Operativo de las máquinas virtuales manipular los recursos a nivel nativo.

Por este motivo el *SO Guest* desconoce la existencia del entorno virtualizado y no requiere ser modificado.

**Paravirtualización**

Bajo este término se denomina a la técnica que permite al *SO Guest* saber que está ejecutándose bajo un entorno de virtualización, es decir es consciente que el acceso a los recursos físicos no se hace directamente sino que es gestionado por el *Hypervisor*. De este modo el *SO Guest* es modificado para poder garantizar un rendimiento cercano a la virtualización nativa. Esta técnica es utilizada principalmente con SO basados en LinuxBSD y BSD, ya que estas permiten modificación del kernel, a diferencia de SO Windows y MAC en la que no es posible esta modificación.

### 2.1.4. Xen

Xen es una plataforma de virtualización de código abierto creada por la Universidad de Cambridge y desarrollada por la comunidad Xen bajo licencia GPL. El objetivo principal como plataforma de virtualización es proveer aislamiento e independencia entre las distintas instancias de sistemas operativos (MV) que son ejecutados en el mismo hardware. Por lo que proporciona a cada MV su propias interfaces de red, disco y memoria.

En contraste a los tradicionales emuladores como el VMware o VirtualBox que emulan los procesos a nivel de software, Xen lo hace ejecutando todo el software directamente en el procesador y por consecuencia de forma más rápida. Para ello Xen requiere portabilidad de los sistemas operativos (*SO Host*). Xen destaca principalmente por el rendimiento, por la gestión de CPU y memoria óptima, por el soporte a *full virtualization* con soporte de hardware con el Intel VT o el AMD -V y permite mover en caliente una máquina virtual

entre distintos equipos físicos.

Figura 2.4: Logotipo de Xen

#### *2.1.4.1. Elementos de Xen*

**Xen Hypervisor**

Es el núcleo de Xen. Éste se ejecuta directamente sobre el hardware convirtiéndose en la interface para todas la peticiones de hardware como CPU, espacio de almacenamiento, I/O (entrada/salida) para los *SO Guest*. Permite la separación de los *SO Guest* del hardware para garantizar la ejecución de múltiples sistemas operativos de forma segura e independiente.

**Domain0**

Este dominio es el *SO Host* con soporte de virtualización. El Domain0 tiene privilegios de administración al *Xen Hypervisor*. Estos privilegios le permite gestionar todos los aspectos de los DomainU, desde los recursos asignado a el arranque, parado, entrada/salida,etc.

**DomainU**

Este dominio no presenta los privilegios del Dom0 pero son ejecutados mediante el Domain0 que arranca un SO con un kernel modificado (máquinas virtuales). Bajo este dominio se encuentran las máquinas virtuales que puede aplicar la técnica de paravirtualización o *full virtualization*.

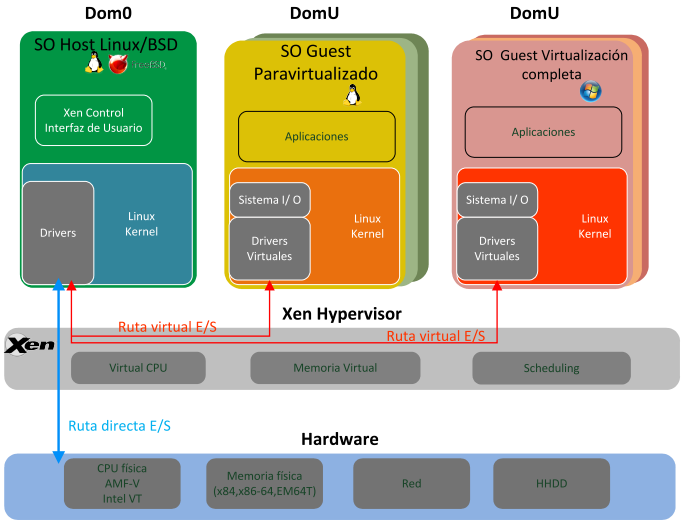


Figura 2.5: Estructura Xen

## 2.2. Arquitectura del sistema

Este TFC está basado en la arquitectura del sistema diseñada por el MediCAT de la Fundación i2CAT [1]. La arquitectura del sistema está basada en el modelo IaaS, *Infrastructure as a Service* . Se trata de un modelo de nube privada bajo una plataforma virtualizada, dinámica y distribuida formada por dos tipos de nodos. Un nodo principal que proporciona capacidad de almacenamiento y diversos nodos secundarios con alta capacidad de procesamiento y memoria. Es una plataforma virtualizada por que utiliza los conceptos de computación en la nube y virtualización utilizando la plataforma libre Xen OpenSource. Es una plataforma dinámica ya que permite la distribución de los recursos en función de la demanda requerida. Es distribuida al no tratarse de un sistema aislado sino clusterizado donde cada tipo de nodo tiene una función determinada. Es importante mencionar que podría parecer una contradicción el hecho de que los nodos secundarios estén distribuidos pero el Máster con la capacidad de almacenamiento este centralizado. Pero no es así ya que el máster puede ser replicado obteniendo cierto grado de seguridad si este cae. A continuación se definen los principales elementos del sistema.

### 2.2.1. Elementos del sistema

#### *2.2.1.1. Servidor Máster*

Este servidor ha sido pensado para llevar a cabo las tareas de gestión y control de recursos de la nube,es por ello que es capaz de controlar a los servidores esclavos. Además está conectado a la cabina de discos.

#### *2.2.1.2. Cluster de servidores esclavos*

Esta formado por N servidores esclavos. Han sido diseñados para soportar la carga de RAM y CPU demandada por la nube por lo que no presentan capacidad de almacenamiento. El almacenamiento es brindado por la cabina de discos centralizada en el servidor Máster.

#### *2.2.1.3. Máquinas virtuales*

Las máquinas virtuales se ejecutan en los nodos esclavos. Para ello el máster se encargará de gestionar los recursos disponibles por la nube (RAM,CPU, almacenamiento) y procederá a asignarlas a un nodo esclavo idóneo, este nodo se decide mediante el módulo IA que es el objetivo de este TFC. En la figura 2.6, tomada de [1], se observan los elementos que conforman el sistema.

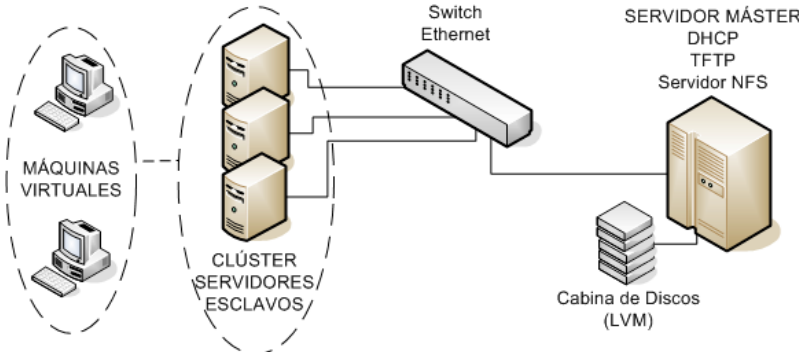


Figura 2.6: Arquitectura del sistema [1]

### 2.2.2. Funcionamiento del sistema

En un primer paso el sistema inicializa los nodos. De este modo se procede a cargar los SO de los servidores esclavos a través de la red utilizando el protocolo PXE.

Este protocolo en una primera instancia emplea el servicio DHCP. Este permite al servidor esclavo obtener una dirección IP dinámicamente cada vez que arranca, del mismo modo permite localizar el servidor máster. Luego utiliza el servicio TFTP, mediante el cual se transfiere un microkernel desde el máster hasta el esclavo. Una vez descargado el kernel, éste es compilado para soportar virtualización basada en Xen. Luego es ejecutado directamente en la RAM del servidor esclavo.

Una de las características principales de esta arquitectura es que es un sistema dinámico. Esto se debe a que el sistema de ficheros de los servidores esclavos se reserva en la cabina de discos utilizando LVM y son montados empleando NFS, que permite montar las particiones en un sistema remoto y usarlas como si se tratara de un sistema de archivos local. Mediante este sistema se ejecutan las máquinas virtuales sobre los servidores esclavos.

En la figura 2.7, tomada de [1], se observa un ejemplo del funcionamiento del sistema

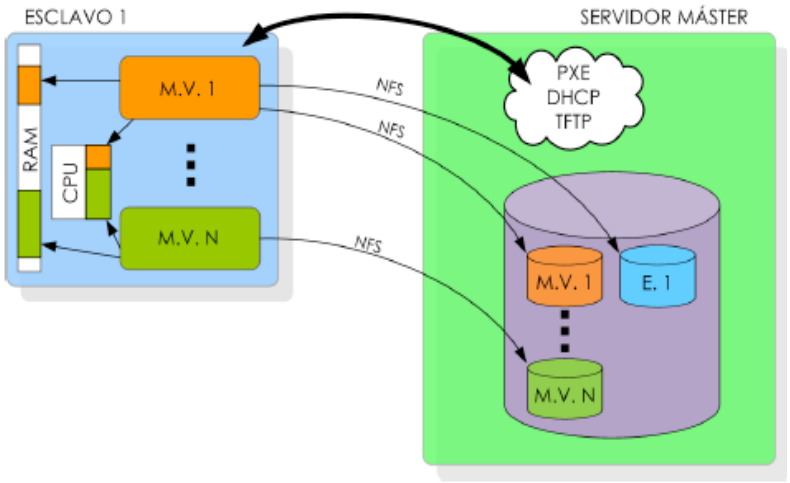


Figura 2.7: Funcionamiento del sistema [1]

## 2.3. Sistema de monitorización y gestión de recursos

Es importante mencionar que la arquitectura definida ya estaba implementada y en funcionamiento antes de abarcar este proyecto. El sistema de monitorización y gestión de recursos se ha trabajado de forma paralela junto al MediaCAT de la Fundación i2CAT.

### 2.3.1. Problemática del sistema gestor de recursos

Para gestionar los recursos de una nube bajo plataformas virtualizadas surgen una serie de problemas que derivan de garantizar el aislamiento de las MV, definir los diversos parámetros a optimizar.

La primera restricción que se encuentra al optimizar los recursos es garantizar el correcto aislamiento de cada máquina. Por otra parte, se requiere establecer un sistema capaz de monitorizar constantemente los recursos utilizados por las diversas MV, sin afectar el rendimiento del sistema.

Establecer un algoritmo que tenga en cuenta los múltiples objetivos a optimizar. Como son la RAM, la utilización de la CPU, y el espacio en disco. Es importante el reto ya que se requiere asignar grados de importancia a los recursos.

Si se requiere establecer asignaciones adaptativas de los recursos, se observa que es necesario un ajuste óptimo que permita mantener un buen grado de optimización de los recursos frente al rendimiento del sistema.

## 2.4. Propuesta de gestión de recursos y monitorización

Para implementar un sistema que consiga cumplir las metas propuestas al inicio de este documento se requiere un sistema capaz de monitorizar eventos que se puedan producir en la nube y que tenga suficiente inteligencia para actuar en consecuencia con el objetivo principal de optimizar los recursos disponibles minimizando el consumo energético. Una vez conocido los objetivos y los problemas que supone gestionar los recursos del sistema, se procede a determinar una solución.

Este trabajo propone la utilización de agentes en los diferentes nodos que permiten monitorizar los recursos utilizados por cada uno de ellos. Mediante esta información y las políticas de asignación de recursos apropiadas se intenta minimizar los efectos que se puedan afectar al funcionamiento de la nube. En la figura 2.8, se detalla la propuesta con las tecnologías utilizadas en su implementación. La implementación de esta propuesta se ha realizado de forma paralela. El sistema de monitorización y agentes se ha ido construyendo por el MediaCAT la vez que este trabajo se ha ido centrando en la gestión de los recursos.

Se pueden definir los siguientes módulos que intervienen en el sistema:

### 2.4.1. Interfaz Usuario (Virt-Manager)

Mediante una interfaz web se permite al usuario interactuar con todo el sistema. Permite la creación de máquinas, la personalización de sistemas operativos, el listado de máquinas previamente creadas y además es capaz de presentar la monitorización gráfica del consumo de CPU y RAM de los diferentes nodos. Este módulo se comunica con el agente principal para que proporcione los recursos que solicite el usuario.

### 2.4.2. Agente Principal

Este bloque se encarga de recibir información sobre las características de las máquinas virtuales generadas por el usuario a través de la interfaz web y definir los recursos necesarios. Recibe información sobre las condiciones de la nube mediante los agentes secundarios. Este módulo además se encarga de interactuar con Xen; por lo que es el agente encargado de establecer las comunicaciones entre los Dom0's y el exterior.

Mediante la recopilación de información el sistema es capaz de actuar en consecuencia a las condiciones de la nube, es por ello, que contiene el módulo IA (Inteligencia Artificial). La IA será capaz de optimizar los recursos en función de las condiciones de la nube.

### 2.4.3. Agentes Secundarios

Se encarga de recopilar información sobre los recursos que utiliza cada nodo y cada máquina virtual de éste. Al igual que el agente principal estos módulos están programados en Python bajo la librería Libvirt utilizando el protocolo XML-RPC para las comunicaciones entre agentes. Este protocolo utiliza XML para codificar los datos y HTTPS para enviar los mensajes.

### 2.4.4. Módulo IA

Este módulo es parte del agente principal. Se encarga de la heurística de la nube, brindando al sistema la capacidad de realizar cambios automáticos para conseguir la optimización de los recursos disponibles. El funcionamiento de la IA viene determinado por la política de asignación de recursos para los diferentes casos de uso. Dentro de este módulo se implementarán los algoritmos y heurística para la gestión de los recursos del sistema, convirtiéndose en el núcleo del estudio en este TFC y en el cual los capítulos siguientes estarán centrados.

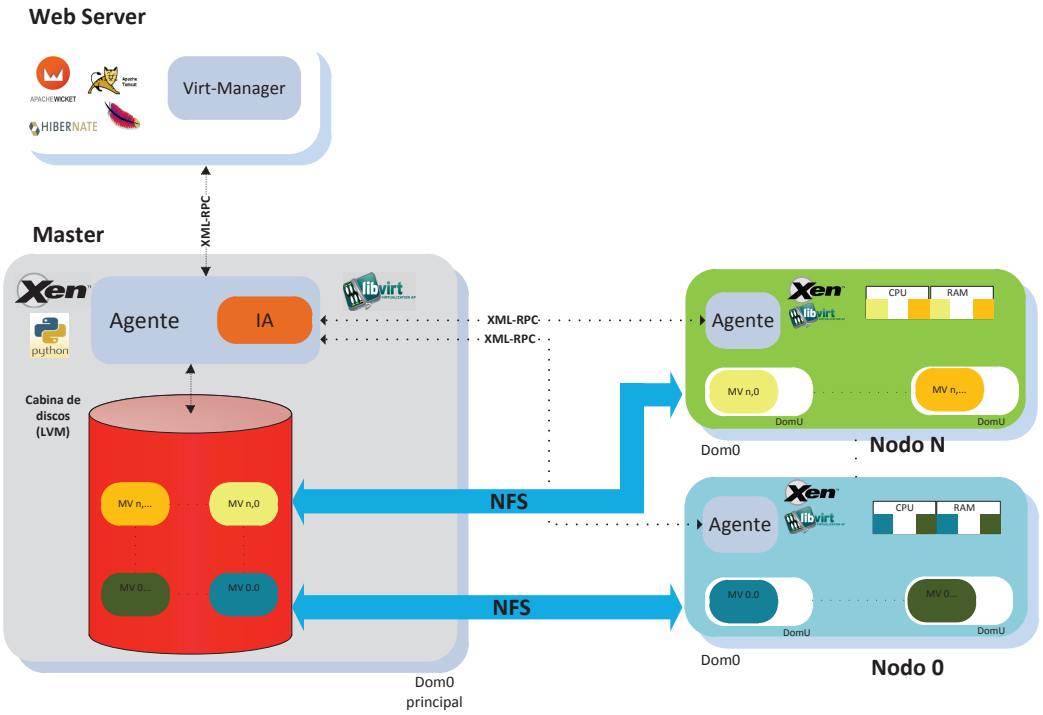


Figura 2.8: Arquitectura del sistema de monitorización y gestor de recursos

# CAPÍTULO 3. ESTADO DEL ARTE DE LA GESTIÓN DE RECURSOS EN SISTEMAS VIRTUALIZADOS

Este capítulo estudia las técnicas actuales empleadas para gestionar sistemas virtualizados, desde las técnicas nativas empleadas por Xen para gestionar los recursos de un único servidor a las implementaciones comerciales sobre sistemas clusterizados, pasando por las publicaciones que proponen soluciones óptimas.

## 3.1. Sistemas nativos Xen para la gestión de recursos

Las herramientas que proporciona Xen para la gestión de recursos son válidas para un único servidor ó Dom0, a diferencia de la infraestructura presentada en este TFC, el cual trata de un sistema clusterizado (multidominio) con diversos nodos y diferentes hypervisores. A nivel local de cada servidor ó Dom0 los recursos son gestionados por el Xen CPU schedulers y el gestor de memoria de Xen. Por este motivo la arquitectura requiere un sistema que sea capaz de gestionar los recursos globales del sistema (interdominio) y no sólo a nivel local.

### 3.1.1. Gestión de CPU en Xen

Es importante gestionar estos recursos de forma efectiva para que se pueda garantizar la modificación dinámica del espacio de las MVs, la migración de las MVs entre distintos nodos y que pueda gestionar este cambio de condiciones. Básicamente los algoritmos de gestión de CPU en Xen se encargan de asignar tiempo de CPU a las CPUs virtuales (VCPU) a cada MV. Estos schedulers intentan asegurar que la CPU minimice los ciclos posibles. El *Hypervisor* de Xen proporciona 3 diferentes propuestas de algoritmos.

#### *3.1.1.1. Borrowed Virtual Time (BTV)*

Este scheduler está basado en el concepto de tiempo virtual.Permite asignar proporcionalmente tiempos de CPU. Se caracteriza por la baja latencia para aplicaciones en tiempo real. Permite conservar el trabajo de un dominio, por ejemplo en caso de tener solo un dominio activo se le asigna toda la CPU disponible sin limitar su uso.

#### *3.1.1.2. Simple Earliest Deadline First Scheduling (SEDF)*

Este algoritmo permite especificar un límite inferior de recursos CPU utilizados tanto a Dom0 como los DomU. Es ideal en entornos donde se requiera aplicaciones en tiempo real.

#### *3.1.1.3. Credit Scheduler*

Actualmente tanto BVT como SEDF están en desuso a favor del Credit Scheduler. Este algoritmo esta diseñado para minimizar el derroche de tiempo de CPU.

Permite ajustar el uso de los recursos de CPU de acuerdo a la demanda de los DomU. Ofrece un sistema que permite asignar créditos a las VCPU (Virtual CPU) y ordenarlas en colas en función de una prioridad determinada.

Bajo este algoritmo se le asigna a cada dominio un *weight* y un *cap*, el *weight* es una parte proporcional de CPU para cada MV y el *cap* indica el limite superior de tiempo de CPU que una MV puede consumir. El scheduler se encarga de transformar el *weight* en créditos de asignación para cada VCPU, los créditos son consumidos en dos categorías, MV activas y no activas. Las MV activas consumen créditos a ritmo fijo y bajo dos prioridades Under y Over, la prioridad Over permite consumir créditos excediendo la división proporcional que se le asigna de cada CPU y Under no permite que se sobrepase el límite. Las MV inactivas no consumen créditos.

Debido a los cambios y avances que se dan en entornos virtualizados, sistemas de gestión de consumo energético y el hardware actual, este sistema de créditos presenta una serie de puntos débiles que hacen que no sea una solución óptima con los actuales *Hypervisors*. Es importante mencionar que este algoritmo no es escalable a las nuevas CPUs con un número elevado de núcleos. Por este motivo actualmente se está implementando el Credit 2 Scheduler [8].

Este algoritmo funciona de tal modo que todas las MV empiezan con valores de crédito fijo. Los créditos pueden ser consumidos en ratios variables, a diferencia de su antecesor. Además debería proporcionar, entre otras características, un balanceo de carga instantáneo entre los núcleos y mejor respuesta respecto al consumo energético.

### 3.1.2. Gestión de memoria en Xen

Las técnicas para optimizar recursos de CPU están ampliamente documentadas e implementadas. Sin embargo la técnicas empleadas para gestionar memoria son escasas, probablemente por la dificultad de gestionar el tiempo compartido de memoria física. Este problema resulta ser importante ya que produce cuellos de botella en los sistemas virtualizados. De este modo frenan el rendimiento final del sistema.

Las particiones de memoria en Xen son espacios de memoria empleados por los distintos elementos que lo conforman. La memoria utilizada por el SO del Dom0, la utilizada por el *Hypervisor* y la memoria utilizada por los SO de los DomU, tal como se observa en la fig . 3.1. La memoria restante se denomina memoria *fallow* o memoria no explotada. Xen aplica una serie de mecanismos para gestionar la memoria, estos son el particionamiento estático, dinámico y el *Trascendent Memory*.


Figura 3.1: Partición de memoria en Xen

#### *3.1.2.1. Particionamiento Estático*

El particionamiento estático permite asignar a una nueva MV iniciada una cantidad fija de memoria física. De este modo cada DomU o Guest utiliza bajo este límite la memoria que requiera. El Dom0 sin embargo se reserva su propia memoria. Esta asignación implica que en la mayoría de tiempo exista memoria desperdiciada (*idle memory*), debido a la infrautilización de memoria física máxima de cada MV. Este valor sumado a la memoria no explotada da como resultado una considerable cantidad de recursos sin utilizar. Por este motivo este mecanismo tiene sus limitaciones.

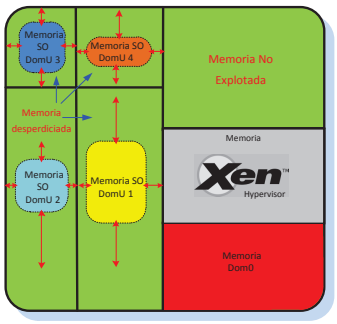


Figura 3.2: Particionado Estático

#### *3.1.2.2. Particionamiento Dinámico*

El particionamiento dinámico consiste en aplicar la técnica del *ballooning*. Esta técnica implementa un *"ballon driver"* o dispositivo globo que se asigna a cada MV. El *"ballon driver"* se encarga de absorber o liberar memoria física que emplea cada SO de DomU o *Guest*. Este dispositivo permite transferir espacio de memoria de un *SO Guest* a otro por medio del *Hypervisor*. Esta transferencia se realiza en función de las MVs que requieran recursos y aquellas que cedan recursos. Esto permite reducir la memoria no explotada del sistema. Este método tiene múltiples restricciones. Principalmente el problema radica en determinar exactamente cuales son las MVs que deben liberar recursos y cuales requieren absorberlos. Este problema se suma al hecho de depender de los *SO Guest* para la cesión de memoria. En la fig. 3.3 se observa como el *balloning* va cediendo recursos entre MVs. Se observa que del paso de a) a c) la cantidad de memoria no explotada se va reduciendo.

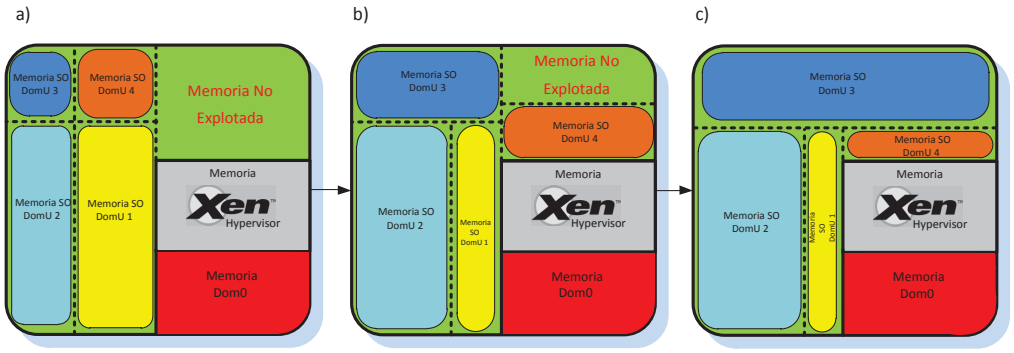


Figura 3.3: Particionado Dinámico

### *3.1.2.3. Trascendent Memory(TMEM)*

El método de memoria sin límites es una implementación en desarrollo pensada para solucionar los problemas de los mecanismos anteriormente explicados.

El objetivo de este método es mejorar la utilización de la memoria física en un entorno virtualizado. En un primer paso el método consiste en reclamar toda la memoria no explotada del sistema. Luego procede a recoger memoria desperdiciada de los diferentes *Guests*. El conjunto de memoria no explotada y desperdiciada es recogida en un *pool* de memoria: el *Transcedent Memory Pool (TMP)*. La memoria del *TMP* es gestionada por el *Hypervisor* y el Dom0. El acceso a esta memoria se provee a partir de una API que permite definir las restricciones de uso de esta memoria.

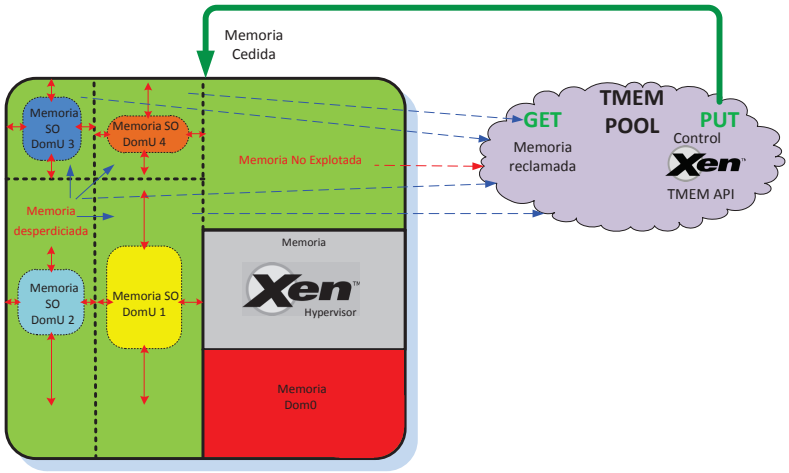


Figura 3.4: Trascendent Memory (TMEM)

Esta implementación contempla la gestión de memoria de Dom0 aislados pero la gestión entre dominios actualmente está en fase de investigación. Precisamente este el objetivo de este TFC, garantizar la gestión de recursos entre dominios.

## 3.2. Trabajos relacionados

La gestión de recursos en sistemas virtualizados es uno de los problemas más importante a solucionar por el aumento de servicios basados en la nube. Por este motivo se realizan muchos estudios e implementaciones de algoritmos que busquen optimizar globalmente los recursos de un **sistema distribuido**.

### 3.2.1. Implementaciones Comerciales

#### *3.2.1.1. XenServer Citrix*

El Xenserver es una plataforma de virtualización basada en Xen. Es la versión comercial del XenOpenSource que se utiliza en este TFC. Esta plataforma comercial presenta un serie de herramientas para conseguir consolidación de servidores, optimización y gran tolerancia a fallos.

La diferencia entre el XenOpenSource y el XenServer es principalmente el grado de estabilidad y fiabilidad que pueden ofrecer. El XenOpenSource esta mantenida por una comunidad de usuarios y programadores pero que entre sus puntos débiles destaca que no puede garantizar la estabilidad de algunos componentes del sistema.

La versión comercial XenServer se encuentra bajo el amparo de Citrix. Esto le permite obtener productos y herramientas rigurosamente testeadas y mejor adecuadas para su uso en producción.

El **Workload Balancing** [12] es la herramienta que permite gestionar el balanceo de carga de los servidores bajo la plataforma XenServer. Esta herramienta permite evaluar la utilización de los recursos y el mejor host que pueda contener a determinada MV. Es capaz de determinar el mejor host en la que una MV puede arrancar, el mejor host en el que una MV puede ser asignada luego de ser reanudada, el mejor host al que mover una MV luego de la caída del host de origen. Todas estas configuraciones pueden ser aplicadas automáticamente así también como el apagado de servidores activos para minimizar el consumo energético.

El Workload Balancing es un sistema muy flexible ya que permite modificar parámetros límite de utilización de RAM o CPU. La configuración de la carga de trabajo de los servidores lo hace de acuerdo a dos conceptos. Por una parte lo hace en función del rendimiento, es decir la utilización de recursos como la CPU, RAM, red y almacenamiento de un servidor o nodo. De este modo al maximizar el rendimiento el sistema es capaz de recomendar el host adecuada para cada MV.

Por otro lado lo hace en función de la densidad, es decir el número de MVs por *Host*. De esta manera si se require maximizar la densidad el sistema, éste es capaz de recomendar la mejor reasignación de MVs que asegure minimizar el número de servidores activos sin afectar el rendimiento. Tal como se comentaba al principio, este sistema permite optimizar y ejecutar las configuraciones de manera automática en base recomendaciones. Estas recomendaciones están basadas en la superación de una serie de lindares de utilización.

En función de los márgenes de estos lindares se permite definir si la utilización es crítica o baja.

#### *3.2.1.2. VMware ESX*

El VMware ESX [13] es una plataforma de virtualización desarrollado por WMware, Inc.Las versiones posteriores como el VMware ESXi y el Vsphere están basadas en esta plataforma. Este entorno posee herramientas de gestión de recursos y ahorro de energía.

**VMware Distributed Resource Scheduler (DRS)** es el gestor de recursos de esta plataforma, se encarga de asignar y balancear los recursos que requieran las MVs para optimizar los recursos físicos globales. Esta herramienta monitoriza continuamente los recursos disponibles que presenta el sistema y mediante una serie de políticas de gestión predefinidas asigna los recursos entre las MVs de forma dinámica. La asignación de recursos a las MVs se realizan mediante migración a otro servidor con mas recursos disponibles o liberando espacio en el mismo servidor migrando otras MVs a distintos servidores. Para realizar el *Live migration* utiliza la herramienta VMware vMotion, que le permite realizar la migración de MVs de forma transparente al usuario.Esta herramienta funciona en modo automático y manual. En modo automático define la mejor distribución posible de las MVs entre los diferentes nodos y hace la migración automática de las MVs a los nodos adecuados. En modo manual sugiere recomendaciones sobre la reasignación de las MVs para que finalmente el administrador del sistema decida el cambio.

**VMware Distributed Power Management (DPM)** es la aplicación que utiliza el WMware ESX para supervisar el consumo energético del cluster. El DRS como en líneas anteriores se ha comentado permite ahorrar consumo energético al consolidar dinámicamente la carga de trabajo del clúster. El DPM va más allá respecto al ahorro de energía ya que incluso es factible minimizar el consumo en periodos donde la utilización de recursos sea baja.

Para conseguirlo se encarga de gestionar el apagado/encendido de los nodos del sistema. De este modo cuando las necesidades de recursos disminuyen, consolida las cargas de trabajo reduciendo el consumo energético del clúster. Si las necesidades de recursos de las cargas de trabajo aumentan, el DPM activa nuevamente los nodos para garantizar los niveles de servicio.

Ambas herramientas proporcionan a la plataforma uno de los más completos sistemas de gestión de recursos de las plataformas comerciales.

#### *3.2.1.3. Microsoft Hyper-V*

Es la plataforma de virtualización desarrollada por Microsoft, también conocido como Windows Server Virtualization. Esta plataforma cuenta con el **System Center Virtual Machine Manager** [14] que provee las herramientas para garantizar la consolidación de los servidores de la infraestructura virtualizada. Dentro de las características que presenta este gestor de recursos, se encuentran la capacidad de proporcionar optimización dinámica y de optimizar el uso de consumo energético. Aunque principalmente destaca por la

compatibilidad que presenta con diversos hypervisores, como el Hyper-V, WMware y Xen Hypervisor. De este modo es independiente a los gestores de recursos de cada plataforma.

Todas las características que presentan estos gestores de recursos son buenas para un sistema como el que es tratado en este TFC. Son herramientas pensadas para entornos corporativos por lo que tienen en común la licencia propietaria. Esto implica que no definen los algoritmos ni la heurística que utilizan para gestionar los recursos de sus infraestructuras.

### 3.2.2. Publicaciones

Existe una distinción entre los trabajos analizados. Por una parte se encuentran los trabajos que analizan e implementan modelos para conseguir un tipo de consolidación estática y semi-estática.

Al hablar de consolidación estática se refiere a la gestión de MV que son asignadas en servidores físicos en un rango de tiempo largo. Es decir que no trabajan sobre los recursos demandados en tiempo real sino que intentan analizar la tendencia de utilización y conseguir patrones de uso a largo tiempo.

En caso de la consolidación semi-estática se habla de tiempos no muy largos, por ejemplo una planificación,gestión y migración de MV (recursos) diarias. Tanto la consolidación estática como semi-estática son los tipos de gestión donde se mantiene un control de las migraciones de MV entre servidores físicos por parte del administrador.

Por otro lado la consolidación dinámica permite una gestión automática e inmediata de los recursos de la nube. Se trata de gestionar las MVs y los recursos que demandan estas bajo variaciones inmediatas de carga de trabajo.

En *Server workload analysis for power minimization using consolidation* [15] analiza las potenciales formas de reducir el consumo energético de un conjunto de clusters de servidores. Este trabajo hace énfasis en la utilización de la correlación entre los picos de utilización de cada MV para establecer la configuración más efectiva. El tipo de consolidación conseguida en todo caso es estática o semi-estática y para conseguirlo utiliza valores históricos de utilización de los servidores.

En *A mathematical programming approach for servers consolidation. Problem in virtualized data centers* [16] se centra en la optimización de recursos para nubes de media escala, superior a 600 servidores. Este trabajo propone un método de consolidación de servidores estática, combinando un análisis de datos históricos de utilización para caracterizar variaciones y métodos heurísticos para definir un algoritmo que asigne los recursos de manera óptima. Es importante mencionar que este trabajo no solo está preparado para gestionar en casos de infrautilización de los recursos del cluster sino que también puede resolver casos donde ocurren picos de demanda de recursos. Aunque el principal logro de este trabajo es que consigue evaluar de manera experimental cargas de trabajo de entornos reales.

En la publicación *Gerência de Memória Adaptativa no XEN* [17] propone un sistema de gestión dinámica o adaptativa de memoria. La propuesta se basa en monitorizar los recursos de memoria capaz de identificar las necesidades de las máquinas virtuales y la distribuir este recurso de forma más justa. Para gestionar la memoria se basa en las herramientas nativas que ofrece Xen en la monitorización de recursos, utilizando la información de tendencia momentánea de memoria utilizada y un *daemon* o agente que continuamente está espiando esta información. Esta publicación muestra resultados simulados e implementación real bajo entorno Xen OpenSource.

En el trabajo *Gossip-based Resource Management for Cloud Environments* [18] se estudia e implementa una solución para el problema de la optimización de recursos para estructuras de nube a gran escala. La gran diferencia respecto a los anteriores trabajos mostrados es que permite hacer la gestión de forma dinámica. Es decir tiene en cuenta los recursos tanto CPU y RAM en función del tiempo. El objetivo principal de este trabajo es minimizar el consumo energético de los servidores, para ello se basa en algoritmos de tipo *Gossip* y una heurística propia que permite minimizar el consumo energético en caso de infrautilización de los recursos del sistema. Este sistema permite gestionar clusters entre 10 mil y 100 mil servidores o nodos. Esta publicación no muestra pruebas de implementación real pero muestra mediante simulaciones resultados que cumplen con los objetivos que se propone. A pesar que la parte de consolidación de servidores, en la que define el método para reducir el número de servidores activos está en desarrollo.

Se observa que existen muchas opciones para conseguir la optimización de recursos, desde nativas a herramientas externas. Por lo que este trabajo intenta recoger estos conceptos para proponer un gestor que se adapte a las necesidades de nuestro sistema.

# CAPÍTULO 4. ESTUDIO DEL PROBLEMA DE LA ASIGNACIÓN DE RECURSOS

Este capítulo se centra en el problema de la gestión de recursos que el módulo IA ha de solventar. Se definen conceptos previos, se modela el problema y posteriormente se explica la propuesta empleada.

## 4.1. Definición del problema

La interfaz de web permite al usuario definir las características de la MV a implementar. El usuario puede elegir el S.O. a instalar, el espacio en disco que requiere y lo más importante: la cantidad máxima de RAM que demanda.

El espacio en disco está centralizado en el servidor principal y no requiere gestión alguna ya que no es un recurso distribuido en los diferentes nodos. Por lo que la IA gestionará la utilización de RAM del clúster de servidores y tendrá en cuenta el porcentaje de utilización de CPU.

Para definir el problema de la asignación de recursos es importante situarse en el modo de funcionamiento de este. La secuencia se inicia con la demanda de recursos por parte del usuario. Esta petición es gestionada por la IA, que intenta asignar esta demanda a un espacio libre dentro de algún nodo disponible del clúster. La IA recurre a los datos de utilización de recursos de cada nodo, proporcionados por los agentes que están en cada uno de los nodos monitorizando continuamente la utilización tanto de CPU como de RAM. La IA procesa la información según las políticas de asignación de recursos y efectúa la colocación de la MV dentro del nodo elegido.

Precisamente el problema de la asignación de recursos es definir en que nodo disponible irá a parar la futura MV con las características demandas por el usuario. Por lo que a partir de aquí el estudio se basa en esta premisa.

En un primer momento parece que el problema es sencillo ya que simplemente bastaría en colocar estos recursos en el primer nodo disponible. Esta no es una solución válida para un sistema que busque la optimización global del sistema. Ya que de esta manera sólo se aseguraría elegir la mejor opción en cada demanda. Es decir buscaría la optimización local en cada paso. Definir el nodo más adecuado en función solo del espacio libre no garantiza al final que se tenga una optimización precisa cumpliendo con todas los objetivos que se marca el sistema.

Un símil de este problema sería el caso de intentar colocar una serie de cajas en diversos contenedores. Cada caja tendría una conjunto de características que podrían ser el ancho y alto o el volumen. Del mismo modo los contenedores. En este caso supongamos que el número de contenedores es limitado y que cada uno de ellos tiene el mismo ancho, alto y volumen, son iguales. El problema se vislumbra al intentar definir limitaciones al momento de asignar estas cajas dentro de los contenedores. Cada una de éstas supone un grado de complicación al problema.

Por ejemplo se pediría que se coloquen las cajas utilizando el mínimo número de contenedores. Para el caso que interesa sería igual a cumplir con la consolidación de los servidores.

Otra limitación puede ser garantizar que el número de cajas esté equitativamente repartido en cada contenedor. Sería igual que garantizar la carga de trabajo de los servidores.

Se requiere además que el tiempo en asignar todas las cajas en los contenedores sea mínimo.

Ahora se ha de imaginar que el número de cajas disponibles es incierto y que aleatoriamente van apareciendo y del mismo modo son substraídas. Pero aún estos inconvenientes sea capaz de asignar de forma optima las cajas que actualmente se tengan garantizando las limitaciones anteriores.

Se puede ver la complejidad que muestra el problema de asignar las MVs (cajas) dentro de los nodos (contenedores) y que requiere ser analizado por partes.

Para definir un modelo que siga fielmente las características y limitaciones del sistema gestor de recursos es importante explicar algunos conceptos propios de la teoría de complejidad. Este tema es bastante complejo y requiere una rigurosidad para ser tratado o abordado a fondo. Debido a la extensión de este documento y básicamente que está fuera del objetivo principal de este trabajo, se procederá a definir algunas nociones básicas.

### 4.1.1. Complejidad Computacional

Al diseñar algoritmos uno de los factores que indica su idoneidad es la complejidad temporal, es decir el número de pasos de ejecución de un algoritmo para resolver un problema. El espacio es otro parámetro importante ya que cuantifica la memoria utilizada para resolver un problema.

Normalmente la complejidad temporal está expresada con la notación tipo O. Es una notación libre de constantes multiplicativas y minimiza el número de términos.

La complejidad de tipo polinomial hace referencia al algoritmo cuyo tiempo de ejecución está limitado por una expresión polinomial. Matemáticamente consiste en una suma de términos, cada uno de ellos incluye variables elevadas a una potencia o multiplicadas por una constante. Ejemplos de complejidad temporal son los de tipo logarítmicos ($log(n)$), los lineales ($n$), cuadráticos ($n^2$), etc.

Con la capacidad actual de cómputo se pueden resolver problemas mediante algoritmos que tienen como máximo un coste computacional polinómico. Los problemas de este tipo están agrupados en el conjunto P. A este tipo de problemas se les denomina tratables, pues son abordables en la práctica.

#### *4.1.1.1. Problemas NP*

Existen problemas en las que aparentemente la única solución posible es resolverlos de forma exhaustiva. Esto conlleva a tener tiempos de resolución exponenciales o factoriales. Es decir, a medida que los valores de entrada van aumentando, el tiempo de resolución seguirá estas tendencias y hace que este tipo de solución sea impracticable. Este tipo de problemas son los denominados NP.

Para este tipo de problemas con la capacidad computacional actual no pueden ser resueltos en un tiempo razonable. Según [25], la clase de problemas NP (*Non-deterministic polynomial*) son aquellos problemas que son *verificables* en tiempo polinomial. De esta forma si existiese un *certificado* que valide una solución, entonces se podría verificar éste en tiempo polinomial.

Este tipo de problemas son importantes ya que bajo estas características existen muchos problemas de optimización. En las que es necesario saber si existe una solución o una mejor que las conocidas. Uno de estos problemas es el que en este trabajo busca solución, que consiste en definir el nodo al que será asignada una determinada MV con tal de minimizar globalmente el número de nodos utilizados.

Debido a su importancia se han realizado muchos trabajos para resolver este tipo de problemas en un tiempo polinomial.

Los **problemas NP-Completo** son los que presentan extrema complejidad. Los problemas NP-Completo tienen una característica especial. Ésta dice que si cualquier tipo de problema de clase NP-Completo es resuelto en tiempo polinomial, entonces todos los problemas de tipo NP podría ser resueltos en tiempo polinomial. De este modo se cumpliría $P = NP$. Por lo que los problemas impracticables podrían ser resueltos en tiempos polinomiales.

De hecho se han hecho muchos esfuerzos para encontrar dicha solución y actualmente se da por aceptado que no debe existir. Aunque no se ha demostrado la imposibilidad de su existencia. Por tanto existe actualmente un problema abierto y de extensa literatura. Este problema radica en comprobar si verdaderamente $P = NP$.

Los **problemas de tipo NP-Hard** realmente no son un subcategoria de los tipo NP, a pesar del nombre. Se considera que un problema NP-Hard si, suponiendo que se pueda resolver en tiempo constante, se pueda utilizar estas soluciones para resolver un problema NP-completo en tiempo lineal. Debido a esto se dice que un problema es NP-Hard si es tanto o más complicado de resolver que uno NP-complete, independiente a su pertenencia a NP o no.

En la figura 4.1 se puede observar el que actualmente es la más probable relación entre las distintas clases.

Los algoritmos de tipo NP-Complete y NP-Hard conocidos emplean tiempos exponenciales con respecto al valor de la entrada. De este modo aplicar un algoritmo de búsqueda exhaustiva resulta impracticable. Por lo que se utilizan una serie de técnicas para reducir la complejidad temporal del problema. Entre ellos destacan:

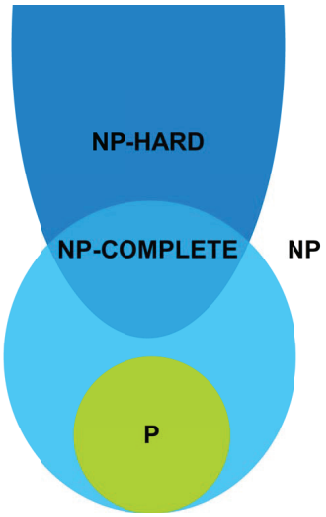


Figura 4.1: Diagrama de Venn de la familia de problemas NP

- Aproximación: Al aplicar algoritmos de aproximación se consiguen soluciones rápidas, pero no garantizan que sean soluciones óptimas. Por lo cual se estima la calidad de la solución bajo un rango de error. Existen muchos problemas de tipo NP-completo en los que es suficiente aplicar un algoritmo de aproximación, pero no todos los problemas de este tipo tienen algoritmos de aproximación.
- Probabilístico: Aplicando algoritmos probabilísticos permite obtener en promedio una buena solución al problema.
- Heurística: Este tipo de algoritmos son rápidos, pero no existe una medida para saber la calidad de la respuesta. Suelen trabajar razonablemente bien en muchos casos.
- Restricciones: Mediante este método se consiguen soluciones rápidas en base a restringir la estructura de las entradas.
- Algoritmos genéticos: Estos algoritmos mejoran las posibles soluciones hasta encontrar una que se acerque al óptimo. No existe forma de garantizar la calidad de la respuesta.
- Casos particulares: Existen casos particulares del problema para los cuales existen soluciones rápidas.

### 4.1.2. Problema de tipo Bin-Packing

Al definir el problema de la asignación de recursos se había hecho la comparación con el problema de tipo *Bin-Packing*. Este es un problema combinacional de tipo NP-Hard.

Considerando el problema para 1D dimensión pero extensible a $p$ dimensiones. Estas dimensiones pueden ser la altura,ancho,peso,volumen,coste, u otro tipo de medidas propias del objeto.

El problema consiste en empaquetar un número determinado de objetos o cajas en un número dado de contenedores tal que el total del peso,volumen,largo,etc no exceda la capacidad del contenedor. Siendo el número de contenedores utilizados el mínimo posible.

A medida que el número de dimensiones (medidas) aumenta, la complejidad se incrementa tremendamente.

Formalmente:

Dada en una serie de $n$ objetos $j \in \mathrm{N} = \{1, ...., n\}$ , con parámetros( peso,coste,altura,etc) $w_j \forall j \in \mathrm{N}$, con un número $m$ de idénticos contenedores $i$ de capacidad $W$.

El 1BPP(Bin Packing Problem de una dimensión) consiste en asignar todos los objetos (4.3) en cada contenedor que no exceda $W$ (4.2) y que el número de contenedores utilizado sea el mínimo posible (4.1). Teniendo en cuenta que existen $n$ potenciales contenedores con variables de decisión (4.6) y objetos con variables de decisión (4.7).

$$min \sum_{i=1}^{m} y_i \tag{4.1}$$

sujeto a;

$$\sum_{j=1}^{n} w_j x_{ij} \leq W \quad i \in \mathrm{N} = \{1, ...., m\} \tag{4.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad j \in \mathrm{N} \tag{4.3}$$

$$y_i \in 0, 1 \quad i \in \mathrm{N} \tag{4.4}$$

$$x_{ij} \in 0, 1 \quad i, j \in \mathrm{N} \tag{4.5}$$

donde;

$$y_i = \begin{cases} 1 & \text{si el contenedor } i \text{ es utilizado} \\ 0 & \text{otros casos} \end{cases} \qquad (i = 1, ...., n) \tag{4.6}$$

$$x_{ij} = \begin{cases} 1 & \text{si el objeto } j \text{ es asignado al contenedor } i \\ 0 & \text{otros casos} \end{cases} \qquad (i, j = 1, ...., n)(j = 1, ...., m) \tag{4.7}$$

Los métodos de resolución básicos que se aplican para la resolución de este problema están situados en los apéndices.

### 4.1.3. Problema de Suma de Subconjuntos

Otro problema que tiene una relación que más tarde se verá en el desarrollo del algoritmo para la gestión de recursos es el problema de suma de subconjuntos (*Subset Sum Problem - SSP*).

El SSP es un problema de tipo NP-Complete. Este problema consiste en encontrar un subconjunto cuya suma sea lo más cercano,pero no más grande, que un límite determinado. Donde el límite es un número positivo y el subconjunto está formado por $n$ números enteros positivos.

Formalmente:

Dado un conjunto de enteros positivos $A = \{a_1, ...., a_n\} \in \mathrm{N}$ con $B \in \mathrm{N}$ que representa el límite. El problema consiste en encontrar un vector $X = \{x_1, ...., x_n\}$ ,subconjunto de A, que representa el subconjunto solución, cuya suma sea el máximo valor cercano al límite $B$ (4.8) pero sujeto a no superar este valor (4.10):

$$max \sum_{i=1}^{n} a_i x_i \tag{4.8}$$

donde;

$$x_i = \begin{cases} 1 & \text{si } a_i \text{ está en el subconjunto solución} \\ 0 & \text{otros casos} \end{cases} \qquad (i = 1, ...., n) \tag{4.9}$$

sujeto a;

$$\sum_{i=1}^{n} a_i x_i \leq B \quad (i = 1, ...., n) \tag{4.10}$$

Los métodos más usuales de resolución a este problema pueden verse en los apéndices. El planteamiento de este problema servirá para modelar una situación determinada dada al implementar el gestor de recursos.

## 4.2. Modelado y propuesta

El problema de la asignación de recursos en los diferentes nodos es similar a los problemas de tipo *Bin-packing*. Las soluciones que se han visto para resolver este problema son diversas (apéndices). El objetivo tal como se pudo determinar en apartados anteriores es realizar la asignación de las MV en diferentes nodos. Teniendo en cuenta que los parámetros que definen una MV son los recursos de CPU y memoria.

Bajo este problema la restricción más importante es la de reducir el número de nodos utilizados. A su vez se ha de minimizar el número de reasignaciones para evitar sobrecargas de red, debido a que éstas implican migraciones en caliente de MVs. Por otra parte se ha de garantizar un balanceo de carga óptimo.

Bajo estas restricciones se implementará un sistema que permita realizar todo el proceso de asignación de recursos que intente reducir el tiempo ejecución del algoritmo.

Aplicando programación dinámica la complejidad temporal para este problema es de tipo exponencial $n^{O(k)}$, siendo $n$ el número de objetos o MV y $k$ el número de contenedores o nodos.

Evidentemente el coste es elevado, por lo que la primera sugerencia es aplicar algoritmos de aproximación tal como FFD, BFD ó WFD (ver apéndice A). Utilizando el First-fit decreasing las MVs son ordenadas de forma decreciente según recursos y las asigna en el primer nodo que encaje. Best-fit decreasing ordena las MVs de forma decreciente y las asigna en el nodo más ocupado posible. El Worst-fit decreasing ordena también de modo decreciente pero las asigna en el nodo menos ocupado. Para una información más completa sobre las características de estos algoritmos ver apéndice.

Aplicando estas aproximaciones se pueden obtener tiempos polinomiales del orden $(nlogn)$.

El problema que surge al implementar estos algoritmos deriva en que no garantizan una óptima solución para la asignación de recursos en los nodos. Estos algoritmos no garantizan un correcto balanceo de carga, ya que solo tienen en cuenta la minimización del número de nodos activos.

Para garantizar un balanceo de carga correcto es necesario un algoritmo más complejo temporalmente. Esto se debe a que se tendrían que realizar más cálculos para determinar la idoneidad de la asignación de las MV en determinados nodos.

De este modo se procede a dividir el problema en dos. Un algoritmo que se encargue del balanceo de carga de los nodos, y otro dedicado a la consolidación de estos.

### 4.2.1. Algoritmo de Balanceo

El algoritmo de balanceo tiene como función asignar las máquinas virtuales que el usuario va creando en el mejor nodo disponible. El mejor nodo disponible será aquel que mediante la asignación de la máquina virtual guarde un valor de carga equilibrado con respecto a los otros nodos. Este algoritmo observa la situación del sistema, el grado de ocupación de los nodos y el número de recursos disponibles para poder determinar el nodo idóneo al que se le asignará el recurso.

El principal problema de aplicar un algoritmo que garantice el balanceo de carga es la complejidad temporal que supone definir el nodo correcto.

Por este motivo se ha planteado la implementación de dos tipos de algoritmos diferentes. Uno de menor complejidad y que actúa automáticamente y otro que lleva la carga más importante del trabajo y que es ejecutado sólo cuando es estrictamente necesario.

De esta forma se aplicará un algoritmo online, con respuesta inmediata. Un algoritmo offline que analiza las restricciones impuestas y determina las migraciones de maquinas virtuales entre nodos.

#### *4.2.1.1. Algoritmo Online*

El algoritmo Online consistirá básicamente en la implementación de un First Fit y Max Rest Fit según se den dos condiciones:

Se aplicará el algoritmo First Fit cuando exista únicamente un nodo (líneas 1-9 fig.4.2). Se irá aplicando este algoritmo que consiste en asignar la MV con los recursos demandados en el primer nodo activo disponible.

Se aplicará el algoritmo Max Rest Fit para el resto de casos(líneas 9-15 fig. 4.2), es decir cuando la cantidad de nodos activos sea superior a un nodo. Este algoritmo consiste en la asignación de la MV en el nodo activo menos ocupado (líneas 9-15 fig. 4.2)

Se aplicará este algoritmo mientras las condiciones lo permitan. Las condiciones vendrán determinadas por si el número de nodos activos pueden ser capaz de acoger los recursos demandados por la máquina virtual.

**input** :
$m_i \leftarrow$ MV actual
$n_j \leftarrow$ Nodo actual

1 **Inicialización**
2 $C_i \leftarrow$ Carga de la Mv actual
3 $M \leftarrow contNodos()$ // Numero de nodos activos
4 $T \leftarrow calTarget()$ // Valor del target
5 **if** $M \leq 1$ **then**
6 **if** $m_i$ *encaja en el nodo* $n_j$ *y* $C_i$ *es inferior a* $T$ **then**
7 Asignar $m_i$ en $n_j$
8 **else**
9 Activar nuevo nodo $n_{j+1}$ y asignar $m_i$
10 **else**
11 Determinar $k = min\{i|c_i = min_{j=1}^{j=M} c_j\}$,es el índice del nodo con máxima capacidad restante
12 **if** $m_i$ *encaja en el nodo* $k$ **then**
13 Asignar MV $m_i$ en nodo $k$
14 **else**
15 Activar nuevo nodo $n_{j+1}$ y asignarle $m_i$

Figura 4.2: Pseudocódigo del algoritmo online

Como se observa en la figura 4.3 el algoritmo delimita una valor (target) a partir del cual el nodo se considera ocupado. El valor del target es un parámetro variable en función de la

carga total del sistema. Por defecto el target está definido de tal forma que el espacio libre de la suma de todos los nodos activos sea igual a la carga de un nodo. Se ha procedido a fijar este valor de target para garantizar la redundancia de un nodo que permita suplir las posibles caídas del sistema.

En el apartado a) de la fig. 4.3 la aplicación es inmediata ya que se trata de un solo nodo y al añadirle los recursos máximos de la maquina virtual entrante no supera el target definido. En el apartado b) de la fig. 4.3 la máquina virtual entrante es asignada en el nodo de menor carga. Para este caso no solo se ha tenido en cuenta el nodo con más recursos libres como máximo sino el que más recursos libres tiene realmente. Para ello se utilizan los valores que proporciona el módulo de monitorización de recursos del sistema. En el apartado c) de la fig. 4.3 se observan nodos cargados y que al asignar una máquina virtual superaría el target estimado. De este modo el algoritmo procede a activar un nuevo nodo y volver a implementar el algoritmo y asignar los recursos de la máquina virtual.

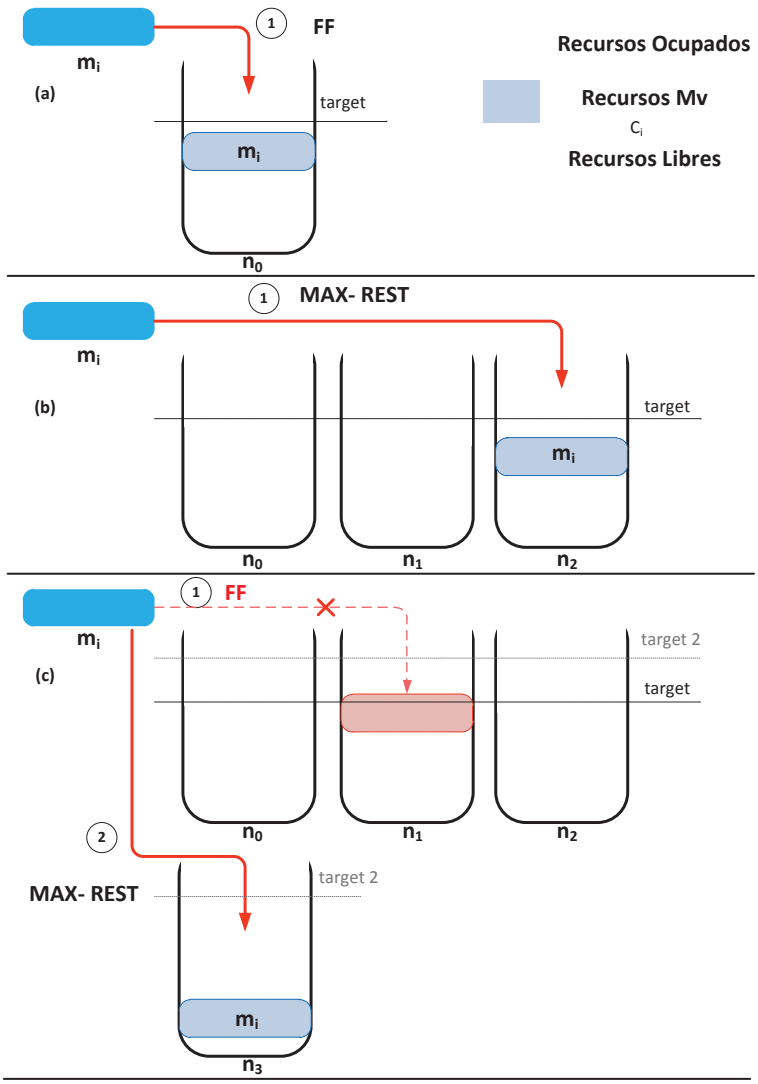


Figura 4.3: Ejemplo de implementación de Algoritmo Online

#### *4.2.1.2. Algoritmo Offline*

Este algoritmo tiene una complejidad temporal más elevada, ya que determina el grado de ocupación de los nodos.

En líneas anteriores se expuso la idea por la cual se utilizan dos algoritmos para la implementación del balanceo de carga de los nodos. Era necesario establecer esta separación para disminuir la complejidad temporal que conlleva asignar de forma óptima los recursos cada vez que sean requeridos por una nueva máquina virtual. Evidentemente a medida que el número de máquinas virtuales aumenta el tiempo de aplicación del algoritmo se incrementa.

Otro de los motivos por lo cual se aplica esta idea es cumplir con otra de las restricciones definidas con anterioridad, la de minimizar la carga de red que supone la reestructuración. Es decir, no era viable que el sistema sea reestructurado cada vez que se añadía una máquina virtual nueva. Ello supondría migraciones muy frecuentes de MVs entre nodos y en consecuencia produciría congestión en la red.

Es por ello que se establecen los targets de ocupación de los nodos. Estos targets delimitan el valor de carga a partir del cual se puede considerar ocupado un nodo. Estos targets son variables en función del número de nodos y carga total de recursos del sistema.

Precisamente la aplicación del algoritmo offline vendrá precedida por un intento de sobrecarga de un nodo. Es decir, si los recursos de una nueva máquina virtual agregada sobrepasan el target del nodo menos cargado se procede primero a la activación de un nuevo nodo y luego a asignarla a éste. A partir de aquí se procede a aplicar el algoritmo offline.

Este algoritmo se divide en diferentes fases.

Primero se encarga de calcular la carga actual de los nodos activos. A partir de este valor estima el valor medio de carga que debería tener los nodos para garantizar un balanceo equilibrado (línea 2 fig. 4.4). A este valor se le denominará *balance*.

El siguiente paso consiste en calcular la desviación respecto a la media estimada (línea 5 fig. 4.4). Esto permite definir aquellos nodos que estén por encima y debajo del valor medio de la carga de trabajo ideal o *balance*, Nodos Over y Under respectivamente.

El algoritmo intentará compensar la carga de los nodos por debajo de la media con la migración de máquinas virtuales provenientes de nodos sobrecargados. Para realizar este trabajo se ha tener en cuenta la restricción de minimizar el número de migraciones.

El problema es escoger la colección correcta de máquinas virtuales que sumadas no superen los objetivos marcados $B_j$, definido como la cantidad de recursos que sobrepasan el *balance* (Nodos Over) o la cantidad de recursos que son necesarios para conseguir llegar al *balance* (Nodos Under). Este el clásico problema de suma de subconjuntos (SSP), expuesto previamente.

Bajo este tipo de problemas existe la restricción en la cual la suma de los valores del conjunto no ha de superar nunca al valor objetivo. Respecto a esta restricción propia

del SSP se puede ser flexible respecto a la consecución del valor objetivo. Es decir se establece un margen de valores alrededor de los valores objetivo, en los cuales se puede considerar el resultado final como óptimo.

Para solucionar este SSP se utiliza un método de aproximación rápida. Este método consiste en realizar una aproximación de la solución ideal, pero que permite reducir el tiempo de aplicación.

De este modo el problema SSP definido con anterioridad sufre una modificación debido a las nuevas restricciones demandadas para solucionar este problema, formalmente:

Dado un conjunto de enteros positivos $A = \{a_1, ...., a_n\} \in \mathrm{N}$ con $B \in \mathrm{N}$ que representa el límite. El problema consiste en encontrar un vector $X = \{x_1, ...., x_n\}$ que representa el subconjunto solución, cuya suma sea el máximo valor cercano al límite $B$. Con la restricción de minimizar el error absoluto en comparación a una solución óptima cuyo vector está representado por $S$ y cuya suma de valores $V_s$ es lo más cercano a $B$ (4.13):

$$max \sum_{i=1}^{n} a_i x_i \tag{4.11}$$

donde;

$$x_i = \begin{cases} 1 & \text{si } a_i \text{ está en el subconjunto solución} \\ 0 & \text{otros casos} \end{cases} \qquad (i = 1, ...., n) \tag{4.12}$$

sujeto a;

$$min\ \delta(x) = V_s - \sum_{i=1}^{n} a_i x_i \tag{4.13}$$

El vector $S$ es un subconjunto de A y que constituye una solución óptima al problema. En (4.13)se define el error absoluto para una solución $X = \{x_1, ...., x_n\}$.

El problema radica en encontrar aquel vector cuya suma minimice el error en comparación a una solución óptima. Como se puede observar en la nueva definición del problema no existe restricción respecto a superar el valor $B$.

Para solucionar este problema se procede a estimar, para cada nodo $j$ por encima del valor medio de carga calculado, el subconjunto de máquinas virtuales a migrar cuyas suma de recursos permitan acercarse al valor objetivo $B_j$. Este valor objetivo es directamente la desviación respecto al valor medio de carga calculado (líneas 7-40 fig. 4.4).

El proceso de elección de máquinas virtuales es el siguiente:

Se analizará la idoneidad del subconjunto de MVs teniendo en cuenta que el resultado final puede ser inferior o superior al objetivo $B_j$.

La máquina cuyos recursos encajen directamente con el objetivo $B_j$ para que ese nodo alcance el balanceo serán marcados directamente como candidatos a ser migrados (líneas 11-13 fig. 4.4) y se pasará al análisis del siguiente nodo.

Para cada nodo se irá determinando el subconjunto de máquinas cuya suma de recursos encaje con el valor objetivo $B_j$ determinado. La idoneidad de la asignación de las MVs será evaluada en función del error respecto a la media. Este error es el valor de recursos necesarios hasta conseguir el valor objetivo $B_j$ dado una vez la máquina haya sido designada como parte del subconjunto (líneas 14-35 fig. 4.4). Es por ello que el valor de error se va actualizando a medida que se van designando las máquinas que pertenecerán al subconjunto final. Para la correcta selección del subconjunto de máquinas virtuales, se ha estimado analizar de forma paralela dos tipos de ramificaciones. La primera es la que permite obtener un subconjunto de máquinas virtuales cuya suma de recursos sea inferior al objetivo $B_j$. La otra ramificación permite superar este valor.

Una vez se ha realizado el proceso de definición de ambas listas de máquinas virtuales, se procede a estimar la mejor solución posible. Para ello se compara el error absoluto que se ha obtenido mediante ambas ramificaciones (líneas 33-39 fig. 4.4)y se elige la combinación que minimice el error. Consiguiendo finalmente la lista de máquinas que pueden ser migradas de este nodo sobrecargado.

Una vez definidos los nodos infrautilizados se procede a migrar el subconjunto de máquinas desde los nodos sobrecargados. Para ello se utiliza el valor de utilización real que tienen los nodos infrautilizados (línea 2 fig. 4.5) mediante los datos que proporciona el agente de monitorización.

Este algoritmo de aproximación para resolver un SSP está basado en el trabajo [22]. La diferencia de esta publicación respecto al problema que se resuelve en este TFC radica en que para la asignación de MVs en los diferentes nodos no requiere hacer una búsqueda exhaustiva de posibles conjuntos solución. Esto se debe a que el espacio de valores del conjunto total de MVs es limitado, en otras palabras los valores de RAM utilizados por el usuario están limitados a valores conocidos [128,256,512,1024,2048,...,máx. RAM de un nodo]MB y por lo tanto el campo de búsqueda es reducido. Esto permite utilizar dos tipos de subconjuntos candidatos a la mejor solución. Aquellos que sobrepasan el límite y aquellos que están por debajo de él. De este modo se reduce el tiempo de estimar otros posibles subconjuntos solución.

En la figura 4.6 se puede observar una implementación del algoritmo offline. En un primer paso se observa el estado inicial en la que se encuentran los nodos para este caso. Se puede ver que proviene de aplicarse un Max-Rest y posee un alto nivel de descompensación de carga.

Para solventar este problema se recurre al algoritmo offline. Primero fija el valor de *balance* y estima que el nodo $n_4$ está infrautilizado. Del mismo modo que estima que los nodos $n_1$ y $n_2$ están sobrecargados y el nodo $n_0$ compensado.

Se procede a determinar el subconjunto de máquinas virtuales de los nodos sobrecargados que serán migrados al nodo infrautilizado, tal como se muestra en el último gráfico de la figura 4.6. El resultado final es un conjunto de nodos cuyos recursos están equitativamente repartidos por lo tanto están balanceados.

```
1  Inicialización
2  Definir el valor medio de la carga (balance) m̂
3  NUnder := [0,...,0],MvUnder := [0,...,0] , MvOver := [0,...,0]
4  for Todos los Nodos activos n_j j = 1,2,....N do
5      D_j ← Calcula el valor de la desviación respecto al valor medio de carga del nodo
6      resetListas()
7      if D_j > 0 then
8          Definir:
9          δ1, δ2,B_j ← D_j                                  // error1,error2,target
10         for Todas las MVs activas m_i i = 1,2,....N do
11             if Recursos de m_i → C_i = B_j then
12                 Encajar maquina m_i en n_j y saltar al n_{j+1}
13             else
14                 if C_i = δ1 then
15                     Añadir m_i en MvUnder
16                     Reinicializar δ1
17                 else
18                     if C_i > B_j then
19                         abs ← |B_j − C_i|
20                         if abs < δ1 & abs < δ2 then
21                             δ2 ← abs
22                             resetMvOver()
23                             Añadir m_i en MvOver
24
25                     else
26                         if C_i < δ1 then
27                             δ1 ← |δ1 − c_j|
28                             Añadir m_i en MvUnder
29                         else
30                             Abs ← |c_i − δ1|
31                             if abs < δ1 & abs < δ2 then
32                                 resetMvOver()   // reinicializa lista MvOver
33                                 Copia lista MvUnder a MvOver
34                                 δ2 ← abs
35                                 Añadir m_i en MvOver
36
37
38         if δ1 < δ2 then
39             Asigna las MVs situadas en la lista MvUnder como candidatas a migrar
40         else
41             Asigna las MVs situadas en la lista MvOver como candidatas a migrar
42     else
43         Añadir nodo n_j en lista NUnder
```

Figura 4.4: Pseudocódigo del algoritmo offline

```
1 for Todos los Nodos activos de NUnder ó infrautilizados n_j j = 1,2,....N do
2     Q_j ← monitoringResource()                    // Valor real de carga
3     γ ← |Q_j − m̂|                                 // Valor absoluto error
4     OrderMv()
5     for Todos los Mvs activas candidatas a migrar m_i i = 1,2,....M do
6         if C_i <= γ then
7             Asignar MV m_i en nodo n_j
```

Figura 4.5: Pseudocódigo de ejecución algoritmo offline

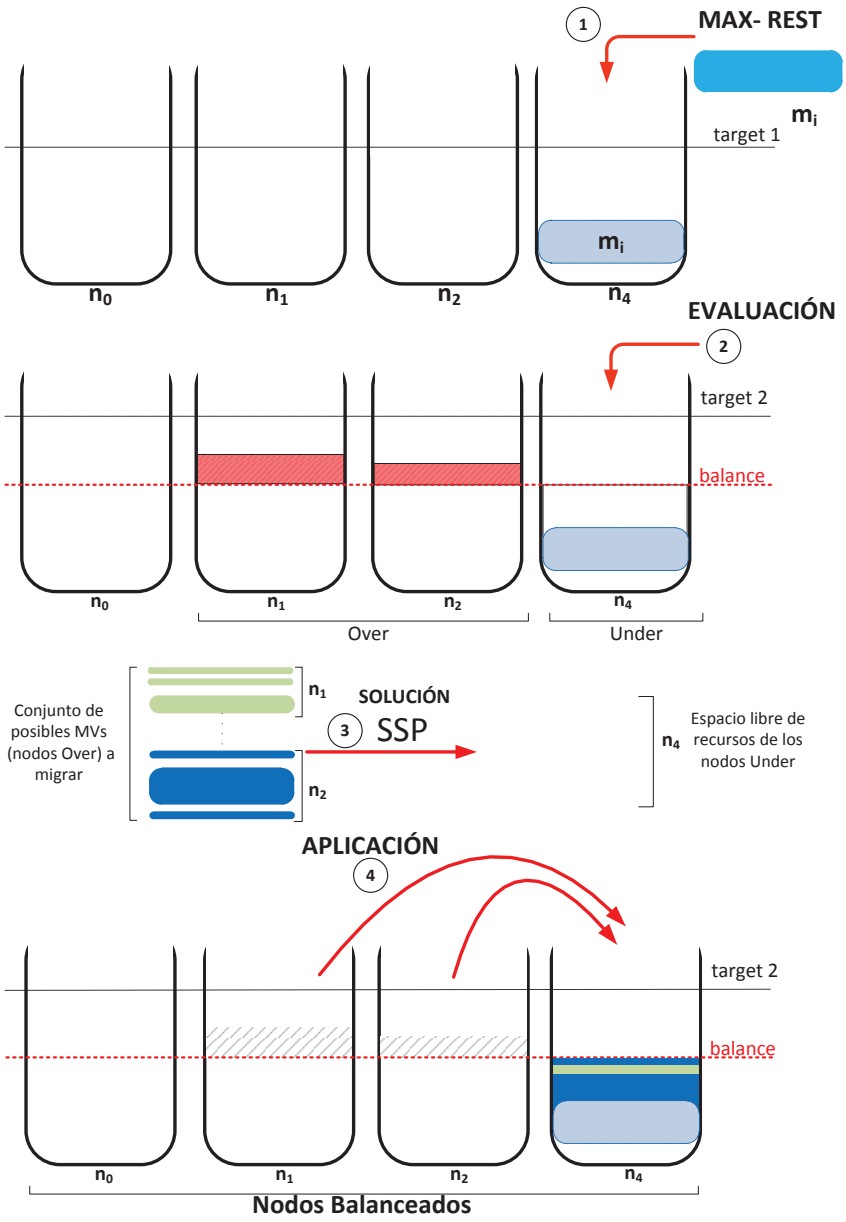


Figura 4.6: Ejemplo de implementación de Algoritmo Offline

### 4.2.2. Algoritmo de Consolidación

Del mismo nodo que es necesario implementar un sistema de balanceo de carga cuando las condiciones sean las adecuadas, es importante minimizar el número de nodos activos según las necesidades de recursos tenga el sistema.

La aplicación de la consolidación vendrá precedida por la eliminación o apagado de una máquina virtual, es decir la liberación de recursos.

La liberación de recursos será el criterio a partir del cual se iniciará todo el proceso de consolidación.

**input** : Nodo al que se libera recursos
**output**: Aplicación de Consolidación

```
1  δ ← Umbral límite de carga
2  Ω ← resourceTotal()
3  M ← maxRecurso()
4  K ← maxFreeNodo()
5  if contNodos() > 1 then
6      if El número de nodos es inferior a 3 then
7          τ ← M/2
8      else
9          τ ← [contNodos() − 2]·maxNodoResource()
10     Δ = τ - Ω
11 if M > K ó Δ < δ then
12     return False
13 else
14     return True
```

Figura 4.7: Pseudocódigo de la evaluación de Consolidación

La implementación del algoritmo está definida en dos fases.

La primera fase consiste en determinar la idoneidad de la eliminación de un nodo y la migración de sus MVs a otros garantizando un correcto balanceo de carga. En la figura 4.7 muestra el pseudocódigo del algoritmo que evalúa la consolidación. Cuando una máquina virtual es eliminada o apagada se procede a activar este código. La entrada del código es el nodo al que pertenece la MV a liberarse. En las líneas 1-4 de la fig. 4.8 se definen los valores del umbral de límite de carga ($\delta$). Este valor es variable y permite definir un margen de valores a partir del cual se considera correcto aplicar la consolidación. Mediante la implementación de este parámetro se puede controlar la periodicidad de intentos de consolidación, por ejemplo para evitar que la eliminación de una máquina virtual con recursos ocupados mínimos y con poca transcendencia en el global del sistema pueda provocar una reestructuración.

Además, se define el recurso total ocupado máximo por el conjunto de máquinas ($\Omega$). Precisamente de la diferencia del parámetro ($\tau$), valor a partir del cual el sistema se con-

sidera cargado, y $\Omega$ da como resultado $\delta$. Esta diferencia se compara con el umbral límite de carga para proceder a aplicar el algoritmo (línea 11). En la línea 3 se define el valor $M$ que expresa los recursos máximos del nodo candidato a ser eliminado. En la línea 4 se estima el valor $K$ que permite obtener los recursos disponibles del nodo más liberado de carga de trabajo. Estos parámetros son obtenidos al emplear información aportada por el agente monitorizador de recursos.

Ambos valores se comprueban para dar paso finalmente a la respuesta de implementación del algoritmo de consolidación.

La segunda fase consiste en aplicar la liberación de la máquina virtual en caso que la evaluación previa haya sido positiva, por lo que procede a migrar las MVs del nodo a eliminar. En la figura 4.8 se muestra el pseudocódigo del algoritmo empleado para ejecutar la consolidación. Se introduce como entrada la máquina virtual a eliminar y el nodo que la hospeda. El algoritmo va colocando cada máquina virtual del nodo a eliminar en los demás nodos disponibles en función del espacio libre que tengan estos (líneas 3-7 fig. 4.8). Para ello se emplean los datos que proporciona el agente de monitorización de recursos.

```
   input :
     n_D ← nodo a eliminar
     m_D ← máquina a liberar
1  Inicialización
2  orderMv()
3  for Todos las máquinas activas del nodo n_D m_i i = 1,2,....M do
4      orderNodos()
5      for Todos los nodos activos excepto n_D n_j j = 1,2,....N do
6          if monitoringResource() > C_i then
7              Migrar m_i de n_D a n_j
8              break
9  if No existe máquinas activas en n_D then
10     Eliminar n_D
```

Figura 4.8: Pseudocódigo de consolidación

En la figura 4.9 se muestra un ejemplo de funcionamiento del algoritmo de consolidación. Se observa que el primer paso consiste en liberar los recursos del nodo $n_0$ que contiene a la MV a liberar. Luego se procede a evaluar si es adecuado proceder a la consolidación. Se determina que el nodo menos cargado es el $n_1$ y finalmente la máquina virtual del nodo candidato a eliminar $n_0$ con más carga para determinar si se procede finalmente a la aplicación de la consolidación. A continuación si la evaluación es positiva se procede a migrar los distintas máquinas virtuales de $n_0$ al resto de nodos. Estas máquinas se van asignando al resto de nodos manteniendo un equilibrio de carga entre todos ellos. Finalmente el nodo es eliminado y el sistema ha sido consolidado.

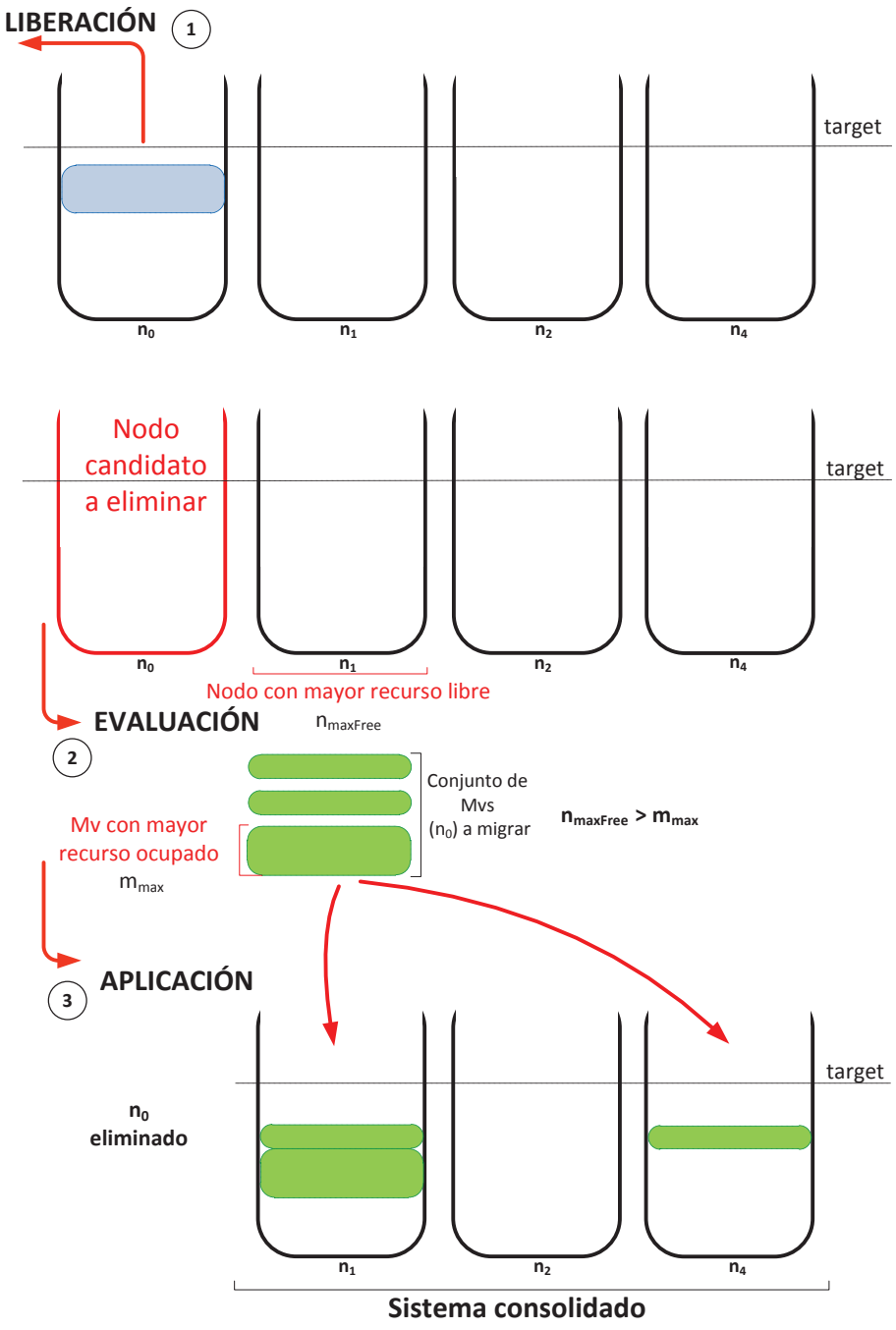


Figura 4.9: Ejemplo de implementación del algoritmo de consolidación

# CAPÍTULO 5. IMPLEMENTACIÓN

## 5.1. Consideraciones iniciales

Para implementar el módulo IA dentro del sistema inicial se han utilizado tecnologías ya establecidas antes de iniciar este TFC. Esto se debe a que el proyecto abarca otros grupos de trabajo que han realizado de manera paralela la implementación de cada bloque que corresponde con el sistema gestor de recursos.

Por este motivo, antes de implementar el algoritmo se establecieron una serie de condiciones iniciales. Entre ellas el tipo de lenguaje de programación a utilizar, el modelo de base de datos que se empleará, el acceso a la base de datos y la gestión de los métodos empleados para unificar proyectos.

### 5.1.1. Lenguaje de programación empleado: Python

Se ha utilizado Python principalmente porque es el lenguaje de programación empleado anteriormente por el grupo de trabajo del MediaCAT y se requiere compatibilidad para garantizar la unificación de los proyectos.

Python es un lenguaje de programación interpretado y multiparadigma, ya que puede soportar programación orientada a objetos como programación funcional. Este lenguaje de programación fue ideado con el objetivo principal de implementar código legible. Es un lenguaje de programación compatible con la licencia GNU por lo que es de libre distribución. Las características que se destacan de este lenguaje es que es multiplataformas (Unix,Ms Windows, Mac,etc), utiliza indentación en bloque y posee una sintaxis sencilla similar al pseudocódigo.


Figura 5.1: Logotipo Python

Una de las ventajas que aporta este lenguaje en la implementación del módulo asignador de recursos es su legibilidad. Este es un factor importante para el mantenimiento del código en futuras mejoras por distintos grupos de trabajo.

Al ser un lenguaje interpretado el tiempo de ejecución es menos rápido que uno compilado como C. Esto se puede traducir en una menor eficiencia en aplicar el algoritmo pero los tiempos que se manejan no son muy críticos, debido principalmente a que el tiempo estará limitado por la capacidad del sistema de aplicar las migraciones. Los tiempos que

el sistema utilice para definir las MV, su instalación, configuración son más largos en comparación a la propia aplicación de los algoritmos contenidos en el módulo IA.

### 5.1.2. Sistema de Base de datos (RDBMS): MySQL

MySQL es un sistema de gestión de bases de datos relacional (RBDMS por sus siglas en Inglés) que funciona como un servidor que provee acceso a múltiples usuarios a una serie de bases de datos. El código fuente de MySQL está disponible bajo licencia GNU GPL pero con restricciones para uso privado. Se caracteriza por ser soportado por una gran variedad de tecnologías como PHP, Perl, Java ó Python y por tener módulos y extensiones ampliamente documentados.


Figura 5.2: Logotipo MySQL

La base de datos se emplea principalmente por el módulo de monitorización para incluir los valores de utilización de recursos del sistema. El módulo IA emplea esta base de datos para que los algoritmos contenidos en él puedan realizar consultas puntuales y también para guardar la configuración luego de aplicar una reestructuración.

La estructura de la base de datos que se ha empleado para las consultas y modificaciones que realiza el módulo IA se puede observar en la figura 5.3

Principalmente el modulo IA utiliza las tablas *nodes* y *virtualmachines* para hacer las consultas y guardar los resultados de las asignaciones. Una vez ejecutado el algoritmo de asignación(balanceo ó consolidación) los resultados se ven reflejados en el campo $temp\_alloc$ que contiene el posible nodo al que irá la MV. La entrada $check\_alloc$ es un valor temporal que utiliza el módulo IA para definir si una MV ya ha sido asignada. Ambos en la tabla *virtualmachines*. Por otra parte, al momento de realizar las estimaciones para poder establecer el nodo más adecuado es necesario tener un campo que permita guardar los valores de recursos libres del nodo, este campo es el $rr\_temp$ situado en la tabla *nodes*.

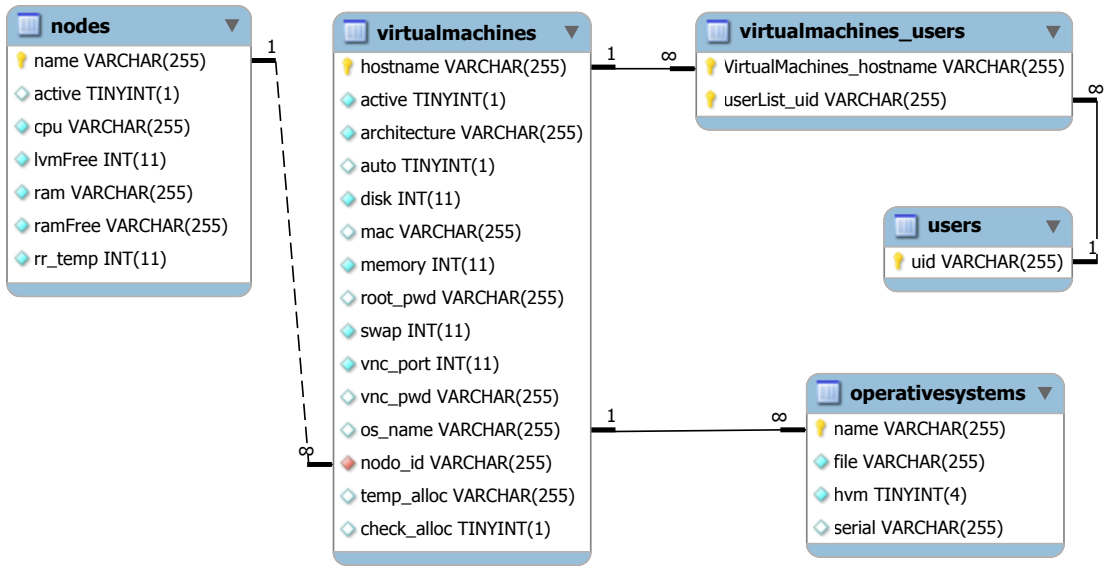


Figura 5.3: Diagrama EER

### 5.1.3. Sistema ORM: SQLAlchemy/SQLSoup

Para que el módulo de la IA pueda interactuar con la base de datos se requiere emplear una herramienta que permita el mapeo objeto-relacional o ORM por sus siglas en inglés.

El mapeo objeto-relacional es una técnica de programación que permite vincular datos persistidos (datos guardados que mantienen su estado a los procesos que lo manipulan) en objetos virtuales. De este modo el contenido de la BBDD pueden ser utilizados por lenguajes de programación orientado a objetos.

SQLAlchemy es una librería de acceso a SQL y a su vez un ORM para el lenguaje de programación Python. SQLAlchemy permite ejecutar sentencias SQL utilizando un lenguaje de programación de alto nivel (Python) sin modificar sustancialmente el código en este lenguaje. Proporciona herramientas que permiten mapear clases y objetos en tablas de BBDD .

Para las modificaciones en el espacio reservado y creación de entradas que requiera implementar el módulo IA se utiliza SQLAlchemy.


Figura 5.4: Logotipo SQLAlchemy

SQLSoup es una extensión de SQLAlchemy que permite acceder a las tablas de una base de datos ya creada sin necesidad de declarar clases mapeadas.

Se utiliza SQLSoup para los casos que no requieran modificación de la base de datos, para realizar consultas directas utilizando python sobre la base de datos.

## 5.2. Características del algoritmo

Para implementar los algoritmos de balanceo y consolidación del sistema gestor de recursos se ha dividido el código en diversas clases. A continuación se explica la acción que realiza cada método o función y las relaciones que existen entre ellas y con otros módulos externos que forman parte de la monitorización.

### 5.2.1. Funciones

El núcleo del código del módulo IA se encuentra en el módulo python *Manager* tal como se puede observar en la fig 5.5. El módulo Manager esta compuesto por la clase *Virtualmachines* (Recursos.py) es empleada para instanciar objetos con las características de las MVs reduciendo el acceso a la BBDD. La clase *Migrate* contiene el método estático que permite la aplicación de las migraciones de MVs entre nodos.

La clase *Ia* (Principal.py) contiene los métodos necesarios para inicializar la asignación de las MV en los diferentes nodos del sistema utilizando los métodos que dispone el módulo.

La clase *Scheduler* contiene todos los métodos o funciones necesarias para llevar a cabo la asignación de las máquinas virtuales en los nodos. Se sitúan los métodos que implementan los algoritmos de balanceo y consolidación. La descripción de cada método es la siguiente:

- **addResource()**: Este método se encarga de gestionar la asignación de una máquina virtual nueva. Aplica el algoritmo online para asignar la MV y determina si es posible aplicar la asignación offline por lo que da paso a la aplicación del algoritmo de balanceo.
- **removeResource()**: Gestiona la liberación de recursos al eliminar una máquina virtual. Decide si es aplicable el algoritmo de consolidación.
- **onMv()**: Tiene la misma función que el *addResource()* pero es aplicable a máquinas virtuales que son activadas luego de haber estado apagadas. Por lo que también determina la aplicación de balanceo a emplear.
- **offMv()**: Gestiona la liberación de recursos debido al apagado de una máquina virtual. Decide la aplicación del algoritmo de consolidación.
- **exchangeResource()**:Este método tiene como argumento de entrada la acción a realizar producto de la modificación de una MV. Una máquina virtual puede ser modificado cuando el usuario decida asignar o reducir más recursos de la MV. Este método se emplea para definir el paso a una consolidación o balanceo, y los tipos de algoritmo a emplear.
- **detectConsolidation()**: Este método sirve de soporte a *exchangeResource* ya que permite determinar si la modificación de una MV da lugar a la aplicación de un algoritmo de consolidación o balanceo.

- **ff()**: Método que implementa el algoritmo First Fit sin control de target.
- **ff2()**: Método que implementa el algoritmo First Fit con control de target para definir el valor a partir del cual se considera un nodo ocupado.
- **ff2Exchange()**: Método que aplica el algoritmo First Fit para el caso de aumento de recursos de una MV ya creada.
- **maxRest()**: Este método aplica el algoritmo Max-Rest sin control de target.
- **maxRest2()**: Aplica el algoritmo Max Rest con control de target.
- **maxRest2Exchange()**:Método que aplica el Max-Rest para el caso de aumento de recursos de una MV ya creada.
- **algBalanceo()**:Este método implementa el algoritmo Offline. Este método devuelve las nuevas asignaciones de las MVs luego de aplicar la reestructuración.
- **algConsolidacion()**: Mediante este método se implementa el algoritmo de consolidación. Por lo cual se encarga de determinar el nodo a eliminar y las nuevas asignaciones de las MVs que pertenecen a este.
- **alConsoliExchange()**: Implementa el algoritmo de consolidación para el caso de reducción de recursos de una MV ya creada.
- **evalRemove()**: Evalúa la idoneidad de implementar el algoritmo de consolidación para el caso en el que se elimina o apaga una MV, librándose recursos.
- **evalRmExchange()**:Este método implementa el algoritmo que evalúa la idoneidad de la consolidación en los casos de reducción de recursos de una MV (edición MV).
- **calCarga()**:Este método se encarga de calcular la desviación respecto al balance ideal que presenta un nodo.
- **ramTotal()**: Devuelve la cantidad real de RAM empleada por el sistema. Retorna consulta sobre la base de datos.
- **cpuTotal()**: Devuelve la utilización real de CPU empleada por el sistema. Retorna consulta sobre la base de datos.
- **ramOcu()**: Devuelve la cantidad real de RAM utilizada de un nodo que se introduce como entrada.
- **ramFree()**: Devuelve el espacio libre de RAM real que emplea un nodo que se introduce como entrada.
- **cpuUsage()**: Devuelve la utilización de CPU real de un nodo que se introduce como entrada.
- **contNodos()**: Devuelve el número de nodos activos del sistema.
- **nodoMv()**: Devuelve el nodo que corresponde una determinada MV.
- **machineMigrate()**: Método que se encarga de determinar si una MV puede ser migrada.

- **salidaIp()**: Método que devuelve la IP del nodo al que finalmente va asignado una MV.
- **ipNodoMachine()**: Método que devuelve la IP del nodo que contiene una MV.
- **ipNodotemp()**: Método que devuelve la IP del nodo que contiene la MV luego de una asignación temporal.
- **resetTemp()**: Este método reinicializa los valores de los campos empleados por la IA para realizar las estimaciones.

En la figura 5.5 se puede ver la interacción que tiene el módulo *Manager* con los otros módulos que corresponden tanto a la interfaz con el usuario como el módulo de monitorización.

En la figura 5.6 se puede observar la dinámica del código. Se observa que existen 5 casos de uso que tiene en cuenta el módulo *Manager* Estos casos de uso son tratados por la clase *Principal* que deriva las acciones sobre los métodos de la clase *Scheduler*.

Para los casos donde se añaden recursos, que son la creación de una nueva MV o la activación de una MV ya creada, se aplican los métodos *addResource()* y *onMv()* respectivamente. En caso que solo exista un nodo y la máquina virtual nueva o activada no sobrepase el target estimado, el proceso consiste en aplicar el método *ff2()* y consigue la asignación empleando el algoritmo online. En caso exista más de un nodo se procede directamente a la aplicación del método *maxRest()*, y de esta forma asignar la MV nueva o recursos incrementados. En caso se sobrepase el target estimado se procede a agregar un nuevo nodo y posteriormente a aplicar el método *maxRest()*. A partir de aquí se aplica el método *algBalanceo()* que procede a aplicar el algoritmo offline para conseguir la asignación del recurso o MV agregada y el balanceo de carga de los nodos.

Para los casos en los que se liberan recursos, que corresponden a la eliminación o apagado de una MV se aplican los métodos *removeResource()* y *offMv()* respectivamente. El proceso consiste en aplicar la evaluación de la consolidación con el método *evalRemove()*. Sí el resultado es positivo se procede a aplicar el algoritmo de consolidación con el método *algConsolidacion()* y finalmente eliminando el nodo infrautilizado y reasignando sus MVs. En caso negativo se aplica el método *algBalanceo()* para balancear la carga de los nodos.

En los casos de modificación de las características de las MV se aplica el método *exchangeResource()*. En el caso de incremento de recursos se procede a aplicar el método *maxRestExchange()* o *ff2Exchange()* en función del número de nodos que exista. Si los recursos incrementados superan el nivel de target se genera un nuevo nodo y se aplica el método *maxRestExchange()* para proceder a la aplicación del balanceo con el método *algBalanceo()*.

En el caso de reducción de recursos de una MV se procede a aplicar la evaluación de la consolidación, que consiste en aplicar el método *evalExchange()*. A partir de aquí sí el resultado es positivo se aplica el algoritmo de consolidación *consolidExchange()* y posteriormente la eliminación del nodo. En caso negativo se aplica el balanceo de carga.

### 5.2.2. Modos de funcionamiento

A partir de los métodos implementados y la dinámica del código es posible aplicar modos de funcionamiento para la ejecución en entornos reales.

#### *5.2.2.1. Modo automático*

Es el modo por defecto en el cual trabaja el módulo IA. Bajo el modo de funcionamiento automático el módulo IA aplica el algoritmo de balanceo como el de consolidación de manera automática en base a estimaciones del target, balanceo, estimación de margen de aplicación de consolidación.

Estos parámetros son los establecidos por defecto en el código pero son modificables dependiendo del grado de balanceo o sensibilidad de consolidación se requiera.

#### *5.2.2.2. Modo semi-automático*

El modo de funcionamiento semi-automático permitirá efectuar los algoritmos pero no ejecutarlos directamente. Por lo que la ejecución final de la reestructuración debido a balanceo o consolidación tendrá que tomarla el administrador. Al implementar campos extras para la gestión de la IA en la base de datos se permite guardar los resultados de la aplicación de los algoritmos pero no aplicarlos directamente al sistema. En casos de problemas de red o falta de provisionamiento de recursos, este modo permite al administrador tener la potestad de ejecutar o no la reestructuración.

#### *5.2.2.3. Modo manual*

El modo manual de funcionamiento permitirá efectuar la reestructuración cuando el administrador crea conveniente. Los métodos empleados anteriormente están dispuestos para que sean ejecutados en caso sea necesario efectuar reestructuraciones puntualmente. Empleando este modo permitirá aplicar la reestructuración debido a balanceo o consolidación en periodos determinados de tiempo.

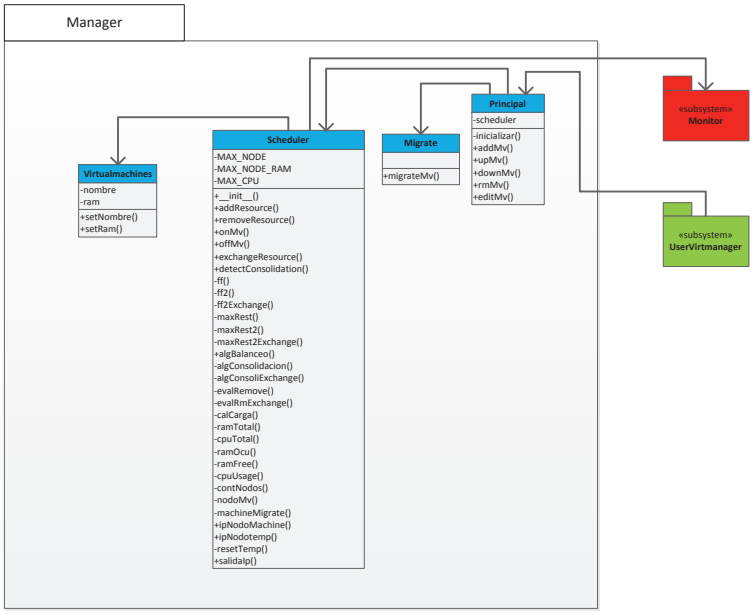


Figura 5.5: Diagrama de clases

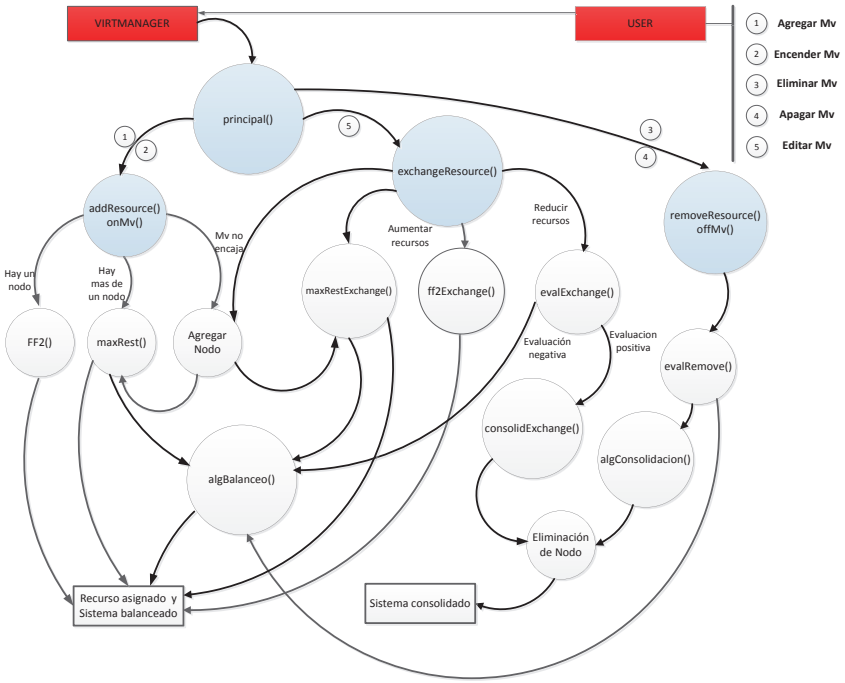


Figura 5.6: Diagrama DFD

# CAPÍTULO 6. TEST Y SIMULACIÓN

Para extraer resultados que permitan demostrar la idoneidad del algoritmo se ha procedido a simular el funcionamiento del módulo IA. La simulación se ha realizado sobre un equipo con procesador Intel®Core2 Duo @ 2.33GHz y memoria RAM de 2 GB.

## 6.1. Simulación

Para poder realizar la simulación del funcionamiento del módulo IA se han definido los valores de las MV considerando situaciones más probables. Se han definido una serie de MVs con parámetros RAM siguiendo la frecuencia de aparición que muestra la tabla 6.1. Donde se ha aplica más frecuencia de aparición a aquellas MV con cantidades de RAM alrededor de 1024 ó 2048 MB, se ha de tener en cuenta que el sistema principalmente brinda servicios de escritorio virtualizado y estos valores serán los más comunes.

| **RAM MV** | **256MB** | **512MB** | **1024MB** | **2048MB** | **3072MB** |
|---|---|---|---|---|---|
| (%)Frecuencia | 9.1 | 18.2 | 36.3 | 27.3 | 9.1 |

Cuadro 6.1: Frecuencia de utilización MV

Se han considerado 88 MVs y la entrada de estás al sistema gestor de recursos se ha realizado de manera aleatoria. Se puede ver un ejemplo de la asignación que va realizando el módulo IA en el apéndice D. Se compara el algoritmo de balanceo empleado con el algoritmo WFD con método voraz para evaluar y comparar la solución. A continuación los resultados obtenidos de la simulación.

### 6.1.1. Coste Temporal

Se puede observar en la fig. 6.1 el coste temporal que supone aplicar la solución empleada. Se observa que el resultado para el algoritmo de balanceo y consolidación el coste temporal presenta una tendencia logarítmica en función de las maquinas virtuales empleadas. Esto se debe a que el algoritmo de balanceo realmente no hace una búsqueda exhaustiva sobre todas los nodos, sino que delimita aquel nodo desbalanceado y aplica la aproximación para un problema de tipo SSP. Esto hace que se reduzca el campo de búsqueda y por tanto el tiempo empleado en realizar el equilibrado de carga de recursos de los nodos. En comparación a la solución que emplea el WFD y método voraz, en la que cada vez que se desborda un nodo se procede a aplicar una completa reestructuración. Esta reestructuración supone que cada nodo sea revisado. Esto se debe a que asigna la máquina virtual en el nodo menos cargado (WFD) y en caso de igualdad de condiciones (múltiples máquinas virtuales para múltiples nodos) se aplique el método voraz para evaluar la mejor óptimo local, aunque muchas veces supone revisar todos las MVs de manera exhaustiva.

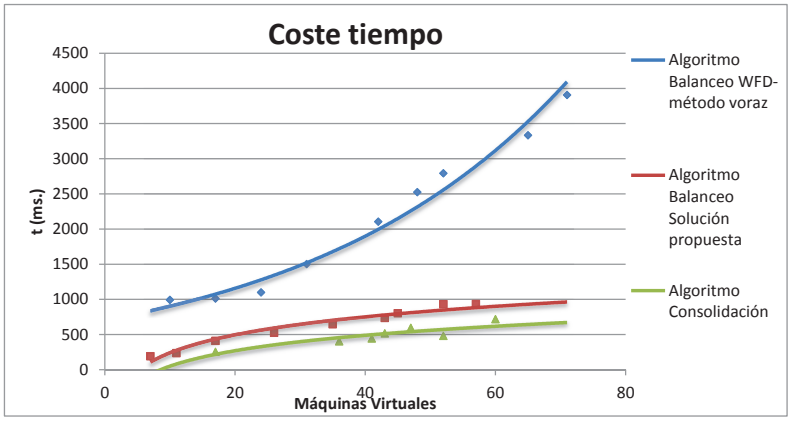


Figura 6.1: Coste temporal

### 6.1.2. Coste Carga de red

El coste debido a las múltiples migraciones que son necesarias para balancear la carga de los nodos es un parámetro importante que se definió al principio como prioritario. La minimización de recursos migrados al aplicar la solución para el balanceo hace que sea practicable en entornos reales.

Se observa en la fig. 6.2 el coste en MB migrados en función de las máquinas virtuales empleadas por el sistema sugiere una tendencia lineal en función de las máquinas establecidas en el sistema. Para la propuesta de algoritmo de balanceo,el valor de recursos migrados nunca supera la carga de un nodo. Esto es producto de que gestiona el equilibrado de carga sin reestructurar completamente las máquinas ya establecidas. Es decir se migran las máquinas estrictamente necesarias para garantizar un valor óptimo de balanceo o equilibrado de carga. El resultado es distinto para el algoritmo WFD con método voraz ya que este algoritmo prioriza la compensación estricta de carga en comparación con el número de recursos migrados. Esto produce un exceso de migraciones que llegan a superar en creces el valor de carga de un nodo.

De igual modo en la fig. 6.3 se muestran los mismos resultados para el número de máquinas virtuales migradas.

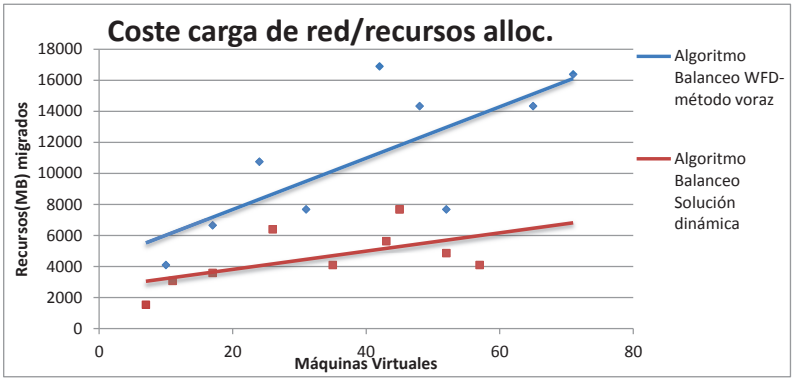


Figura 6.2: Coste migración

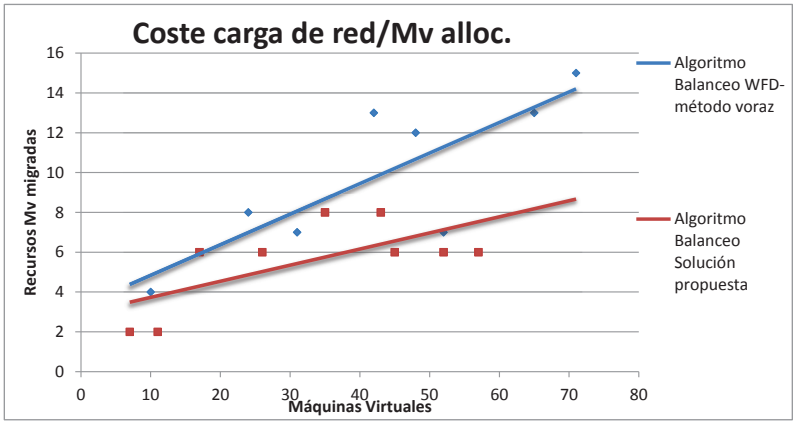


Figura 6.3: Coste migración MV

### 6.1.3. Grado de Balanceo

Para estimar el grado de balanceo final se ha definido el coeficiente de variación. Consiste en determinar el grado de dispersión que presenta respecto a la media. Este valor expresa la relación de la desviación típica entre la media de carga de los nodos. En la fig. 6.4 se puede observar las medidas para ambos algoritmos.

Como era de esperar el algoritmo WFD con método voraz obtiene mejores resultados. Esto se debe a que el WFD permite obtener un perfil equilibrado de los recursos que gestiona. El algoritmo propuesto es más relajado en este aspecto pero tal como se puede observar el grado de descompensación es inferior al 10 % respecto a un balanceo ideal.

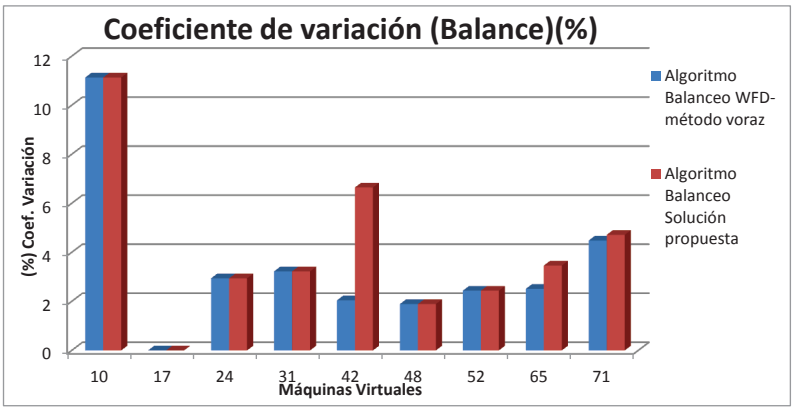


Figura 6.4: Coeficiente de variación de Balanceo

### 6.1.4. Grado de Utilización

El grado de utilización refleja el número de nodos empleados finalmente al utilizar el método propuesto. Se compara el resultado obtenido por el algoritmo propuesto, con el valor ideal que refleja directamente el número de nodos necesarios en función de los recursos totales empleados. El número de nodos debido a una asignación ideal refleja el mínimo

número de nodos necesarios para garantizar la asignación de todos los recursos. Tal como se observa en la fig. 6.5 durante las transiciones donde se emplea el algoritmo de balanceo de carga siempre resulta un nodo superior al valor ideal. Esto se debe al margen por defecto impuesto en el algoritmo. Este margen es producto del target o valor límite a partir del cual un nodo se puede considerar cargado. Esta relajación del objetivo permite garantizar que se eviten sobrecargas y ralentización de los nodos, que pueden afectar al correcto funcionamiento de las máquinas virtuales y al usuario final.

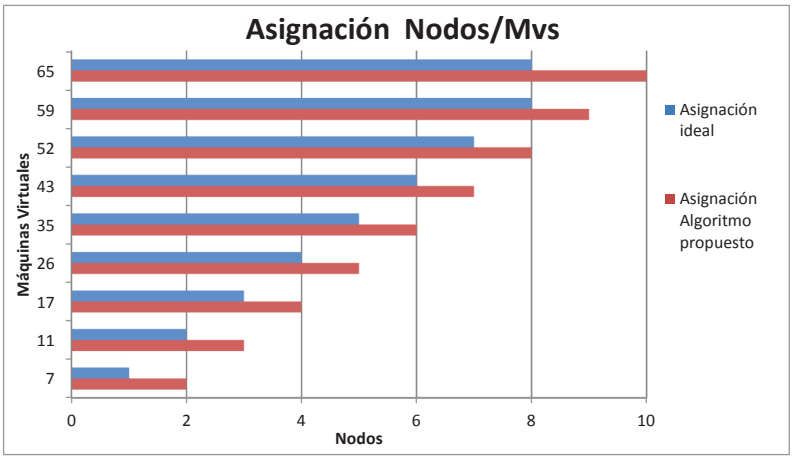


Figura 6.5: Utilización

# CAPÍTULO 7. ESTUDIO MEDIOAMBIENTAL

El continuo crecimiento de aplicaciones y servicios disponibles desde Datacenters está implicando el incremento de la capacidad tanto de infraestructura como de consumo energético de los servidores. Es por ello que el concepto de computación en la nube es imprescindible ya que ayuda a optimizar los recursos del sistema.

El hecho que el consumo energético representa un gran problema en los Datacenters actualmente se ve reflejado en un informe realizado por la Agencia de Protección Medioambiental americana en el que se estimó que entre el 2000 y 2006 el consumo energético se había duplicado y que esperaba se duplicaría otra vez para 2011 [32]. A pesar de ello en un informe independiente realizado por el departamento de ingeniería medioambiental de la Universidad de Stanford, sobre el consumo energético de los Datacenters entre 2005 y 2010 afirma que a pesar que el incremento es significativo se observa una tendencia a contraerse [33].

Entre las posibles causas que se exponen en este informe afirman que se debe a la aparición de tecnologías más eficientes energéticamente y a la implantación cada vez más común de virtualización de servidores.

Precisamente la virtualización que emplea la computación en la nube para proporcionar servicios representa el principal método para reducir el consumo energético.

Profundizando en el ahorro de consumo energético actualmente se utilizan herramientas que permiten la gestión en tiempo real de los recursos del sistema.

Por este motivo se ha empleado un gestor que tiene como principal objetivo el balanceo de carga de recursos del sistema sin dejar de lado la idea de minimizar la utilización de servidores activos. De este modo al implementar el algoritmo una de las restricciones ha sido considerar la idea de la consolidación de servidores. Para ello este trabajo plantea la idea de implementar un *pool* de servidores activos y otro en el cual los servidores se encuentren en modo inactivo, cuya implementación práctica está supeditada a la capacidad que tiene el hardware para soportar modos de ahorro de energía y activación/desactivación de forma remota.

En caso que el hardware lo permita, el algoritmo está preparado para utilizar los servidores activos necesarios minimizando la infrautilización de recursos y con capacidad de redundancia de un nodo. Esta solución supone de forma directa la reducción del consumo energético debido a la infrautilización de recursos de los servidores.

Esta implementación supone un paso inicial para garantizar el mínimo impacto medioambiental del sistema que este TFC está tratando. Este primer objetivo se engloba en lo que el concepto Green IT propone. Precisamente este concepto hace referencia al estudio, practica, diseño, fabricación y utilización de servidores y subsistemas implementados eficientemente con el mínimo impacto medioambiental [34]. Bajo el Green IT se amparan áreas y actividades que incluye la eficiencia energética, la gestión de consumo energético, el diseño de Datacenter y virtualización de servidores.

# CAPÍTULO 8. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURO

Este trabajo se ha centrado en la implementación de un sistema gestor de recursos para una arquitectura diseñada por el MediaCAT de la Fundación i2CAT basada en un sistema clusterizado que permite brindar servicios de escritorio y servidores remoto utilizando virtualización.

Para llevar a cabo el cometido final se han planteado una serie de objetivos a cumplir que a continuación serán revisados.

**El estudio de las condiciones iniciales y estado actual del sistema** han permitido conocer los conceptos básicos en cual se fundamenta la arquitectura del sistema. Del mismo modo se han revisado las distintas técnicas empleadas tanto por Xen como por las soluciones comerciales y publicaciones cuyas ideas han servido para elaborar el sistema gestor de recursos final.

Se ha comprobado el funcionamiento del **live migration** en la arquitectura ya establecida. Esta comprobación permite garantizar que las funciones del gestor de recursos son completamente aplicables en una implementación real.

Por lo que respecta a la **tolerancia a fallos**, el algoritmo final implementado garantiza una resistencia a caídas equivalente a la carga de un nodo.

**El balanceo de la carga de trabajo** es una característica importante del gestor de recursos. Los algoritmos implementados, en base a los resultados obtenidos, garantizan en el peor de los casos una desviación del 10 % respecto a balanceo de carga ideal.

**La escalabilidad** va ligada a los resultados de tiempo de aplicación obtenidos. Los resultados muestran una tendencia lineal en función del número de MVs en comparación a métodos exhaustivos. La reducción del tiempo de aplicación del algoritmo de balanceo es producto de aplicar métodos de aproximación y la reducción del campo de búsqueda de posibles valores que caracterizan a las MVs.

**La monitorización de los recursos del sistema** es un objetivo marcado por el grupo de trabajo MediaCAT y que gracias a su implementación permite al gestor de recursos obtener la información sobre la utilización de los recursos de cada nodo.

Se ha cumplido con el objetivo de **minimizar la información de control y carga de red** implementando un algoritmo de balanceo modelado en base a problemas suma de subconjuntos, en la cual se ha aplicado aproximación para su resolución. Tal como se muestra en los resultados el número de MVs a migrar para garantizar el balanceo es el estrictamente necesario para garantizar un margen reducido alrededor del balanceo de carga ideal. Además se aplican sólo cuando las condiciones lo requiera.

Se garantiza que las reestructuraciones producidas al gestionar los recursos del sistema son **transparentes al usuario final**. La implementación ha tenido en cuenta la reducción del impacto sobre el usuario final minimizando el número de migraciones y la periodicidad

de las reestructuraciones. Este resultado es producto de dividir el problema del balanceo de carga en el algoritmo online, de aplica una asignación inmediata, y en el algoritmo offline aplicado cuando las condiciones lo requieran y que supone migraciones interdominio.

Se ha conseguido garantizar la **consolidación de servidores** implementando un algoritmo que mediante parámetros de utilización estudia la viabilidad del apagado del nodo más infrautilizado. Los resultados demuestran que el número de nodos activos resultantes superan en una unidad el valor mínimo ideal de nodos activos utilizados.

**La implementación del gestor y monitorización de recursos sobre un sistema real** es un objetivo que actualmente está en desarrollo. Los primeros resultados garantizan un correcto funcionamiento del algoritmo de balanceo. Esta primeras pruebas dependen de la unificación de proyectos tanto del monitorizador como del gestor de recursos que actualmente se están desarrollando.

Para cumplir el objetivo marcado de **minimizar el consumo energético** el gestor de recursos define un conjunto de servidores activos e inactivos. Para entornos reales esta funcionabilidad dependerá de la capacidad del hardware de los nodos de garantizar el funcionamiento en modo de ahorro de energía y la opción de activarlos de forma remota.

Finalmente este trabajo con los objetivos iniciales establecidos representa un primer paso para automatizar la gestión de recursos para un sistema basado en el concepto de computación en la nube con arquitectura clusterizada y con servidores virtualizados.

Para **futuras implementaciones** será conveniente aplicar patrones de utilización y estadísticas del consumo de recursos para mejorar la consolidación.

Del mismo modo será conveniente estudiar las herramientas proporcionadas por Xen para gestionar el mecanismo del *Transcendent Memory*. Para ello sería importante trabajar con la API que Xen proporciona para gestionar este mecanismo e implementarlo dentro del gestor de recursos.

Para garantizar la consolidación de servidores será importante implementar en el futuro un sistema de activación/desactivación de nodos del sistema. Éste ha de garantizar la activación/desactivación de los servidores de manera automática y remota, en base a las reestructuraciones que aplique el módulo IA.

Para un mayor control de los recursos del sistema en el futuro será importante descentralizar la cabina de discos, de este modo se pueda añadir la gestión del espacio del disco duro como también la posibilidad de trabajar con nodos con capacidad de almacenamiento.

En trabajos futuros será importante que el sistema gestor de recursos y monitorización pueda ser implementado en diversas plataformas de virtualización, tal como *VMware*.

# BIBLIOGRAFÍA

[1] Arbós,N., Amorós,L.M., González,D., Oller,A., Alcober,J., “Servicios telemáticos sobre nubes privadas en plataformas virtualizadas y distribuidas”, *Publicación JITEL*, 2010.

[2] Meinser,D., Gold,B., Wenish,T., “PowerNap: Eliminating Server Idle Power”, *Publicación ASPLOS*, 2009.

[3] Furht,B., “Cloud Computing Fundamentals”, *Handbook of Cloud Computing*, Furht, B., Escalante, NY, USA, 2010.

[4] Wang,L., Laszewski,G., Jie,T., Kunze,M., Canales,A., Kramer,D., Wolfgang,K., “Scientific Cloud Computing: Early Definition and Experience”, *Publicación HPCC* , 2008.

[5] Jones,M., “Anatomy of a Cloud storage Infraestructure”, *Articulo developerWorks,IBM*, 2010.

[6] Takemura,C., Crawford,L., *The Book of Xen*, No Starch Press,San Franciso,2010.

[7] Chisnall,D., *The Definitive Guide to the Xen Hypervisor*, Prentice Hill,Westford,2007.

[8] Dunlap,G. , “Scheduler development update”, *Publicación Citrix Systems R&D*, 2009.

[9] Devine,J., “An Empirical Evaluation of Methods for Improving Eficiency in Xen Virtual Machine CPU Scheduling”, *Informe Técnico CS10-01,Allegheny College*, 2010

[10] Cherkasova,L., Gupta,D., Vahdat,A., “Comparison of the three CPU schedulers in Xen”, *ACM SIGMETRICS Performance Evaluation Review*, **Volumen 35**(2),2007

[11] Magenheimer,D., Mason,C., McCracken,D. Hackel,K., “Transcendent Memory and Linux”, *Publicación para el Linux Symposium 2009*.

[12] Citrix©, “Citrix Workload Balancing Administrator’s Guide”, *Publicación de Citrix* , 2011.

[13] VMware®, “VMware Distribuited Power Management: Concepts and Use”, *Publicación de WMware*, 2009.

[14] Microsoft®, “System Center Virtual Machine Manager”, *Publicacion de Microsoft*, 2008.

[15] Verma,A., Dasgupta,G., Nayak,T., De,P., Kothari,R., “Server Workload Analysis for Power Minimization using Consolidation”, *Usenix Annual Technical Conference*, 2009.

[16] Speitkamp,B., Bichler,M., “A mathematical programming aproach for servers consolidation. Problem in vitualized data centers”, *IEE Transactions on Services Computing*, **Volumen 3**(4),2010

[17] Baruchi,A., Midorikawa,E., “Gerência de Memória Adaptativa no XEN”, *Informe Técnico para CSBC São Paulo*, 2009.

[18] Wuhib,F. , Stadler,R , Spreitzer,M., “Gossip-based Resource Management for Cloud Environments”, *Publicado para International Conference on Network and Service Management, Niagra Falls, Canada*, 2010.

[19] Korf, R., “A new algorithm for optimal bin packing”, *Publicado para National for Artificial Intelligence*, 2002.

[20] de Niz,D., Rajkumar,R., “Partitioning Bin Packing Algorithms for Distributed Real-Time Systems”, *Publicado en International Journal of Embedded Systems*, 2005.

[21] Rieck, B., “Basic Analysis of Bin-Packing Heuristics”, *Publicado por Interdisciplinary Center for Scientific Computing. Heildelberg University*, 2010.

[22] Przydatek, B., “A Fast Approximation Algorithm for the Subset-Sum Problem”, *International Transactions in Operational Research*, **Volumen 9**(4),2002.

[23] Dasgupta,S. ,Papadimitriou,C., Vazirani,U., *Algorithms*, Mc Graw Hill,2006.

[24] Sedgewick,R., *Algorithms*, Addison-Wesley, CA, USA, 1984.

[25] Cormen,T., Leiserson,C., Rivest,R., Stein,C., *Introduction to Algorithms*, MIT Press,2009.

[26] Skiena,S., *The Algorithm Design Manual*, Springer, NY, USA,2008.

[27] González,R., *Python para todos*, Creative Commons,2008.

[28] Pilgrim,M. , *Dive Into Python*, GNU Free Documentation License,2004.

[29] Hetland,M. , *Python Algorithms*, Apress, NY,USA,2010.

[30] Copeland,R. , *Essential SQLAlchemy* O’Reilly,Sebastopol,CA,USA, 2008.

[31] Bayer,M. , *SQLAlchemy Documentation* SQLAlchemy,2011.

[32] U.S. Environmental Protection Agency, “Report to Congress on Server and Data Center Energy Efficiency” 2007.

[33] Koomey,J., “Worldwide electricity used in data centers”, *Environmental Research Letters*, **Volumen 3**(3),2008.

[34] Murugesan,S., “Harnessing Green IT: Priciples and Practices” *Journal IT Professional* **Volumen 10**(1),2008.

# APÉNDICES

# APÉNDICE A. ALGORITMOS DE RESOLUCIÓN DE BBP

Al problema de tipo *Bin-Packing* a pesar de ser de tipo NP-Hard, se pueden aplicar soluciones heurísticas que garantizan tiempos de aplicación competitivos.

Por otro lado si quiere garantizar una solución óptima es necesario establecer métodos más complejos y esto supone el aumento de la complejidad temporal, ya que esto conlleva a revisar cada posible solución. Para valores elevados de entrada el tiempo de aplicación es elevado y hacen impracticables ciertos métodos.

Por lo que para aplicar soluciones válidas se pueden incrementar la complejidad del algoritmo o por otro lado relajar la restricción y obtener tiempos de aplicación aceptables.

Existen soluciones rápidas pero que no representan soluciones óptimas como el Next-Fit, First-Fit,Worst-Fit o Best-Fit.

## A.1. Algoritmo Next-Fit

Este algoritmo es el más básico empleado en la resolución rápida del BPP. Este algoritmo consiste en la asignación de items u objetos a partir del contenedor actual. De este modo si un objeto es grande en comparación al espacio disponible por el contenedor actual, agrega uno nuevo que pasa a ser el actual y la siguiente asignación no tiene en cuenta el nodo anterior. Continua este proceso hasta que se asignan todos los objetos. Este algoritmo es el menos eficiente pero tiene beneficios prácticos.

```
1 for Todos los objetos i = 1,2,....n do
2     if Objeto i encaja en el actual contenedor then
3         Asigna objeto i en el actual contenedor.
4 else
5     Crear un nuevo contenedor, convertirlo en el actual contenedor y encajar objeto i.
```

Figura A.1: Pseudocódigo Next Fit

## A.2. Algoritmo First-Fit

Este algoritmo es el más claro ejemplo de un tipo de algoritmo voraz. Se trata de asignar una serie de items u objetos que están ordenados arbitrariamente seleccionando el primer contenedor disponible que encaje. En caso no exista este contenedor se procede a agregar uno nuevo. Este algoritmo es más eficiente que el Next-Fit.

```
1 for Todos los objetos i = 1,2,....n do
2     for Todos los contenedores j = 1,2,.... do
3         if Objeto i encaja en el contenedor j then
4             Asigna objeto i en el contenedor j.
5             Break
6 if Objeto i no encaja en ningún contenedor disponible then
7     Crear un nuevo contenedor y encajar el objeto i
```

Figura A.2: Pseudocódigo First Fit

## A.3. Algoritmo Max-Rest o Worst-Fit

A pesar del nombre este algoritmo es más eficiente que el método First-Fit. Consiste en asignar una serie de objetos seleccionando en cada movimiento el nodo que tenga el espacio libre más grande.

```
1 for Todos los objetos i = 1,2,....n do
2     Define k = min{i|c_i = min_{j=1}^{j=M} c_j}, cómo el índice del contenedor con capacidad
      restante máxima.
3     if Objeto i encaja en el contenedor k then
4         Asigna objeto i en el contenedor k.
5         Break
6     else
7         Crear un nuevo contenedor y encajar el objeto i
```

Figura A.3: Pseudocódigo Max-Rest

## A.4. Algoritmo Best-Fit

Este algoritmo a diferencia del Worst-Fit se encarga de asignar los objetos o items seleccionando el nodo que tenga menos espacio disponible. Dependiendo de la naturaleza de los parámetros de los objetos este algoritmo puede ser más efectivo que el Worst-Fit.

```
1 for Todos los objetos i = 1,2,....n do
2     for Todos los contenedores j = 1,2,.... do
3         if Objeto i encaja en el contenedor j then
4             Calcular la capacidad restante después que el objeto haya sido asignado.
5     Encajar el objeto i en el contenedor j, siendo j el contenedor con la minima
      capacidad restante después de agregar el objeto.
6     if No existe el contenedor then
7         Abrir un nuevo contenedor y agregar el objeto.
```

Figura A.4: Pseudocódigo Best Fit

Todos estos algoritmos pueden ser mejorados mediante la ordenación descendente en función de los parámetros de los objetos. Por otro lado implementar el ordenado supone incrementar la complejidad temporal.

# APÉNDICE B. ALGORITMOS DE RESOLUCIÓN DE SSP

## B.1. Algoritmo Exact-Subset-Sum

El Exact -subset -Sum $(S,t)$ es un algoritmo exacto de tiempo exponencial. El par $(S,t)$ es una instancia del SSP, donde $S$ es el subconjunto de enteros positivos cuya suma es el valor $t$ o target. Precisamente ese es el objetivo al resolver un $SSP$, encontrar un subconjunto $S'$ cuya suma se aproxime al máximo al valor target $t$ sin sobrepasarlo. Este algoritmo consiste en aplicar en cada iteración $i$ la suma de valores del subconjunto $S'$, con el objetivo de evaluar si a partir de los valores de $S'$ es posible conseguir $t$.

```
1 t ← Target
2 n ← |S|
3 L_0 = 0
4 for i = 1,2,....n do
5 |   L_i = MergeLists(L_{i-1}, L_{i-1} + x_i)
6 |   Eliminar de L_i cada elemento que es más grande que t
7 return el elemento más grande de L_n
```

Figura B.1: Pseudocódigo Subset-Sum Exacto

Este algoritmo emplea el método de ordenación por mezcla. Este consiste en combinar la array definida en la iteración $i-1$ con la misma array sumando el valor $i$ del subconjunto $S'$. Si algún valor sobrepasa el valor $t$ es eliminado en cada iteración. Tomando como ejemplo la primera asignación que se puede ver en el anexo, en la que se observa una nodo $00$ de 5376 MB y el nodo $01$ con una carga de 2048 MB. El balance ideal de ambos nodos se sitúa 3712 MB por lo que es necesario reducir carga del nodo $00$ y migrarla al nodo $01$. Idealmente se requiere que migrar 1664 MB para garantizar un balanceo ideal. A partir de aquí se puede definir el objetivo $t$ =1664 MB y el conjunto de enteros como carga de las MV del nodo $00$:

$$\begin{aligned} S &= \{1024, 2048, 1024, 512, 256, 512\} \\ t &= 1664 \end{aligned} \tag{B.1}$$

$$\begin{aligned} L_0 &= \{0\} \\ L_1 &= L_0 \cup (L_0 + 1024) = \{0, 1024\} \\ L_2 &= L_1 \cup (L_1 + 2048) = \{0, 1024, \cancel{2048}, \cancel{3072}\} = \{0, 1024\} \\ L_3 &= L_2 \cup (L_2 + 1024) = \{0, 1024\} \\ L_4 &= L_3 \cup (L_3 + 512) = \{0, 512, 1024, 1536\} \\ L_5 &= L_4 \cup (L_4 + 256) = \{0, 256, 512, 768, 1024, 1280, 1536\} \\ L_6 &= L_5 \cup (L_5 + 512) = \{0, 256, 512, 768, 1024, 1280, 1536\} \end{aligned} \tag{B.2}$$

Al aplicar el algoritmo se llega a la conclusión que el máximo valor de la suma del conjunto sin sobrepasar $t$ es 1536 MB. Esto conlleva que se emplea una MV de 1024 y otra de 512 MB que están presentes en la lista de candidatas a migrar. Como anteriormente se ha mencionado este algoritmo es de tipo exponencial por lo que en base a la longitud de los valores de entrada $L_i$ puede dar resultados $2^i$.

## B.2. Métodos Voraces

El algoritmo voraz más común empleado para resolver el SSP consiste en iterar sobre el subconjunto ordenados de forma descendente en función de su valor. Se irán asignando al subconjunto solución aquellos números cuyo valor no exceda la diferencia de la suma del subconjunto actual y el valor objetivo $t$.

```
1 t ← Target
2 n ← |S|
3 L ← {0}
4 sum ← 0
5 for i = 1,2,....n ordenados descendentemente en funcion de su valor do
6     if valor i i(t − sum) then
7         sum ←sum+valori
8         Agregar i a L
```

Figura B.2: Pseudocódigo Método Voraz para SSP

La complejidad temporal de este método es de $O(n\lg n)$. Existen variaciones a partir de cual se prescinde del ordenado de los números en función de su valor y son introducidos en el algoritmo en modo aleatorio, esto permite reducir la complejidad temporal a $O(n)$.

## B.3. Algoritmos de aproximación para el SSP

Existe otra solución para resolver un SSP basada en esquemas de aproximación de tiempo polinomial (PTAS y FPTAS). Estos permite aproximar la solución reduciendo las instancias del problema. Las soluciones se evalúan en función del valor de error $\varepsilon$. Por lo tanto la complejidad temporal esta evaluada en función de la calidad de la aproximación$O(n^{2\varepsilon}$.

Un ejemplo para conseguir una aproximación reduciendo la complejidad temporal del algoritmo consiste en reducir el campo de búsqueda. En el caso del SSP se puede reducir el valor de elementos de $L$. En los casos en que existen muchos elementos cuyos valores son cercanos se puede mantener uno de ellos reduciendo $L$.

Para definir el grado de aproximación se establece el parámetro $\delta$, donde $0 < \delta < 1$. De este modo se puede establecer que un elemento $x$ se aproxima al elemento $y$ si se

considera $y/(1+\delta) \leq x \leq y$ Aplicando el algoritmo de reducción se eliminan el máximo número de elementos que se aproximen a un valor dado.

En la fig. B.3 se observa que son evaluados todos los elementos de $L$ y eliminando del subconjunto final aquellos elementos que son aproximados.

---

1 $m \leftarrow$ Longitud de $L$
2 $L_0^{'} \leftarrow 0$
3 $ultimo \leftarrow y_1$
4 **for** $i = 2,3,....n$ **do**
5 | **if** $y_i > ultimo \cdot (1+\delta)$ **then**
6 | | Agregar $y_i$ dentro del final de $L^{'}$
7 | | $ultimo \leftarrow y_i$
8 **return** $L^{'}$

Figura B.3: Pseudocódigo Reducción de lista

Si se aplica esta idea al Algoritmo Exact-Subset-Sum, de manera que en cada iteración se evalúe la aproximación y reduzca el subconjunto final se consigue un algoritmo de aproximación al SSP.

---

1 $t \leftarrow Target\ n \leftarrow |S|$
2 $L_0 \leftarrow 0$
3 **for** $i = 1,2,....n$ **do**
4 | $L_i \leftarrow MergeLists(L_{i-1}, L_{i-1} + x_i)$
5 | $L_i \leftarrow Trim(L_i, \varepsilon 2n)$
6 | Eliminar de $L_i$ cada elemento más grande que $t$
7 **return** el elemento más grande de $L_n$

Figura B.4: Pseudocódigo Algoritmo de aproximación SSP

# APÉNDICE C. ALGORITMO WFD - MÉTODO VORAZ

Se ha utilizado este algoritmo para comparar los resultados con la solución finalmente propuesta.

Este algoritmo emplea el método Worst-Fit Decreasing para reasignar las MVs. El WF se caracteriza por asignar las MV en aquellos nodos menos cargados o con más capacidad restante. Pero lo que le hace ser más complejo temporalmente en comparación a la solución propuesta en el ordenado decreciente previo de las MVs en función de sus recursos. Esta ordenación permite que la asignación de las MVs sea más eficiente. Al ordenar las MV de modo decreciente hace que las MV con mayor capacidad se sitúen en los primeros nodos para que posteriormente las MVs de mínima carga se adapten en los espacios disponibles de cada nodo. Esto produce un una distribución equilibrada de los recursos en los nodos.

Una posibilidad que puede ocurrir frecuentemente es la repetición de valores de carga de las MV. Ante este problema la asignación utiliza un método voraz que consiste en aplicar la asignación garantizando el óptimo local. Es decir si existe un conjunto de MVs que tienen los mismos recursos máximos establecidos y diversos nodos al que pueden ir asignados se decide por aquella máquina y aquel nodo cuya reasignación no implique migración. Por lo tanto existe una preferencia de asignación aquellas MV que coinciden con algún nodo candidato. A pesar de ello se pueden producir múltiples opciones de MVs para que sean reasignadas a múltiples opciones de Nodos por lo que irremediablemente signifique definir la MV y Nodo adecuado de manera exhaustiva dando como resultado mayores tiempos de aplicación (casi exponenciales) si aparece esta peculiaridad.

Esta opción garantiza el óptimo local respecto a minimizar el número de migraciones. Sin embargo y tal como se observa en los resultados de la simulación no garantiza la optimización global con respecto a minimizar el número de migraciones.

En la comparación con la solución propuesta, este algoritmo se comporta ligeramente mejor garantizando un reparto equitativo de los recursos. Además obtiene valores similares de ocupación de nodos. Por el contrario es coste temporal es exponencial y el número de migraciones o reestruturaciones que realiza es elevado. Por este motivo este algoritmo es poco práctico para la gestión de recursos.

En la fig. C.1 se muestra el pseudocódigo de este algoritmo.

```
1  contNodos ← 0
2  contMV ← 0
3  for Todas las MVs activas ordenadas descendentemente por capacidad i = 1,2,....n
   do
4      contMv ← Numero de MVs con la capacidad de MV i
5      subListM[0,....,0] ← MVs con capacidad de i
6      for Todos los nodos activos , no asignados y ordenados descendentemente
7      por capacidad restante j = 1,2,....m do
8          contNodos ← Numero de Nodos con capacidad restante de j
9          subListN[] ← Nodos con capacidad restante de j
10         if contNodos = 1 & contMv > 1 then
11             for Todas las MVs de subListM ii = 1,2,.... do
12                 if La MV ii actualmente pertenece nodo j then
13                     Dejar asignada la MV ii al nodo j
14                     break
15             if No existe MV de subListM que pertencezca al nodo j then
16                 Asignar la primera máquina de subListM que encaje al nodo j
17         else if contNodos > 1 & contMv = 1 then
18             for Todas los nodos de subListN jj = 1,2.... do
19                 if La MV i actualmente pertenece al nodo jj then
20                     Dejar asignada la MV i al nodo jj
21                     break
22             if No existe Nodo de subListN que contenga a la MV i then
23                 Asignar la MV i al primer nodo que encaje de subListN
24         else if contNodo > 1 & conMv > 1 then
25             for Todos los nodos de subListN jj = 1,2... do
26                 for Todas las MVs de subListM ii = 1,2,.... do
27                     if La MV ii actualmente pertenece jj then
28                         Dejar asignada la MV ii al nodo jj
29                         break
30             if No existe nodo en subListN que contenga alguna MV de subListM then
31                 Asignar la MV ii al primer nodo que encaje de subListN
```

Figura C.1: Pseudocódigo WFD con método voraz

# APÉNDICE D. SIMULACIÓN: EJEMPLO DE APLICACIÓN DEL SCHEDULER

| Id. | Nodos |
|---|---|
| | Nodo 00 |
| | Nodo 01 |
| | Nodo 02 |
| | Nodo 03 |
| | Nodo 04 |
| | Nodo 05 |
| | Nodo 06 |
| | Nodo 07 |
| | Nodo 08 |
| | Nodo 09 |

| | Número Mv | Id Mv | ASIGNACIÓN | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 5376/2048 | | | | | | | | | | | | | | | | | |
| MÁQUINAS VIRTUALES | 1 | 1 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 2 | 2 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 3 | 3 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 4 | 4 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 5 | 5.1 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 |
| | 6 | 5 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 7 | 6 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 8 | 7 | 1024 | 2m/1536 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 9 | 8 | 2048 | 0,192 s | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 10 | 9.1 | 256 | | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 |
| | 11 | 9 | 512 | | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 12 | 10 | 2048 | | | 2m/3072 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 13 | 11 | 512 | 3840 | | 0,236 s | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 14 | 12 | 3072 | 3584 | | | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 |
| | 15 | 13 | 3072 | | 5888 | 3840 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 |
| | 16 | 14 | 512 | | 4864 | 3840 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 17 | 15 | 2048 | | 512 | 3584 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 18 | 16 | 2048 | | | | | 6m/3584 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 19 | 17 | 2048 | | | | | 0,41 s | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 20 | 18 | 1024 | | | | | | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 21 | 19.1 | 256 | | | | 6912 | 5632 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 |
| | 22 | 19 | 3072 | | | | 7424 | 5632 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 |
| | 23 | 20 | 512 | | | | 6144 | 5632 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 24 | 21 | 1024 | | | | 2048 | 5632 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 25 | 22 | 1024 | | | | | | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 26 | 23 | 512 | | | | | | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 27 | 24 | 2048 | | | | | | | 6m/6400 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 28 | 25.1 | 256 | | | | | | | 0,5246 s | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 |
| | 29 | 25 | 1024 | | | | | | | | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 30 | 26 | 512 | | | | | | 8960 | 6912 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 31 | 27 | 512 | | | | | | 8192 | 6656 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| | 32 | 28 | 1024 | | | | | | 8704 | 6656 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 33 | 29 | 2048 | | | | | | 7680 | 6912 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 34 | 30 | 1024 | | | | | | 512 | 6912 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 35 | 31 | 3072 | | | | | | | | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 |
| | 36 | 32 | 2048 | | | | | | | | | 8m/4096 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 37 | 33 | 1024 | | | | | | | | | 0,647 s | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 38 | 34.1 | 256 | | | | | | | | | | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 |
| | 39 | 34 | 3072 | | | | | | | | 7936 | 7680 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 | 3072 |
| | 40 | 35 | 1024 | | | | | | | | 8448 | 7680 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 41 | 36 | 1024 | | | | | | | | 8704 | 7680 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 42 | 37.1 | 256 | | | | | | | | 7936 | 7680 | 256 | 256 | 256 | 256 | 256 | 256 | 256 | 256 |
| | 43 | 37 | 2048 | | | | | | | | 9472 | 7680 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 44 | 38 | 1024 | | | | | | | | 3072 | 7168 | | 8m/5632 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 45 | 39 | 2048 | | | | | | | | | | | 0,736 s | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 46 | 40 | 1024 | | | | | | | | | | | | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 47 | 41 | 1024 | | | | | | | | | | 8704 | 8192 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 48 | 42 | 2048 | | | | | | | | | | 8704 | 8192 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 49 | 43.1 | 256 | | | | | | | | | | 8704 | 8192 | 256 | 256 | 256 | 256 | 256 | 256 |
| | 50 | 43 | 1024 | | | | | | | | | | 8192 | 7936 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 51 | 44 | 2048 | | | | | | | | | | 10752 | 8192 | 2048 | 2048 | 2048 | 2048 | 2048 | 2048 |
| | 52 | 45 | 1024 | | | | | | | | | | 9216 | 7936 | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |
| | 53 | 46 | 2048 | | | | | | | | | | 2048 | 7680 | | 6m/7680 | 2048 | 2048 | 2048 | 2048 |
| | 54 | 47 | 1024 | | | | | | | | | | | | | 0,803 s | 1024 | 1024 | 1024 | 1024 |
| | 55 | 48 | 2048 | | | | | | | | | | | | | | 2048 | 2048 | 2048 | 2048 |
| | 56 | 49 | 512 | | | | | | | | | | | | 9216 | 8192 | 512 | 512 | 512 | 512 |
| | 57 | 50 | 2048 | | | | | | | | | | | | 10496 | 8448 | 2048 | 2048 | 2048 | 2048 |
| | 58 | 51 | 2048 | | | | | | | | | | | | 9216 | 8192 | 2048 | 2048 | 2048 | 2048 |
| | 59 | 52 | 3072 | | | | | | | | | | | | 9984 | 8960 | 3072 | 3072 | 3072 | 3072 |
| | 60 | 53 | 1024 | | | | | | | | | | | | 10240 | 8192 | | 6m/4864 | 1024 | 1024 |
| | 61 | 54 | 2048 | | | | | | | | | | | | 8960 | 8448 | | 0,932 s | 2048 | 2048 |
| | 62 | 55 | 512 | | | | | | | | | | | | 8704 | 8704 | | | 512 | 512 |
| | 63 | 56 | 512 | | | | | | | | | | | | 1024 | 8704 | 10240 | 9216 | 512 | 512 |
| | 64 | 57.1 | 256 | | | | | | | | | | | | | | 10496 | 9216 | 256 | 256 |
| | 65 | 57 | 2048 | | | | | | | | | | | | | | 10240 | 9216 | 2048 | 2048 |
| | | | | | | | | | | | | | | | | | 8960 | 8960 | 6m/4096 | MV/RAM(MB) MIGRADOS |
| | | | | | | | | | | | | | | | | | 9216 | 9216 | 0,934 s | COSTE TEMPORAL |
| | | | | | | | | | | | | | | | | | 8960 | 8960 | | CARGA DE NODOS (MB) |
| | | | | | | | | | | | | | | | | | 8704 | 8704 | 9216 | 9216 Nodo 00 |
| | | | | | | | | | | | | | | | | | 10752 | 9216 | 9216 | 9216 Nodo 01 |
| | | | | | | | | | | | | | | | | | 3072 | 7936 | 9216 | 9216 Nodo 02 |
| | | | | | | | | | | | | | | | | | | | 9472 | 8704 Nodo 03 |
| | | | | | | | | | | | | | | | | | | | 9216 | 8704 Nodo 04 |
| | | | | | | | | | | | | | | | | | | | 9216 | 8704 Nodo 05 |
| | | | | | | | | | | | | | | | | | | | 10752 | 8704 Nodo 06 |
| | | | | | | | | | | | | | | | | | | | 9216 | 9216 Nodo 07 |
| | | | | | | | | | | | | | | | | | | | 9472 | 9216 Nodo 08 |
| | | | | | | | | | | | | | | | | | | | 2048 | 6144 Nodo 09 |

Agradecimientos:

Agradezco la oportunidad dada para realizar este trabajo al coordinador del MediaCAT y director del proyecto David González y Toni Oller como supervisor del proyecto. De igual forma los agradecimientos son extensibles a Luismi Amorós y Borja Ruiz que han cedido tiempo y paciencia con el fin de ayudarme a desarrollar este TFC.