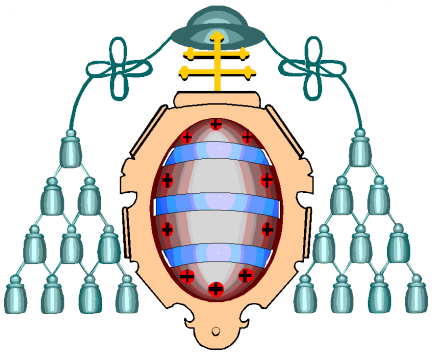


UNIVERSIDAD DE OVIEDO

ESCUELA POLITÉCNICA SUPERIOR DE INGENIERÍA DE GIJÓN

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

PROYECTO FIN DE CARRERA Nº 1102015

GESTIÓN AUTONÓMICA DE ENERGÍA BASADA EN MÉTRICAS DE CALIDAD DE SERVICIO

MEMORIA

RAMÓN MEDRANO LLAMAS
ABRIL 2011

TUTOR: DANIEL F. GARCÍA
COTUTOR: JOAQUÍN ENTRIALGO

A todos aquellos que me han hecho ser quien soy.

Y a la persona que me cambia cada día.

RESUMEN

El consumo de energía de grandes clústeres de ordenadores afecta a los costes de explotación de centros de datos, en particular incrementando el coste total de posesión (*TCO*). Por lo tanto, la utilización de algoritmos de minimización de la energía es esencial para una explotación rentable de los centros de datos.

La mayoría de los algoritmos disponibles no garantizan un mínimo de calidad del servicio (*QoS*) proporcionado por el clúster y muchos otros requieren modelos complejos del clúster y la evolución de la carga de trabajo, la solución del problema de optimización off-line y la tabulación de los resultados que se pueden utilizar más adelante en línea.

Por el contrario, el algoritmo presentado en este proyecto sólo utiliza dos modelos de tipo *input-output* para cada tipo de equipo del clúster. Estos modelos caracterizan el tiempo de respuesta (*QoS*) y el consumo de energía en función de la carga de trabajo y pueden obtenerse fácilmente a partir de un experimento de carga.

En base a estos modelos, el algoritmo trabaja en línea manteniendo encendidos los equipos más eficientes para prestar el servicio con la calidad deseada. La simplicidad de implantación de este nuevo algoritmo contribuirá significativamente a la integración de las políticas de Green IT en centros de datos existentes.

ABSTRACT

The power consumption of large computer clusters affects the exploitation costs of datacenters notably. Therefore, the utilization of power minimization algorithms is essential for a profitable exploitation of datacenters.

Most of available algorithms do not guarantee a minimum quality of the service (QoS) provided by the cluster and many others require complex models of the cluster and the workload evolution, solving the optimization problem off-line and tabulating results that can be used later on-line.

Contrarily, the algorithm introduced in this project only uses two input-output models for each computer type of the cluster. They characterize the response time (QoS) and the power consumption as a function of the workload and can be easily obtained from a load experiment.

Based on these models, the algorithm works on-line maintaining turned on the most power efficient computers needed to provide the service with the desired quality. The deployment simplicity of this new algorithm will contribute significantly to integrate Green IT policies in datacenters.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

1 INTRODUCCIÓN

El proyecto a desarrollar tendrá como objetivo principal la gestión dinámica de la energía de un clúster que aloja un sistema transaccional. Los principales puntos de innovación son la integración de la gestión de la energía con el módulo de gestión de la calidad del servicio y el enfoque autonómico de la gestión.

Hoy en día, la utilización de clústeres para alojar sistemas transaccionales es muy común, debido a la gran escalabilidad que proporcionan, así como el nivel tolerancia a fallos que ofrecen. Sin embargo, se precisa de una gestión energética adecuada, pues los gastos derivados de consumo energético y refrigeración de grandes centros de datos llegan ya a ser equiparables al gasto en hardware [Koomey07], alargando notablemente el tiempo de amortización de la compra del clúster e incrementando el coste total de operación (*TCO, total cost of ownership*).

La gestión dinámica de la energía de un clúster tiene como objetivo integrar en el sistema un módulo de tal forma que permita ofrecer una calidad de servicio acorde a las políticas que se definan para el clúster, pero manteniendo un consumo energético mínimo para cada nivel de servicio.

En una definición de alto nivel, el gestor de energía se encargará de mantener encendidos tantos nodos del clúster como sean necesarios para satisfacer las políticas de calidad de servicio, pero minimizando el conjunto de nodos que permanecen en operación, bien reduciendo su número o el consumo que representan.

Según el modelo de lógica de negocio, existen muchos períodos a lo largo de un día donde un clúster tiene una carga de trabajo notablemente inferior a punta de trabajo que puede llegar a alcanzarse durante tan sólo dos o tres veces cada día. En estos largos tiempos con una carga de trabajo reducida parece conveniente apagar uno o más nodos de dicho clúster, pues puede garantizarse la calidad de servicio para dicho nivel de carga con un subconjunto de nodos.

Sin embargo, en momentos de elevada carga de trabajo, el sistema de gestión dinámica de energía ha de encender todos los nodos del clúster para satisfacer todas las peticiones en un nivel de carga más elevado.

Además, dado que se desea hacer esta gestión de forma autonómica, será el propio sistema quien decida cuándo alterar el número de nodos del clúster y bajo qué condiciones, reduciendo y simplificando la gestión y administración del clúster y, por tanto, los costes de operación derivados de mantenimiento y energía.

1.1 Organización de un sistema on-line *de gestión energética*

El sistema de gestión dinámica de la energía propuesto para controlar el sistema del clúster se basa en tres bloques fundamentales cuyas relaciones de los mismos pueden observarse en la Figura 1.

1.1.1 Monitor de energía

Este módulo se encarga de obtener información acerca de cómo está funcionando el sistema y sus correspondientes componentes. Esta información será de utilidad a la hora de tomar decisiones sobre la gestión energética y es de especial interés que sea capturada en tiempo real, para conocer el estado instantáneo del sistema y su evolución, de forma que se puedan establecer predicciones sobre el comportamiento del mismo.

La obtención de esta información se representa por las líneas verdes en la Figura 1.

1.1.2 Actuador energético

El módulo actuador será el encargado de realizar modificaciones en el estado energético de los diferentes componentes del clúster mediante tecnologías que permitan interactuar con el sistema de gestión energético del sistema operativo. Este flujo de información se representa por las líneas rojas en la Figura 1.

1.1.3 Gestor de energía

Este módulo toma decisiones sobre la gestión de energía en base a las mediciones ofrecidas por el monitor y por las órdenes recibidas desde el monitor de calidad de servicio. Este es el componente autonómico, ya que será el encargado de realizar la gestión de forma autónoma, en base a unos objetivos de calidad de servicio, políticas de administración y el estado del sistema.

El monitor de calidad de servicio se encarga de aceptar nuevas transacciones siempre que se pueda satisfacer la política de calidad.

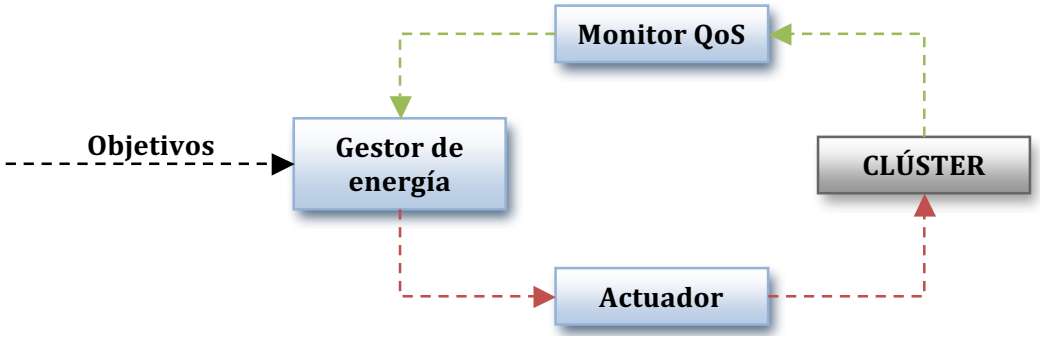


Figura 1: Diagrama de bloques del gestor autonómico

De cara a la operación del gestor de energía, tal y como se mencionó antes, es necesario proporcional políticas y objetivos de gestión que permitan mantener una línea de operación acorde a las características del modelo de operación del clúster.

Modelar estas políticas es un problema que ha de ser resuelto para minimizar los parámetros de operación del sistema y, a la vez, proporcionar herramientas consistentes al administrador.

1.2 Objetivos del proyecto en detalle

Los objetivos del proyecto de investigación son:

- Analizar y documentar las técnicas de gestión energética tanto a nivel hardware, como a nivel de sistema operativo.
- Diseñar un algoritmo de optimización en línea que permita al sistema autogestionarse.

- Monitorizar la utilización de los dispositivos de cada nodo del clúster así como de su consumo energético y su estado de operación (suspendido, hibernado, totalmente funcional, etc.).
- Integrar un actuador que permita arrancar y parar un nodo desde otro computador. El arranque y la parada podrían ser progresivos, pasando por los distintos estados de suspensión e hibernación que soporte cada nodo completo del clúster y sus dispositivos individuales.
- Diseñar y realizar un gestor combinado de calidad de servicio y energético del clúster. Este módulo es el que dota de inteligencia al sistema y permita realizar la provisión dinámica de nodos de forma automática en función de los requisitos de calidad de servicio.

1.3 Organización del documento

La memoria de este proyecto está orgnizada como sigue: en la sección 2 se realiza un repaso a las diferentes tecnologías de gestión enerética actuales, en la sección 3 se analiza el estado del arte existente relativo a la gestión autonómica de la energía en clústers de computación, en la sección 4 se introduce una nueva propuesta para la gestión autonómica de la energía en clústers de computación, basada en modelos sencillos que garantizan la calidad de servicio.

En la sección 5 se repasan conceptos de modelado de sistemas, así como se introduce una forma de modelado para el algoritmo de gestión presentado anteriormente. En la sección 6 se analizan aspectos avanzados del algoritmo, para dar mayor claridad a aspectos que requieren un estudio más detallado. En la sección 7 se presenta el diseño experimental y los resultados de las múltiples pruebas de validación del algoritmo en diferentes entornos.

Finalmente, las secciones 8, 9 y 10 se reservan para conclusiones, aspectos de diseño del prototipo que acompaña al proyecto y un resumen de la gestión del proyecto.

2 FUNDAMENTOS TECNOLÓGICOS

A continuación se presenta un resumen de las tecnologías para la gestión energética desarrolladas en los últimos años. Mientras que unas se comentan como mero repaso histórico, otras, derivadas de éstas, son de directa aplicación en el prototipo a desarrollar para validar la estrategia diseñada, tales como *Wake on LAN* o *Dynamic Voltage Scaling*.

Unas tecnologías se controlan directamente en la provisión dinámica, pero otras simplemente se dejan manejar por el control automático de cada nodo del clúster. Pese a esto, es necesario tenerlas en cuenta para el modelado y caracterización del servicio.

2.1 Tecnologías de actuación y monitorización

Las tecnologías de monitorización son tecnologías que tienen como objetivo el control y seguimiento del consumo energético de un sistema. Generalmente, además del consumo, permiten medir numerosos parámetros, tales como la temperatura o la velocidad de giro de los ventiladores.

Las tecnologías de actuación, aunque muchas veces incluidas dentro de las de monitorización, tienen un objetivo diferente, pues permiten modificar los estados energéticos de diferentes elementos de un sistema para cambiar su consumo (y rendimiento).

2.1.1 Gestión energética de dispositivos

Para realizar una adecuada gestión de la energía, los dispositivos que forman cada uno de los nodos ha de ofrecer unas determinadas características relativas a la gestión de la energía de los mismos. Deben permitir ser apagados y encendidos bajo demanda y, a ser posible, contar con

estados intermedios de funcionamiento que permitan una mayor granularidad a la hora de realizar una gestión adecuada.

A continuación se repasan las principales técnicas y tecnologías disponibles en la actualidad para la gestión dinámica de la energía:

2.1.1.1 Microprocesadores

El procesador es el componente que más energía puede llegar a consumir dentro de un servidor de alto rendimiento, así como ser el componente que más calor llega a disipar.

La gestión energética de los procesadores actuales es fundamentalmente autónoma en lo que se refiere al mismo, no siendo necesaria la administración directa (aunque es posible y los sistemas operativos mayoritarios la incorporan para mejorar la relación potencia/consumo) desde el sistema operativo u otros dispositivos en la mayoría de los casos (aunque si existen soluciones más elaboradas [Behle08]). Las técnicas más empleadas son el escalado dinámico de frecuencia y/o voltaje (*DVS, Dynamic Voltage Scaling*) y el apagado de unidades funcionales [Bianchini04].

Estas tecnologías se administran, generalmente, de forma automática por parte del hardware, pero en numerosas ocasiones se encuentran desactivadas en las configuraciones por defecto de las placas base.

Toda la energía que consume un microprocesador se disipa en forma de calor. Esta potencia depende de tres factores importantes [Hamacher03]:

$$P = C \cdot V^2 \cdot f$$

Donde P es la potencia disipada y C la capacitancia del circuito, que es constante para un mismo microprocesador. Los términos V y f corresponden al voltaje y a la frecuencia, respectivamente.

Como se puede deducir de la fórmula anterior, incrementos de frecuencia producen incrementos lineales en la potencia disipada, pero incrementos de voltaje producen incrementos de orden cuadrático en la potencia disipada.

Las técnicas de escalado dinámico de frecuencia y escalado dinámico de voltaje permiten a los microprocesadores operar en distintos modos de voltaje y frecuencias y cambiar entre ellos de forma dinámica a lo largo del tiempo de funcionamiento, de forma que se minimice el consumo energético. Normalmente, estas técnicas se implementan de forma combinada y se delega la gestión directamente al propio microprocesador.

Otra técnica muy utilizada en la actualidad es la que permite apagar unidades funcionales que no estén en uso. Esta técnica, conocida como *power gating*, está implementada en los procesadores de última generación y ya no sólo permite apagar (directamente, desconectar de la alimentación, eliminado así también el *leakage*[1]) unidades funcionales, sino núcleos enteros en el caso de procesadores multinúcleo.

Por lo general, el propio hardware del procesador se encarga de la gestión de energía del mismo, no siendo necesario hacer uso de mecanismos externos. Sin embargo, para cambiar a modos de ahorro de energía tales como la suspensión o hibernación, ha de ser el sistema operativo, bien por orden del usuario o del gestor de energía del mismo, quien ordene al procesador la entrada en dichos modos tras asegurar la información de la sesión.

Los microprocesadores actuales normalmente recurren a técnicas de predicción de salto y ejecución especulativa para incrementar el rendimiento ejecutando fuera de orden los programas secuenciales. Sin embargo, las predicciones incorrectas pueden producir un incremento del consumo energético. Por este motivo, existen procesadores que implementan en hardware pocas funcionalidades de ejecución especulativa para permitir tener un consumo de energía menor, compensando la pérdida de rendimiento con técnicas de paralelismo (un ejemplo de este tipo de procesadores es la gama de procesadores *Atom* de Intel, que aunque se presentan en varias microarquitecturas, todas son de ejecución en orden y ofrecen paralelismo a nivel de hilo).

Por último, existen procesadores, como el UltraSPARC T2 de Sun Microsystems, que mediante un diodo controlan la temperatura del *die* y puede regular el flujo de instrucciones e hilos dinámicamente [Shah07].

La combinación de una gran cantidad de técnicas de gestión energética, junto con la administración automática de las mismas en los procesadores, hace que la predictibilidad del comportamiento de los mismos sea difícil de modelar [Lefurgy03].

[1] Todos los circuitos electrónicos, y más los de un grado de miniaturización elevado, pierden energía por el hecho de estar conectados y calientes. El *leakage power* se refiere a estas pérdidas.

2.1.1.1.1 AMD PowerNow!

Esta tecnología es la propuesta de implementación de la técnica de *DVS* para sus procesadores de 64 bits. Se basa en la reducción de la frecuencia del procesador, de forma que se reduce también el voltaje y el consumo.

Requiere tener soporte del sistema operativo, de la placa base, del chipset y del sistema operativo (por medio de controladores adecuados), además del microprocesador. En este caso, el control de la carga se realiza cada $1/30\ s$.

El nombre comercial de esta implementación es *Cool'n'Quiet* (para equipos de propósito general) u *Optimized Power Management* (para equipos servidores, como los AMD *Opteron*).

2.1.1.1.2 Intel Enhanced SpeedStep Technology

Se trata de una tecnología similar a la implementada por AMD. Desde la versión 2.1 de ésta también permite realizar escalado de frecuencia y de voltaje.

En los procesadores más modernos de Intel, esta tecnología se combina con otras, tales como el *power gating*, *frequency gating*, o el *Turbo Boost*. Esta última tecnología permite, en sistemas multinúcleo, incrementar la frecuencia de alguno de los núcleos de procesamiento por encima de la frecuencia nominal aprovechando la baja frecuencia del resto para optimizar el rendimiento de aplicaciones monolíticas sin sobrepasar la envoltura térmica del chip.

2.1.1.2 PCI Express

PCI Express es el estándar de facto para conectividad de tarjetas de expansión. El diseño de este bus de comunicaciones serie, de forma similar a las arquitecturas de red, en capas, incluye gestión de energía integrada, llamada *Active State Power Management* (*ASPM*) y diseñada para ser compatible con el estándar para gestión energética por parte de los sistemas operativos *ACPI* (como se explica más adelante).

ASPM es una tecnología que permite ahorrar energía cuando un canal *PCI Express* no se utiliza. Los enlaces se mantienen siempre activos para mantener la sincronía de las comunicaciones. La

implementación en los controladores de dispositivo del soporte para *ASPM* es obligatoria [Solari03].

PCI Express establece una tabla de equivalencias entre los estados S de *ACPI* y una nueva serie de estados de enlace, los estados L, de forma que se independiza la gestión energética del dispositivo, determinado mediante los estados D, y de los enlaces *PCI Express* (aunque hay ciertas combinaciones de estados no permitidas) [Kwa02].

Adicionalmente, *ASPM* permite la gestión diferenciada de transceptores, enlaces y diversos elementos del dispositivo. Para la gestión, se basa en el cálculo de latencias entre cambios de estado, definiendo, principalmente, los estados L0, L0s y L1 cuando el dispositivo está en el estado ACPI de encendido completo (D0) [Marshall06b].

Para gestionar esta funcionalidad es necesario, además de que el hardware soporte la gestión de estados y ofrezca ciertas características en cuanto a latencia en los cambios de estado, que el sistema operativo sea capaz de gestionar *ASPM* para permitir ir reduciendo la energía suministrada a los enlaces conforme van siendo menos utilizados [Microsoft06].

2.1.1.3 S.M.A.R.T.

La tecnología *S.M.A.R.T.* (*Self-Monitoring Analysis and Reporting Technology*) tiene como objetivo fundamental monitorizar el funcionamiento de los discos duros e informar al sistema operativo en caso de fallo o de predicción de fallo.

S.M.A.R.T. es una tecnología estándar que permite monitorizar de forma automatizada diferentes parámetros del funcionamiento de un disco duro y almacenarlos en una zona reservada del mismo. En base a valores límite y de umbral, se puede detectar y predecir fallos de funcionamiento.

Aunque se trata de una tecnología relativa a la gestión de fallos, ofrece ciertos atributos de interés, tales como los que permiten leer la temperatura, tiempo de encendido, ciclos de encendido o el tiempo que el disco tarda en cambiar de estado de energía.

2.1.2 Sistemas de gestión de la energía

Las siguientes tecnologías representan los sistemas de gestión que ofrece el hardware de forma unificada para manipular el estado del sistema a nivel global. Están pensados para establecer una interfaz común y estándar entre el sistema operativo y el hardware, aunque adicionalmente, éste puede ser controlado mediante gestión directa a través de los controladores de cada dispositivo.

2.1.2.1 Advanced Power Management

Advanced Power Management (APM) fue uno de los primeros acercamientos a la gestión energética, aunque ya ha caído en desuso y no está soportado por los sistemas operativos más utilizados.

Estaba controlado por la *BIOS* mediante la utilización de códigos de estado y de función que permitían cambiar y obtener el estado energético del sistema [Intel96].

2.1.2.2 Advanced Configuration and Power Interface

Advanced Configuration and Power Interface (ACPI) es un estándar abierto para el control y la monitorización de la energía en diversos dispositivos.

ACPI se considera, por tanto, una tecnología tanto de actuación como de monitorización, ya que especifica métodos para la obtención de información relativa a la gestión energética de los dispositivos y para cambiar los estados energéticos de los mismos.

ACPI es una evolución de estándares antiguos, como *APM*, manejados a nivel de *BIOS*. En su lugar, *ACPI*, se maneja a nivel de sistema operativo, permitiendo una flexibilidad y potencia mucho mayor.

Actualmente, el estándar *ACPI* es el más implantado para la gestión energética, tanto en los dispositivos hardware como en los sistemas operativos. El 16 de junio de 2009 se publicó la revisión 4.0 del mismo, en el que se añade, por ejemplo, soporte para el nuevo interfaz *USB 3.0* [ACPI09].

El objetivo fundamental de *ACPI* es ofrecer una interfaz estable y claramente definida a los sistemas operativos de forma que éstos puedan realizar una gestión de la energía económica e inteligente. Esta gestión de la energía por parte del sistema operativo se divide en dos módulos:

- ***ACPI*:** es la parte hardware, los fabricantes pueden implementarla o no, siendo opcional ofrecer los interfaces estándar.
- ***OSPM* (*OS directed Power Management*):** es el componente del sistema operativo que se encarga de consultar y modificar el estado energético de los dispositivos. Permite implementar complejas y sofisticadas políticas de gestión energética sin necesidad de incrustarlas en las *ROMs* de los dispositivos físicos. Todo ello redunda en flexibilidad y escalabilidad.

OSPM es un componente que depende de la implementación de cada sistema operativo y de las *APIs* que éste ofrezca a los desarrolladores; sin embargo, la interfaz entre hardware y software (*ACPI*) sí es invariable entre fabricantes. El modo de almacenamiento y gestión de esta información por parte del hardware queda a decisión del propio fabricante, por tanto, *ACPI* es independiente de las soluciones que los fabricantes desplieguen en sus productos y define un interfaz concreto para acceder a la información de energía.

Se trata de un estándar grande y complejo, de bajo nivel, en el que se describen todas las tablas que el hardware ha de implementar, así como el *byte code* para acceder a las mismas.

Para completar la especificación, existe una implementación neutral de referencia del estándar *ACPI* lista para que los fabricantes puedan utilizarla denominada *ACPI Component Architecture* (*ACPICA*) [Intel09a]. Para utilizar esta implementación basta con hacer un puente entre los servicios proporcionados por la librería y los servicios del sistema operativo.

2.1.2.3 Simple Network Management Protocol

SNMP (*Simple Network Management Protocol*) es un protocolo de capa de aplicación de la suite de protocolos de Internet cuyo objetivo es permitir la monitorización de equipos conectados a una red [Case90].

Para permitir la monitorización, *SNMP* se compone de un protocolo de red y un esquema estándar de objetos y datos.

El escenario típico de utilización de *SNMP* se da cuando se cuenta con ciertos equipos que van a ser monitorizados (y posiblemente administrados) por uno o más equipos monitores,

denominados *NMS* (*Network Management Systems*). Cada uno de los equipos que van a ser administrados cuenta con un agente SNMP que expone a la red información sobre ese equipo en forma de variables, tales como la cantidad de memoria libre o la ruta por defecto de dicho equipo.

Los *NMS* podrán consultar la información ofrecida por los agentes *SNMP* mediante la utilización de software creado a tal efecto y, en ciertos casos, podrán editar esas variables, permitiendo no sólo la monitorización, sino también la actuación sobre los dispositivos administrados.

Para obtener información de los dispositivos, no sólo es posible hacerlo mediante el sistema de *polling* (consultando manualmente), sino que se ofrece un sistema notificaciones (*traps*) que permiten a los agentes notificar cambios de forma asíncrona a los *NMS*.

La información de administración se organiza en una base de datos jerárquica, *MIB* (*Management Information Base*), de forma que se pueda acceder de manera sencilla a la información, desde diferentes protocolos y que los fabricantes de dispositivos puedan agregar sus propios atributos sin interferir con el estándar [McCloghrie91].

Aunque *SNMP* y *MIB* en particular fueron diseñados para la administración de redes, existen ciertas extensiones experimentales que pueden utilizarse para la gestión energética, de forma que puedan leerse o modificarse atributos como estados *ACPI* de los dispositivos [Blanquicet08].

2.1.2.4 Windows Management Infrastructure

Windows Management Infrastructure (*WMI*) es la implementación de Microsoft para Windows de los estándares del *Distributed Management Task Force* (*DTMF*). Por su diseño, permite acceder de forma cómoda a estándares y tecnologías relacionadas, tales como *CMI*, de la propia *DTMF* [DTMF09], y *SNMP*.

WMI se construye como una extensión de la infraestructura de controladores de dispositivos de Windows que permiten la administración, monitorización y actuación sobre decenas de elementos de administración de Windows desde los lenguajes de programación más comunes.

Adicionalmente, es posible la administración remota, puesto que *WMI* incluye un mecanismo basado en *DCOM*. También soporta la realización de consultas en un leguaje similar a *SQL*, tanto local como remotamente y ofrece una herramienta de consola de comandos para obtener información del sistema de forma manual.

El funcionamiento de *WMI* se basa en una arquitectura consumidor-productor. En primer lugar, los productores (llamados en terminología *WMI* proveedores) son cualquier tipo de entidad que permita publicar datos *WMI*, por ejemplo, existen entidades *SNMP*, Win32 o incluso entidades de código manejado .NET.

Estos proveedores ofrecen sus datos a la infraestructura *WMI* que, a su vez, mediante una interfaz *COM*, ofrece dichos datos a los consumidores de los mismos.

Los consumidores pueden ser cualquier tipo de proceso, bien sea una aplicación o un servicio. Puede estar escrito en multitud de lenguajes que tengan acceso a una librería WMI o a un adaptador, tales como lenguajes estáticos, dinámicos o de script.

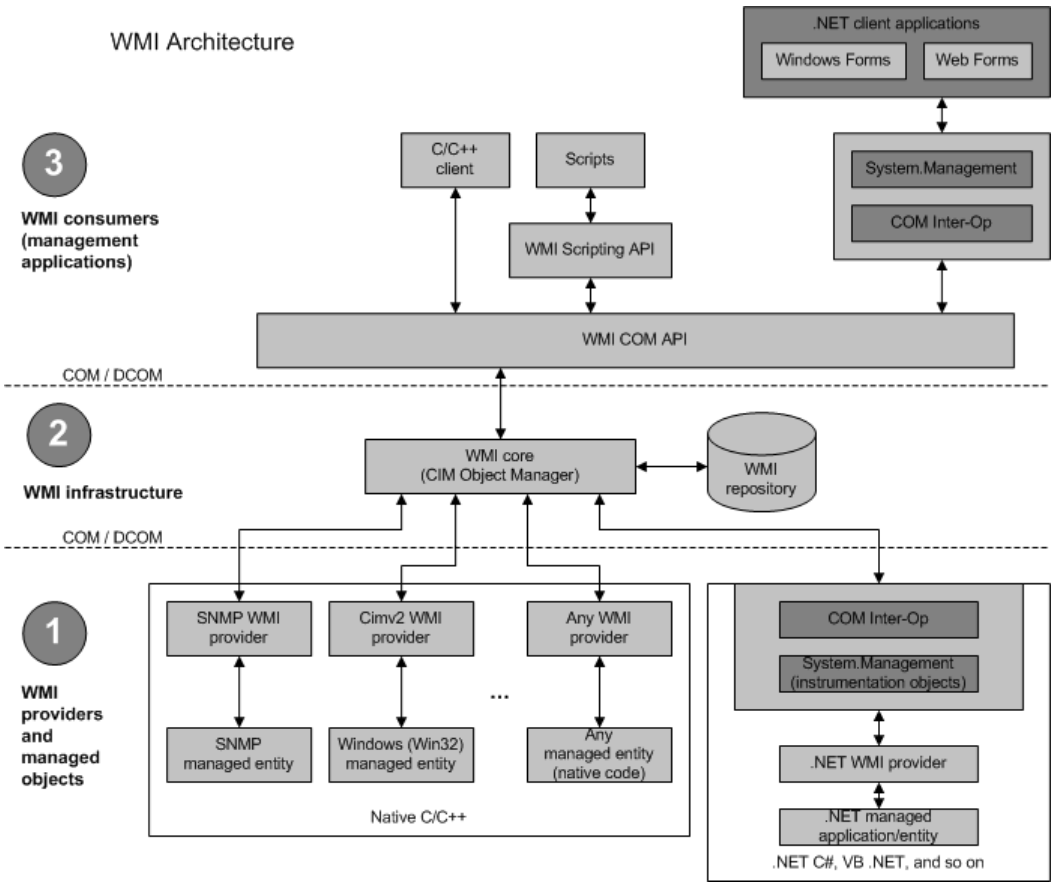


Figura 2: Arquitectura de la infraestructura WMI.

La Figura 2 representa un resumen arquitectónico de la infraestructura de *WMI*. Como se puede ver, todo el acceso desde las aplicaciones consumidoras de datos deben de acceder a través de una capa *COM*, lo que permite acceder desde múltiples lenguajes. También existen métodos de acceso basados en *ODBC* y *Active Directory*.

El otro componente fundamental son los proveedores. Éstos actúan como intermediarios entre el núcleo *WMI* (*WMI core*) y los objetos manejados (*managed objects*).

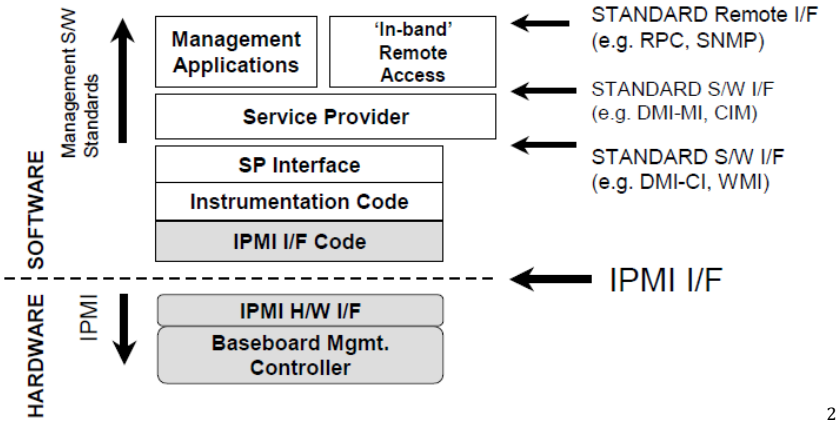
Los objetos manejados son entidades, bien físicas o lógicas, descritas en MOF (*Managed Object Format*, un lenguaje de descripción de objetos orientado a objetos) y almacenadas en el repositorio *WMI*.

Para acceder a estos datos, *WMI* hace uso de los proveedores. Cuando el núcleo recibe una petición que no es capaz de resolver (ya que sólo es capaz de resolver determinadas peticiones estándar), la redirige al proveedor que sí es capaz de resolverla.

Esta arquitectura permite ser extendida con nuevos proveedores y objetos manejados, por ejemplo, mediante los controladores de dispositivos de los fabricantes.

2.1.2.5 Intelligent Platform Management Interface

Intelligent Platform Management Interface (*IMPI*) es una especificación pública desarrollada por Intel en colaboración con otras empresas del sector, como HP o NEC. Permite implementar y ofrecer al sistema operativo funciones de gestión autonómica de diferentes aspectos del computador directamente desde el gestor del *firmware* del mismo [Intel09b].



[2]

Figura 3: Capas de tecnologías de monitorización y actuación

IPMI se integra con otras tecnologías de monitorización y actuación de más alto nivel, tal y como se puede observar en la Figura 3. De esta forma, es el puente encargado de proporcionar la

[2] Figura obtenida de [Intel09b]

información desde el hardware a capas más altas, como *WMI* en caso de un sistema operativo Windows.

2.2 Niveles de operación energética

Además de las tecnologías de monitorización y actuación, se pueden destacar los diferentes niveles de operación energética que los diversos componentes permiten tener.

2.2.1 Niveles de operación genéricos

Se pueden definir una serie de niveles de operación energética de los computadores de forma genérica que cualquier tipo de estándar y sistema operativo actual soportan. A continuación se describen de menor a mayor ahorro energético:

- **En funcionamiento:** el sistema está funcionando completamente. En este estado, los procesadores están completamente activos, así como la memoria principal. Algunos dispositivos de entrada/salida podrían apagarse, como discos o monitores.
- **Suspensión:** la memoria principal mantiene su contenido y, por tanto, debe de ser refrescada, sin embargo el procesador para de ejecutar instrucciones y su estado se almacena temporalmente en la memoria principal para una recuperación rápida. Éste se introduce en un nivel de ahorro energético casi total.
- **Hibernación:** el contenido de la memoria se vuelca a disco y tanto el procesador como la memoria se apagan. El resto de dispositivos así lo hacen, aunque el sistema puede responder a impulsos externos, como eventos *USB*, que lo despierten. A todos los efectos, el sistema está apagado y la gestión de la recuperación depende del sistema operativo.
- **Apagado:** se trata de un estado energético igual al anterior, pero en el que no se guarda información de recuperación ya que el sistema operativo ha de ser cerrado por completo. El sistema sigue consumiendo energía para mantener alimentados elementos como las tarjetas de red latentes.
- **Apagado mecánico:** además del cierre del sistema operativo, se corta por completo la alimentación a cualquier dispositivo del sistema.

2.2.2 Niveles de operación definidos por el estándar *ACPI*

Los niveles más utilizados y extendidos son los definidos por el estándar *ACPI*, debido a la implantación de éste. Casi cualquier dispositivo y equipo informático permite su gestión mediante *ACPI* y la mayoría de los sistemas operativos mayoritarios lo implementan.

Este estándar define una serie de niveles de operación energética en función del tipo de dispositivo, de esta forma, se pueden tener:

- **Niveles globales:** indican el estado de todo el sistema de forma conjunta.
- **Niveles de dispositivo:** indican el estado de dispositivos periféricos concretos.
- **Niveles de CPU:** indican el estado energético de la *CPU*.
- **Niveles de rendimiento:** además de los niveles energéticos, los dispositivos y la *CPU* permiten indicar y establecer estados de rendimiento. Ejemplos de estos estados son las tecnologías Intel *SpeedStep* o AMD *Cool'n'Quiet*.

2.2.2.1 Niveles de operación globales

Los niveles de operación globales describen el estado energético de todo el sistema de forma conjunta. El estándar define hasta 4 niveles de operación con diferente relación de consumo energético, rendimiento y latencia de arranque. En la Tabla 1 se muestra un resumen comparativo de los mismos:

- **G0:** indica que el sistema está en funcionamiento normal (también se denomina S0).
- **G1:** en este nivel el sistema está suspendido. Existen 4 subniveles de suspensión:
    - **S1:** las cachés del procesador se apagan y éste deja de ejecutar instrucciones. Aunque se mantiene la alimentación a la *CPU* y la memoria principal, el resto de dispositivos se apagan (salvo que indiquen explícitamente lo contrario).
    - **S2:** la *CPU* se apaga completamente.
    - **S3:** se mantiene la alimentación a la memoria principal (este nivel se conoce generalmente como suspensión).
    - **S4:** la memoria principal también se apaga, almacenando su contenido en un dispositivo no volátil, como un disco duro (este nivel se conoce como hibernación o suspensión no volátil).

- **G2:** todos los dispositivos se apagan. El sistema operativo deberá ser cargado de nuevo al encenderlo puesto que no almacena su estado. Sin embargo, algunos dispositivos, tales como una tarjeta de red, mantienen su alimentación para permitir el encendido del sistema mediante estímulos externos. Este estado también se denomina S5.
- **G3:** en este estado todos los dispositivos salvo el reloj de tiempo real se apagan y se les retira la alimentación (se conoce generalmente como apagado mecánico). Sólo se podrá volver a encender el sistema de forma manual.

Estado	Consumo	Latencia de arranque
G0	Máximo – depende del gestor energético del S.O.	–
G1	Moderado	Mínima
G2	Mínimo	Larga
G3	Ninguno[3]	Larga (sin funcionalidad)

Tabla 1: Resumen de estados ACPI globales

2.2.2.2 Niveles de operación de dispositivo

Los niveles de operación de dispositivo permiten conocer y establecer el estado energético de los diversos periféricos del sistema. En este caso, su implementación es dependiente del dispositivo, pero han de satisfacer:

- **D0:** el dispositivo está plenamente funcional.
- **D1, D2:** estados de consumo intermedio, que dependerán del dispositivo. Se espera que el estado D1 ahorre menos energía que D2, pero mantenga más funciones u ofrezca una latencia de reciclado menor. Definir el estado D2 no es obligatorio.
- **D3:** el dispositivo está completamente apagado.
- **D3hot:** el dispositivo está apagado, pero mantiene el contexto almacenado.

2.2.2.3 Niveles de operación de CPU

Estos niveles son específicos de la *CPU* y pueden variar ligeramente su significado. En general, establecen niveles de parada de la *CPU* que cuestan cada vez más recuperarse de ellos (aunque también el consumo es menor):

[3] El consumo nunca puede llegar a ser nulo, pues sólo mantener el reloj de tiempo real tiene un coste de batería. Sin embargo, en este estado es seguro manipular el hardware sin riesgos de electrocución.

- **C0:** la *CPU* está totalmente operativa.
- **C1:** la *CPU* está encendida, pero no ejecuta instrucciones. La salida de este estado es inmediata y transparente para el software que se encuentra en ejecución.
- **C2:** la *CPU* se mantiene visible al software, pero para además el reloj interno. Este estado también es transparente para el software que se encuentra en ejecución.
- **C3:** la *CPU* ya no mantiene la coherencia de caché y está prácticamente apagada (aunque mantiene cierto estado). Este modo es el de menor consumo y el que más tiempo necesita para recuperar.

2.2.2.4 Niveles de rendimiento

Los niveles de rendimiento, de la forma Px, son aplicables tanto a la *CPU* como a cualquier dispositivo y tienen sentido cuando éste está completamente funcional (niveles D0 ó C0).

Estos niveles indican el escalón de rendimiento en el que se encuentra funcionando un dispositivo y son totalmente dependientes del dispositivo, tanto en número de niveles como en significado, aunque el nivel P0 siempre indica el nivel máximo rendimiento y no habrá un nivel superior a P16.

Las implementaciones más extendidas de estos niveles son las tecnologías de gestión energética de procesadores Intel *SpeedStep* o AMD *Cool'n'Quiet*.

2.2.2.5 Relaciones entre estados ACPI

A continuación se muestra la relación existente entre los diversos estados *ACPI* globales:

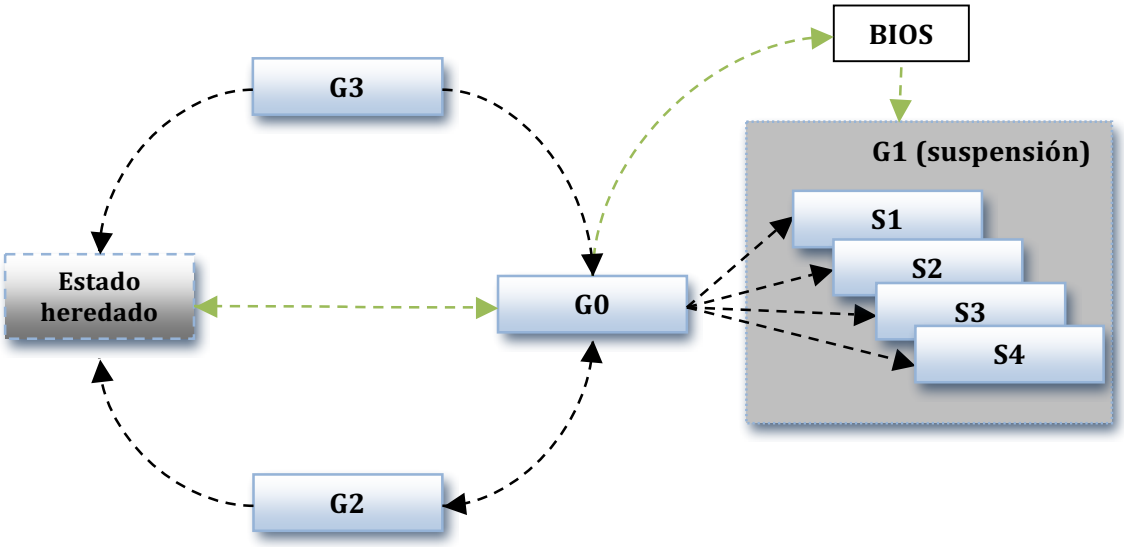


Figura 4: Diagrama de estados ACPI

En la Figura 4 puede observarse cómo es posible acceder desde el estado G0 a todos los estados de suspensión, pero cómo no puede cambiarse entre cada uno de ellos directamente.

Los equipos que soportan mecanismos antiguos de gestión de energía arrancan en un estado heredado y luego pasan al estado G0 de funcionamiento normal. Para entrar en estados de suspensión hacen uso de rutinas básicas de la *BIOS* utilizando las transiciones y estados especiales marcados a color verde en la Figura 4.

Nótese cómo entrar en el estado G3 sólo es posible mediante la desconexión de la alimentación de forma manual, ya que no hay una transición de entrada definida.

2.2.3 Equivalencia de estados energéticos *ACPI*

En la Tabla 2 se muestra una tabla de equivalencias entre los estados genéricos definidos en la sección 2.2.1 y los estados del están dar *ACPI*:

Nivel genérico	**Estado *ACPI***
En funcionamiento	G0
Suspensión	S1, S2 y S3
Hibernación	S4
Apagado	G2 (S5)
Apagado mecánico	G3

Tabla 2: Equivalencias de estados energéticos genéricos y estados ACPI

2.3 Gestión energética en Windows Server 2008

La versión 2008 de Windows Server es un sistema operativo que satisface totalmente el estándar *ACPI* (de hecho, sólo soporta *ACPI* [Marshall06a]). En principio, Windows Server cuenta con un modo de gestión energética automático que hace un balance de la carga y de la energía. El principal problema de este modo es que se restringe al estado de una única máquina. Además, ofrece una granularidad mucho mayor que versiones anteriores para la gestión de los estados P del procesador [Berard07] y nuevos estados de energía compuestos [Stemen06].

Adicionalmente, se proporcionan herramientas, como *powercfg*, que permiten configurar y cambiar todos los parámetros de gestión energética del sistema creando nuevas políticas de configuración energética. Además, esta herramienta, permite generar informes para determinar la existencia de problemas de consumo energético [Microsoft09].

Windows Server no proporciona una *API* convencional para el acceso y control de las funciones y tablas *ACPI* del computador, sino que el acceso se realiza mediante el uso de controladores (*drivers*), que tienen la capacidad de evaluar métodos *ACPI* sobre los dispositivos. Una estructura común de aplicaciones que realizan acceso y modificación de los datos *ACPI* es la formada por dos componentes: el *driver* que accede al hardware y proporciona funciones a la aplicación que ofrece el acceso al usuario o posee la inteligencia de control [Microsoft07].

2.3.1 Gestión automática de dispositivos en Windows Server 2008

Dentro de un nodo de computación de un clúster, existen múltiples dispositivos que, mediante su control, permiten optimizar el consumo energético de un nodo. Los dispositivos que más energía consumen y, por tanto, los mejores candidatos a controlar son:

- Estado del procesador.
- Discos (otro tipo de unidades, como *DVDs*, sólo cuando están en uso).
- Tarjetas de expansión.
- Módulos de memoria.

Todos estos dispositivos están bien gestionados por el propio sistema operativo. En el caso de Windows Server 2008, mediante el establecimiento de una política adecuada de gestión de energía se pueden alterar todos los parámetros y dejar que el sistema operativo

automáticamente, y en función de la carga del sistema, gestione el estado energético de todos los dispositivos que tiene instalados.

En primer lugar, el sistema operativo permite establecer intervalos de frecuencias mínimas y máximas para las que los procesadores del nodo van a funcionar, permitiendo reducir dinámicamente la carga de la *CPU* para que ésta reduzca su frecuencia.

Respecto a la configuración de los discos, permite apagarlos de forma automática, aunque en este caso, la configuración sólo permite establecerse en base a umbrales de desuso temporal.

Windows Server permite gestionar el consumo energético de las tarjetas *PCI Express*, tales como tarjetas gráficas, tarjetas de red, tarjetas *RAID* u otras tarjetas de expansión mediante el ajuste dinámico de los vínculos *PCI Express*, reduciéndolos o activándolos según sea necesario. Además, si el controlador de la tarjeta a controlar es compatible con *ACPI*, las tarjetas no utilizadas entran en un estado de bajo consumo.

Por último, relativo a la gestión de los módulos de memoria principal, grandes consumidores de energía, éstos no se pueden apagar bajo demanda, sino que entran en la gestión de la suspensión. Windows Server permite realizar suspensiones escalonadas en función de políticas temporales.

En primer lugar, superado un umbral de inactividad, el sistema entra en una suspensión S3, estado en el que se entra y del que se sale de forma rápida. En este estado sólo se alimenta la memoria, todos los demás dispositivos se apagarán, salvo que la política los excluya explícitamente.

Superado otro umbral de tiempo, el sistema pasa a una suspensión más profunda, S5, todo el sistema se apaga y se almacena el contenido de la memoria en un fichero de hibernación. Todo el hardware está apagado, a excepción de dispositivos capaces de activar el equipo, tales como un concentrador *USB* o una tarjeta de red compatible con *Wake On LAN*.

Para salir de este estado es necesario cargar el sistema operativo completamente (generalmente, se hará una carga ligera para restaurar el archivo de hibernación), por lo que la entrada y salida de este estado conlleva más tiempo que en una suspensión ligera S3.

3 GESTIÓN ENERGÉTICA AUTONÓMICA

La gestión de la energía se ha vuelto un problema capital en los últimos tiempos en el que se ha llegado a necesitar una mayor inversión en consumo energético y refrigeración que en los propios equipos y software [Pinheiro01].

Por este motivo es fundamental realizar una optimización en el consumo energético de los equipos de grandes centros de datos. La principal área de mejora se sitúa en el hardware, donde existen numerosas tecnologías que permiten monitorizar y controlar la energía, pero también puede escribirse software energéticamente más eficiente [Steigerwald08] y ser optimizado mediante tecnologías de compiladores sensibles al consumo [Hsu03]. Ciertamente, en la gestión energética de un equipo, todos y cada uno de los componentes que lo forman son susceptibles de ser optimizados y mejorados [Vahdat00].

Adicionalmente, la densidad de consumo energético se está volviendo otro problema grave, sobre todo en dispositivos como microprocesadores, que requieren disipadores cada vez más grandes y eficientes [ChenK06] y de la optimización en diversos aspectos, como la distribución física de los equipos [Ranganathan10].

Existen diversas organizaciones que se encargan de certificar la eficiencia energética de los dispositivos y sistemas, tales como *Energy Star*, *TCO* o *EPEAT*. Por tanto, es fundamental buscar las marcas de certificación en los dispositivos a la hora de comprarlos.

En este capítulo se repasarán las diferentes técnicas para la gestión dinámica del consumo energético [Chedid02] y se estudiará cómo integrarlas en un sistema autonómico, además de analizar y catalogar los diferentes trabajos que la comunidad científica ha llevado a cabo en el área de la gestión autonómica de energía en grandes centros de datos.

3.1 Computación autonómica

En [Kephart03] se establece la primera definición de computación autonómica derivada del manifiesto producido por IBM en [Horn01], que indicaba que la crisis del software iba a convertir el manejo de trillones[4] de dispositivos conectados a Internet en una pesadilla. Adicionalmente, el desarrollo y mantenimiento de sistemas software con millones de líneas de código iba a dejar de ser viable.

Por este motivo, la única salida que se ve al problema es la computación autonómica, definida como "sistemas de computación que pueden administrarse a sí mismos dados una serie de objetivos de alto nivel por parte del administrador".

Tal y como se puede deducir de esta definición, obtener sistemas capaces de auto administrarse, lo que incluye configurarse, adaptarse y repararse, es prácticamente un sueño a día de hoy y sólo puede ser conseguido incrementado poco a poco el nivel de abstracción de la manipulación del sistema y evolucionando los sistemas y modelos existentes.

Esto es a lo que la segunda parte de la definición se refiere. Ésta indica que han de establecerse una serie de objetivos de alto nivel para el comportamiento del sistema, esto es, definiendo una política en base a objetivos que el sistema ha de satisfacer en su auto configuración y adaptación.

Por este motivo es fundamental mejorar y optimizar los modelos que representan los sistemas, porque es a través de ellos como se permite a los propios sistemas tener conciencia de sí mismos y satisfacer políticas autonómicas. En general, una política autonómica completa debería de basarse en los siguientes puntos:

[4] Tomado literalmente, se trata de trillones en la concepción americana, esto es, billones.

Concepto	Computación convencional	Computación autonómica
Auto-configuración	Utilizar varias plataformas y configuraciones en un mismo centro de datos es prácticamente imposible.	Interoperabilidad entre plataformas automática.
Auto-optimización	Los sistemas tienen cientos de parámetros con dependencias no lineales que hay que ajustar a mano.	Pocos parámetros de muy alto nivel, estimables dinámicamente en tiempo de funcionamiento.
Auto-reparación	Problemas para depurar las fuentes de los problemas y errores.	Los sistemas indican dónde se están produciendo los fallos para reparaciones rápidas.
Auto-protección	Detección y recuperación de ataques manual y compleja.	Los sistemas se defienden automáticamente de los ataques, los detectan y atajan.

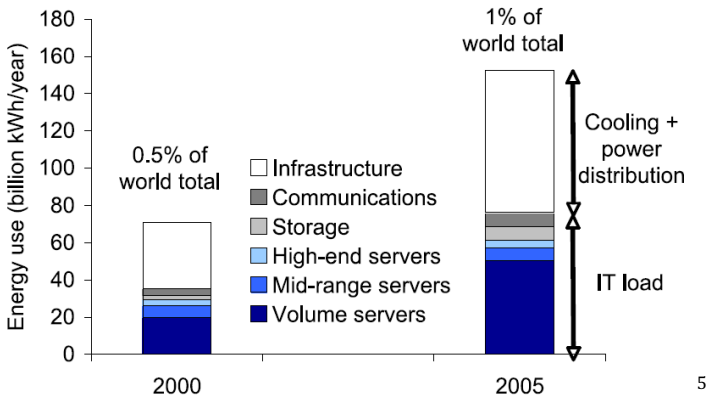
Tabla 3: Resumen de objetivos de computación autonómica

Como se observa en la Tabla 3, el satisfacer todos los objetivos de un sistema autonómico permitiría reducir en gran medida los esfuerzos de mantenimiento y administración que un gran centro de datos conlleva, permitiendo hacer crecer los centros de datos y la densidad de integración de máquinas en los mismos.

Los sistemas autonómicos se basan en la aplicación de conceptos de aprendizaje automático (basado en modelos) que obtienen datos de sensores en el sistema y deciden cambios mediante actuadores en el mismo. Para la toma de decisiones, se toman como entrada también los objetivos marcados por el administrador.

3.2 El problema de la energía en los grandes centros de datos

En los tiempos que corren, cada vez más servicios están siendo proveídos por servidores. Normalmente, estos servidores se agrupan en grandes centros de datos, dada la masividad de servicios como proveedores de vídeo bajo demanda o gestores de correo electrónico, que cuentan con millones de usuarios en todo el mundo, tales como *Gmail*, que cuenta con más de 170 millones de usuarios en todo el mundo [Tang08].



[5]

Gráfico 1: Estimación de la distribución del consumo y refrigeración

Por este motivo, no sólo la escalabilidad del servicio en términos de calidad de servicio (*QoS*) se han vuelto un problema clave, sino también la escalabilidad y densidad del consumo energético conforme la densidad de integración de máquinas en centros de datos aumenta [Barroso03].

La tendencia actual es que los centros de datos van a requerir mayor esfuerzo económico para ser alimentados y refrigerados que para comprar el hardware de computación que realiza el trabajo, que se mantiene en un coste constante a lo largo del tiempo [Barroso05]. De hecho, en 2005, el coste del consumo energético para refrigeración ya era igual que el coste de la infraestructura informática de los centros de datos de todo el mundo, que suponen el 1% del consumo energético global [Koomey08].

Como consecuencia de esta tendencia, se han realizado numerosos trabajos relativos a la optimización del consumo energético aprovechando tecnologías existentes de gestión, determinando cuáles son las configuraciones óptimas para clústeres y centros de datos y desarrollando algoritmos que permiten auto-modelar y configurar servicios para la estimación y configuración de centros de datos y clústeres con el objetivo de minimizar el consumo energético [Rajamani08].

El resto de esta sección, donde se repasan los trabajos previos a éste, se estructura de forma en la que se revisan los algoritmos de provisión dinámica, algoritmos de gestión de *DVS* y *DFS*, algoritmos híbridos, así como los diferentes métodos de modelado y estimación. También se hará una separación entre los sistemas de gestión para centros de datos, para clústeres y para computadores, sin olvidar de los sistemas jerárquicos que combinan varios de estos niveles.

En primer lugar, y antes de pasar a la revisión de trabajos, conviene diferenciar qué entornos de aplicación son de interés para los algoritmos y sistemas analizados:

[5] Gráfico obtenido de [Koomey08]

Nivel	Características
Grid / *Cloud*	Es una abstracción de muy alto nivel, en general no son aplicables políticas de gestión energética directamente, debido a la federación de clústeres y centros de datos geográficamente muy dispersos.
Centro de datos (*CPD*)	Aloja uno o varios clústeres, permite una gestión integrada de los mismos y un balanceo de carga de muy alto nivel.
Clúster	Pueden ser homogéneos o heterogéneos, en ambos casos se tiene información integrada muy detallada y permiten mayor granularidad en la gestión. El balanceo de carga es muy detallado.
Computador	La gestión es automática, pero dirigida por el hardware y el sistema operativo sin contar con información global de niveles superiores.

Tabla 4: Niveles de aplicación de interés

Fundamentalmente, existen dos niveles de aplicación de técnicas de gestión autonómica, que difieren en la información con la que se cuenta para tomar las decisiones: a nivel de clúster se tiene información relativa a todo el sistema y a nivel de computador, donde, en principio, se cuenta con sólo información local. Esta diferencia de niveles determinará el algoritmo a implementar y la forma en la que se integran los componentes. No obstante, existen alternativas híbridas que combinan ambas aproximaciones.

3.2.1 Gestión energética proporcional

Tal y como se apunta en [Barroso07], el objetivo de la optimización que debe proporcionar una gestión autonómica para un centro de datos ha de ser la proporcionalidad en la relación entre el consumo energético y los niveles de utilización de los computadores.

La tendencia actual en el desarrollo de procesadores es dirigirlos a sistemas masivamente paralelos, incorporando más nodos de procesamiento, nodos de propósito dedicado e incrementado las memorias caché. De esta forma, el rendimiento por ciclo se mantiene estable, pero se incrementa mucho el *throughput* de un solo microprocesador, ofreciendo grandes prestaciones en sistemas servidores, pero necesitando mucho más ancho de banda de memoria que las arquitecturas precedentes (un efecto similar se produjo al introducir los primeros

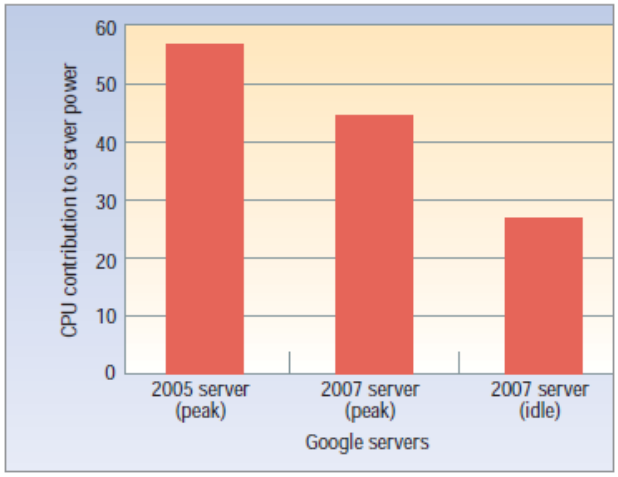
procesadores segmentados, que requerían incrementos de ancho de banda iguales al grado de segmentación implementado).

En este sentido, se incrementan los canales de memoria y se implementan estrategias de predicción de accesos y paralelismo en la memoria, pero el salto de prestaciones (*memory gap*) entre la *CPU* y la memoria es cada vez más sustancial.

En grandes centros de datos, la eficiencia energética es fundamental, de forma que exista una proporcionalidad entre el nivel de operación del computador y el consumo energético del mismo. Esto permite incrementar la densidad instalada, cambiando a entornos *blade*, y reducir el consumo energético en disipación de calor.

3.2.2 Aportación y composición del consumo de un servidor

En función del tipo de servidor y carga, puede variar, pero generalmente, la mayor parte del consumo de un servidor típico se concentra en la CPU y la memoria principal [Schlatter05].



[6]

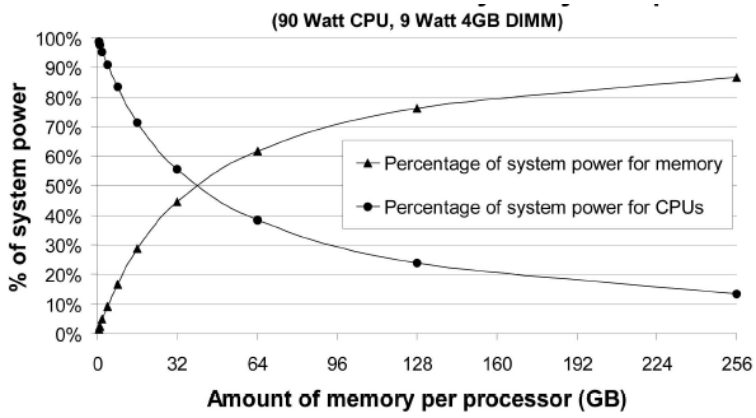
Gráfico 2: Contribución al consumo de la CPU en dos generaciones de servidores de Google

Además de la contribución de la CPU al consumo total del computador, la gestión energética del mismo permite gestionar de forma más proporcional el consumo del mismo, permitiendo la aparición de un salto entre el consumo del mismo cuando está sin utilización (*idle power*) y completamente utilizado (*peak power*). Esta diferencia, que puede llegar a los 70 puntos porcentuales en procesadores (hasta 50 en memorias RAM, 25 en discos duros y sólo 15 en

[6] Gráfico obtenido de [Barroso07].

equipamiento de red), tal y como muestra el Gráfico 2 (en el caso de cargas de servidor, donde las bajas utilizaciones son escasas, el salto en caso de CPU suele estar en el entorno de los 30 puntos), es la clave que permite mejorar la proporcionalidad del consumo energético, tanto en entornos de computación de servidores y centros de datos como en entornos personales.

En cuanto a la memoria, es especialmente destacable cómo en sistemas de alta capacidad de memoria principal, como servidores de bases de datos, el consumo de la misma, puede llegar a dominar al consumo de las CPUs [Matthew09]:



[7]

Gráfico 3: Impacto en el consumo combinado de la cantidad de RAM

Hay que tener en cuenta que la composición del consumo total del computador depende de la carga que se aplique al mismo, esto es, un servicio intensivo en procesamiento de datos, como procesamiento de datos procedentes de experimentos de física, tendrá un consumo de CPU mucho más elevado que un servicio de procesamiento de correo electrónico, donde el peso del consumo de disco será más alto [Economou06].

Adicionalmente, pueden realizarse modelos complejos de la aportación de cada componente al consumo del computador, en [Cai06] y [Gurumurthi02] se realizan sendos análisis detallados del consumo del disco y memoria en los diferentes estados de operación de los mismos. Tener en cuenta estos modelos es de interés de cara a realizar un modelo muy detallado del sistema, que aunque preciso, puede ser complejo de estimar debido al alto número de parámetros de los que dependa.

Finalmente, y con respecto a la aportación particular de discos duros en los servidores convencionales basados en un computador que realiza su almacenamiento principal en servidores de almacenamiento remoto, como redes *NAS* (*Network-attached Storage*, por ejemplo

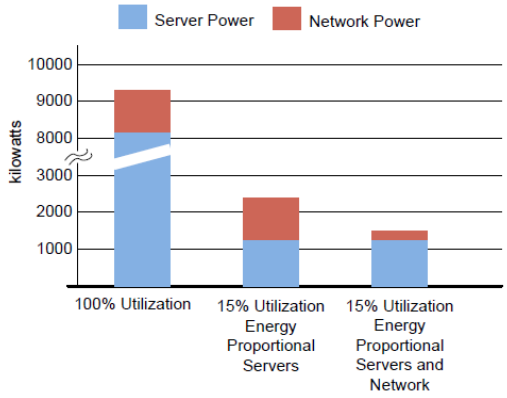
[7] Gráfico obtenido de [Matthew09].

servidores *NFS* ó *SMB*) o SAN (*Storage Area Network*, por ejemplo cabinas de discos de fibra o *iSCSI*), es necesario destacar que el almacenamiento local se utiliza para cachés temporales y, dependiendo de la carga, espacios de trabajo locales. En [Carrera03] se destaca la posibilidad de sustituir estos espacios de almacenamiento locales por dispositivos de menor consumo y, sobre todo, con un alto grado de proporcionalidad energética respecto a la utilización de los mismos (ver sección 3.2.1 para más detalles sobre proporcionalidad energética) que permita una gestión combinada en algoritmos jerárquicos (ver sección 3.8) y permita reducir más el consumo en cargas con poca importancia de procesamiento de datos locales.

Adicionalmente, en [Gurumurthi03] se ponen de manifiesto, mediante el modelado detallado de los discos, las ventajas del uso de discos con *DRPM* (*Dynamic Revolutions Per Minute*), de forma que estos discos son mucho más proporcionales en el consumo respecto a la utilización.

En una última referencia a la optimización de los discos duros, cabe destacar la aplicación de técnicas diseñadas para equipos portátiles, que al estar equipados con baterías, deben optimizar el consumo energético. En [Heath02] se modela el consumo del disco de forma analítica y se proponen dos mecanismos para su optimización (basados en la desactivación del mismo) que cuentan con la participación de las aplicaciones en el proceso de optimización. Este método, definido para el disco, puede ser extendido a más dispositivos.

Además del disco, otro elemento que cobra un peso importante en el consumo y optimización de un centro de datos son los equipos de red [Abts10].



[8]

Gráfico 4: Comparativa de proporcionalidad de elementos de red

Tal y como se observa en el Gráfico 4, en un clúster no optimizado, los elementos de red tienen un peso muy importante, cercano al 15%. Cuando se optimiza el consumo de los elementos de

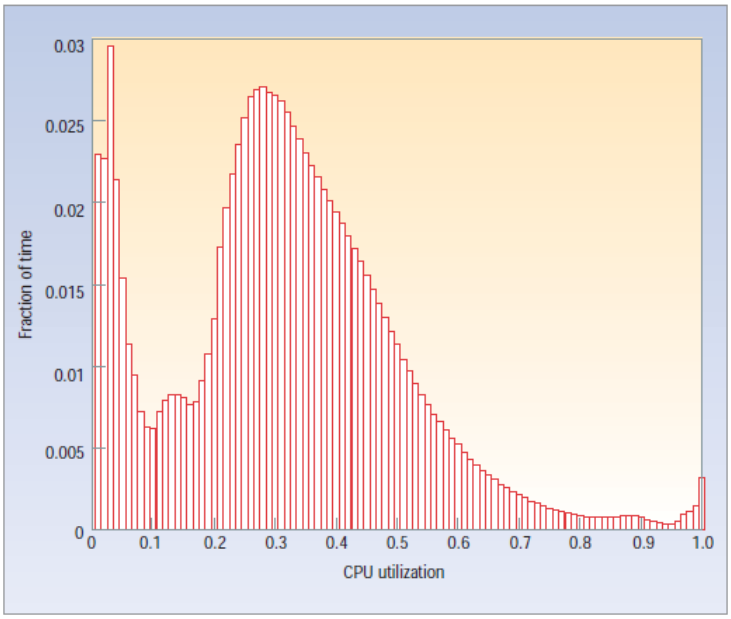
[8] Gráfico obtenido de [Abts10].

computación, haciéndolos proporcionales a la utilización, los elementos de red pueden llegar a pesar hasta un 50% en el consumo total, por lo que son objeto de una nueva optimización [Simunic02a].

3.2.3 Modelado de cargas de servidor y eficiencia energética

Las primeras generaciones de sistemas integrados de gestión de energía trataban de optimizar el consumo medio del computador, sin tener en cuenta la utilización y el modelado de la misma.

Ha de tenerse en cuenta, además, que el modelado del consumo y utilización difiere notablemente en los entornos clásicos de servidor y computador personal.



[9]

Gráfico 5: Histograma de utilización de 5.000 chunk-server GFS en Google durante 6 meses

Tal y como puede observarse en el histograma mostrado en el Gráfico 5, en un entorno de servidor es difícil estar en un estado de no utilización (no así en entornos de computación personal), pero también es raro llegar a puntos donde la utilización de la CPU es completa.

De esta forma, se puede observar cómo la distribución de la utilización de la CPU en entornos de servidor tiene su centro de masas fundamentalmente entre el 10% y el 50% de utilización.

[9] Histograma obtenido de [Barroso07].

Esto es debido fundamentalmente a que un servidor no cuenta prácticamente con *idle time* en su funcionamiento, mientras que un computador de escritorio interactivo cuenta con largos períodos de *idle time*, lo que permite que existan grandes períodos de tiempo en los que el procesador puede reducir su consumo mediante técnicas como el *power gating*, *DVS* o *DFS*. En el trabajo de [Bai07] se realiza un análisis de los tiempos efectivos que cuesta poner un computador en estado de suspensión y, además, los diferentes dispositivos que componen el mismo, resultado que en determinadas combinaciones, los cambios rápidos de estado son contraproducentes para la optimización del consumo, por lo que hay que minimizarlos en lo posible (ver sección 4.3.1.1).

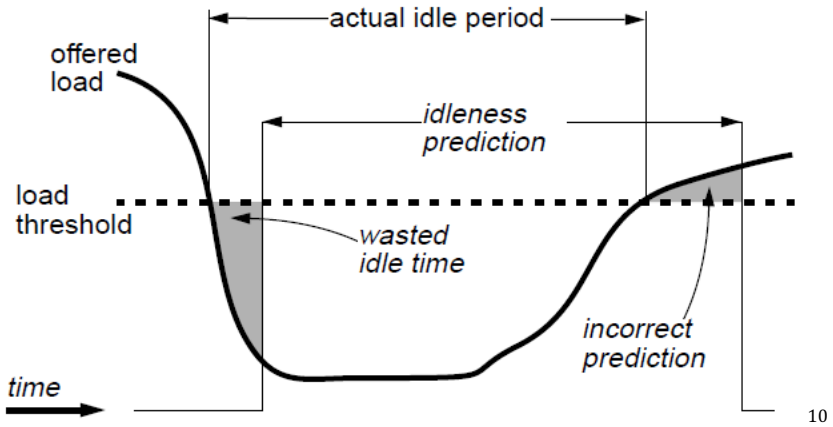
Estas tecnologías (*Dynamic Voltage Scaling* y *Dynamic Frequency Scaling*) permiten variar de forma dinámica la frecuencia y/o el voltaje de la CPU para obtener ahorros de energía dada la relación cuadrática de la frecuencia y el voltaje con el consumo final del procesador.

3.2.3.1 Variabilidad en las cargas de servidor

Otro punto que hay que reseñar de la carga de un servidor de cara a su modelado es la variabilidad de la misma. Normalmente, las curvas de carga que representan el uso de servicios son curvas periódicas (más correctamente, estacionales, pues el período no es constante y tienen tendencia, pero normalmente en la literatura de computación autonómica vienen referenciadas como periódicas y, de hecho, existen modelos con curvas periódicas, como senoidales) que permiten ser estimadas de forma sencilla mediante modelos ARMA (*Autoregressive Moving Average*), por ejemplo.

Esta variabilidad, reseñada en [Bohrer01] es lo que permite introducir técnicas de gestión autonómica que concentren la carga en un subconjunto de nodos más reducido en las partes de las curvas de carga más bajas.

En relación al tiempo de inactividad (*idle time*), hay que reseñar la diferencia existente entre las cargas de escritorio y de servidor, tal y como se remarcará en la sección 3.6.2. El tiempo de inactividad de un servidor es prácticamente nulo, incluso con bajos niveles de carga, por ello, y tal como se indica en [Golding96], es necesario establecer límites más laxos a la definición de inactividad, de forma que sea posible una concentración de carga efectiva. Adicionalmente, habrá que ser capaz de detectar estos tiempos de inactividad de forma efectiva para evitar desaprovechar potencia de computación si se hace tarde, o incurrir en penalizaciones energéticas si se hace de forma prematura, tal y como muestra el Gráfico 6.



[10]

Gráfico 6: Efecto de predicciones de tiempos de inactividad incorrectos

En vista de la variabilidad de la carga soportada por los centros de datos, han aparecido propuestas para optimizar el consumo de los computadores y del centro de datos en completo mediante un correcto balanceo de carga, de forma que se maximice la localidad de los datos. En [LeeH05] se propone un sistema de distribución de carga que minimiza la distancia del nodo asignado para servir la petición Web a los ficheros que la componen. De esta forma se optimizan los accesos a disco y se reducen las transmisiones por la red interna.

3.3 Calidad de servicio en sistemas distribuidos

En primer lugar, de cara al modelado del sistema para una correcta representación del mismo en el elemento que toma decisiones (*manager*), es necesario tener en cuenta la percepción de la calidad de servicio para los usuarios del servicio, que en muchas ocasiones, además de usuarios, se pueden considerar clientes de un servicio de pago o mecanismos *B2C*, por ejemplo.

El término calidad de servicio (*QoS, Quality of Service*) se empezó a utilizar en entornos de red de conmutación de paquetes con el objetivo de ofrecer un cierto nivel de garantías en tiempos de entrega, tiempos de ida y vuelta y fiabilidad de las redes. Formalmente, se definió en la recomendación E.800 del consorcio ITU-T [ITU94] como: "el esfuerzo colectivo para garantizar el rendimiento del servicio, el cual determina el grado de satisfacción del usuario de dicho servicio".

[10] Figura obtenida de [Golding96].

En este sentido, la definición anterior es directamente aplicable a cualquier tipo de servicio, y en particular, a los servicios de Internet, como comercio electrónico, multimedia y servicios transaccionales en general.

Hoy en día, todos los servicios anteriormente mencionados, dada la masividad de usuarios que tienen, se proporcionan desde grandes centros de datos mediante software distribuido de gran complejidad. Esto dificulta la aplicación de políticas de *QoS* sencillas y, sobre todo, hace que la integración de servicios de gestión de energía sea, además de también compleja, más que necesaria.

3.3.1 Métricas de calidad de servicio

Es interesante hacer hincapié en lo centrada que está en el usuario la definición de QoS. Este hecho hace que una correcta selección de métricas de *QoS* sea fundamental para modelar el servicio.

Mientras existen muchas métricas para describir un servicio, como la productividad medida en peticiones/s que se están ofreciendo, sólo unas pocas permiten reflejar el grado de satisfacción del usuario del servicio, tales como el tiempo de respuesta.

En general, modelar el servicio con un conjunto reducido de métricas es ideal, pues simplifica la operación del modelo en línea y, sobre todo, permite alcanzar un objetivo de computación autonómica como es la reducción de los parámetros de funcionamiento del sistema de forma directa.

No obstante, no se debe de olvidar que reducir mucho el conjunto de métricas utilizadas puede hacer que se sintetice demasiado el modelo, dejando de representar correctamente la realidad o no respondiendo adecuadamente en ciertas situaciones.

Respecto a las métricas de eficiencia, aunque existen muchas que permiten relacionar de forma directa el rendimiento con la energía, como instrucciones/Julio [Weiser94], no se debe olvidar que no representan la calidad del servicio, sino información interna de aprovechamiento en la ejecución que puede ser utilizada para discriminar entre un computador u otro a la hora de, por ejemplo, decidir sobre cuál de ellos encender o apagar.

3.4 Clústeres homogéneos y heterogéneos

Es más que común contar con un clúster heterogéneo como objetivo de la gestión, debido a las ampliaciones y actualizaciones que sufren a lo largo de los tiempos de operación que introducen diferentes generaciones de equipamiento en las granjas de servidores.

De hecho, una correcta optimización de este tipo de clústeres permite incluso ahorrar cierta cantidad de energía y alargar la duración de los equipos, tal y como se introduce en [Nathuji07].

Aunque la gestión es más sencilla en entornos homogéneos, no se puede olvidar que los clústeres heterogéneos suponen una gran mayoría de los sistemas a gestionar y el algoritmo de optimización presentado en la sección 4 debe ser capaz de tener en cuenta las particularidades de los mismos.

Fundamentalmente, los puntos a tener en cuenta se centran en tener en cuenta qué nodo encender o apagar cuando llegue el momento, para lo que habrá que conocer las prestaciones (capacidad) del mismo y su consumo energético.

[WangL09] es uno de los primeros trabajos en contemplar de forma explícita las particularidades de estos sistemas heterogéneos, para lo cual utiliza un complejo modelo de colas y sistemas de listas ordenadas que permiten optimizar a qué servidor enviar cada petición en función de sus prestaciones.

3.5 Provisión dinámica de nodos

Los algoritmos de provisión dinámica de nodos vienen siendo estudiados desde hace mucho tiempo. Concretamente, en el trabajo de [Hwang97] se propone por primera vez un sistema que permite apagar y encender nodos de computación. En este caso, al tratarse de una propuesta primitiva, sólo se contemplan dos estados posibles en los nodos, pero ya se tiene en cuenta el periodo de encendido y una predicción primigenia del *idle time* de los nodos.

Cabe destacar el año 2001 por el trabajo de [Chase01a], donde se propuso una agenda de investigación para este tipo de sistemas. Este tipo de algoritmos se centra en realizar una gestión dinámica de la configuración del clúster donde son aplicados de forma que mediante apagado y encendido de nodos se mantiene un subconjunto de nodos encendido capaz de absorber la

demanda de los usuarios del mismo manteniendo la calidad del servicio dentro de unos umbrales para las métricas utilizadas para la estimación del modelo.

En [Pinheiro01] y [Pinheiro03] se desarrolló el primer sistema que realizaba esta gestión de forma dinámica; con el objetivo de minimizar el número de nodos encendidos, se trataba de concentrar la carga en el conjunto más pequeño posible.

Tal y como se menciona en [Chung99a], la calidad del algoritmo que decide cuándo aprovisionar nodos o reducir el número de nodos en operación es fundamental, y para ello es necesario que este algoritmo aprenda de la carga de trabajo que tiene el servicio (y por tanto, del usuario, que es el punto central de la satisfacción en términos de calidad de servicio). En su trabajo, realizan un complejo modelado de aprendizaje basado en algoritmos de aprendizaje automático, pero no tienen en cuenta modelos de sistema completos.

Otro trabajo de interés es [Aikebaier09], donde se realiza un modelado del sistema muy detallado, partiendo de datos obtenidos en un experimento de carga y normalizándolos para poder soportar clústeres heterogéneos.

Existen trabajos que tienen en cuenta varios estados de operación para cada nodo, más allá de los básicos *encendido* y *apagado*, como el introducido en [Irani02]. Fundamentalmente, el tener en cuenta estos estados intermedios, como el de suspensión e hibernación (ver sección 2.2), permite tener unos tiempos mejores de respuesta al lanzar órdenes de encendido y apagado, pero trae consigo un aumento muy grande del número de transiciones a manejar (aunque pueden simplificarse) y, sobre todo la obligatoriedad de moverse a un modelo temporizado de decisión, en lugar de un modelo basado en eventos, como el propuesto (ver sección 4 para más detalles).

Además de realizar una provisión dinámica de nodos físicos, es posible realizar una provisión en dos niveles mediante el uso de máquinas virtuales, como en [Kumar09] o [Petrucci10]. De esta forma, se proveen nodos físicos que se van utilizando con máquinas virtuales (para las que se hace una gestión energética mediante estados de procesador ficticios).

Cuando las máquinas virtuales requieren más recursos, se migran (ver sección 3.5.1) a máquinas físicas más descargadas. En el caso de que una máquina física quede muy descargada, se migran sus instancias a otras máquinas físicas y se procede a su apagado.

La visión para el cliente es que su máquina virtual siempre está corriendo, pero de forma adaptativa obtendrá más recursos cuando los necesite y migrará de máquina de forma transparente si es necesario.

La principal contrapartida de esta gestión es la complejidad que entraña hacer una asignación de recursos en dos niveles y determinar cuándo una máquina virtual requiere más recursos. Además, la forma de asignación de recursos generalmente no permite modificar dinámicamente el número de *CPUs* mientras la máquina virtual está en ejecución.

Incluso, es posible realizar una gestión de provisión dinámica de dispositivos, tal y como se expone en [Lu02], de forma similar a la provisión de nodos, en función de la carga que el nodo está soportando a nivel de dispositivo y, en particular, de los procesos que están haciendo uso de los mismos, el sistema operativo es capaz de apagar y encender dispositivos bajo demanda. El principal problema de esta iniciativa es la necesidad de agregar soporte explícito a los sistemas operativos para realizar esta gestión de la forma propuesta, debido a que ya suelen tener mecanismos de gestión basados en tiempos de inactividad. Otro enfoque similar, consiste en retirar procesos *calientes* de la *CPU*, para reducir el consumo (perjudicando el rendimiento) [Rohou99]. En [Yao95] se describe un planificador de procesos para un sistema operativo que es capaz de asignar cuantos de ejecución en base a parámetros energéticos.

Otra iniciativa similar a la anterior es la descrita en [Mandagere07], donde se intenta optimizar el consumo de los sistemas de almacenamiento de forma similar a un sistema de provisión dinámica de nodos, pero con discos duros, en lo que denominan un sistema *MAID* (*Massive Array of Idle Disks*), donde se cuentan con *pools* de discos encendidos y apagados y un sistema de ficheros distribuido que permite realizar el balanceo. En [WangJ08] se presenta una iniciativa similar, optimizado cabinas de RAID existentes.

En [Zhu04] se pretende optimizar el consumo mediante un configuración de caché optimizada para el acceso de forma que la rotación de los discos se minimiza, invalidando entradas de caché de forma inteligente.

Por último, ambas gestiones, a nivel de sistema y de dispositivo, pueden combinarse en un solo algoritmo que realice una gestión jerárquica, como los descritos en la sección 3.8 (aunque a otro nivel) [Rong05].

3.5.1 Migración y priorización de carga

Un reto que plantea la provisión dinámica de nodos es que, al realizarse apagados y encendidos dinámicos de los mismos, el conjunto de nodos trabajadores es variable. En un clúster tradicional, que cuenta con un distribuidor de carga y n nodos que realizan el trabajo, tal y como se muestra en la Figura 5, el conjunto de nodos trabajadores no varía, de forma que las sesiones

asignadas a cada uno de ellos siempre permanencen en dicho nodo, desde el principio hasta su conclusión. Si el conjunto de nodos varía, la carga asignada a cada uno de ellos puede variar de igual manera.

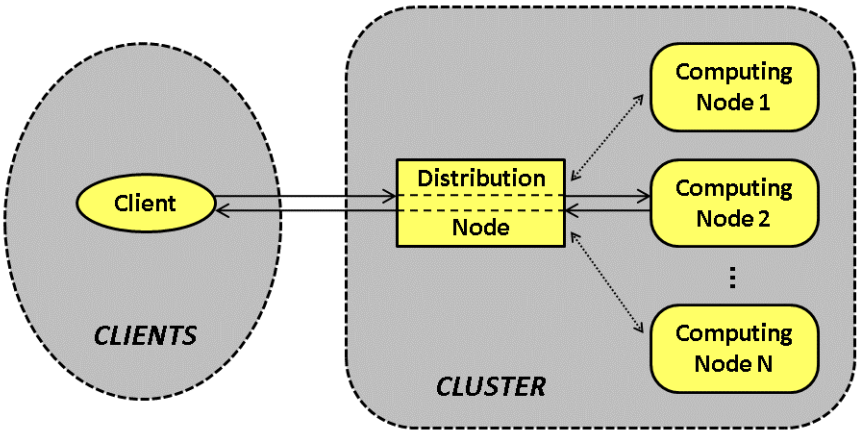


Figura 5: Diagrama de bloques de un clúster clásico

De esta forma, apagar uno de los nodos de trabajo cuando se tienen sesiones abiertas o trabajos en ejecución en el mismo hace que, o bien se espere a la terminación de dichos trabajos para apagar dicho nodo, lo cual necesita garantizar que esto va a producirse en un tiempo finito y cercano, o se migren esos trabajos a otro nodo, lo que no es técnicamente posible en todas las situaciones y hace que los estados de carga de cada nodo varíen drásticamente al recibir sesiones de otro nodo de golpe.

En general, cuando los trabajos que se asignan a los nodos de computación son de duración finita y, sobre todo, responden a una distribución de tiempos estimable, una buena solución es esperar a su terminación, descargando el nodo de trabajo antes de apagarlo. No obstante, si la duración de los mismos es muy larga, puede hacer que los cambios de estado del sistema tomen mucho tiempo.

Aunque se trata de un trabajo relacionado con técnicas de gestión de disco, en [Lu99] se introduce un mecanismo de predicción del tiempo de una sesión de forma sencilla mediante estimaciones estadísticas del tiempo de reflexión.

Con un mecanismo de balanceo adecuado, tal y como el que se introduce en [SharmaRK03] es posible reducir el consumo energético. En este caso, se asignan las tareas en función de la temperatura ambiente de la sala, de forma que se distribuye la generación de calor y se optimiza la refrigeración del centro de datos.

Adicionalmente, en ciertos sistemas, como sistemas de tiempo real o sistemas con priorización de clientes, realizar una migración de carga puede ser problemático, perjudicando la calidad de servicio a clientes prioritarios o no satisfaciendo requisitos temporales tal y como indica [Amirijoo07]. Contrariamente, en [Horvath07a] se aplica un mecanismo de asignación de políticas para clientes priorizados que permite optimizar el consumo energético en ciertas situaciones.

En general, tal y como se indica en [Tsai08], sea cuál sea la política escogida en este apartado, será necesario conocer los tiempos de encendido y apagado de los nodos para una correcta adaptación del algoritmo utilizado. En el caso del algoritmo presentado en la sección 4, es fundamental para el ajuste de los parámetros que rigen su funcionamiento.

3.6 Control de DVS y DFS

Las tecnologías de control de energía con las que cuentan los procesadores modernos permiten a los mismos autogestionarse para minimizar el consumo energético mediante cambios de voltaje y, por tanto, frecuencia con un alto grado de granularidad e independencia.

Estas tecnologías, *Dynamic Voltage Scaling* (*DVS*) y *Dynamic Frequency Scaling* (*DFS*), tal y como se explica en la sección 2.1.1.1, son clave en la gestión energética de los dispositivos más exigentes en términos de energía de un servidor moderno[11].

Es necesario destacar la posibilidad que ofrecen los sistemas operativos de última generación, mediante la implementación de las interfaces *ACPI* [ACPI09] (ver sección 2.1.2.2), permitiendo controlar de forma explícita ciertos comportamientos de las *CPU*, como establecer umbrales de funcionamiento en el caso de Windows Server 2008 (ver sección 2.3).

Adicionalmente, además del control de la frecuencia de forma explícita, se puede controlar también las nuevas tecnologías de *power gating*, apagando unidades funcionales y núcleos completos de forma centralizada para reducir más el consumo energético, tal y como se introduce en [Leverich09].

Por último, no sólo la gestión del *DVS* se realiza en máquinas físicas, sino que también es posible realizarla en máquinas virtuales, de forma que se produzca una relación (*mapping*) entre las

[11] Las *GPUs* tienen unos requisitos energéticos que son ya mayores que las *CPUs*, aunque, por el momento, no son comunes en instalaciones de servidor.

CPUs virtuales y físicas de forma eficiente [Pedram10]. Otra opción interesante mostrada en [WangX09] es inyectar estados P ficticios en los procesadores virtuales desde el *hipervisor*, de forma que los sistemas operativos virtuales tengan la conciencia de estar en un estado energético de menor perfil y se auto ajusten de forma consistente.

Otro trabajo de interés es el descrito en [Zhong07]. En éste se gestiona de forma avanzada la frecuencia de los procesadores en base a la búsqueda de invariantes en la asignación de intervalos de frecuencias.

3.6.1 Control explícito y control independiente

En el caso de la aplicación de estas tecnologías a la gestión autonómica de la energía, se tienen fundamentalmente dos opciones a día de hoy.

En primer lugar, después de los primeros trabajos de gestión autonómica, como los ya mencionados de [Hwang97], [Chase01a] y [Pinheiro01] que no incluían gestión de frecuencia de forma explícita, han aparecido trabajos que manejan y administran la frecuencia de los procesadores de las máquinas del clúster de forma explícita, tales como los de [Elnozahy03a], [Elnozahy03b], el de [Bertini10a] para clústeres heterogéneos y [Bertini10b] para homogéneos.

Existen trabajos completos, como el de [ChenY05] que además de contemplar el uso explícito de tecnologías de control de energía como *DVS*, tienen en cuenta tiempos de arranque y permiten ser configurados en clústeres homogéneos y heterogéneos.

El trabajo de [Heath05] también incluye gestión explícita del *DVS*, así como el de [Kusic09]. El trabajo de [Ishihara98] define un algoritmo de planificación de voltaje basado en problemas de programación lineal con restricciones enteras.

Los trabajos de [Rusu06] y [SharmaV03] incluyen gestión explícita del DVS y mantenimiento del SLA, pero con la desventaja de utilizar en ambos un modelado muy complejo del sistema, que incluye la definición de múltiples colas de servicio y la dependencia en exceso del servicio proporcionado, en este caso, un servidor Web basado en *Apache*.

La otra opción es, en lugar de realizar un control centralizado del *DVS*, realizar un control independiente en cada uno de los nodos. Pese a que se cuenta con menor cantidad de información (sólo local), la gestión es realizada de forma automática por el hardware y el sistema operativo en cada uno de los nodos, minimizando el consumo local de cada nodo de

forma independiente. Esto permite simplificar los modelos del sistema global y la toma de decisiones, teniendo en cuenta que, a día de hoy, el *DVS* no permite ahorrar tanta energía como el apagado de un nodo.

Uno de los primeros trabajos que contemplan el control integrado y distribuido de la frecuencia de los equipos de una granja de servidores es [Horvath07b]. En este trabajo, mediante un algoritmo de optimización basada en restricciones lineales, se consigue hacer una gestión que satisface restricciones de tiempo de respuesta completo en un servicio de tres capas. Esta gestión sólo incluye la aplicación de técnicas de control de frecuencia y no de provisión dinámica de nodos.

De cara a la comparación de la integración de las dos opciones en un sistema de gestión de energía, el trabajo de [Heo07] proporciona una comparativa de tres combinaciones de gestión, resultando que la provisión dinámica es la responsable de la mayor parte de los ahorros energéticos, pero que en determinadas cargas, combinarla con la gestión del DVS puede ser contraproducente.

En cualquier caso, la efectividad de la gestión del DVS es dependiente del peso que la computación en la CPU tenga en la carga que esté soportando el servidor (ver sección 5.2).

3.6.2 Impacto del DVS en el ámbito de servidores

Las tecnologías de control de frecuencia son importantes en cualquier entorno, pero ha de entenderse también que en el ámbito de servidores pueden tener un impacto menor al esperado por la diferencia del comportamiento del sistema.

Fundamentalmente, el mayor cambio es el tiempo de inactividad (*idle time*) que se tiene en los servidores, que están siempre atendiendo clientes, a diferencia de los computadores de usuario, donde los tiempos de inactividad son altos.

Para el modelado de los tiempos de inactividad en entornos de servidor se dispone del tiempo de reflexión (tiempo entre peticiones de un usuario emulado), sin embargo, pese a que cuando se emulan usuarios humanos en entornos *B2C* el tiempo de reflexión es más alto (ver sección 9.3.1), en cuanto se cuenta con un número suficiente de usuarios, los tiempos de inactividad se anulan.

De hecho, trabajos como el de [Freeh07] sugieren que con la situación actual de implementación de tecnologías del tipo de DVS y DFS, la provisión dinámica es más efectiva que la gestión dinámica y centralizada de la frecuencia de los procesadores. En este sentido, es más eficiente ejecutar programas distribuidos en un número mayor de máquinas configuradas directamente con la frecuencia mínima que proveen y fijarla en ese punto.

3.6.2.1 Impacto en servicios de tiempo real

Adicionalmente, es remarcable el impacto que la aplicación de estas tecnologías puede llegar a tener en sistemas de tiempo real, donde la predictibilidad del sistema es fundamental de cada a aplicar técnicas de asignación de trabajos que garanticen la compleción en tiempo de los mismos.

En el trabajo de [Hong98] se analizan métodos y algoritmos de *scheduling* de trabajos que tienen en cuenta la variabilidad de la frecuencia de los procesadores y, por tanto, de la variabilidad de la duración de los mismos en entornos controlados. Asimismo, el trabajo realizado por [LeeS00] es uno de los primeros en incorporar *feedback* basado en métricas de aplicaciones para modificar la frecuencia de los procesadores, manteniendo restricciones de tiempos límite.

En este tipo de sistemas, también es necesario tener en cuenta las penalizaciones variables por fallos de caché en distintos núcleos, si se cuenta con un procesador SMP basado en replicación de núcleos. En [Hong99] se refina el algoritmo de asignación de trabajos de forma que también es capaz de seleccionar a qué núcleo enviarlo y optimizar el acceso a caché. De esta manera, se puede reducir el consumo medio del procesador, reduciendo el tiempo de acceso a memoria e incrementando la proporción de tiempo de ejecución efectivo.

Para modelar el comportamiento de estos nuevos microprocesadores, con múltiples núcleos, frecuencia variable, cachés de varios niveles se pueden usar modelos detallados, como el propuesto en [Huang09], que modela no sólo el procesador, sino también la carga y los procesos ajenos a la CPU que influyen en su funcionamiento variable.

3.7 Punto único de fallo y algoritmos distribuidos

Tal y como muestra la Figura 5, generalmente un clúster de distribución de carga clásico tiene un nodo que centraliza la distribución de carga, repartiendo los trabajos entre los nodos trabajadores.

Este punto de distribución suele estar centralizado y representa un punto de fallo único, disminuyendo la fiabilidad y disponibilidad del sistema.

Una opción para incrementar la fiabilidad es replicar el nodo distribuidor, construyendo, por ejemplo, un clúster de disponibilidad para el mismo que sea capaz de alternar de nodo distribuidor en caso de fallo.

Otra opción importante es diseñar el algoritmo de control de forma que sea distribuido y tolerante a fallos de nodos de forma natural. Un ejemplo de este modelo de diseño es el algoritmo introducido en [Kandasamy06] que mediante un diseño basado en descomposición de trabajos (cada trabajo es una solicitud de optimización) y el uso de varios controladores de procesamiento (los nodos encargados de ejecutar el algoritmo) idempotentes permite tener una gestión totalmente distribuida.

3.8 Algoritmos híbridos y jerárquicos

Los algoritmos híbridos y jerárquicos son algoritmos que combinan los dos enfoques anteriores: la provisión dinámica de nodos y el control explícito de DVS.

Esta gestión se realiza en dos niveles en el caso de los algoritmos jerárquicos (o distribuidos): provisión dinámica englobando todo el clúster y control de frecuencia en cada uno de los nodos de forma explícita.

Los algoritmos híbridos (también conocidos como coordinados) realizan una gestión centralizada de ambos enfoques, de forma que deben de decidir entre, por ejemplo, encender un nuevo nodo en condiciones de incremento de carga o incrementar la frecuencia de los nodos que ya están funcionando [Raghavendra07] [Ren04].

En este caso, se debe tener un modelo capaz de representar ambas situaciones, siendo potencialmente más complejo y con más parámetros a estimar o configurar.

Todos los algoritmos mencionados en la sección 3.6 como ejemplo de algoritmos de gestión de *DVS,* son, además, algoritmos híbridos, siendo raro encontrar un algoritmo que sólo maneje el control de frecuencia y no la provisión dinámica de nodos.

Especialmente destacable es el trabajo de [Das08]. En este trabajo se define la arquitectura de un modelo jerárquico que incluye provisión dinámica y control de *DVS* de forma integrada con la posibilidad de introducir los diferentes agentes de decisión en múltiples nodos distribuidos, lo que termina con el problema del punto único de fallo y convierte al sistema en un modelo muy escalable. Como contrapartida, los modelos utilizados para representar el sistema son complejos y el sistema es difícil de integrar en productos en producción.

Otro algoritmo jerárquico a destacar es el definido en [Khargharia07], que permite la gestión dinámica de nodos (provisión dinámica) para realizar la optimización de eficiencia. Por el contrario, no mantiene restricciones de rendimiento, sino de escalabilidad.

3.9 Modelado de sistemas

El modelado del sistema es un punto fundamental para la correcta operación y optimización del gestor autonómico. Obtener un modelo completo, correcto y sencillo, de forma que tenga pocos parámetros que estimar es una tarea compleja que ha evolucionado a lo largo del tiempo [Rivoire07].

3.9.1 Modelado a priori

El modelado a priori hace referencia a obtener un modelo del sistema antes de ponerlo en operación, de esta forma el gestor autonómico tiene un modelo inicial para comenzar la operación.

Obtener un modelo del sistema de forma previa requiere realizar, por ejemplo, experimentos de carga para ver cómo responde el sistema a la carga y predecir en tiempo de funcionamiento el estado del sistema.

Generalmente, al realizar un modelo a priori, éste se usa de forma estática a lo largo del período de funcionamiento del sistema, aunque puede ser dinámicamente actualizado en tiempo de ejecución.

Un trabajo interesante al respecto del modelado previo es el de [Pettis09], donde se realiza un gestor que permite optimizar el consumo de energía en base a la aplicación de políticas, pero sin tener conocimiento previo de las cargas de trabajo o modelos del sistema.

Otro trabajo de interés al respecto es el de [Ramanathan00], que permite modelar sistemas en base a históricos de utilización del mismo.

3.9.2 Modelado en línea

El modelado en línea generalmente supone una evolución del modelo del sistema obtenido en una fase de modelado a priori, haciendo que mientras el sistema parte del modelo estático calculado previamente, a lo largo del tiempo de funcionamiento del mismo, el modelo va siendo evolucionado y adaptado a nuevas situaciones, como cambios en los tiempos de respuesta de servicios externos o variaciones internas en el servicio que hacen que la respuesta no tenga un tiempo constante como se predijo en el modelo a priori.

También es posible contar con un sistema capaz de aprender un modelo de forma dinámica en línea sin necesidad de un modelo a priori inicial, pero eso requiere un tiempo de adaptación del sistema al comenzar su ejecución, de forma que el modelo en línea cuenta con una estabilidad suficiente para una operación adecuada.

Dos modelos sinteresante de captura en línea son los expuestos en [Tsai07] y [Valledor09]. El primero permite reestimar la capacidad del servidor mediante el uso de métricas de red y el segundo permite la reestimación mediante curvas adaptadas en estados estacionarios que mantienen garantías de monotonía.

3.9.3 Modelos óptimos y estocásticos

Adicionalmente, se pueden encontrar y desarrollar modelos óptimos, esto es, que modelan el sistema de forma certera y tienen en cuenta todos y cada uno de los estados de forma que para cambiar de situación se optimiza una función de coste que indica cuál es el estado adecuado en cada momento, y modelos estocásticos, que modelan el sistema en base a métricas sencillas con una cierta distribución probabilista y sistemas de colas [Qiu99] y modelos de Markov [Qiu00].

Es importante realizar una separación entre el modelado del clúster en sí mismo como sistema, ya sea con modelos de colas o modelos más detallados, y el modelado del problema a resolver, que marcará el método para resolverlo.

Realizar un modelo óptimo del clúster es ciertamente complejo, dada la gran cantidad de parámetros que intervienen en el sistema y, asimismo, realizar un modelo óptimo del problema de optimización energética es igual de complejo, debido al alto número de estados por los que pasa el sistema, por lo que la mayoría de sistemas se quedan con aproximaciones estocásticas y modelos simplificados que permiten hacer una gran estimación de los sistemas de forma sencilla y rápida.

Casi todos los métodos y algoritmos analizados, ya sean óptimos o estocásticos, hacen uso de modelos de estados para controlar y modelar el problema de optimización energética, tales como el de [Andrew09], que predice la minimización de una función resultante de la combinación de la energía y el tiempo de respuesta (aunque no garantiza el SLA).

En el trabajo de [Chung99b] se introduce un método de gestión de estados de múltiples peticiones, que es radicalmente diferente a la gestión de estados de las diferentes máquinas del clúster. Esto permite tener más granularidad en la gestión, pero complica el modelado, siendo necesario recurrir a cadenas de Markov o redes de Petri.

Otros modelos, como el de [Chase01b] presentan estrategias de optimización basadas en técnicas de optimización de costes. En este sentido, se establece una función de coste basada en la energía que consume cada cliente y dispositivo y se asignan recursos mediante un algoritmo devorador. Además de perder la optimalidad (en pos de mejorar los tiempos de decisión), se pueden dar casos en los que el consumo energético degrade el servicio y no se garantice el cumplimiento del SLA.

En cuanto a los métodos de modelado estocástico y estadístico, existen trabajos como el de [Chung02], que realizan modelado por cadenas de Markov del sistema a procesar y generan modelos dinámicos que dependen del estado del sistema. En [Gandhi10] se propone un modelo similar basado en modelos de redes de colas y la introducción de la métrica de energía-respuesta, que garantiza su optimalidad desde el punto de la minimización del consumo, pero no así desde el cumplimiento del SLA del servicio.

Otros tipos de modelos estocásticos son, por ejemplo, el definido en [Jung08]. En este modelo, basado en decisores de cadenas de Markov, se modela el sistema de forma que lo que se trata de minimizar es el coste de la energía que requiere un determinado nivel de rendimiento. Para la representación, se realiza a un nivel de detalle muy alto, llegando a niveles microarquitectónicos,

de manera que se permiten realizar modificaciones de voltaje y frecuencia (ver secciones 2.1.1.1 y 3.6) y de otro tipo, como de tamaño de caché o grado de asociatividad de la misma. Otro modelo de este tipo es el expuesto en [Simunic02b].

En el trabajo de [Mastroleon05] se utilizan modelos basados en heurísticos y programación dinámica para tratar de optimizar el consumo, de forma que se tienen conjuntos de nodos encendidos y apagados y se trata de optimizar el tamaño de cada uno de ellos para minimizar una función de coste basada en la relación entre la energía y el rendimiento.

En [Pakbaznia07] se utilizan modelos basados en programación lineal, con el objetivo esta vez de realizar una consolidación de la carga en el menor número de servidores (aunque todos los algoritmos alcanzan este corolario, éste lo hace de forma directa, pues es el objetivo a optimizar).

Adicionalmente a los modelos de colas tradicionales ya analizados, trabajos como [Simunic01] utilizan sistemas diferentes de modelado, como un modelado dirigido por eventos discretos (no confundir con la simulación orientada a eventos discretos, que es una técnica de validación). Estos modelos son sencillos y computacionalmente eficientes de ejecutar. La principal contrapartida es que sólo pueden tomar decisiones cuando existen eventos en el sistema, por lo que la selección de qué se considera un evento es crucial para mantener un equilibrio entre la sobrecarga introducida por el algoritmo y la capacidad de decisión de los algoritmos.

3.9.4 Modelado energético empírico y teórico

De cara al modelado del consumo energético del clúster, tradicionalmente se han utilizado modelos teóricos donde la relación entre el consumo y la carga se hace por medio de la frecuencia, tal y como se explica en la sección 2.1.1.1.

Esto ya no es válido dada la complejidad de los procesadores de hoy en día (multinúcleo, con cachés de tercer nivel, capaces de desactivar unidades funcionales por completo, etc.) y el incremento de peso de los demás componentes del equipo, como las cantidades ingentes de memoria principal o los discos de altas revoluciones.

Por este motivo, es necesario modelar el comportamiento energético de forma empírica, para tener la relación real entre la demanda de servicio y el consumo energético, teniendo en cuenta todos los componentes y factores del sistema.

En trabajos como [WangX08] se realiza un modelado en línea del consumo basado en métricas directas del servicio, que se obtienen mediante la instalación de aparatos de medición y sistemas de adquisición de datos. Fundamentalmente, la principal desventaja de estos métodos es la complejidad de la captura de los datos en línea, un problema en sí mismo.

3.10 Conclusiones al análisis

Fundamentalmente, tras realizar el análisis de los trabajos realizados por la comunidad científica en este ámbito, nos encontramos con problemas variados, tales como la complejidad de los algoritmos y métodos de optimización y modelado, la no utilización de métricas de calidad de servicio o la necesidad de modelar el sistema previamente a la utilización del algoritmo.

3.10.1 Modelos no realistas

En primer lugar, es necesario remarcar que existen trabajos, tales como [ChenY05], que utilizan modelos que no se corresponden con la realidad. En este caso, el modelo es un modelo teórico de consumo de energía de un computador, basado en la curva teórica de consumo en relación a la frecuencia y voltaje del procesador explicada en la sección 2.1.1.1.

Se ha descubierto mediante métricas empíricas que utilizar directamente esta aproximación no refleja de forma correcta la realidad, dado que los procesadores actuales cuentan con muchas técnicas de ahorro de energía, como el *power gating*, que hacen que su consumo ya no dependa exclusivamente de su frecuencia.

Además, otros dispositivos, como discos y bancos de memoria, tienen un alto grado de impacto en el consumo total del computador.

3.10.2 Complejidad de los algoritmos y modelos

Otro problema grave de cara a la implantación de un algoritmo de gestión energética en sistemas existentes es la complejidad de los propios algoritmos de gestión y, sobre todo, de los modelos que se utilizan en su funcionamiento.

Existen trabajos, como [Bertini10a] y [Bertini10b], que requieren de potenciómetros e instalaciones de sistemas hardware adicionales en los servidores, además de modelar el sistema previamente mediante complejos sistemas de estimación.

A medida que se utilizan modelos más complejos, estos cuentan con un mayor número de parámetros y dependencias entre los mismos, lo que dificulta su aplicación y utilización en sistemas ya existentes y, sobre todo, no permite cumplir con el objetivo de simplicidad de la computación autonómica.

3.10.3 Métricas de calidad de servicio no adecuadas o derivadas

Otro problema importante detectado en trabajos como los de [Bertini07], [Bertini10a], [Bertini10b], [Elnozahy03a], y [Elnozahy03b] es la utilización de métricas que no reflejan la satisfacción del usuario del servicio, esto es, métricas que resuman la calidad del servicio.

La satisfacción del usuario, tal y como se repasa en [Lu01], es fundamental de cara a valorar el cumplimiento de objetivos del servicio, debido a que es el usuario quien valora y, en la mayoría de los casos, sufraga, la existencia de este tipo de servicios.

En este sentido, se considera fundamental el uso de métricas que reflejen el servicio que recibe el usuario, dado que son las únicas que permiten operar el sistema en base a datos que interesan al usuario. Otras métricas simplemente explican el comportamiento del servicio de cara a su monitorización y son interesantes sólo para los administradores.

Sólo el trabajo de [ChenY05] tiene en cuenta de forma explícita el *SLA* del servicio. El trabajo de [Xue07], de provisión dinámica de nodos incluye ciertas restricciones sobre la calidad del servicio, pero ninguna garantía.

Adicionalmente, estos trabajos no tienen en cuenta las métricas de eficiencia energética de los nodos, simplemente utilizando aproximaciones para la toma de decisiones, tales como un incremento de frecuencia o el encendido de un nuevo nodo.

3.11 Introducción al algoritmo de optimización en línea

El algoritmo de optimización, introducido en el resultado publicado como [Garcia10], que se presenta en las siguientes secciones intenta resolver de forma sencilla los problemas detectados en los trabajos más modernos relativos a gestión autonómica de energía.

En este algoritmo se tiene en cuenta de forma explícita el *SLA* del servicio mediante la utilización de la única métrica que lo describe de forma completa, el tiempo de respuesta. Además, se modela el sistema como una caja negra de forma que se oculta la complejidad del servicio y se permite definir un único algoritmo para cualquier tipo de servicio.

De cara a la toma de decisiones, se tiene en cuenta el consumo empírico de las máquinas, de forma que se cuenta con información real en todo momento.

4 ESTRATEGIA DE CONTROL

El módulo gestor de energía del clúster necesita de una estrategia de control que maneje de forma autonómica la provisión dinámica de nodos para optimizar el consumo energético.

Típicamente, esta gestión se realiza basándose en la utilización de los nodos que forman el clúster que se está administrando [Pinheiro01]. Esta aproximación no garantiza el aseguramiento de la calidad de servicio (QoS) proporcionada a los clientes ya que sólo tiene en cuenta información sobre el estado de los nodos y no sobre el estado del servicio proporcionado.

La calidad de servicio requerida se suele expresar en un acuerdo de nivel de servicios (SLA, *Software Level Agreement*) entre el proveedor del servicio y el cliente que limita el tiempo de respuesta máximo del clúster en las transacciones a ejecutar (además de, posiblemente, otros parámetros que definan el servicio).

En la mayoría de los casos suele utilizarse una métrica como el 90-percentil del tiempo de respuesta para describir el SLA, de forma que se resume de manera concisa el estado del servicio.

En la propuesta presentada en este proyecto, el objetivo primario es satisfacer el acuerdo de nivel de servicio en base a la métrica seleccionada a tal efecto (como el 90-percentil del tiempo de respuesta). Como objetivo secundario se tiene el ahorro de energía siempre que sea posible en base al estado actual del servicio en relación al SLA acordado.

4.1 Modelo de clúster objetivo de la gestión

El algoritmo de optimización en línea desarrollado permite optimizar la operación de clústeres de escalabilidad o balanceo de carga, esto es, clústeres que cuentan con múltiples nodos de procesamiento con el objetivo de repartir la carga que llega al sistema entre ellos y, de este modo, permitir ofrecer servicio a más usuarios con unos tiempos de respuesta que se

encuentren debajo de unos umbrales concretos, definidos normalmente por el administrador y el cliente en un acuerdo de nivel de servicio (*SLA*).

Los nodos del clúster de balanceo de carga pueden ser idénticos (clúster homogéneo) o de diferentes características (clúster heterogéneo). En este tipo de clústeres siempre suele haber un elemento de reparto de carga, normalmente centralizado (aunque puede ser otro clúster en sí mismo, generalmente dedicado a fiabilidad). Un esquema de la estructura puede observarse en la Figura 5, denotando cómo el elemento de reparto es el punto único de fallo en una estructura sin replicación. Es en este nodo, de hecho, donde la lógica de monitorización, control y actuación se centrará.

Todos los nodos de trabajo del clúster exponen el mismo servicio al nodo distribuidor y, éste hace lo mismo al exterior, ocultando las características internas del clúster a los clientes y permitiendo repartir el trabajo dentro del clúster sin intervención del cliente. Cuando un cliente abre una sesión en un servicio transaccional, lo hace contra el nodo de distribución y, éste encamina la sesión a uno de los nodos trabajadores en función del estado del clúster.

De esta forma, el nodo distribuidor conoce en todo momento el estado instantáneo del clúster y es capaz de medir la carga soportada por cada uno de los nodos y por el servicio en total. Con esta información, el nodo distribuidor ejecuta un módulo de software que, conociendo las características del clúster, del servicio y las políticas definidas en el *SLA*, permite decidir si aceptar o no una sesión y a qué nodo asignarla en caso de aceptarla. También puede priorizar sesiones de determinados clientes, por ejemplo llevándolas a nodos de menor utilización o a nodos dedicados.

Tal y como se observó en la sección 3.5.1, en el caso de hacer provisión dinámica, al apagar un nodo hay dos opciones: una es migrar las sesiones en ejecución a otro nodo y apagar el nodo o, si no se contempla o el servicio no soporta migración, esperar a que las sesiones asignadas al nodo a apagar se terminen, lo que se conoce como descarga de nodo. El algoritmo que se ha diseñado utiliza este segundo concepto, compatible con todos los servicios transaccionales y que maximiza la coherencia del servicio, aunque introduce un tiempo transitorio en el que un nodo se está descargando de sesiones, que posiblemente sea desconocido si no se conoce la duración de una sesión.

Para el enrutamiento de sesiones, en el prototipo desarrollado para la validación del algoritmo, descrito en la sección 9, se utilizan directamente los estados de conexión *TCP*, de forma que el nodo distribuidor es un túnel de conexiones *TCP*, lo que tiene un peso muy ligero para el nodo distribuidor. De esta forma, cuando un cliente abre una conexión contra el clúster, el distribuidor

la recibe, aplica el control de admisión y comienza a redirigir las peticiones a los nodos trabajadores. La sesión se asigna a un nodo trabajador y todas las peticiones serán resueltas por el mismo, de forma que las repuestas serán enviadas al distribuidor y ésta las reenvía al cliente. Esto permite manejar sesiones con estados complejos sin necesidad de tener un segundo nodo central de coherencia.

4.2 Objetivos de la optimización

Los objetivos que el algoritmo de optimización energética ha de satisfacer son los siguientes, de forma priorizada:

1. El tiempo de respuesta de cada petición debe estar por debajo de un umbral establecido en el *SLA* del servicio. Normalmente, se usará la métrica de percentil-90 del tiempo de respuesta.
2. El consumo energético del conjunto del clúster ha de ser el mínimo posible, pero suficiente para garantizar el cumplimiento del *SLA*.
3. La disponibilidad del servicio debe ser maximizada.

Además, hay que tener en cuenta que la carga durante la operación del servicio puede variar, las capacidades de los nodos también e, incluso, el *SLA* de los nodos podría modificarse durante la operación, además de establecer diferentes *SLA* para diferentes nodos internos[12] y permitir priorización se sesiones. Es por esto que la optimización ha de ser realizada en línea, mientras el clúster está en operación y no mediante modelado y optimización previos a la operación.

4.3 El algoritmo de optimización

El algoritmo diseñado y prototipado se ejecuta en el nodo distribuidor de forma centralizada y tiene como objetivo la toma de decisiones en dos vertientes: asistir al control de admisión

[12] Es importante destacar que los clientes ven el servicio como un único computador, de forma que sólo existe un *SLA* para ellos. La ventaja que puede obtenerse de establecer diferentes *SLAs* en el servicio es mantener nodos que estén saturados para ese nivel de carga fuera de saturación, de forma que los tiempos de respuesta son más estables (aunque la eficiencia energética menor). Adicionalmente, puede ser beneficioso en servicios con priorización de clientes.

informando sobre la capacidad del clúster en cada momento y tomar decisiones de control energético, esto es, apagar o encender nodos para satisfacer el *SLA* del servicio.

Para tomar las decisiones, el algoritmo necesita tener conocimiento sobre el comportamiento del tiempo de respuesta de cada nodo, denotado por R_i, en función de la carga actual, denotada por N_i, y del consumo energético de cada nodo, también en función de la carga soportada.

El comportamiento del tiempo de respuesta se mide previamente a la operación, mediante un experimento de carga sencillo con el servicio real que permita relacionar de forma directa el tiempo de respuesta esperado con la carga en número de sesiones activas. Esto permite obtener un modelo sencillo y global del sistema y con alta representatividad, pues proviene de datos empíricos. Este modelo medido a priori se considera el modelo inicial que se le proporciona al sistema para operar, y en la operación del algoritmo podría ser adaptado y actualizado dinámicamente si las características de la carga o de los nodos varían en tiempo de ejecución.

De esta forma, el tiempo de respuesta es medido para cargas de usuarios crecientes y ajustado mediante un modelo de estimación apropiado. La Figura 6 muestra tres ejemplos de modelos obtenidos para tres nodos de trabajo de características diferentes.

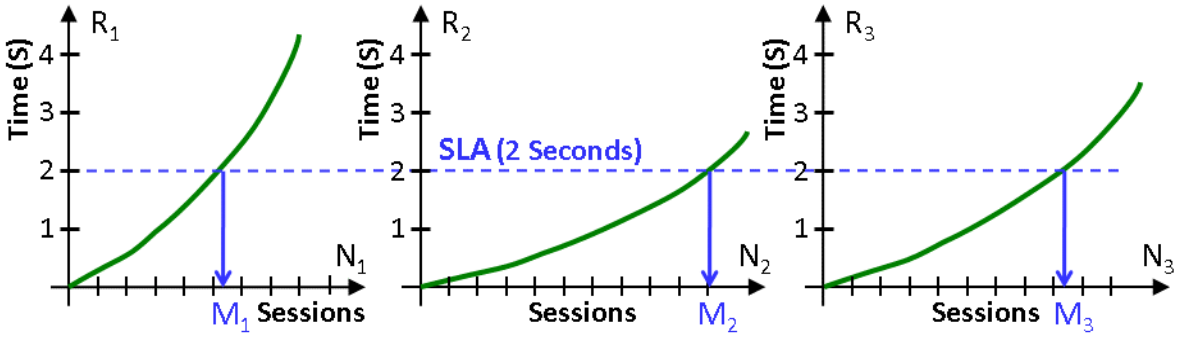


Figura 6: Ejemplos de estimación del tiempo de respuesta

Como se puede observar, en base al *SLA* (en la figura, 2 segundos como ejemplo) es posible calcular la capacidad que tiene un nodo sin llegar a violar el tiempo de respuesta límite (denotada por M_i). En el ejemplo, el nodo de mayor capacidad es el segundo. Para el cálculo de esta capacidad, teniendo la curva polinómica de ajuste, basta con ejecutar un procedimiento de cálculo de raíces analítico (será clave su optimización dada la importancia que tiene en la toma de decisiones la estimación de varias curvas).

Para el consumo energético de los nodos también se va a usar un modelo de estimación dinámico. Los computadores de hoy en día no consumen siempre la misma cantidad de energía y habrá que establecer una relación entre la carga que soporta un nodo y el consumo energético.

De los algoritmos estudiados en la sección 3, aquellos que tienen en cuenta de forma explícita el consumo energético, lo hacen mediante estimaciones de carácter teórico que, tras examinar y experimentar de forma empírica no se muestran como válidos para la optimización, porque no representan el consumo total del computador ni de los procesadores modernos con tecnologías de gestión avanzada. En Figura 7 se muestran resultados para los tres ejemplos típicos de modelos de estimación de consumo.

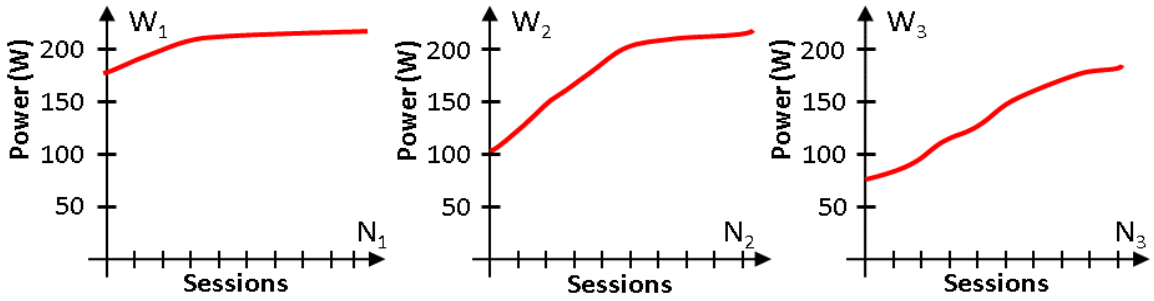


Figura 7: Ejemplos de estimación del consumo

Como se puede observar, en primer lugar, la forma de las curvas no tiene mucho que ver con los modelos cuadráticos teóricos que relacionan la frecuencia con el consumo mediante una relación casi directa. Estas curvas muestran el consumo básico (*idle power*) y el consumo de pico (*peak power*) y lo relacionan con una métrica directa y fácilmente obtenible por el módulo distribuidor, como es el número de sesiones concurrentes.

El primer nodo corresponde con un nodo poco eficiente (casi siempre se encuentra rozando el consumo de pico), mientras que los otros dos nodos son más eficientes.

El segundo nodo es el nodo paradigmático de un nodo con un muy buen control del *DVS* en el procesador, que permite escalar de forma lineal hasta que éste se satura o ya no tiene tiempo de reflexión entre las peticiones a atender y todas sus unidades funcionales están ocupadas, dando lugar a que el *DVS* no consigue obtener ahorros de ahí en adelante.

Finalmente, el tercer nodo es un claro ejemplo de un nodo muy eficiente, dónde además de una buena gestión de *DVS*, hay necesariamente una buena gestión de todos los dispositivos, lo que permite crecer de forma lineal en consumo.

El consumo de pico, en general tiende a estabilizarse en un punto, de forma que incrementos de usuarios no suponen un incremento del consumo, lo que incrementa la eficiencia (ver sección 5.2 para más detalles sobre eficiencia energética).

De cara a analizar la eficiencia de cada nodo, además de buscar la curva con valores mínimos, es conveniente buscar dos aspectos: la mayor diferencia entre el consumo básico y de pico, lo que permitirá gestionar de forma más profunda el computador y la mayor proporcionalidad, esto es, el incremento lo más lineal posible del consumo (o infralineal, si es posible, aunque todos los modelos tienen a ser supralineales). Esta proporcionalidad es la clave que permite optimizar el consumo, tal y como se detalla en la sección 5.2.

A partir de estas curvas, medidas empíricamente, se pueden deducir otras curvas de eficiencia de forma fácil, y utilizar métricas de eficiencia, como se analizará posteriormente.

4.3.1 Estados de operación de los nodos

Con el objetivo de realizar la optimización del funcionamiento de los nodos, cada uno de ellos podrá pasar por un estado de funcionamiento diferente que determinará las operaciones que está permitido hacer con él y si acepta carga o no.

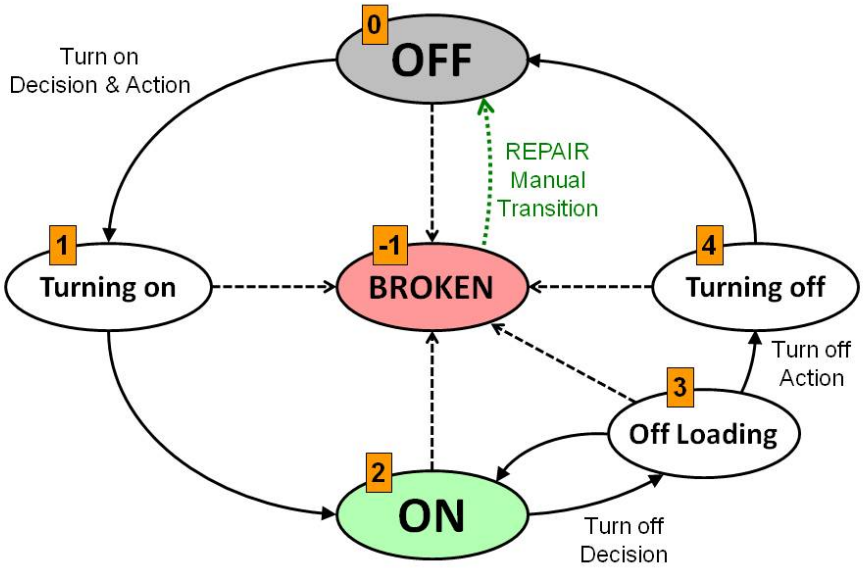


Figura 8: Diagrama de estados de operación de un nodo

La Figura 8 presenta el diagrama de estados de operación para un nodo del clúster (muestra adicionalmente el código numérico de cada uno de los estados) así como las transiciones permitidas.

Normalmente, los nodos estarán durante períodos largos en los estados estacionarios, marcados en color en el diagrama; estos estados, *ON* y *OFF*, además de *BROKEN*, son deseables que sean

estacionarios pues una minimización del número de transiciones minimiza la histéresis del algoritmo y conserva el hardware.

En la transición entre cada uno de los estados estacionarios, el nodo pasará por los diferentes estados transitorios y transiciones.

Cuando el gestor energético decide encender un nodo que está apagado (*OFF*), inmediatamente le envía un paquete mágico *WoL* y ejecuta la transición al estado transitorio *TURNING_ON*. Después de un periodo transitorio durante la recuperación o arranque del nodo, este notificará su disponibilidad al nodo distribuidor y pasará a estar operativo (*ON*) en estado estacionario. En este punto el nodo puede empezar a aceptar sesiones.

Si transcurrido un tiempo desde el envío del mensaje de encendido, el nodo no notifica su disponibilidad, el nodo será marcado como estropeado (*BROKEN*). En este estado el nodo está fuera de servicio y se deberá notificar al administrador del clúster el problema para que repare el nodo y éste vuelva al conjunto de nodos operativos.

Cuando el algoritmo decide apagar uno de los nodos, puesto que no se contempla la migración de carga, éste se pasa a un estado transitorio (*OFF_LOADING*) en el que no acepta sesiones y está planificado su apagado cuando se termine de ejecutar las transacciones que tiene asignadas. Cuando esto ocurre, pasa a un segundo estado transitorio (*TURNING_OFF*) tras enviarle una señal de apagado. Pasado un tiempo predefinido para el nodo, se marca como apagado (*OFF*).

Adicionalmente, cuando un nodo está en descarga y se decide encender un nodo, éste se puede marcar como encendido de nuevo, de forma que el rescate es inmediato y la respuesta del clúster en incrementos de carga mejora. Además, se conserva el hardware reduciendo los apagados y encendidos.

Dado que siempre se escoge para apagar el nodo menos eficiente de los encendidos y para el encender el más eficiente de los apagados, se garantiza que el subconjunto de nodos encendidos siempre va a ser el más eficiente (más detalles en las secciones 4.3.2 y 4.3.3).

4.3.1.1 Bloqueos de transiciones de estados y protección de histéresis

Para minimizar las transiciones entre estados estacionarios y mejorar la respuesta, se han estudiado posibles bloqueos para ejecutar transiciones dependiendo del estado de otros nodos.

El primer bloqueo se produce en el encendido de un nodo, de forma que cuando hay un nodo en estado *TURNING_ON*, no se puede encender otro nodo. El motivo es sencillo y es que la capacidad del clúster aún no se ha modificado hasta que este nodo no se termine de encender y, por tanto, en nuevas aperturas de sesión se seguirá solicitando encender otro nodo, de forma que si el tiempo de encendido es alto, podrían ordenarse encender todos los nodos.

En cambio, en el caso del apagado, no ha lugar, pues la capacidad se modifica instantáneamente al poner un nodo en descarga, y dependiendo del tiempo de descarga, posiblemente desconocido, pueden empezarse a descargar otros nodos (sobre todo en transitorios de carga descendente muy rápidamente).

Sin embargo, existe una situación en la que tener varios nodos en descarga puede ser problemática, y es a la hora de sacar un nodo de descarga debido a un incremento repentino de la carga soportada por el clúster.

Extraer nodos del estado *OFF_LOADING* a *ON* es una medida que permite reducir mucho el tiempo de respuesta del algoritmo en casos de cambios bruscos de carga.

Para solucionar este problema cuando existen más de un nodo en estado de descarga se pueden tomar dos opciones: sólo permitir la existencia de un nodo en ese estado, estableciendo un bloqueo similar al introducido en el encendido, lo que puede hacer que el algoritmo tarde en responder en decrementos rápidos de carga, haciendo que el clúster sea menos eficiente o, por otro lado, permitir varios nodos en descarga, pero de cara a seleccionar cuál hay que escoger para volver a poner en operación, aplicar el cálculo de consumos tentativos explicado en las secciones 4.3.2 y 4.3.3, pero reduciendo el conjunto de nodos candidatos al conjunto de nodos en descarga.

Para discriminar nodos de igual eficiencia, se prefieren los nodos que, teniendo el mismo consumo tentativo, tiene una mayor utilización, lo que minimiza el tiempo de balanceo de carga posterior al *rescate*.

Aplicar este cálculo permite garantizar que el conjunto de nodos encendidos siempre contiene a los nodos más eficientes, debido a que se apagan primero los menos eficientes y, en cualquier situación, se escogen para encender los más eficientes. Con las transiciones existentes, no es posible que se dé el caso en el que se encienda un nodo menos eficiente que otro que esté apagado, pues el segundo nodo ha de estar encendido previamente al primero.

4.3.2 Acciones cuando la carga aumenta

En el momento en el que distribuidor recibe una petición de apertura de sesión, el algoritmo de optimización empieza a operar, decidiendo si se acepta la sesión y, en ese caso, a qué nodo se asigna. Además, puede ser necesario incrementar la capacidad del clúster debido a que ahora se está soportando una nueva sesión.

En primer lugar, cuando una sesión llega al distribuidor, se aplica el control de admisión, actualizando los datos de capacidad y utilización de cada nodo que se esté gestionando y esté encendido (aceptando sesiones), esto es, en estado *ON*.

Para el cálculo de la capacidad máxima se realiza el proceso analítico de cálculo de raíces para la curva estimada de respuesta y el *SLA* definido en el servicio, obteniendo M_i sesiones máximas para cada nodo *i*. Para calcular la utilización, teniendo en cuenta que se tienen abiertas N_i sesiones en el momento actual, la utilización del nodo i se define como:

$$U_i = \frac{N_i}{M_i}$$

De esta forma, se sabe qué proporción de sesiones hay actualmente en uso respecto al máximo de sesiones que permite soportar el *SLA*. Nótese cómo al utilizar un modelo sencillo pero empírico, no es necesario realizar uso de métricas de utilización de dispositivos ni latencias de red, pues todo viene representado en el modelo final y, además, es la visión del usuario final del servicio.

Para el control de admisión, si existe al menos un nodo con una utilización inferior a 1,0, se procederá admitir la sesión; en otro caso se rechaza la misma, pues todos los nodos están al máximo de capacidad que permite satisfacer el *SLA*.

En caso de admisión, se asigna la sesión al nodo de menor utilización, de forma que se balancea la carga del sistema, se actualiza la utilización del nodo asignado y se inicia la fase de control de energía.

Estos primeros pasos permiten garantizar el cumplimiento del *SLA*, evitando una utilización excesiva de los nodos de computación. Los siguientes pasos permiten minimizar el número total de nodos encendidos y el consumo de los mismos, siempre manteniendo el SLA.

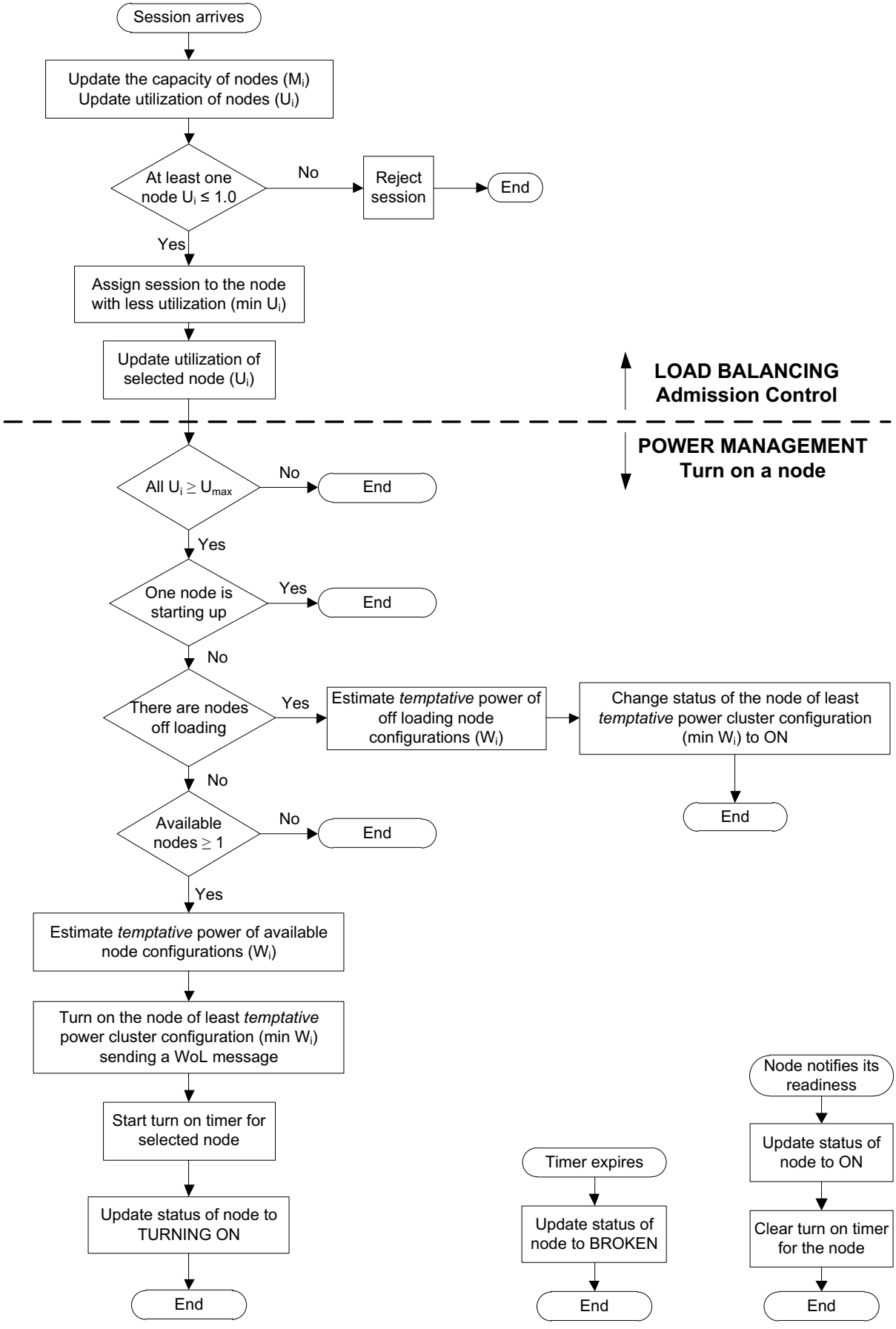


Figura 9: Diagrama de flujo para el incremento de carga

Para ello, el algoritmo comprueba que las utilizaciones de todos los nodos no sobrepasen un umbral definido como el parámetro del algoritmo U_{max}. Este parámetro debe ser inferior a 1,0, pero estar cercano, pues todo lo que se aleje de 1,0, será potencia de cálculo (y energía) desaprovechada. El escoger un valor u otro dependerá del tiempo de encendido de los equipos, que si se mantienen en suspensión en vez de apagados será menor (pero los nodos en estado *OFF* tendrán un consumo no despreciable).

En caso de que las utilizaciones superen el umbral U_{max}, el algoritmo solicitará al gestor de energía el encendido de un nodo, si es que hay alguno disponible en estado *OFF* u *OFF_LOADING*. Para seleccionarlo, se calculará el consumo de todas las configuraciones posibles con los nodos disponibles para el encendido, seleccionando el nodo que genere, para la carga existente, la configuración de mínimo consumo.

Esto garantiza tener siempre el conjunto de nodos más eficiente encendido de forma que el consumo es mínimo. Por último, se actualizan los estados y utilizaciones y se envía el mensaje *WoL*, estableciendo un temporizador de encendido y esperando por el encendido del nodo de forma asíncrona bloqueando, además, nuevos encendidos hasta que éste se encienda y adquiera, por tanto, el estado *ON*.

En este transitorio, dónde el encendido está bloqueado, pueden producirse rechazos de sesión temporales, por lo que el ajuste del umbral $U_{máx}$ es la clave para hacer al algoritmo reaccionar a tiempo y encender un nodo con antelación suficiente como para evitar rechazos indeseados, pero lo suficientemente tarde como para aprovechar la capacidad instalada en el clúster al máximo.

Cuando el nodo notifica su disponibilidad, se le comienza a asignar sesiones y la capacidad del clúster es finalmente incrementada. Si se supera el tiempo límite de encendido, se asigna al nodo el estado *BROKEN*.

Si la notificación llega retrasada, el nodo vuelve a estado *ON*, pero se avisa igualmente al administrador para que compruebe el estado del hardware, que puede ser síntoma de un fallo inminente.

No obstante, las notificaciones retrasadas presentan un problema relativo al incremento de capacidad sin control del algoritmo: cuando el nodo es marcado como BROKEN, el algoritmo escoge otro nodo para encender, si es que está disponible, de forma que incrementa la capacidad disponible. El problema se produce cuando, al llegar la notificación con retraso, el nodo que está marcado como *BROKEN*, pasa a estar en estado *ON*. En este momento se incrementa la capacidad

disponible y el algoritmo, si la carga se mantiene, probablemente decidirá apagar otro nodo (o el recién encendido), de forma que se incrementa la histéresis del algoritmo.

Por esto es fundamental medir de forma correcta los tiempos de encendido y apagado de los nodos, además de incrementarlos en un margen de error apropiado.

4.3.3 Acciones cuando la carga disminuye

El momento en el que se cierra una sesión es el otro punto de entrada al gestor energético. En este caso, además de actualizar los estados de los nodos, se debe decidir si se reduce la capacidad del clúster después de la modificación sufrida en el estado global.

Las acciones a tomar son básicamente las opuestas a las tomadas para el encendido, pero con una ligera diferencia.

En la Figura 10 se muestra el diagrama de flujo de las acciones a tomar por el algoritmo en el caso de un cierre de sesión.

En este caso, se define el segundo y último parámetro de funcionamiento del algoritmo, el umbral de apagado U_{min}, que define una utilización tal, que si la utilización de todos los nodos está por debajo de ela, debe apagarse uno de los nodos. Esta utilización es una estimación de la utilización tras el balanceo de la carga que se produce al apagar un nodo, o utilización de *post-apagado*.

En la sección 6.1 se detalla el porqué de este comportamiento y las implicaciones que tiene, no obstante, como resumen de las mismas, el objetivo de este umbral dinámico es estabilizar la operación del algoritmo, concentrando la carga en los mismos niveles sea cual sea el número de nodos que componen el conjunto de nodos encendidos.

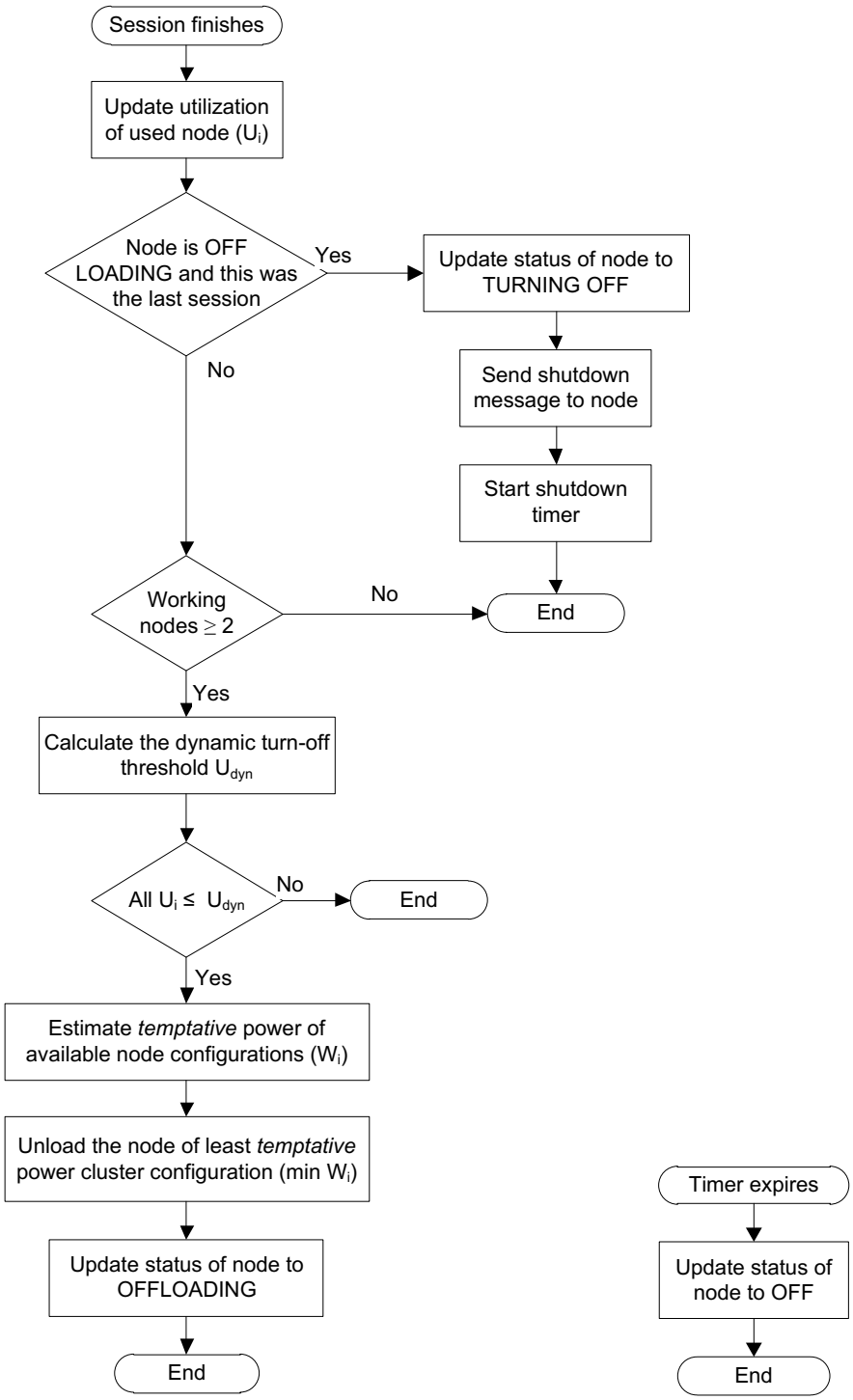


Figura 10: Diagrama de flujo para la disminución de carga

Si todos los nodos cumplen la condición de estar por debajo de U_{din}, (el umbral de post apagado) se decide apagar uno de ellos (quedándose siempre con uno disponible para la operación del clúster).

En este caso, se estima el consumo o eficiencia de los nodos en el momento actual (pueden estimarse tanto el consumo como la eficiencia, aunque no son términos exactamente

equivalentes para este caso, ver sección 4.3.5) para todas las configuraciones posibles con los nodos encendidos, seleccionando el nodo que hace que el consumo sea mínimo para la configuración del clúster con un nodo menos y la carga actual.

El motivo de esta estimación es que las curvas de carga esperadas de servicios transaccionales típicos son muy suaves y no cuentan con oscilaciones abruptas entre aperturas y cierres de sesión (en otras palabras, estas curvas cuentan con alta histéresis). De esta forma, cuando se producen incrementos de carga, es esperable que sigan produciendo incrementos y cuando se producen descensos, es esperable que se sigan produciendo descensos.

Una vez seleccionado un nodo, se actualiza su estado y se pone en descarga. Cuando se termina la descarga, se le envía un mensaje de apagado físico y se inicia un temporizador de apagado. Cuando este expire, se marca el nodo como apagado.

No hay comprobación física de apagado pues no es necesaria para la operación del algoritmo. No obstante, si para un determinado hardware sobre el que opere el algoritmo es necesario, se cuenta con la posibilidad de sustituir este mecanismo por uno de *polling*.

4.3.4 Umbrales de funcionamiento

El algoritmo de optimización ve el clúster como una caja negra de cara a su modelado y las decisiones que se toman sobre esta caja negra son estrictamente dependientes de la garantía de la QoS del servicio.

De esta forma, sólo se necesita configurar dos parámetros (además de proporcionar la descripción del hardware y del servicio que va a controlar mediante los modelos de QoS y potencia), que son los umbrales U_{max} y U_{min}. Estos dos parámetros definen cuándo se va a encender y apagar un nodo y, en definitiva, la precisión con la que el algoritmo va a funcionar.

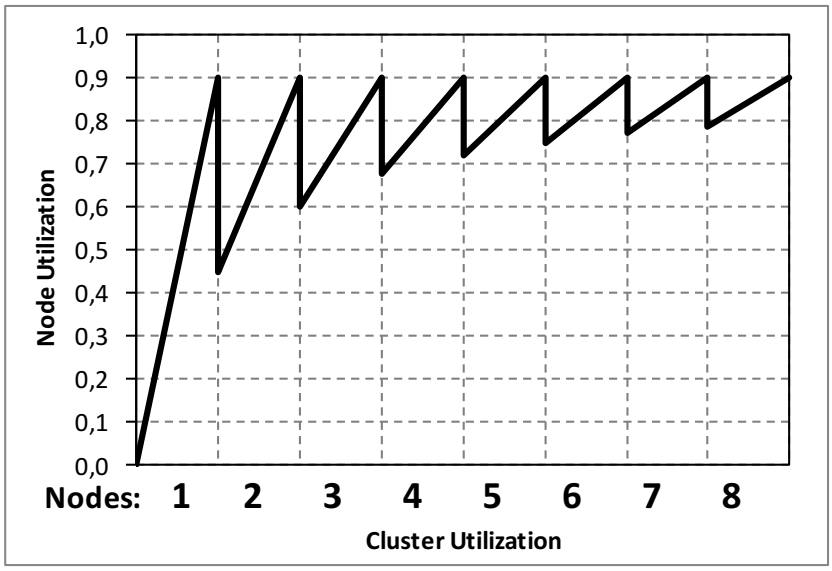


Gráfico 7: Utilización del clúster cuando la carga aumenta

En el Gráfico 7 se representa la evolución de la utilización combinada de un clúster teórico homogéneo utilizando un umbral U_{max} de 0,9. La representación permite comprender cómo evoluciona la utilización del clúster conforme se van enciendo nodos mientras la carga aumenta.

Hay que tener en cuenta que en la Gráfico 7 los balanceos y tiempos de encendido son instantáneos para enfatizar el comportamiento dependiente del umbral.

Cuando el primer nodo llega a superar el umbral de utilización máxima, U_{max}, se enciende el segundo, de forma que la utilización baja a 0,45 por nodo. Al encender el tercer nodo, la utilización baja a 0,33 por nodo, como era esperable.

Con esto, se puede calcular la utilización esperada del clúster después de encender un nuevo nodo y que se complete el balanceo de carga, si se opera con N nodos y se pasa a $N + 1$ nodos:

$$U_i = \frac{N}{N+1} \cdot U_{max}$$

Es remarcable cómo la selección de este umbral U_{max} ha de estar muy próxima a 1,0, pues, como se puede ver en el ejemplo, al utilizar 0,9, se desaprovecha sistemáticamente el 10% de la capacidad, que sólo se usará cuando ya no quedan nodos apagados y durante los transitorios de encendido de los nodos.

La relación entre los dos parámetros es también importante, pues el sistema ha de tener una respuesta que tienda a concentrar la carga y maximizar dentro de lo posible la utilización de los nodos de computación. Los umbrales de post-apagado y la relación entre ambos se encuentran explicados en la sección 6.1.

4.3.5 Consideraciones para clústeres heterogéneos

Si se está manejando un clúster homogéneo, donde todos los nodos tienen la misma capacidad y consumo, el algoritmo funciona como un algoritmo de "seleccionar el primero", pues todas las opciones son iguales.

En el caso de clústeres heterogéneos, el manejo de las curvas de potencia y de respuesta es fundamental para optimizar el funcionamiento, por tanto el algoritmo es plenamente compatible con la administración de clústeres heterogéneos.

Opcionalmente, puede decidirse qué parámetro energético optimizar, o bien directamente el consumo, o la eficiencia. Optimizar el consumo permite garantizar que siempre se tiene el subconjunto de nodos que tiene un consumo mínimo encendidos, no importando la capacidad (el algoritmo garantiza que sea suficiente para no violar el *SLA*).

Optimizar la eficiencia puede ser equivalente, pero según la métrica de eficiencia escogida, permite optimizar el consumo por usuario, permitiendo optimizar el consumo en función de la capacidad habilitada en cada momento. Fundamentalmente, se puede establecer una métrica de eficiencia en términos de energía por usuario (con objetivo de su minimización), de forma que, aunque la tendencia va a ser la concentración de carga, el conjunto de nodos seleccionando puede variar con respecto al seleccionado utilizando una métrica de consumo absoluto para la misma carga.

Si se usa una métrica de energía, se escogerán los nodos de mayor capacidad, con el fin de minimizar el impacto del consumo de base de cada nodo por sesión, mientras que con una métrica de consumo absoluto, siempre se minimiza el consumo, lo que dependerá del modelo de potencia obtenido para cada nodo.

Es importante indicar que para las mediciones de eficiencia es conveniente utilizar métricas que relacionen la potencia consumida con el número de usuarios o métricas de energía por sesión. Si se usan medidas de potencia, se están utilizando métricas de consumo instantáneo, por lo que relacionarlas con la carga es más conveniente que con la unidad de sesión, que puede tener diferentes características según el servicio proporcionado.

El utilizar métricas de energía por sesión ofrece una relación directa con métricas de coste.

4.3.6 Sincronización de toma de decisiones

Las tomas de decisiones, en contextos multihilo y multiproceso han de ser sincronizadas, bien estableciendo bloqueos o un contexto coherente que permita tomar decisiones de forma coordinada en paralelo.

Es fundamental, como se deduce de la estructura del algoritmo, que los encendidos y apagados de nodos (y, en general, cualquier decisión que pueda implicar un cambio de estado) sea tomada de forma sincronizada, para no corromper el seguimiento del estado de cada nodo, que constituye un estado global para todo el sistema de decisiones.

4.4 Integración del sistema en servicios existentes

El objetivo del algoritmo diseñado, más allá de la validación con el prototipo desarrollado, es su integración en entornos operativos con software existente. Por este motivo se ha diseñado de tal forma que los puntos de entrada con un software de distribución de carga sean lo más reducidos posible y facilite la integración, no como en otros sistemas de gestión que requieren grandes cambios en el diseño del mismo [Petrucci09].

Además, el software es integrable con sistemas de monitorización existentes, como *Ganglia*, de forma que se integran en sistemas de monitorización y actuación ya desplegados en grandes organizaciones.

La Figura 11 muestra una posible distribución en módulos del sistema y los puntos de integración.

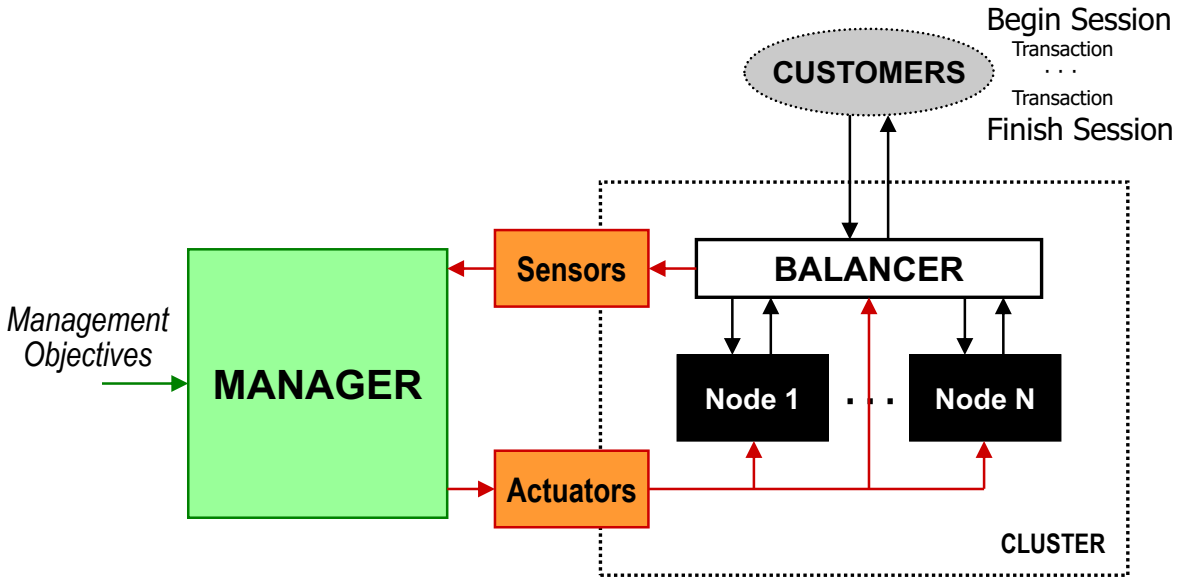


Figura 11: Diagrama de módulos de integración

Como se puede observar, el único puto de integración es el distribuidor (*balancer*), donde hay que establecer dos entradas: una para los sensores, que deben ser capaces de obtener el estado del clúster en una forma estándar y extensible para diversos tipos de servicio y una para los actuadores, que entrarán en funcionamiento en los momentos de apertura y cierre de una sesión, haciendo que el manager tome una decisión en base a la información proveniente de los sensores.

Al tratarse de un algoritmo orientado a decisiones sobre eventos discretos (las aperturas y cierres de sesión), los sensores no tienen por qué estar monitorizando y recolectando información de forma continua, a excepción de que se estén generando los modelos del sistema de forma dinámica, en cuyo caso, una estrategia de ventana y temporización como la mostrada en la sección 9.3.2 permite reducir notablemente la sobrecarga de la captura en línea de datos de tiempo de respuesta.

Para el caso de los actuadores sobre los nodos, al tratarse de protocolos y mensajes estándar (*WoL* para el apagado y monitorización basada en *WMI* o *CIM* para el control de disponibilidad y, si se considera necesario, carga), no es necesario integrar ningún software.

4.5 Estrategias de validación del algoritmo

Finalmente, para la validación de la estrategia se han estudiado dos opciones: a) una validación empírica mediante la implementación de un prototipo de servicio transaccional que permita emular diversos tipos de servicios y comprobar la funcionalidad y optimalidad del algoritmo en diferentes situaciones, y b) validación mediante modelos de colas.

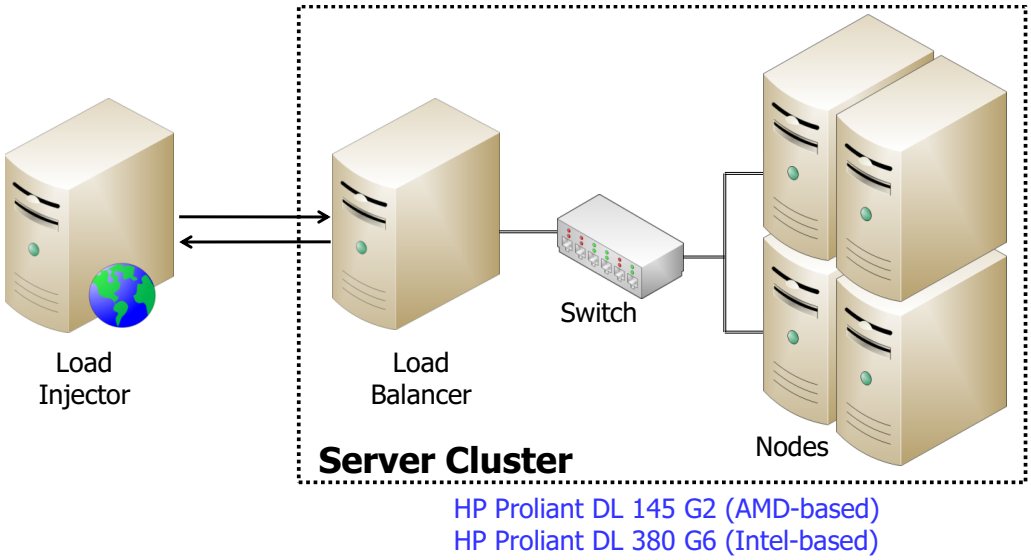


Figura 12: Diagrama del entorno físico de pruebas

En evalidación empírica, seleccionada para este proyecto, se cuenta con un entorno experimental constituido por un clúster heterogéneo de computadores, cuya estructura puede verse resumida en la Figura 12. En este caso, se cuentan con dos tipos de nodos de dos generaciones de hardware, basados en procesadores AMD *Opteron* e Intel *Xeon*, cuyas características en detalle pueden observarse en la sección 10.

La otra opción de validación es mediante una validación formal por modelos de colas, con el paquete QNAP por ejemplo.

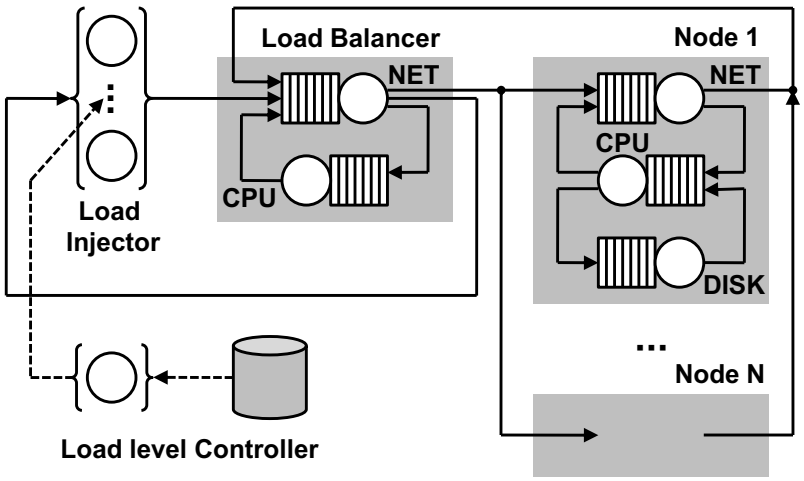


Figura 13: Diagrama de validación del modelo de colas

La Figura 13 muestra un resumen del sistema cerrado de colas para la simulación de los nodos y del distribuidor.

La validación se realizará mediante la implementación de un software prototípico, puesto que la validación por modelos de colas es parte de otro proyecto fin de carrera desarrollado en paralelo a éste.

5 MODELADO DE CALIDAD DE SERVICIO Y ENERGÍA

El modelado del sistema transaccional a controlar es fundamental de cara a permitir operar al algoritmo de optimización. Éste cuenta con tan sólo dos parámetros, pero los modelos son de capital importancia para que las predicciones y decisiones se tomen con información que represente la realidad que está modelando.

5.1 Modelos de ajuste y estimación de curvas

Para estimar la calidad del servicio que se está proporcionando, es necesario estimar la curva que representa el tiempo de respuesta en función del número de usuarios. De esta forma, es posible determinar qué hacer ante la llegada de un nuevo cliente (control de admisión) o modificar mediante provisión dinámica de nodos la configuración del clúster.

Adicionalmente, esta estimación debe realizarse *on-line*, mientras el sistema está en funcionamiento, de forma que la curva se adapte dinámicamente a las condiciones de trabajo reales del sistema.

Existen diversos métodos de estimación de las curvas del tiempo de respuesta. Fundamentalmente, se trata de resolver un problema de regresión polinómica de un determinado orden. Dentro de este conjunto de problemas, cabe destacar dos aproximaciones: el ajuste por mínimos cuadrados (LS, *least squares*) y el ajuste por mínimos cuadrados no negativos (NNLS, *non-negative least squares*).

Las entradas a estos métodos provendrán de un experimento de carga, de forma que se tiene la información de entrada para que el método de estimación comience a operar. Esta información debe reflejar la realidad de operación del clúster, esto es para cada nivel de usuarios esperado, conocer el percentil-90 del tiempo de respuesta y el consumo energético. De esta forma se puede garantizar la capacidad para cubrir el *SLA* y minimizar el consumo.

5.1.1 Ajuste por mínimos cuadrados

El ajuste por mínimos cuadrados (LS) es el modelo de ajuste más común y generalista [Nakamura92]. El problema se puede resumir en un problema de optimización, más concretamente de minimización de una función de error E, definida de forma que si la función $g(x)$ es el polinomio de orden N que define la curva estimadora de los datos, se tiene que:

$$g(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N$$

$$E = \sum_{i=1}^{L} \left(y_i - g(x_i)\right)^2$$

Donde L es el número de puntos dados para el cálculo de la curva de estimación. De cara a buscar el conjunto óptimo de parámetros para la función $g(x)$ (sus coeficientes), se igualan a cero las derivadas parciales de R respecto a cada uno de sus coeficientes:

$$\frac{\partial R}{\partial a_i} = 0, \qquad i = 0,1,2 \dots N$$

Estas derivadas parciales dan como resultado, de forma resumida, la siguiente expresión:

$$\sum_{n=0}^{N} \left[\sum_{i=1}^{L} x_i^{n+k} \right] a_n = \sum_{i=1}^{L} x_i^k \, y_i, \qquad k = 0,1,2 \dots N$$

La importancia de este resultado puede observarse al representarlo (se trata de un sistema de ecuaciones) en forma matricial:

$$\begin{bmatrix} L & \sum x_i & \sum x_i^2 & \cdots & \sum x_i^N \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \cdots & \sum x_i^{N+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum x_i^N & \sum x_i^{N+1} & \sum x_i^{N+2} & \cdots & \sum x_i^{2N} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^N y_i \end{bmatrix}, \qquad i = 0,1,2 \dots L$$

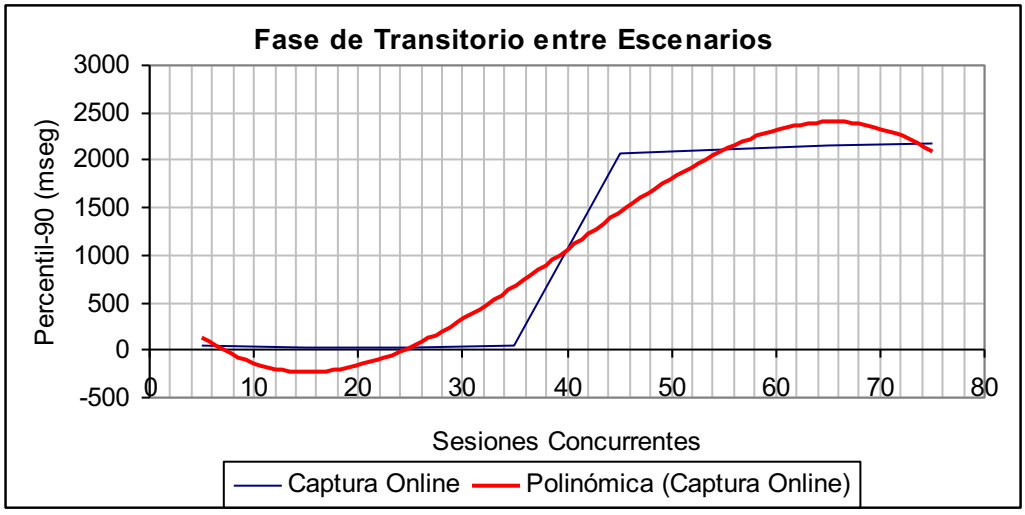
Tal y como muestra la ecuación anterior, puede observarse cómo la matriz de entrada al problema es una matriz de Vandermonde construida a partir de los puntos de entrada.

5.1.1.1 *Análisis del método de los mínimos cuadrados*

En el trabajo de investigación sobre calidad de servicio [Valledor09] y en [Garcia09] se realiza un análisis exhaustivo y experimental a la utilización del método de los mínimos cuadrados para el modelado dinámico de la calidad de servicio.

Fundamentalmente, este método sufre de dos problemas de vital importancia que lo desaconsejan para obtener curvas de servicio de forma dinámica, tal y como ejemplifica la Gráfico 8:

- En ciertas situaciones, se pueden obtener curvas que estimen valores de tiempo de respuesta negativo.
- Además, pueden darse situaciones en las que la curva estimada no sólo no sea creciente, sino que sea decreciente. Esto carece de sentido, pues el incremento de clientes, en general, debe hacer incrementar el tiempo de respuesta promedio.



[13]

Gráfico 8: Problemática del uso de LS para la estimación en línea

La curva estimada debe satisfacer las condiciones de no negatividad y ser monótona creciente, de forma que los tiempos de respuesta estimados sean coherentes.

[13] Gráfico obtenido de [Valledor09].

5.1.2 Ajuste por mínimos cuadrados no negativos

El método de ajuste por mínimos cuadrados no negativos (*NNLS*, por su nombre en inglés) es un método de ajuste ideado por Lawson y Hanson en 1974 [Lawson95]. Este método garantiza que las curvas generadas cumplen con la condición de no negatividad y monotonía creciente.

El método NNLS es muy similar al método de mínimos cuadrados comentado en la sección 5.1.1, pero se le añaden restricciones de inecuaciones lineales para satisfacer las restricciones de no negatividad. Concretamente, y en forma matricial, el método NNLS resuelve, minimizndo el error cuadrático, el problema siguiente:

$$Ex = f, \qquad x \geq 0$$

La restricción de que los coeficientes del polinomio que describe la curva de estimación sean positivos garantiza las restricciones de no negatividad y monotonía creciente que se requieren a las curvas usadas para estimar la calidad de servicio.

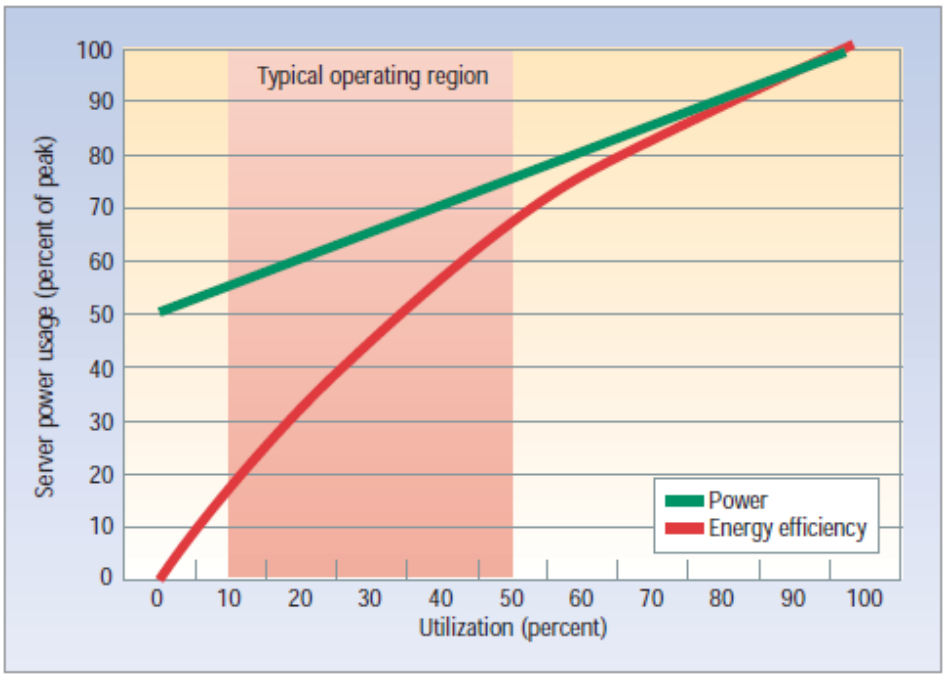
Las matrices de entrada al algoritmo son la matriz de Vandermonde definida para el método de los mínimos cuadrados estándar y el vector de productos también definido para ese método.

Adicionalmente, la implementación ofrecida por Lawson y Hanson tiene un buen comportamiento en términos de complejidad computacional para un conjunto reducido de datos de entrada, $O(n^2)$, aunque existen algunas con un mejor comportamiento en general [ChenD07].

La principal limitación que se tiene utilizando este método es la propia limitación de monotonía creciente que se establece a la curva, que si bien es un requisito para modelos que se están estimado dinámicamente (para evitar curvas decrecientes que relacionen carga y tiempos de respuesta, por ejemplo), en ciertas situaciones puede hacer que el ajuste no sea tan bueno como con las estimaciones polinómicas genéricas mencionadas en la sección 5.1.1.

5.2 Modelado de consumo energético y eficiencia

De cara a mejorar la proporcionalidad en el consumo energético, tal y como se establece en las secciones 3.2.1 y 3.2.3, se ha de establecer una relación entre la eficiencia energética y el nivel de utilización del computador, permitiendo de esta forma conocer las prestaciones energéticas del computador en cuestión y tratar de optimizar su punto de funcionamiento seleccionando uno de máxima eficiencia.



[14]

Gráfico 9: Eficiencia energética para un servidor poco eficiente

El Gráfico 9 permite contrastar el consumo (en porcentaje sobre el pico) con la eficiencia energética, calculada como el cociente entre la utilización y la potencia para un nivel de carga i:

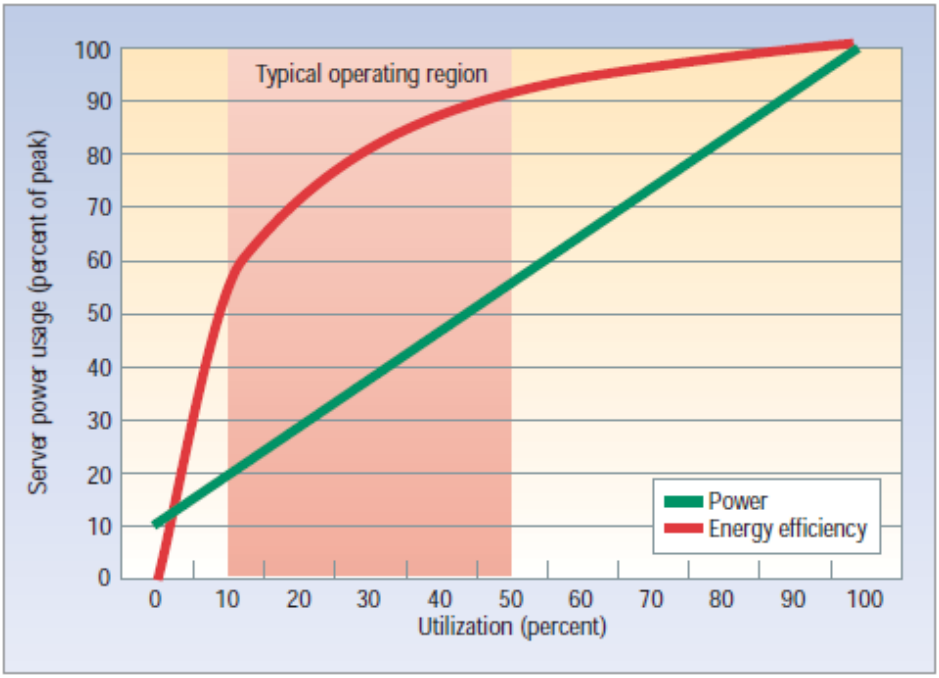
$$E_i = \frac{U_i}{P_i}$$

En el Gráfico 9, con el fin de generalizar, se muestra la eficiencia como una medida de porcentaje respecto a la eficiencia máxima, así como la utilización del sistema como un porcentaje respecto a la utilización máxima, esto es:

$$E_{\%} = \frac{E_i}{E_{max}} \qquad U_{\%} = \frac{U_i}{U_{max}}$$

Como se puede observar en el Gráfico 9, en un servidor de poca eficiencia, esto es, con un *idle power* alto (en este caso, de prácticamente el 50% del consumo de pico), la eficiencia energética es relativamente baja en las zonas de operación típicas entre el 10% y el 50% de utilización.

[14] Figura obtenida de [Barroso07].



[15]

Gráfico 10: Eficiencia energética para un servidor muy eficiente

Si se considera un servidor más eficiente, como el representado en el Gráfico 10, se observa cómo la eficiencia energética crece mucho en las zonas de operación típicas, permitiendo un mejor aprovechamiento de los recursos energéticos durante más tiempo.

Es notable reseñar cómo esto se debe a dos efectos: la reducción del *idle power* y, sobre todo, la mejora en la proporcionalidad en el incremento del consumo con la utilización.

Sin embargo, otra forma de visualizar el comportamiento en términos de eficiencia energética, es mediante la representación directa de la relación usuarios/vatio (métrica, por otro lado, mucho más común en la industria de componentes informáticos).

[15] Figura obtenida de [Barroso07].

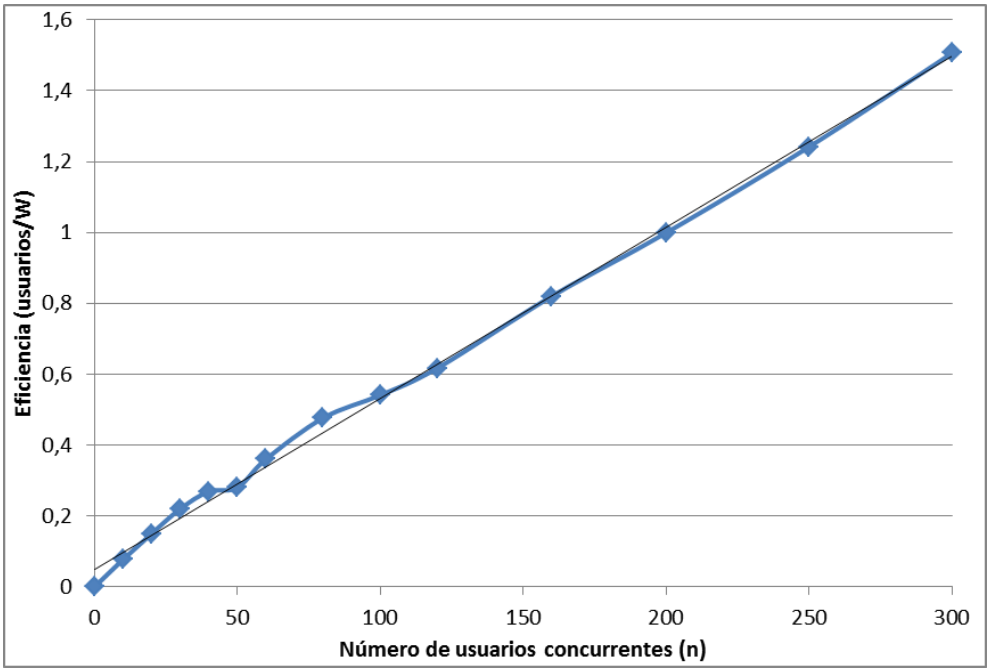


Gráfico 11: Eficiencia energética del procesador Intel *Xeon*

Tal y como se observa en el Gráfico 11, que representa la eficiencia energética para un procesador Intel *Xeon* descrito en la sección 10, lo usual es encontrarse un crecimiento en la eficiencia de carácter plenamente lineal. Es destacable la zona de crecimiento supralineal, al inicio del rango de carga, debido a la mejora en el consumo introducida por el uso de técnicas como el *DVS* y *power gating*. A partir de un cierto nivel de carga (en este caso, unos 125 usuarios), este efecto se ve anulado por la desaparición del idle time en el procesador, clave para el funcionamiento del *DVS*.

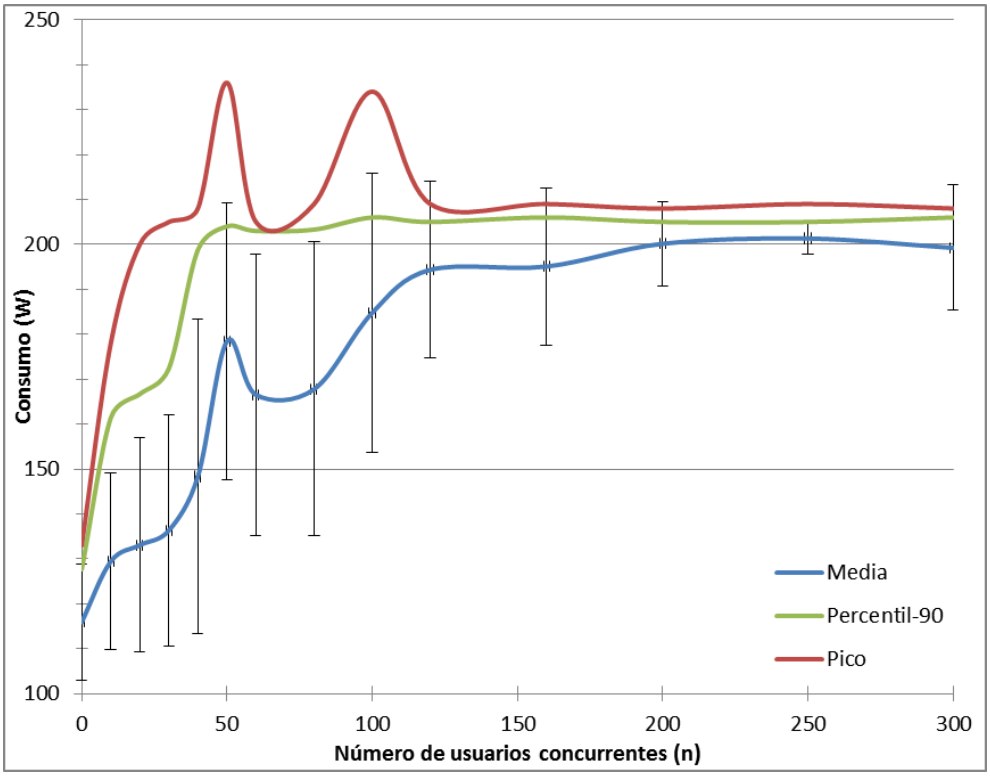


Gráfico 12: Métricas de consumo del procesador Intel *Xeon*

También es necesario indicar que los modelos anteriores son modelos teóricos donde el incremento de consumo se produce con una dependencia lineal respecto a la utilización. En un modelo real, como el mostrado en el Gráfico 12, medido empíricamente sobre el procesador Intel Xeon, el incremento se produce de forma lineal durante un régimen inferior al 100% de utilización (sobre la utilización máxima) y, después se mantiene constante.

De esta forma, la curva de eficiencia energética no tiene una forma tan marcadamente monótona creciente, de hecho, podría modelarse como dos curvas, una de crecimiento lineal y otra constante sobre el pico de consumo y, por tanto, hay puntos donde la eficiencia baja.

Esto convierte el problema de seleccionar qué nodos mantener encendidos y en qué puntos de funcionamiento en un problema de optimización, donde es necesario encontrar el punto de funcionamiento adecuado en cada momento. Por supuesto, con una utilización del 100% se obtiene el máximo nivel de eficiencia, pero esto puede traer problemas de tiempos de respuesta si se combina esta política con una de restricción de tiempos mediante un acuerdo SLA.

5.2.1 Implicaciones del tipo de métrica escogida

De cara al modelado del servicio y a la optimización en línea, pueden tenerse en cuenta indistintamente las métricas de eficiencia y consumo directo, pero teniendo en cuenta que las políticas obtenidas serán diferentes en cada caso.

En caso de utilizar métricas de consumo directo, la política generada en el algoritmo operará reduciendo al máximo el consumo. Si se usan métricas de eficiencia, se operará maximizando la eficiencia.

En general, esto afectará al subconjunto de nodos que el algoritmo escogerá en su operación. En el caso de métricas de consumo directo, se seleccionará siempre el nodo que menor consumo tenga, y puesto que las curvas de consumo generalmente tienen una forma similar, la selección será más o menos directa para todo el rango de operación.

En el caso de utilizar métricas de eficiencia, la selección del nodo a encender o apagar dependerá de la eficiencia dinámica de los nodos, esto es, por ejemplo, en el caso del encendido, habrá que seleccionar el nodo más eficiente para el nivel de operación en el que se espera que el clúster quede tras el balanceo, por lo que la selección en entornos de eficiencia variable no será la misma para unos niveles de carga que para otros.

5.3 Experimentos previos

Previamente a la operación del algoritmo es necesario modelar los nodos de trabajo del clúster para representar su comportamiento de forma sencilla y coherente. Para la validación se van a usar dos tipos de nodos, descritos en la sección 10.

5.3.1 Obtención de curvas de carga estáticas del clúster

De cara a implementar, probar y validar el prototipo de gestión autonómica de la energía del clúster, es necesario obtener y contrastar la curva de carga del servidor de forma experimental, mediante medición directa del servicio y, a continuación y a partir de los datos obtenidos, construir un modelo analítico que permita representar con un polinomio la curva de Calidad de Servicio (QoS) de cada uno de los nodos del clúster. Esta curva representará la relación entre la

métrica QoS escogida, en este caso el percentil-90 del tiempo de respuesta y la cantidad de usuarios activos en el servicio.

Los polinomios de grado 3 permiten un gran detalle en el modelo, que ha de ser monótono creciente como cabe esperar en una curva QoS de tiempo de respuesta respecto al número usuarios activos, tal y como se observó en la sección 5.1, sin incurrir en la inclusión de un gran número de parámetros que compliquen el modelo en exceso.

Como desventaja, el utilizar un modelo polinómico de tercer grado trae consigo la implicación de que, dadas las características de las curvas a modelar, los datos obtenidos con cargas más altas van a tener más peso en el modelado (al tener asociados tiempos de respuesta más altos, la suma de sus cuadrados son más altas) y el ajuste se va a producir mejor en esas zonas. Por tanto, es crucial escoger de forma adecuada el rango de niveles de utilización que se va a introducir al modelo.

Los modelos obtenidos de la realización de mediciones experimentales sobre el clúster de pruebas son modelos estáticos, es decir, no tienen en cuenta las variaciones que puede sufrir el servicio en tiempo de ejecución y deben ser utilizados para el arranque del sistema de gestión autonómica de energía. Este sistema, mediante una reevaluación en línea del modelo irá adaptando las curvas de los nodos conforme el sistema está funcionando, teniendo siempre información realista para la toma de decisiones.

El procedimiento para la obtención de las curvas QoS incluyó el ajuste del servicio transaccional sintético, que permite emular cualquier otro servicio de forma fácil y cómoda, ver sección 9.3.3.

Esto permite caracterizar el servidor donde se van a realizar las pruebas relativas a la gestión energética, de forma que estas curvas permitirán conocer información relativa a:

- Calidad de servicio proporcionada por el sistema en términos de tiempos de respuesta y productividad en función del número de usuarios concurrentes.
- Utilización de recursos por parte del servicio en función del número de usuarios concurrentes.

Para la obtención de dichas curvas de rendimiento se han realizado mediciones en el sistema de pruebas (ver sección 10 para más detalles sobre el mismo) caracterizadas por los siguientes parámetros:

- **Número de clientes emulados en paralelo:** permite medir la capacidad del sistema en términos de productividad cuando hay numerosos clientes operando con el sistema concurrentemente.

- **Tiempo de reflexión de sesión y petición:** permite emular diferentes tipos de servicios sintéticos, tales como los modelos B2B y B2C.
- **Número de peticiones por sesión:** junto con el parámetro anterior, permite modelar distintos tipos de servicio.
- **Parámetros de operación:** permiten definir cómo se comporta el servidor ante una petición, indicando cuánta CPU han de consumir y cuánta memoria reservar, así como la capacidad de disco necesaria. En general, permiten emular cualquier tipo de servicio transaccional.

Los parámetros anteriores, además de permitir obtener las curvas de respuesta para distintos tipos de servicio, permiten configurar pruebas y perfiles de diferentes características para la optimización del gestor energético para satisfacer de forma óptima los requisitos de cada uno de los modelos de servicio.

5.3.1.1 Medición experimental del clúster

Para la medición del sistema se han realizado medidas sobre uno de los nodos del clúster de pruebas de forma que sea posible conocer los tiempos de repuesta y productividades de cada uno de los nodos y poder conformar el modelo de servicio completo.

Además, estas medidas son necesarias para caracterizar las transacciones que forman las sesiones de forma que se obtenga la carga deseada.

Para la obtención de los modelos de rendimiento, se han comparado modelos no-NNLS con modelos NNLS de diversos grados. Tal y como se observaba en la sección 5.1.1.1, los modelos de ajuste de mínimos cuadrados presentan problemas de negatividad y decrecimiento.

Todos los datos mostrados en esta sección se han tomado en el inyector de carga y representan la visión de los clientes del servicio.

5.3.1.1.1 Características de la carga sintética

Para emular un servicio transaccional se ha escogido un servicio intensivo en CPU debido a que no se cuenta con elementos para emular un servicio intensivo en entrada/salida, como un servidor de base datos, que requiere elementos hardware, como cabinas de disco.

Adicionalmente, una saturación por CPU otorga cierta estabilidad, deseable para validar el algoritmo en unas situaciones controladas y hacer que la variabilidad de la experimentación provenga fundamentalmente de factores internos, y no de factores externos, al menos en los primeros estadios de pruebas.

En general, una transacción ejecutará una media de 4.100.000.000 instrucciones aritmético-lógicas, consumirá 4.096 KiB de memoria y supondrá, al menos, dos transacciones a disco de 1.024 KiB.

5.3.1.1.2 Nodos de tipo AMD

Para los nodos con procesador AMD, se han obtenido los siguientes resultados experimentales.

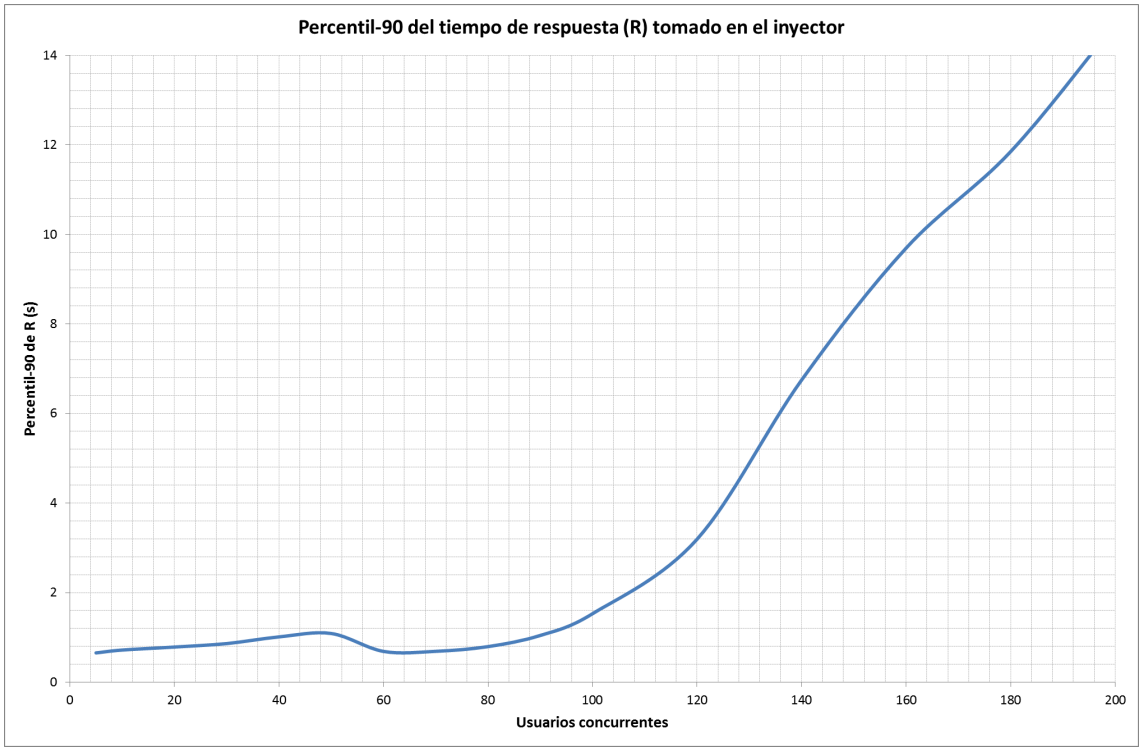


Gráfico 13: Curva de respuesta experimental en nodos AMD

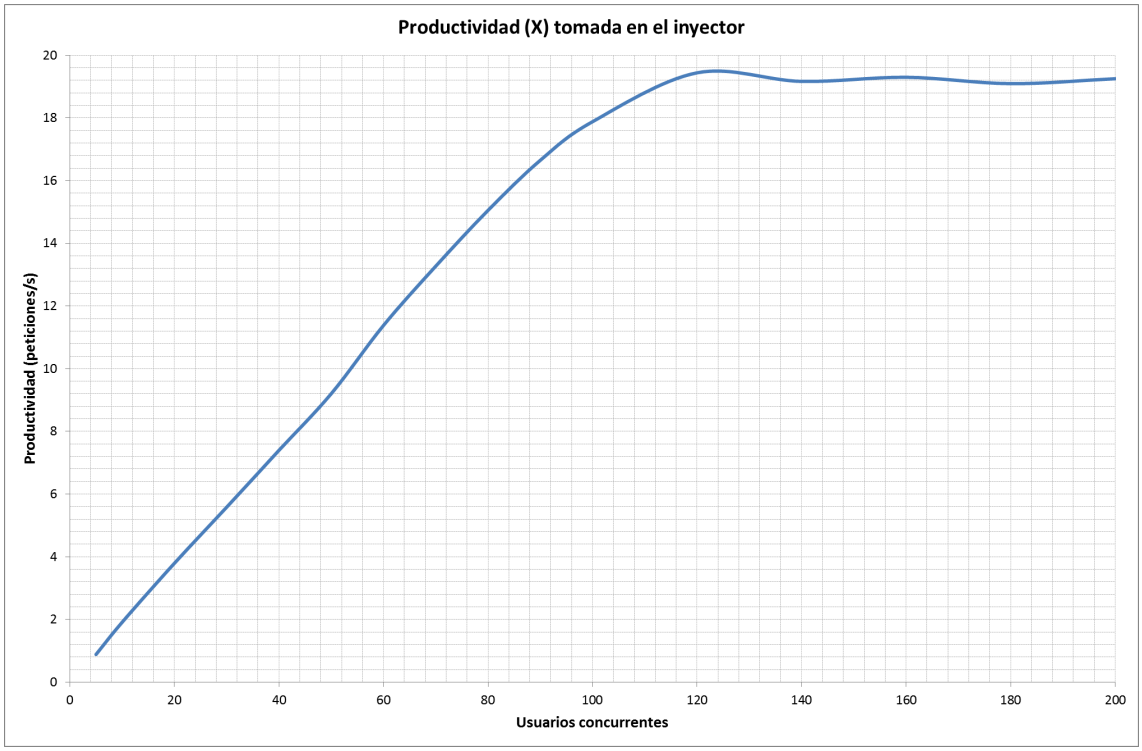


Gráfico 14: Curva de productividad experimental en nodos AMD

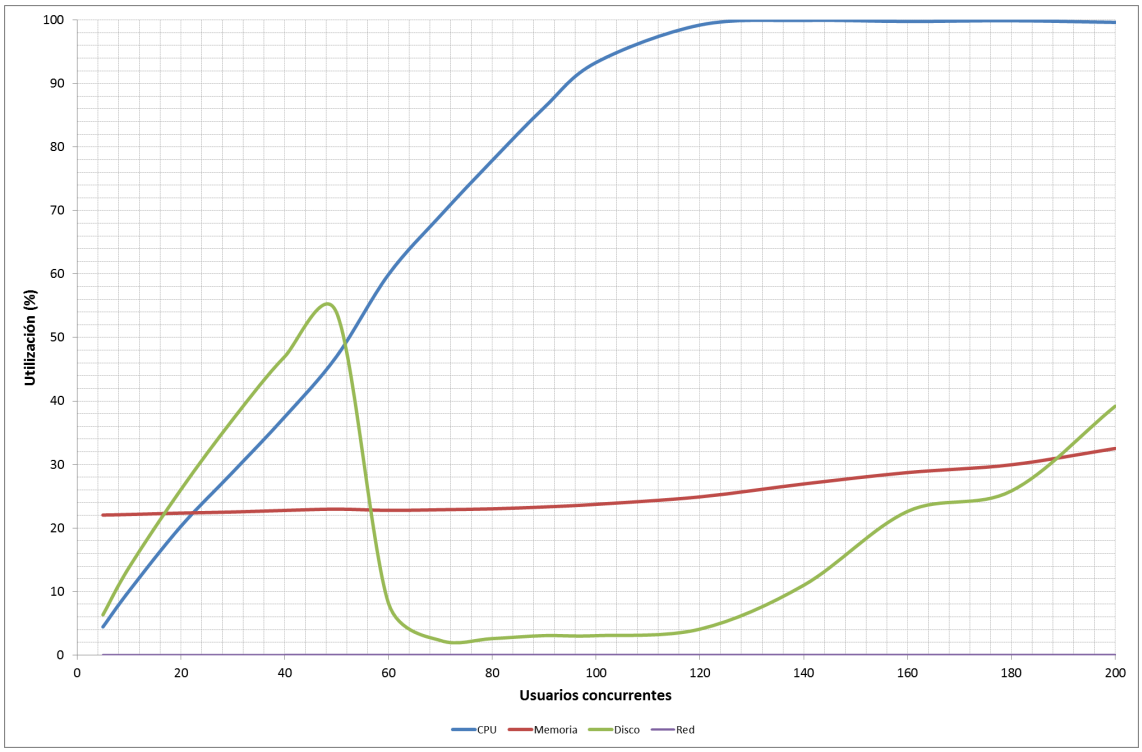


Gráfico 15: Curvas de utilizaciones en los nodos AMD

Tal y como se puede observar en el Gráfico 14 y Gráfico 15, el punto de saturación del servidor para esta configuración de carga de sitúa entre los 100 y 120 usuarios concurrentes

Adicionalmente, tal y como se puede observar en el Gráfico 13, estableciendo un SLA de 2 segundos, se sitúa al servidor en el punto de saturación (limitando el tiempo de respuesta), de forma que la utilización del mismo es la máxima posible, maximizando la eficiencia.

En cuanto a las utilizaciones mostradas en el Gráfico 15, se observa un comportamiento normal, a excepción del alto uso de disco en los estados de baja carga. Este comportamiento no normal se debe a interacciones entre el disco y la CPU, así como del efecto del sistema de caché de ficheros del sistema operativo.

5.3.1.1.3 Nodos de tipo Intel

Los mismos experimentos se han realizado en los nodos con procesador Intel, de cara a obtener el modelo asociado y comparar con los nodos AMD, de menor capacidad.

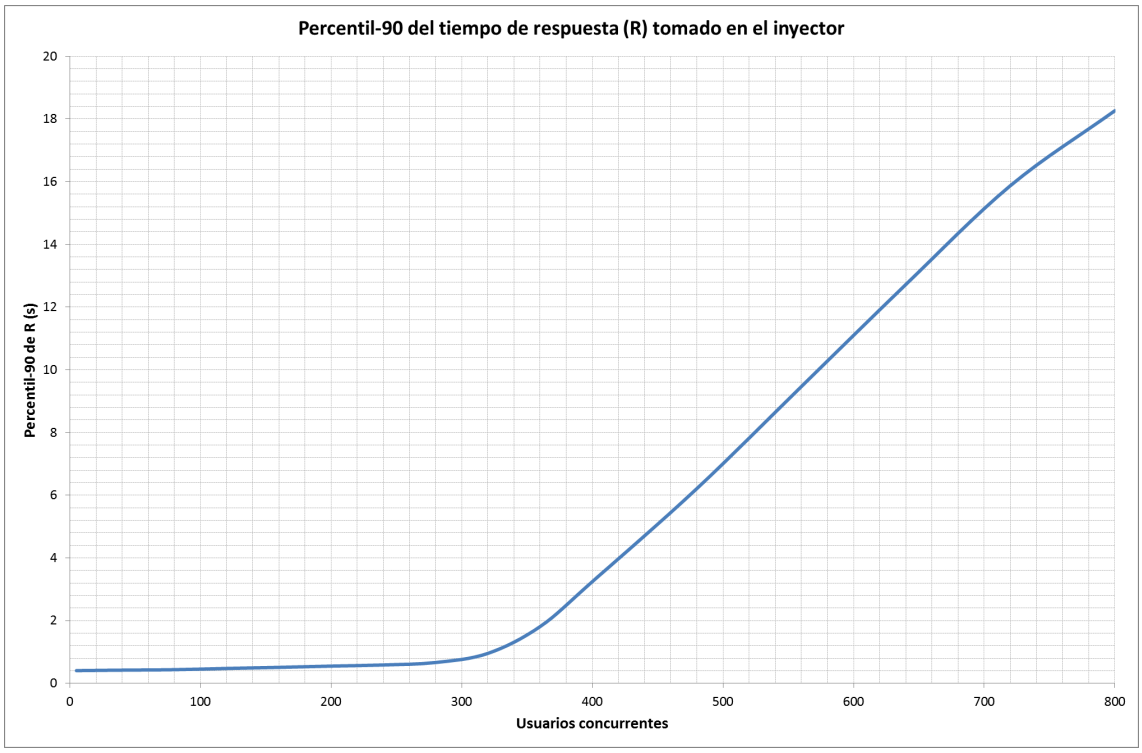


Gráfico 16: Curva de respuesta experimental en los nodos Intel

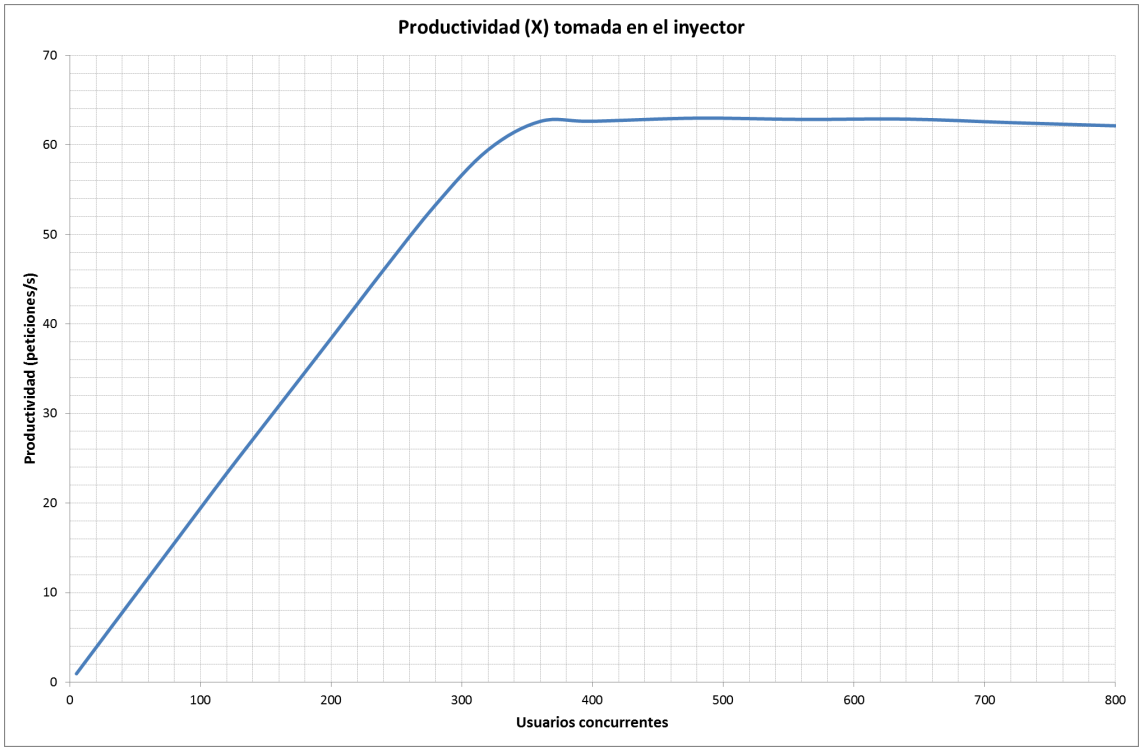


Gráfico 17: Curva de productividad experimental en los nodos Intel

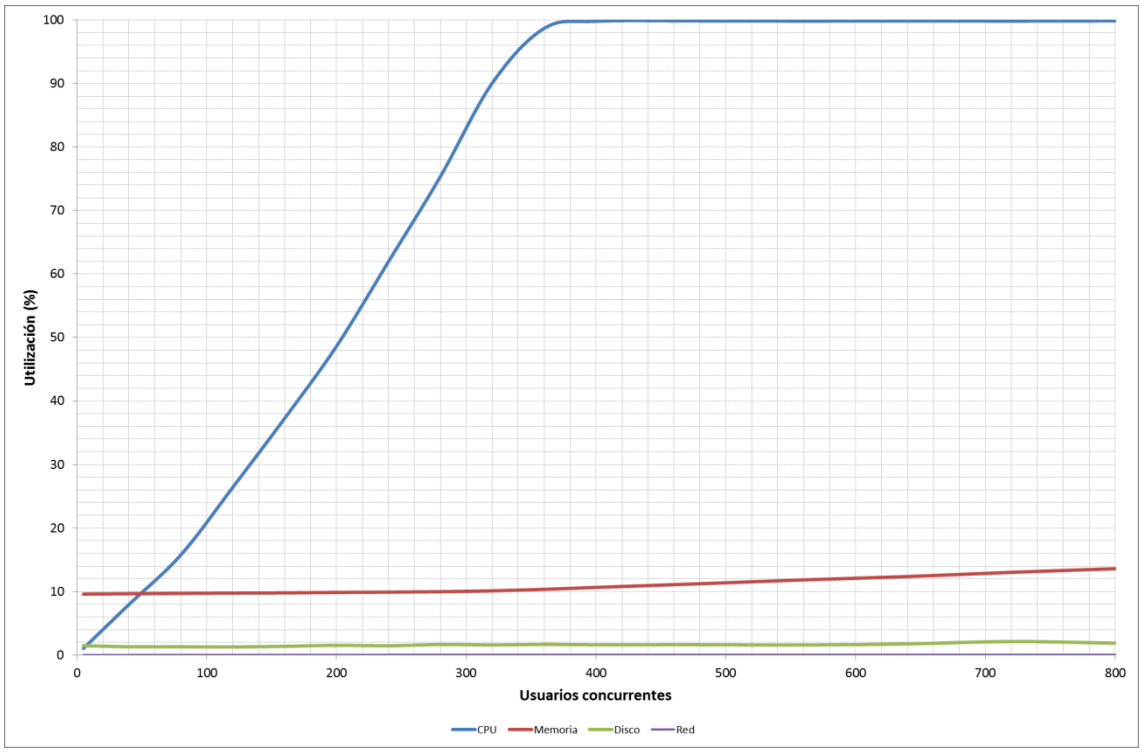


Gráfico 18: Curvas de utilización en los nodos Intel

Tal y como se observa en el Gráfico 17, el Gráfico 18 y Gráfico 19, los resultados para los nodos Intel tienen una forma similar a los obtenidos en el caso de los nodos AMD, con la excepción de

un comportamiento normal de disco y, como era de esperar, una capacidad unas cuatro veces superior a los anteriores, teniendo el punto de saturación en el entorno de los 400 usuarios.

5.3.1.2 Obtención del modelo analítico estático

Una vez completada la fase de obtención de los datos de medición, es necesario obtener un modelo que resuma y represente la información recogida y que cumpla las restricciones que se detallan en la sección 5.1.

De esta forma, se ha implementado un modelo NNLS en base a las mediciones obtenidas en la sección 5.3.1.1. Adicionalmente, se muestran otros modelos de diferente grado y clase para comparar.

5.3.1.2.1 Nodos de tipo AMD

Los modelos asociados a los resultados experimentales son los siguientes:

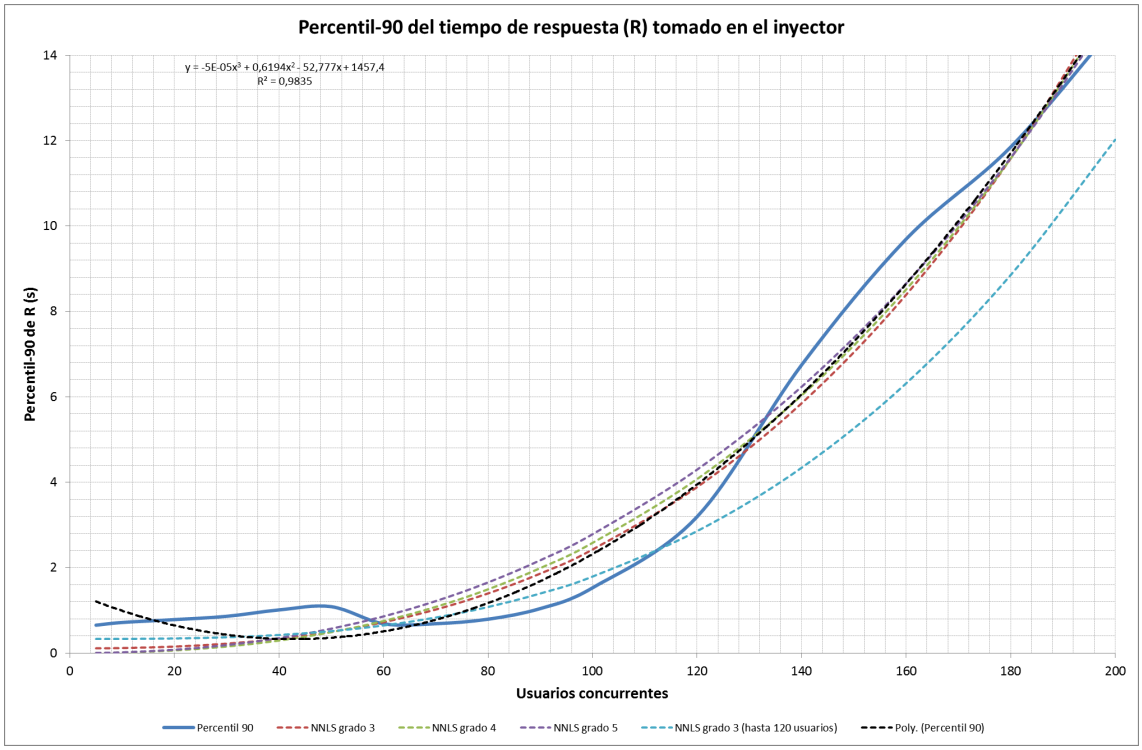


Gráfico 19: Modelos NNLS de respuesta para los nodos AMD

En este caso, y tal y como muestra el Gráfico 19, se ha optado por utilizar un modelo NNLS de grado 3 ajustando hasta los 120 usuarios. A partir de ese punto no se incrementará el número de usuarios en el servidor debido a la gran saturación que sufre el servidor y la degradación del tiempo de respuesta.

Utilizar datos relativos a una cantidad de usuarios mayor disminuye la calidad del ajuste en la zona de operación, puesto que el peso del error es cada vez mayor en esos casos. La siguiente ecuación representa el modelo obtenido:

$$R_{amd}(u) = 0{,}146112 \cdot 10^{-2} \cdot u^3 + 332{,}189670$$

Con este modelo, estableciendo un SLA de 2 segundos, la capacidad máxima del servidor AMD es de 105 usuarios concurrentes.

5.3.1.2.2 Nodos de tipo Intel

Al igual que en el caso de los nodos AMD, para el caso de los nodos Intel se han generado varios modelos para escoger el más adecuado.

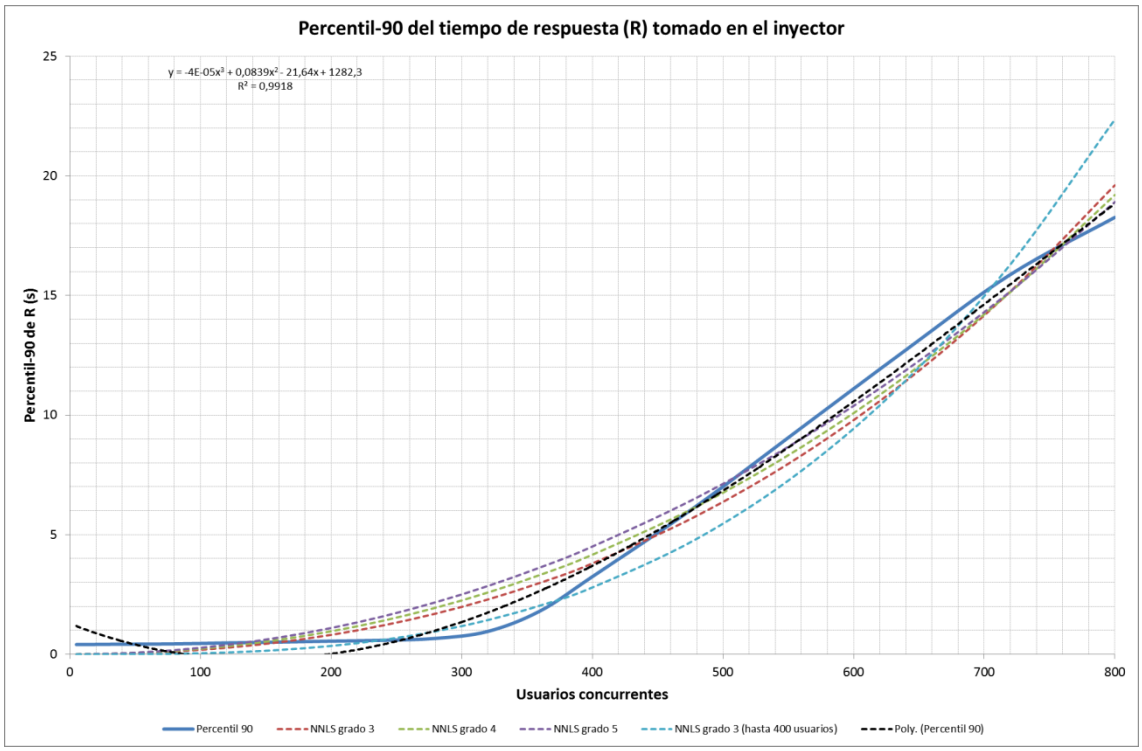


Gráfico 20: Modelos NNLS de respuesta para los nodos Intel

Al igual que ocurría en los nodos AMD, se sobrepasa el SLA de 2 segundos en el entorno del punto de saturación, por lo que la mejor opción (que minimiza el ECM) es usar el modelo teniendo en cuenta datos hasta 400 usuarios, puesto que no se incrementará más el número de usuarios. En este caso, el modelo resultante es:

$$R_{intel}(u) = 0{,}436649 \cdot 10^{-4} \cdot u^3$$

Con este modelo, la capacidad de los nodos Intel es de 358 usuarios concurrentes.

5.3.2 Modelado energético del clúster

De cara a conocer el modelo energético de los nodos, se ha procedido de igual manera que para la búsqueda de los modelos de los tiempos de respuesta, esto es, haciendo un estudio empírico del consumo del nodo servidor del clúster midiendo el consumo que tiene el conjunto total del computador mientras se hace un experimento de carga variable.

5.3.2.1 Medición experimental del clúster

Para el modelado energético se han realizado los estudios en los dos tipos de computadores disponibles, de forma que se puedan obtener los modelos siguientes. El estudio consistió en mantener al computador en niveles estacionarios de carga y medir el consumo durante un período muestral de 60 segundos tomando muestras con un multímetro cada segundo, de forma que se obtiene una muestra de cada nivel de carga de tamaño 60. Para la eliminación de los *outliers*, se ha decidido usar la mediana como medida de centralización.

5.3.2.1.1 Nodos de tipo AMD

El siguiente gráfico resume los resultados obtenidos en los nodos de tipo AMD:

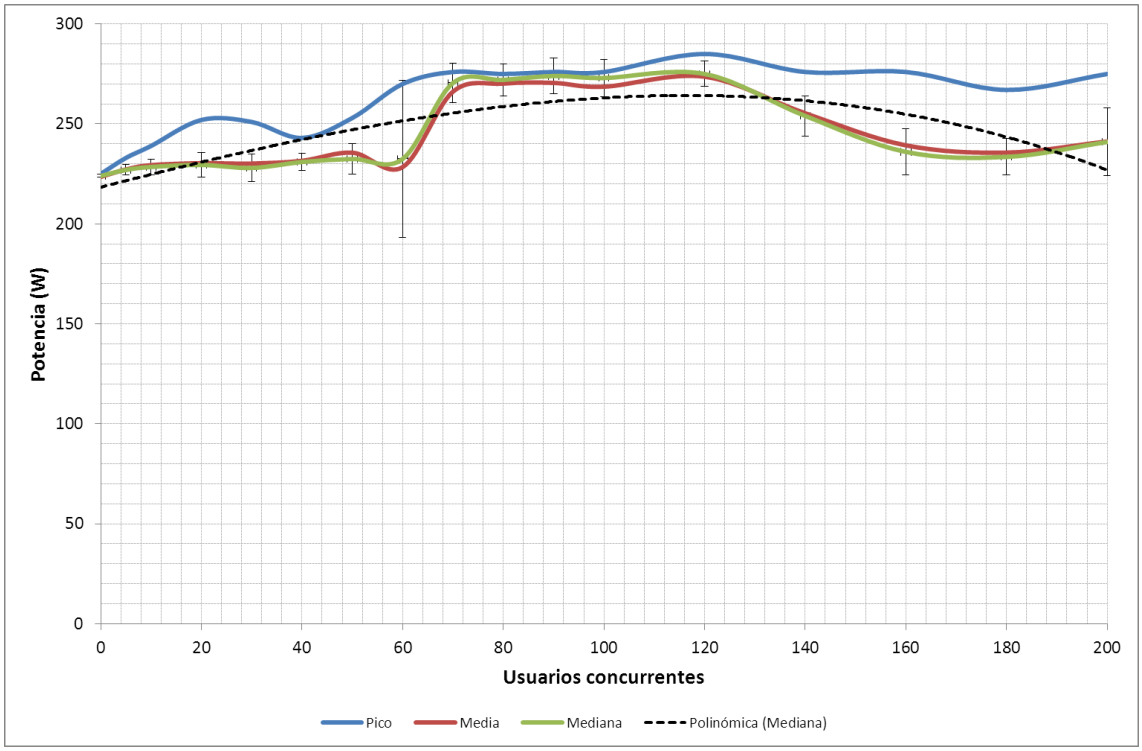


Gráfico 21: Curva de consumo empírico en los nodos AMD

Fundamentalmente se pueden destacar dos puntos: en primer lugar, la variabilidad de los resultados, lo que demuestra que el gestor energético de los procesadores de esta primera generación no es muy avanzado, y, en segundo lugar, lo dispar de los resultados experimentales respecto a los modelos teóricos utilizados en investigaciones previas.

Es interesante, asimismo, ver cómo el consumo energético desciende con la carga a partir de cierto número de usuarios concurrentes. Esto es debido a la contención de las CPUs en el acceso a memoria, lo que les permite apagar unidades funcionales de cálculo, responsables de hasta el 40% del consumo combinado de la CPU.

El modelo a utilizar, basado en la mediana, y realizado con un ajuste de mínimos cuadrados (ahora no se tiene la restricción de monotonía creciente) es:

$$P_{amd}(u) = -10^{-5} \cdot u^3 - 0{,}0011 \cdot u^2 + 0{,}6553 \cdot u + 218{,}3$$

5.3.2.1.2 Nodos de tipo Intel

En el caso de los nodos con procesador Intel, el resultado de las pruebas es más estable:

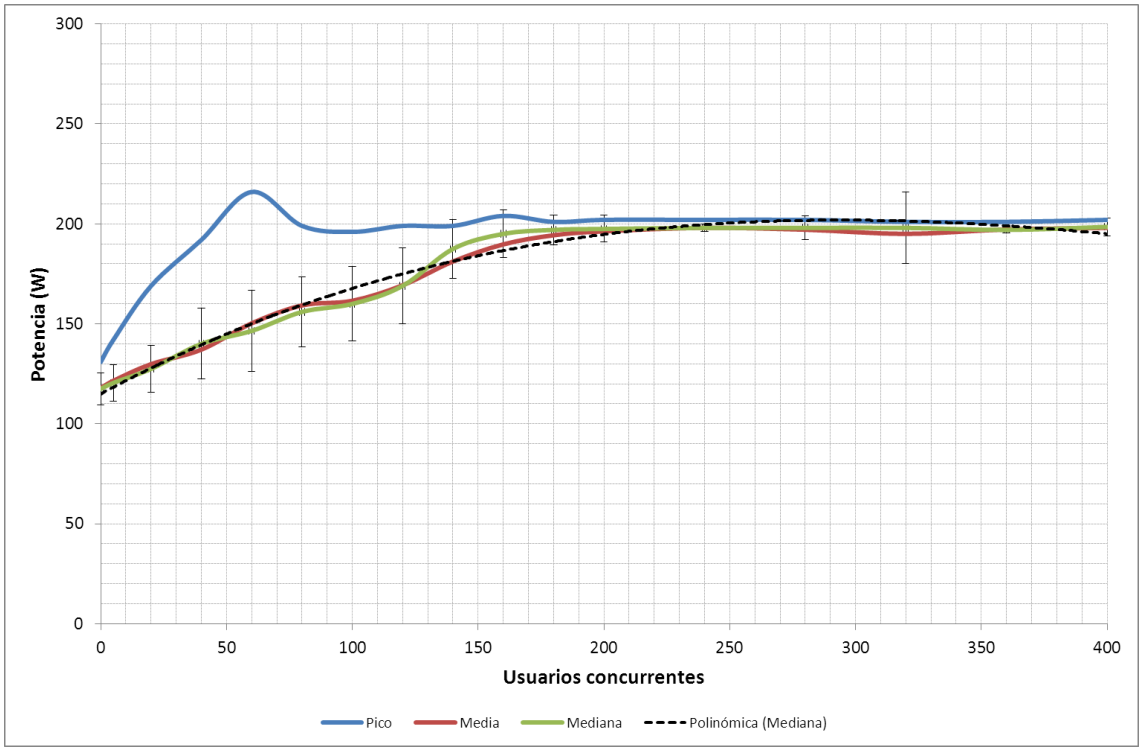


Gráfico 22: Curva de consumo empírico en los nodos Intel

Como se puede apreciar, el consumo de base (*idle power*) es más bajo que en el caso de los nodos AMD y, además, es más proporcional, incrementándose de forma lineal, hasta alcanzar el pico en el entorno de los 150 usuarios, cuando el tiempo de inactividad de la CPU se anula y el DVS deja de tener efecto. Destacan también la estabilidad de los resultados de ese punto en adelante.

El modelo obtenido, mediante un ajuste de mínimos cuadrados es el que sigue:

$$P_{intel}(u) = 10^{-6} \cdot u^3 - 0{,}0016 \cdot u^2 + 0{,}6765 \cdot u + 114{,}96$$

5.3.3 El problema de las sesiones en reflexión

Durante los experimentos de carga, se recopilaron los datos utilizando la herramienta diseñada para realizar los experimentos de carga, el inyector de carga, que permite emular una carga de trabajo de usuarios concurrentes en un servicio para medir el tiempo de respuesta en diferentes estados estacionarios de carga (niveles de carga), de forma que sea posible obtener unos modelos que representen el servidor para su autogestión.

Esta herramienta se proporcionaría como herramienta de configuración, junto con la librería de gestión energética, para su integración en un servicio existente y la posterior configuración del mismo mediante los resultados del experimento de carga.

Tal y como se observa en la sección 9.3.5, además de generar carga dinámica, el inyector de carga permite emular los tiempos de reflexión de usuarios y entre sesiones. Este tiempo representa, mediante una distribución estadística de tipo exponencial, el tiempo que pasa desde que un usuario termina una petición hasta que comienza la siguiente. Por ejemplo, en el caso de un servicio de alojamiento de sitios Web, este tiempo representa el tiempo que el usuario dedica a leer la página, mientras que en un servicio de comercio electrónico B2B, el tiempo que necesita el servidor cliente para procesar los datos. En la sección 9.3.1 se ahonda más en las implicaciones de la duración del tiempo de reflexión en la implementación práctica del prototipo.

En lo concerniente al modelado del servicio, este tiempo de reflexión genera una diferencia entre la carga inyectada con el inyector de carga y la carga efectiva que está activa en el distribuidor.

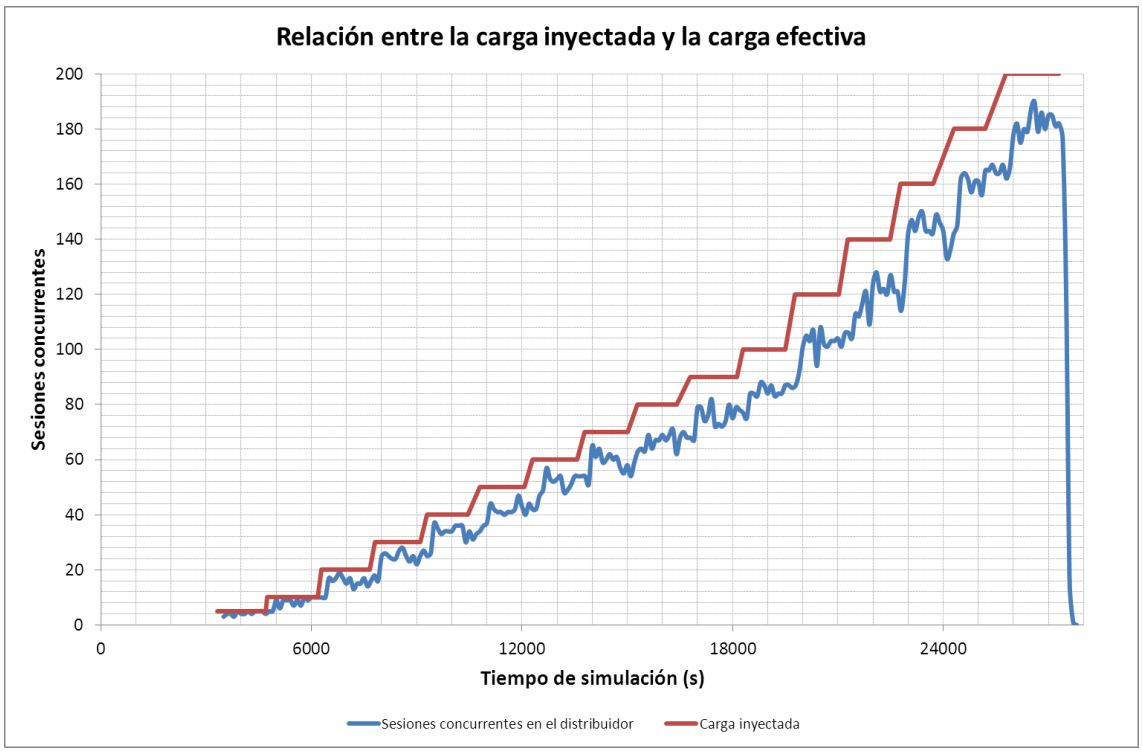


Gráfico 23: Comparación entre la carga inyectada y la carga efectiva (nodos AMD)

Tal y como se observa en el Gráfico 23, la diferencia es proporcional al número de sesiones inyectadas. La carga inyectada con el inyector, indica el número de clientes activos en cada

momento, estén realizando peticiones de forma activa o emulando un tiempo de reflexión entre transacciones mediante un bloqueo del hilo que emula el cliente.

La carga activa en el distribuidor indica las sesiones activas en cada instante, de forma que los clientes que se encuentren reflexionando entre sesiones no tendrán una sesión abierta y, por lo tanto, el distribuidor no tendrá forma de saber cuántos clientes existen, sino que tan sólo podrá conocer las sesiones que tiene abiertas.

Las implicaciones de esta diferencia se hacen patentes al desplegar los modelos obtenidos con datos procedentes del inyector de carga (que incluye las sesiones en reflexión en la cuenta de carga) en el distribuidor, que no tiene forma de conocer la totalidad de los clientes existentes (generalmente, esto nunca se conocerá).

Dependiendo de cómo se hayan realizado los experimentos de carga, esta diferencia puede existir o no (por ejemplo, si se instrumenta el inyector de forma que descarte las sesiones en reflexión, operación que, en el caso del prototipo implementado no es trivial al introducir bloqueos entre los hilos emuladores de clientes).

Para aportar la máxima flexibilidad, se ha extendido el método de cálculo de capacidades del distribuidor de forma que permite establecer un factor multiplicador que representa la proporción de sesiones activas en relación a la carga inyectada, esto es:

$$m_i = \frac{N_{distribuidor,i}}{N_{inyector,i}}$$

Este coeficiente m permite representar esta diferencia de forma sencilla. Si los experimentos en el inyector tienen en cuenta la carga activa, establecer este factor a su valor por defecto, 1, basta para desactivarlo en la descripción del servidor.

En el caso de que no sea así, se puede calcular la diferencia para cada nivel de carga, o para cada instante y realizar un promedio. En el caso de los experimentos con los nodos AMD se obtuvo un promedio para m de 0,89, con una desviación típica de 0,012, lo que demuestra la estabilidad de este parámetro. Para los nodos Intel, se obtuvo un resultado de 0,91 con una desviación de 0,009.

Este resultado sólo depende del tamaño de tiempo de reflexión establecido en la emulación, de forma que a mayor tiempo de reflexión, la proporción m es más pequeña.

Sin embargo, de cara a modelar un sistema para una realizar una validación completa del algoritmo, o bien se comprueba la validez de este este parámetro formalmente o bien se permite realizar los experimentos de modelado de forma estable.

Por este motivo, se ha extendido el inyector de carga para permitir modificar el tiempo de reflexión entre sesiones y permitir un modelado del mismo sin este problema. Adicionalmente, y por flexibilidad, se ofrece la posibilidad de configurar este parámetro.

Todos y cada uno de los modelos obtenidos anteriormente y los experimentos de validación del algoritmo no padecen de este problema, puesto que se ha configurado correctamente los experimentos de carga, de forma que el distribuidor cuenta con modelos estables y coherentes con la carga realmente soportada en cada momento.

6 OPTIMIZACIÓN Y ASPECTOS AVANZADOS

En la presente sección se realiza un análisis detallado de puntos específicos del algoritmo de gestión autonómica del clúster que aún no se han introducido y analizado, pero que son clave para comprender el funcionamiento en detalle del mismo.

6.1 Encendido y apagado físico

Para la correcta operación del algoritmo, es necesario implementar métodos de encendido y apagado físico que permitan manejar los nodos del clúster de forma remota por el nodo distribuidor de carga.

Es deseable que en este aspecto, el acoplamiento sea mínimo y permita modificar el estado físico de los nodos de forma remota mediante protocolos estándar de forma que no haya que implantar procesos y métodos concretos en el servidor de trabajo que incrementen los puntos de integración.

Con las tecnologías *Wake on LAN* y *WMI Remoting* (para servidores Windows) se puede conseguir este objetivo de forma sencilla en el gestor de energía.

El encendido y apagado frecuente de las máquinas acelera su desgaste, por lo que es necesario minimizarlo. El algoritmo es consciente de ello y minimiza los cambios de estado, pero en el caso de un clúster homogéneo, degenera en un algoritmo de "encontrar el primero", dado que los parámetros de consumo energético de los nodos son idénticos.

Por este motivo, es necesario establecer ciertos bloqueos temporales en las decisiones del algoritmo nada más modificado el estado de una de las máquinas y evitar su re-elección en un periodo de tiempo corto.

6.2 Temporización del algoritmo

Para la correcta operación del algoritmo, es necesario establecer temporizadores que controlen los estados transitorios donde existe un potencial fallo en las máquinas que están siendo controladas. En este caso, los puntos de temporización son los encendidos y los apagados.

Principalmente, si durante un encendido de un nodo el distribuidor no recibe una notificación de la disponibilidad de un nodo o bien no ha sido capaz de contactar con él se marca como estropeado y no se utiliza. Para el apagado, y con el objetivo de asegurarlo, no se puede confiar en los mensajes, ya que el nodo puede bloquearse tras enviar el mensaje, por lo que se recurre a un temporizador configurado de tal forma que exceda el tiempo real de apagado.

De esta forma, con el temporizador de apagado, no se requiere intervención del propio nodo y se minimiza el impacto en el software. Hay que tener en cuenta que el servicio transaccional debe ser capaz de manejar apagados y/o suspensiones sin corromper los datos.

Cuando un nodo se apaga o enciende, es conveniente mantenerlo en ese estado durante un cierto intervalo de tiempo. Para ello, se puede recurrir a temporizadores locales para evitar seleccionar estos nodos como los candidatos a ser modificados.

6.3 Umbrales de post-apagado

En las secciones anteriores (fundamentalmente en la sección 4.3.4) se hace referencia a los umbrales de funcionamiento del algoritmo, sus dos únicos parámetros, U_{max} y U_{min}.

El parámetro de encendido U_{max} es directo y se encuentra comentado por completo en la sección 4.3.4. Sin embargo, el parámetro de post-apagado tiene un comportamiento diferente.

El umbral U_{min} representa la utilización que cada nodo del clúster debería de tener para que se apague un nodo tras apagar dicho nodo. Esto es, se realiza la comparación de cada una de las utilizaciones de los nodos tal y cómo deberían de quedar después de apagar el nodo candidato y realizar el balanceo de carga.

Esta modificación del comportamiento de apagado se debe al problema que se detecta al realizar un análisis de las diferencias entre los umbrales de encendido y pre-apagado (el umbral de pre-apagado, aunque no se utilice en el algoritmo, es el equivalente al de encendido).

En primer lugar, si se establecen umbrales iguales, se tiene el problema de la desaparición de la histéresis en el algoritmo, entrando en un ciclo estacionario de encendido y apagado infinito.

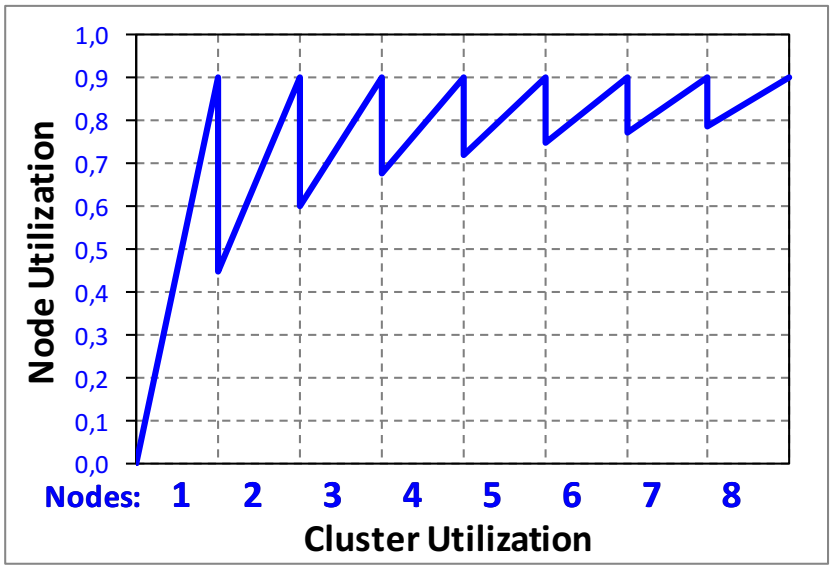


Gráfico 24: Problema de umbrales iguales

En el Gráfico 24 se puede comprobar cómo al establecer los dos umbrales iguales (de encendido y pre-apagado) se entra en un círculo de encendido y apagado infinito, pues en cuanto un nodo alcanza la utilización de 0,9 (en el ejemplo) se encenderá un nuevo nodo. Pero además, tras el cierre de una sesión se apagará de nuevo, ya que la utilización pasa a estar por debajo del umbral de apagado. Utilizar umbrales diferentes y el uso de bloqueos de estados permite mitigar el problema, como se muestra en la **¡Error! No se encuentra el origen de la referencia.**.

Una diferencia grande en los umbrales reduce la probabilidad de reinicios, pero incrementa la capacidad desaprovechada del clúster.

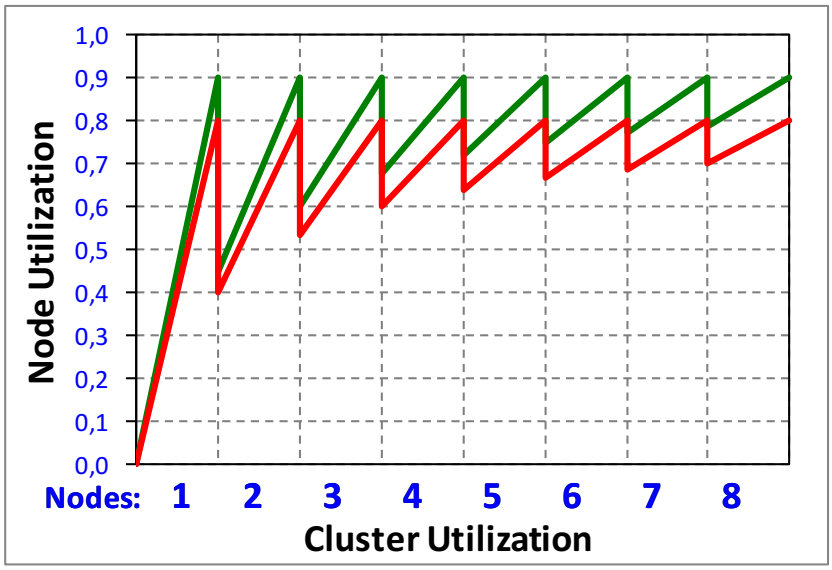


Gráfico 25: Comportamiento con umbrales diferentes

No obstante, aunque los umbrales sean diferentes, el umbral de pre-apagado no es siempre el mismo, y depende de la cantidad de nodos que estén encendidos, tal y como se ve en el Gráfico 25.

Tal y como puede verse en el Gráfico 25, la aportación a la capacidad global del clúster de cada nodo es proporcionalmente menor cuantos más nodos están encendidos, de esta forma, para eliminar el problema del ciclo de apagado sin fin, habría que establecer un umbral de apagado estático inferior a la utilización esperada para un número de nodos mínimo. En el caso de una configuración de 2 nodos equivaldría a establecer un umbral de apagado inferior a 0,45. Esto eliminaría el problema del ciclo de apagados, pero volvería al algoritmo extremadamente ineficiente en situaciones de descenso de carga.

La clave para resolver el problema de forma eficaz es darse cuenta de que el valor del umbral de apagado no es constante, sino que depende del número de nodos encendidos en cada instante. Es más, en situaciones de clústeres heterogéneos, depende de las capacidades de la configuración instantánea del clúster.

Utilizar una curva para el umbral de apagado complica la parametrización del algoritmo, por lo que se ha decido incluir el *umbral dinámico de post apagado*.

Cuando se cierra una sesión se calcula, en base al umbral de post-apagado configurado en el algoritmo, el umbral de apagado dinámico, que depende también del número de nodos encendidos (en la situación de un clúster homogéneo como el de ejemplo), de forma que:

$$U_{din} = \frac{N-1}{N} \cdot U_{min}$$

Una vez calculado el umbral dinámico, se conoce cuál es la utilización que tienen que tener todos los nodos (estar por debajo de la misma) para apagar uno de los nodos. A continuación, se escoge cuál nodo apagar mediante la estimación de consumo.

Hay que hacer notar que el valor de $U_{mín}$ ha de ser estrictamente inferior al valor de $U_{máx}$. De tener valores iguales, se estaría en el valor límite del umbral dinámico que, dependiendo de la implementación del algoritmo (del redondeo establecido en el cálculo del mismo) podría derivar en ciclos de apagado-encendido (si se redondea hacia arriba al calcular U_{din}) o decisiones incoherentes, como utilizaciones por debajo de 0 (si se redondea hacia abajo).

Utilizar este modelo de umbrales calculados de forma dinámica se hace necesario debido a que cada nodo no tiene la misma contribución en términos de utilización en diferentes situaciones del clúster (con más o menos nodos encendidos). De esta forma, si se establece un valor estático del umbral de apagado, se pueden dar situaciones, por ejemplo, con cargas altas, en las que se apaguen nodos con mucha más probabilidad que con cargas bajas, de forma que se multiplican los rebotes en apagado.

Es necesario remarcar cómo, en entornos heterogéneos, la función de cálculo del umbral de apagado dinámico ha de calcularse utilizando las capacidades totales en la situación actual y esperada tras el balanceo, debido a que, como se comentaba anteriormente, la contribución de cada nodo no es siempre la misma en términos de número de sesiones soportables. Sin embargo, dicha fórmula puede usarse como una buena aproximación en el caso en el que el cálculo de la capacidad de cada nodo sea complejo, de forma que se optimize la velocidad de decisión del algoritmo operando en tiempo real.

7 PRUEBAS Y RESULTADOS

En esta sección se realiza la planificación y análisis de los resultados de las pruebas. Fundamentalmente, el objetivo de las mismas no es otro que validar el algoritmo de optimización en línea presentado en la sección 4 y cuantificar la capacidad de optimización del mismo.

7.1 Pruebas de validación del software

Las pruebas de validación del software son un paso previo a la validación del algoritmo y tienen como objetivo probar que el prototipo implementado se comporta de forma estable y no presenta errores de programación.

Debido a la reducida cantidad de recursos destinados a la implementación, se ha decido automatizar este proceso mediante pruebas estáticas y dinámicas sobre el código con el objetivo de reducir el número de experimentos dada la duración temporal que tienen (en el entorno de las 8-12 horas cada uno).

Las pruebas estáticas se han centrado en análisis sobre el código usando herramientas de análisis estático de código, en este caso *PREFAST*, herramienta proporcionada por Microsoft para el análisis de aplicaciones *Win32* escritas en C++. *Astyle* ha sido la herramienta escogida para mantener el código en buen estado, además de realizar revisiones del mismo en el grupo de investigación.

Las pruebas dinámicas incluyen la instrumentación del código con el perfilador *Intel Thread Profiler* para *Win32* y la ejecución de los experimentos con código instrumentado con asertos además de la utilización de métricas de rendimiento proporcionadas por el monitor de rendimiento integrado de *Windows Server*.

De esta forma se ha podido validar el prototipo de forma rápida y sencilla, requiriendo pocos recursos para ello. No obstante, para una implementación no prototípica del algoritmo, una fase de planificación, ejecución y análisis de pruebas ha de ser llevada a cabo.

7.2 Pruebas de validación del algoritmo

Las pruebas de validación del algoritmo tienen como objetivo conocer y ajustar el algoritmo para maximizar la optimización energética que es capaz de llevar a cabo. En este sentido, es necesario conocer cómo se comporta en entornos heterogéneos y homogéneos y que influencia tienen los parámetros que lo definen.

Para ello, se ha desplegado un experimento en los nodos de tipo AMD con el fin de validar el comportamiento general del algoritmo en una situación de carga creciente y decreciente. Se han establecido unos umbrales de $U_{max} = 0{,}9$ y $U_{min} = 0{,}8$ de forma tentativa, para luego ser ajustados en una serie de experimentos posteriores y comprobar la influencia de las variaciones de los mismos en la sección 7.3.

Los gráficos que representan los resultados de las pruebas son los que siguen.

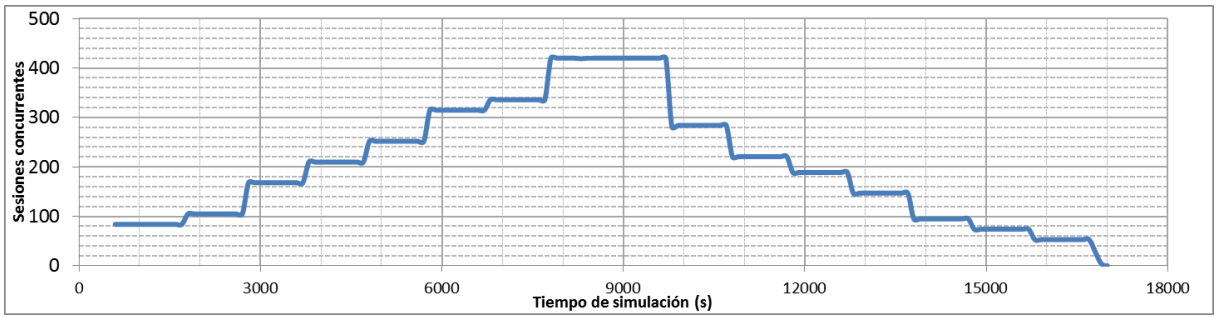


Gráfico 26: Carga soportada en el distribuidor durante el experimento de validación

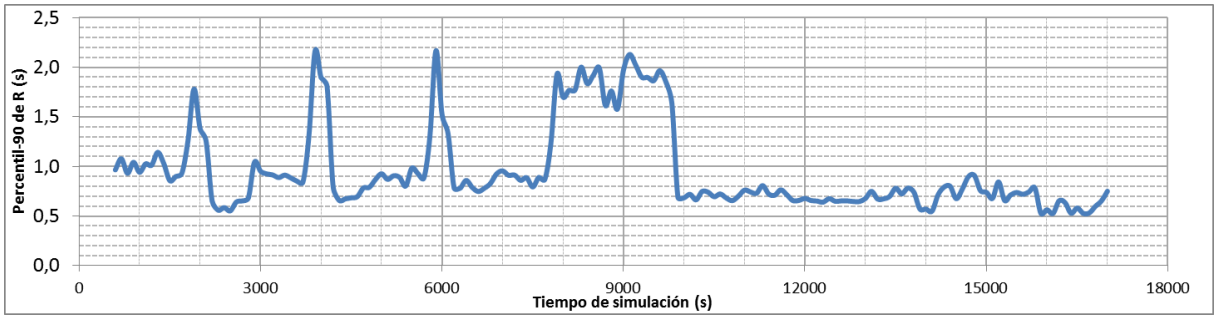


Gráfico 27: Percentil-90 del tiempo de respuesta durante el experimento de validación

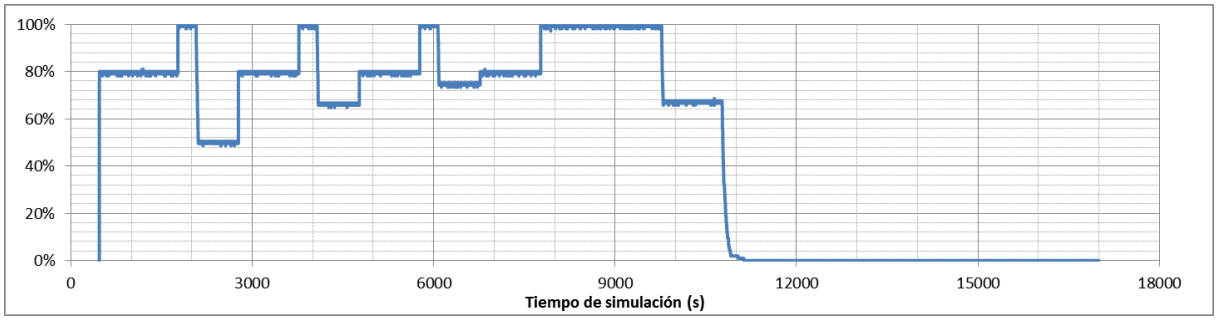


Gráfico 28: Utilización del nodo ATC182 durante el experimento de validación

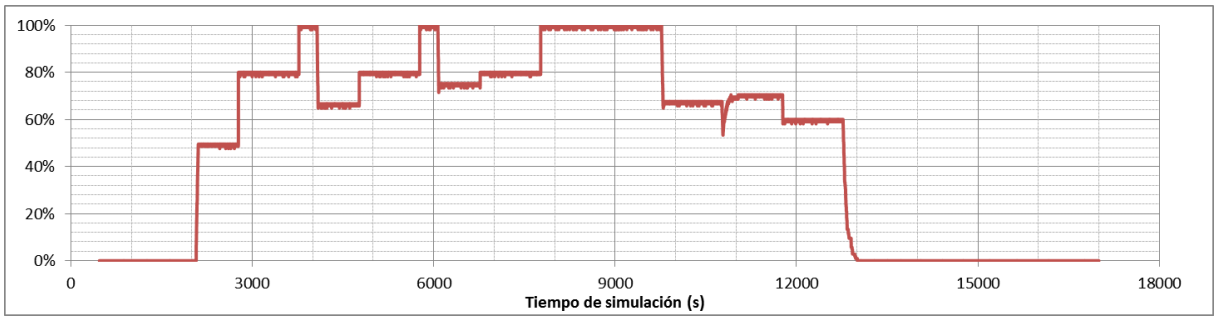


Gráfico 29: Utilización del nodo ATC183 durante el experimento de validación

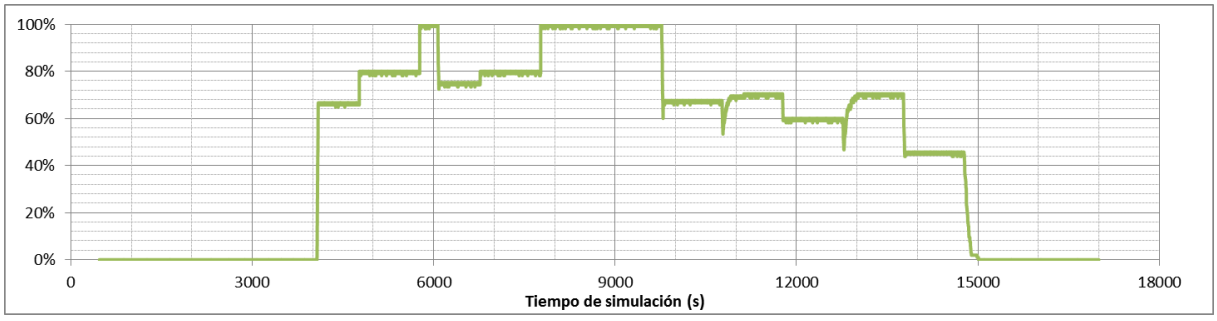


Gráfico 30: Utilización del nodo ATC184 durante el experimento de validación

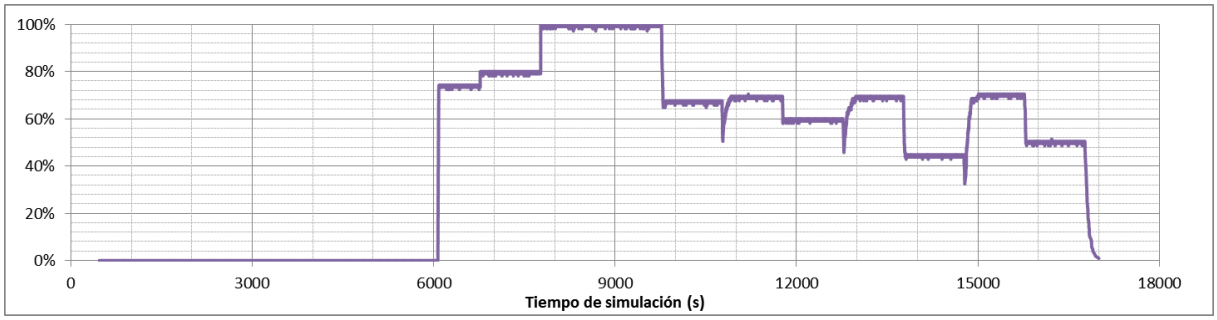


Gráfico 31: Utilización del nodo ATC185 durante el experimento de validación

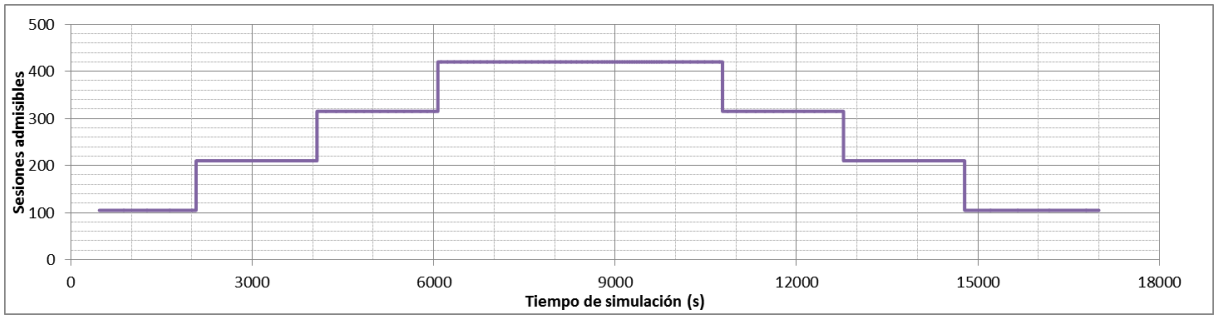


Gráfico 32: Capacidad combinada del clúster durante el experimento de validación

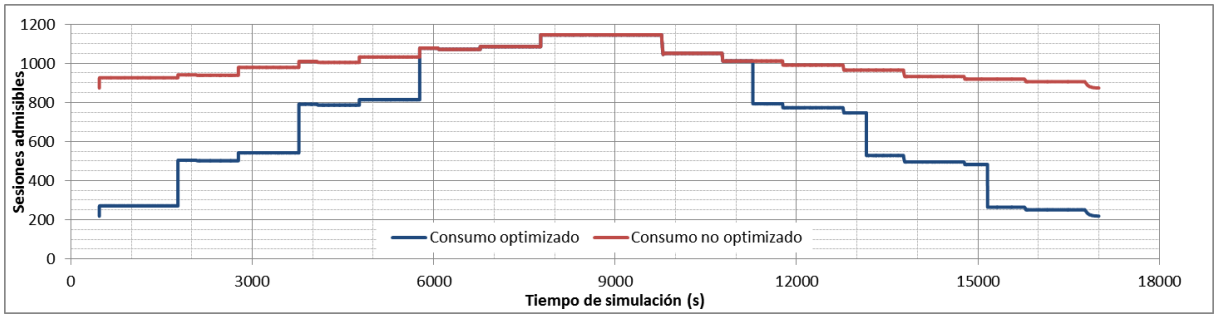


Gráfico 33: Consumo optimizado y no optimizado durante el experimento de validación

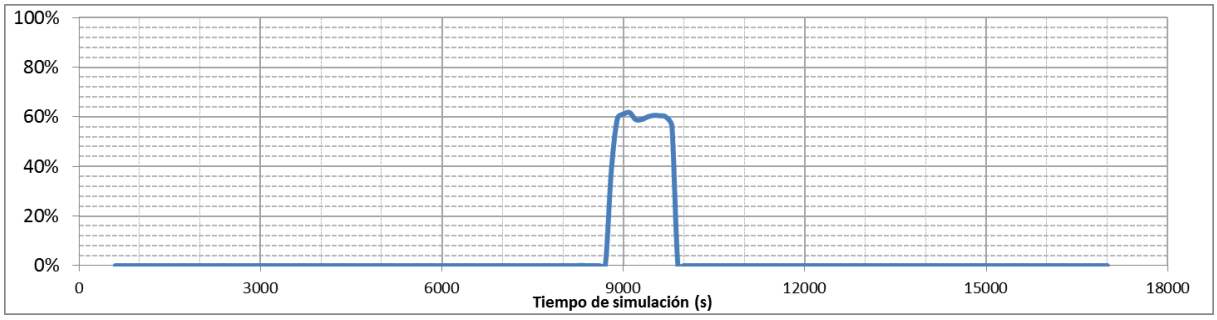


Gráfico 34: Tasa de rechazo de sesiones durante el experimento de validación

Los primeros resultados a destacar corresponden a la utilización de los nodos utilizados para el experimento, en este caso cuatro nodos de tipo AMD. Tal y como se observa en el Gráfico 28, el primero de los nodos arranca encendido y se mantiene así hasta que el algoritmo decide apagarlo.

Al respecto del Gráfico 28, el Gráfico 29, el Gráfico 30 y el Gráfico 31, hay que destacar los siguientes puntos de interés, observando los gráficos de forma combinada:

1. El primero de los nodos comienza encendido, con una utilización inicial aproximada del 80%.
2. A continuación, en el segundo 1800 de simulación, se produce un incremento de carga, que hace que la utilización del primer nodo supere el 90%, motivo por el cual el algoritmo decide encender el segundo nodo. Mientras se completa el encendido y se produce el rebalanceo de carga, la utilización del primer nodo continua siendo alta y, por ese motivo, se produce un incremento en el tiempo de respuesta del servicio, pero sin embargo, no se produce ningún rechazo de sesión, tal y como se observa en el Gráfico 34. Una vez se ha completado el balanceo de carga, los nodos cuentan con una utilización promedio del 45%.
3. En segundo 2700 se produce el siguiente incremento de carga; el clúster, configurado con dos nodos, vuelve a alcanzar una utilización combinada del 80%, lo que hace que se esté operando a un nivel alto de capacidad, pero no hace necesario encender un nuevo nodo, pues no se supera ningún umbral.
4. Es en el segundo, 3600, al incrementar la carga de nuevo, los nodos que están encendidos superan el umbral de encendido, por lo que el algoritmo decide encender el tercer nodo. En este caso, tras el balanceo de carga, todos los nodos quedan con una utilización promedio del 66%, tal y como se demostraba teóricamente en la sección 4.3.4.
5. El siguiente incremento de carga, en el segundo 4800, vuelve a poner a los tres nodos en utilizaciones cercanas al 80%, que al no superar ningún umbral, hace que el clúster opere en un nivel de alta eficiencia.
6. Cuando se produce el siguiente incremento de carga, en el segundo 5600, se enciende el cuarto y último nodo disponible, dejando, tras el encendido y balanceo, todos los nodos del clúster en un 75% de utilización.
7. A continuación, desde el segundo 7800, se incrementa la carga hasta un nivel cercano a la capacidad máxima del clúster, de forma que se opera por encima de los umbrales, pero no se enciende ningún nodo adicional, ya que no hay más nodos disponibles.
8. Por último en la parte de carga ascendente del experimento, desde el segundo 8400, se solicita al clúster más capacidad de la que puede ofrecer con el objetivo de probar el

control de admisión y la garantía del SLA en situaciones de saturación. Tal y como se observa en el Gráfico 34, se produce un alto nivel de rechazos, debido a que el clúster no dispone de capacidad suficiente para soportar todas las sesiones solicitadas sin violar la garantía del SLA. Dicha garantía puede observarse en el Gráfico 27, que muestra cómo el percentil del tiempo de respuesta (capturado en línea en el distribuidor) se mantiene generalmente por debajo del límite establecido de 2 segundos, con unos picos en los momentos de alta solicitación al clúster, que coinciden con los momentos de cambio de estado (al esperar que se complete el encendido de nuevos nodos) y de sobrecapacidad, que, a su vez, coinciden con los momentos de rechazo de sesiones.

9. A continuación sigue el período de disminución de carga del experimento. En este sentido, cabe destacar que se ha establecido el umbral de post-apagado en un valor de 0,8 de forma tentativa, con el fin de comprobar el comportamiento del algoritmo de forma clara. Un mejor ajuste se realizará en la sección 7.3.3, puesto que la eficiencia del algoritmo depende en gran medida de un valor alto de ambos umbrales. Tal y como se observa, al reducir la carga al 66% en el segundo 9.800, no se produce ningún apagado de nodos debido a que, aunque la carga instantánea está por debajo del umbral, la utilización tras el apagado de uno de los nodos, no lo estaría (si se apaga un nodo, hay que balancear las sesiones existentes entre 3 nodos en lugar de 4, lo que hace que la utilización de post-apagado sea superior al 80% con 3 nodos operativos).
10. En este sentido, cuando la carga baja por debajo del 45%, en el segundo 11.700, sí que se decide apagar uno de los nodos, ya que la utilización tras el apagado del mismo es inferior al 80%. Tras el apagado, se produce un balanceo de carga, para dejar los tres nodos restantes al 66% de utilización. Este proceso se repite hasta que se vacía el clúster, quedando un solo nodo encendido. Hay que hacer notar cómo el apagado tiene como objetivo concentrar la carga en conjuntos de nodos más pequeños, de forma que la utilización de los mismos sube, y por tanto la eficiencia en la utilización de la energía.

Los resultados en términos de consumo pueden observarse en el Gráfico 33, que compara el consumo instantáneo (en función de la carga) del clúster operando con el algoritmo de optimización y operando sin él. Las dos series de la gráfica comparan el consumo utilizando la provisión dinámica y no utilizándola, de forma que las variaciones en la energía consumida cuando no se está usando provisión dinámica se deben al *DVS* de los procesadores.

Como se puede comprobar, el DVS sólo es responsable de un ahorro del 18% respecto al consumo de pico, mientras que el algoritmo de provisión dinámica permite alcanzar ahorros de hasta el 85% de energía total del clúster.

Dichos ahorros, son en promedio del 12% respecto al consumo obtenido para el mismo experimento con una configuración sin provisión dinámica de nodos, lo que permite ser optimista para las configuraciones más ajustadas de los umbrales de encendido y apagado.

7.3 Dependencia y ajuste de parámetros

En esta sección se va a proceder a comprobar la influencia de los umbrales en el comportamiento del algoritmo y a realizar un ajuste de los mismos en base a métricas experimentales.

Para las pruebas realizadas en entornos homogéneos se ha optado por realizarlas con una configuración de cuatro nodos trabajadores de tipo AMD (ver sección 10). El objetivo de estas pruebas es el ajuste los parámetros $U_{máx}$ y $U_{mín}$ que definen el algorimto y conocer la interacción que existe entre ellos.

7.3.1 Configuración experimental

Para estos experimentos se ha diseñado una configuración de cuatro nodos trabajadores AMD y dos nodos que actúan como inyector y distribuidor de carga, que son de otro tipo, Intel. Un esquema de esta configuración se ha mostrado en la Figura 12 en la sección 4.5.

Para los experimentos se ha decido utilizar la curva de carga en "pirámide" comúnmente utilizada para la validación de este tipo de algoritmos.

Para la configuración de los experimentos que permitan ajustar el parámetro $U_{máx}$ se usará una carga creciente hasta la capacidad del clúster, además de establecer un tiempo de sobresolicitación, que permita ejercitar el control de admisión, con el fin de comprobar la respuesta en base al SLA.

Para los experimentos de apagado, se ha decido usar la parte de carga decreciente de la curva de carga piramidal.

7.3.2 Ajuste inicial del parámetro $U_{máx}$

El objetivo del ajuste del parámetro $U_{máx}$del algoritmo es conocer qué influencia tiene en el aprovechamiento de los recursos de computación. Establecer un valor muy cercano a 1,0 para este parámetro permite tener un alto aprovechamiento de los recursos, pues sólo hasta que se llegue a una utilización muy alta de éstos no se encenderá un nuevo nodo.

Para el ajuste de este parámetro, la parte de interés de la curva de carga es la parte ascendente, donde se produce el encendido de los nodos.

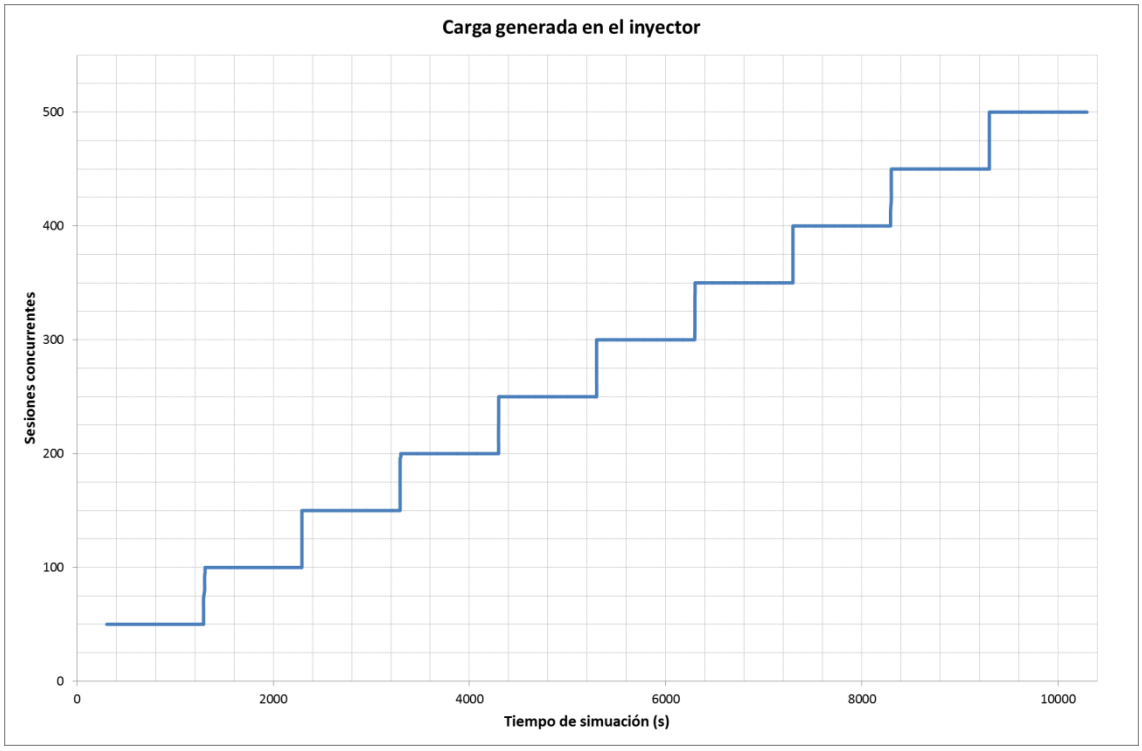


Gráfico 35: Carga inyectada para el ajuste de Umax en los nodos AMD

Sin embargo, tener un valor muy alto de $U_{máx}$ puede redundar en rechazos temporales de sesiones mientras el nuevo nodo a encender realiza su arranque. Por esto, el objetivo de estas pruebas es comprobar el impacto que establecer un valor alto de $U_{máx}$ puede tener en lo relativo al rechazo temporal de sesiones.

Fundamentalmente, el rechazo de sesiones temporalmente durante el encendido de un nodo se debe a la superación de los umbrales de utilización de forma transitoria mientras se enciende el nuevo nodo. De esta forma, los parámetros que definen el posible rechazo de sesiones de forma transitoria son:

- **El propio umbral $U_{máx}$**: como ya se ha indicado, un valor alto de este umbral incrementa la probabilidad de un rechazo de sesión durante el tiempo de encendido.
- **El tiempo de encendido del nodo:** si los nodos tienen un tiempo de encendido largo, se prolonga el período transitorio durante el cual la utilización ha superado el umbral especificado. De esta forma, este factor es dependiente del hardware con el que se cuenta y podrá ser reducido usando estados de apagado intermedios, que reducen el tiempo de encendido, pero incrementan el consumo energético de los nodos no operativos. En los equipos de pruebas este tiempo de aproximadamente 5 minutos.
- **La forma de la carga:** es fundamental tener en cuenta que la forma de la carga determina en gran medida un rechazo de sesiones transitorio. El peor de los casos es una carga con forma ascendente, ya que los nodos estarán ganando utilización constantemente (esta es la situación probada en el experimento).
- **La capacidad total de los nodos actualmente encendidos:** cuantos más nodos haya encendidos en el momento actual (mientras se enciende el nuevo nodo), mayor será la capacidad restante desde que supera el umbral hasta el total y, por tanto, mayor será la capacidad de respuesta, hasta que la capacidad del nuevo nodo es agregada al conjunto de nodos operativos.

Por tanto, el objetivo de este ajuste será relacionar el valor del umbral $U_{máx}$ con la tasa de rechazos media producida en situaciones de solicitación dentro de la capacidad admisible del clúster. En situaciones de sobresolicitación, se producirá un rechazo de sesiones necesario para el mantenimiento de la calidad de servicio (al igual que en los transitorios), pero nada podrá hacerse, pues todos los recursos del clúster estarán habilitados para el servicio.

De esta forma, la carga realmente soportada en el clúster para las pruebas realizadas (con cuatro nodos de 105 usuarios de capacidad) es la que se presenta en el Gráfico 36.

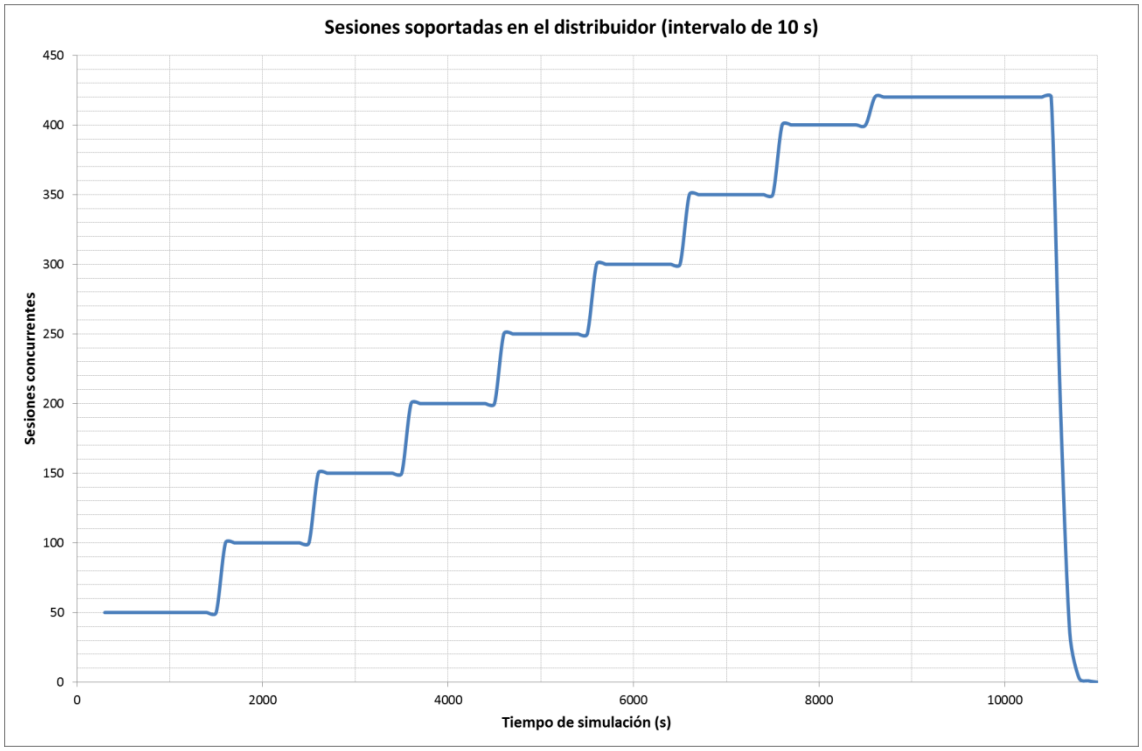


Gráfico 36: Carga soportada durante las pruebas de ajuste de Umax

Como era de esperar, se soporta toda la carga solicitada, a excepción de en la última parte del experimento, donde se produce el control de admisión.

Para esta validación de han configurado experimentos variando el valor de $U_{máx}$ dentro del intervalo $[0{,}5 - 1{,}0]$, de forma que se comprueba el impacto de este valor en la tasa promedio de rechazo de sesiones.

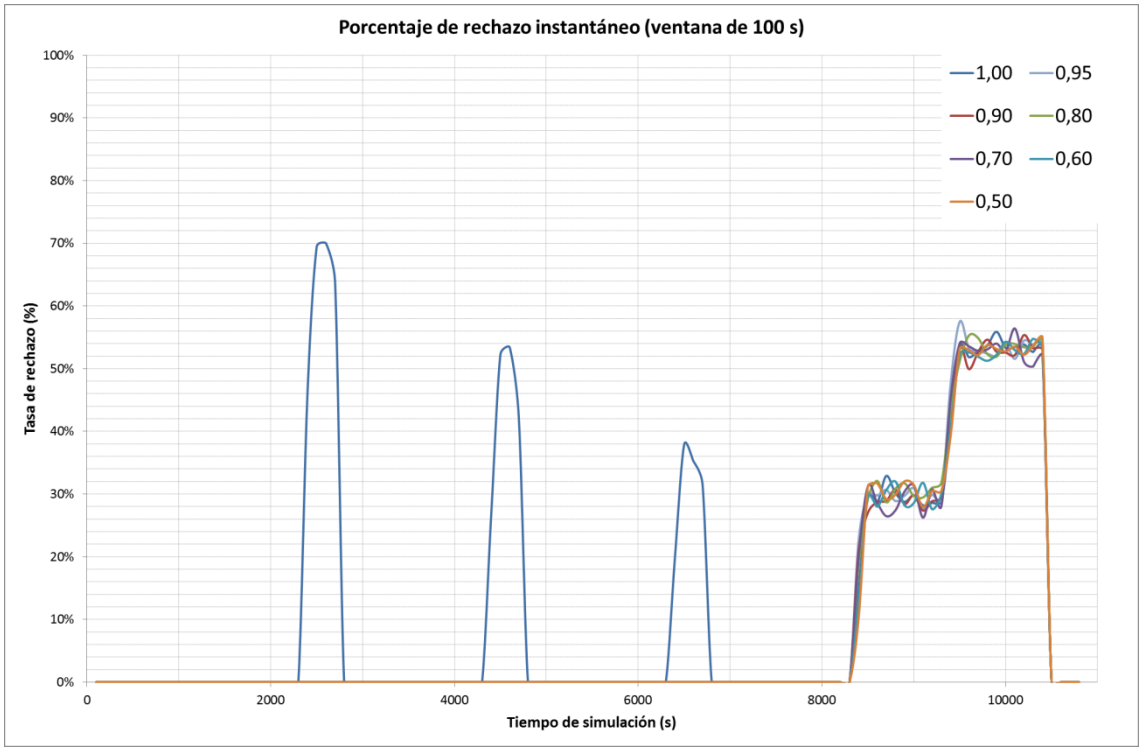


Gráfico 37: Comparación de tasas de rechazo para varios valores de Umax

Tal y como se observa en el Gráfico 37, existen dos momentos reseñables. En primer lugar, sea cual sea el valor de $U_{máx}$, el algoritmo aplica control de admisión de igual forma cuando se está en una zona de sobresolicitación clara, como al final del experimento (desde el segundo 8400).

El otro comportamiento se puede observar en los tres picos de rechazos cuando $U_{máx}$ se establece al valor máximo de 1.0. Estos picos (en los segundos 2300, 4400 y 6300 respectivamente) se corresponden con los momentos en los que el algoritmo ha decido encender uno de los nodos y el clúster está a la espera de recibir la nueva capacidad (debido a que las decisiones de encendido están bloqueadas durante el período de encendido de un nodo, ver sección 4.3.1.1).

Estos picos se producen porque el algoritmo, con un valor de $U_{máx}$ de 1,0 sólo va a ordenar encender un nuevo nodo cuando el conjunto actual está completamente utilizado, de forma que mientras se procede al encendido, se va a estar en situación de sobrecapacidad de forma segura. Por esto, es recomendable establecer el valor de $U_{máx}$ a un valor estrictamente menor que 1.0. Pero ya que este valor determina la eficiencia del clúster habrá que tratar de hacerlo lo más grande posible.

El que los picos tengan una tendencia decreciente se debe a que cada vez hay más nodos encendidos y, por tanto, la capacidad del clúster es mayor, de forma que la capacidad restante

tras superar el umbral es también mayor y se tiene más margen para esperar por el nodo a encender.

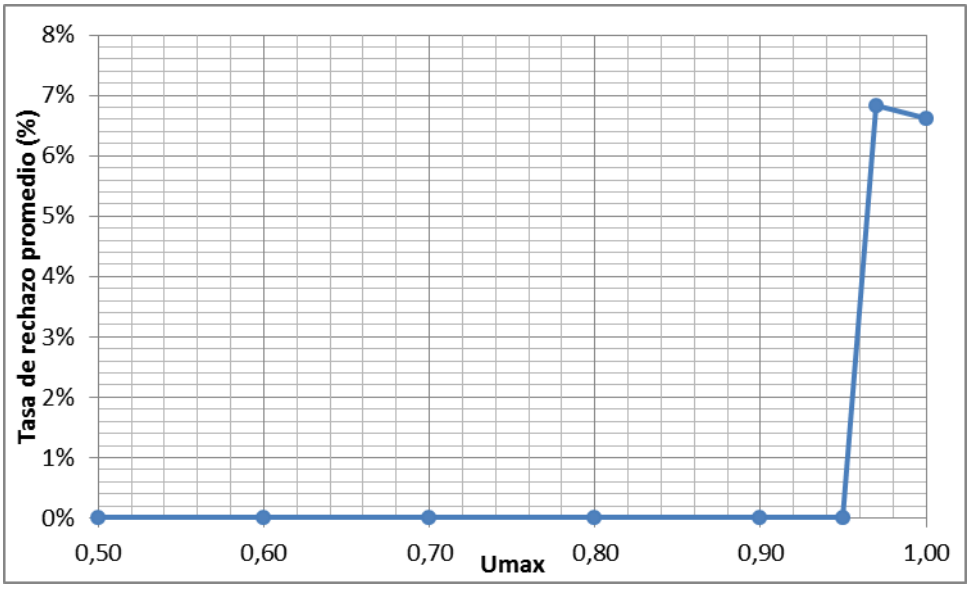


Gráfico 38: Relación entre el valor de Umax y la tasa de rechazos

Tal y como se puede observar en el Gráfico 38, la tasa de rechazos se mantiene a cero hasta que se alcanza el valor de 0,97 (para esta configuración de nodos).

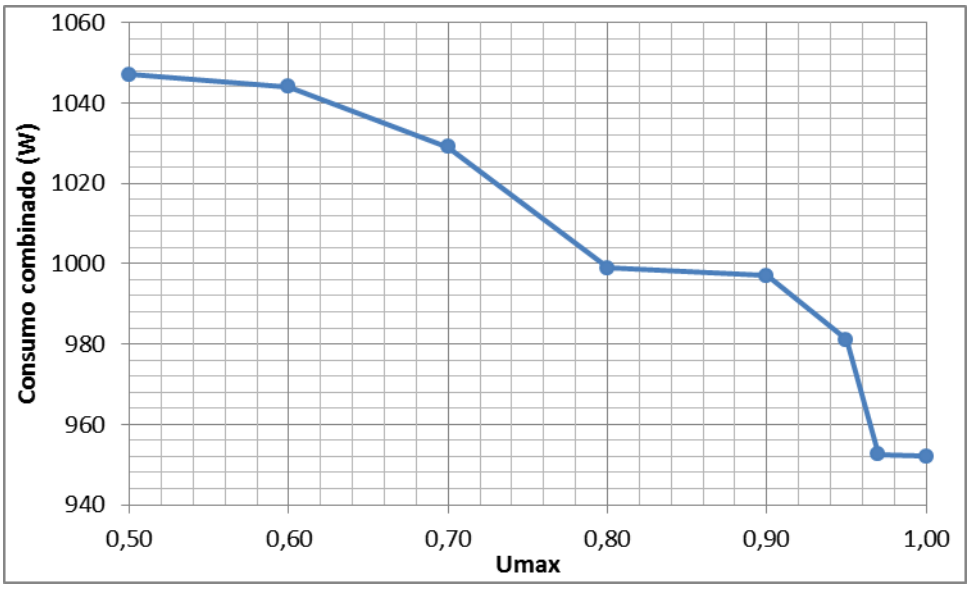


Gráfico 39: Relación entre el valor de Umax y el consumo energético combinado

En el Gráfico 39 puede observarse como, conforme el valor de $U_{máx}$ se incrementa, el consumo energético combinado medio para la prueba desciende. Esto es normal, pues se está retardando el encendido de los nodos cuanto más alto se establece el valor del umbral de encendido.

Para el cálculo del consumo energético se utilizan los modelos estimados previamente (ver sección 5.3.2) de forma que el distribuidor, en base a las utilizaciones de los nodos es capaz de estimar el consumo energético de cada nodo en línea.

Nótese como este este consumo está medido en unidades de potencia y no de energía (ver sección 7.4.2 para más detalles entre la diferencia entre energía y potencia y las implicaciones en entornos heterogéneos). La potencia, no obstante es un buen indicador de la eficiencia energética conseguida, pues está directamente relacionado, siempre que se capture en línea y pueda obtenerse un consumo medio.

Para la captura de datos, se toma una muestra de la potencia actual de cada nodo en cada transacción realizada, lo que ofrece una alta resolución en la medición (los cálculos anteriores contaron con unas 150 000 muestras cada uno, espaciadas unos 500 ms). La medida de potencia mostrada es el promedio de las mediciones anteriores.

Combinando los datos con los obtenidos en el Gráfico 38, se puede tomar la decisión de establecer un valor para $U_{máx}$. En principio, dicho valor ha de ser lo más alto posible, tal y cómo se dedujo de forma teórica en la sección 4.3.4, de forma que se minimice el consumo energético del clúster retardando el encendido de los nodos lo máximo posible.

No obstante, como cota superior ha de establecerse aquel valor de $U_{máx}$ para el que comienzan los rechazos de sesiones, indicando que no hay suficiente capacidad de reserva como para permitir a los nuevos nodos encenderse sin que el conjunto actual se sature, alcanzando el límite de utilización que permite garantizar el cumplimiento del SLA.

En el caso de la configuración de pruebas, un valor para $U_{máx}$ de 0,95 sería lo más apropiado, ya que no se llega a rechazar ninguna sesión y es el valor para el cual el consumo medio de operación del clúster es, dentro del conjunto de valores para los que no se producen rechazos, mínimo (981 W).

7.3.3 Ajuste del parámetro $U_{mín}$

De una forma similar al comportamiento analizado para el parámetro $U_{máx}$, el ajuste del parámetro $U_{mín}$ permite reducir la cantidad de recursos que son desperdiciados por el gestor energético para asegurarse de que no se producen *rebotes*.

Un rebote es el apagado y encendido en un periodo de tiempo corto de una de las máquinas del clúster.

El parámetro $U_{mín}$ se define en función del umbral $U_{máx}$, puesto que ha de ser siempre menor que éste último, por lo que el siguiente análisis se definirá en función del salto entre ambos, esto es:

$$U_{mín} = U_{máx} - salto$$

El análisis comienza con un salto de 0,1 y permite analizar la reducción de los rebotes conforme evoluciona el mismo. En función de este salto, se ha decido explorar los valores de 0,45; 0,55; 0,65; 0,75 y 0,85 para el parámetro $U_{mín}$.

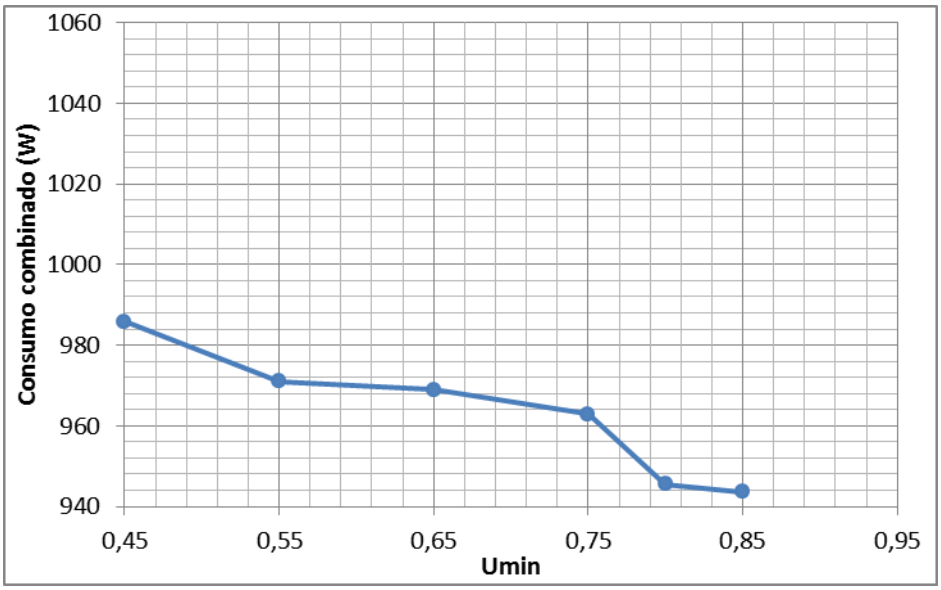


Gráfico 40: Relación entre el valor de Umin y el consumo energético combinado

Tal y como puede observarse en el Gráfico 40, que recoge los resultados en términos de consumo para los diferentes valores de $U_{mín}$, teniendo en cuenta que $U_{máx}$ se ha establecido al valor de 0,95 (tal y como se razonó en la sección 7.3.2) y que $U_{mín}$ ha de ser estrictamente menor a $U_{máx}$ (ver sección 6.3).

En los experimentos de ajuste de $U_{máx}$ se consiguió obtener una cota inferior de consumo promedio de 980 W cuando este umbral vale 0,95. Este es el punto de inicio de los resultados obtenidos para $U_{mín}$ (de forma que esto es coherente con el ajuste de $U_{máx}$) y, mediante el correcto ajuste de $U_{mín}$ se puede mejorar el consumo llegando hasta los 945 W de cota inferior si se establece el umbral de apagado en el valor de 0,85.

Sin embargo, hay que tener en cuenta los *rebotes*. Un rebote podría definirse como, en una situación de descenso de carga el seleccionar para apagar uno de los nodos y, cuando se llega a un estacionario en un tiempo *relativamente corto*[16], seleccionarlo para encenderlo.

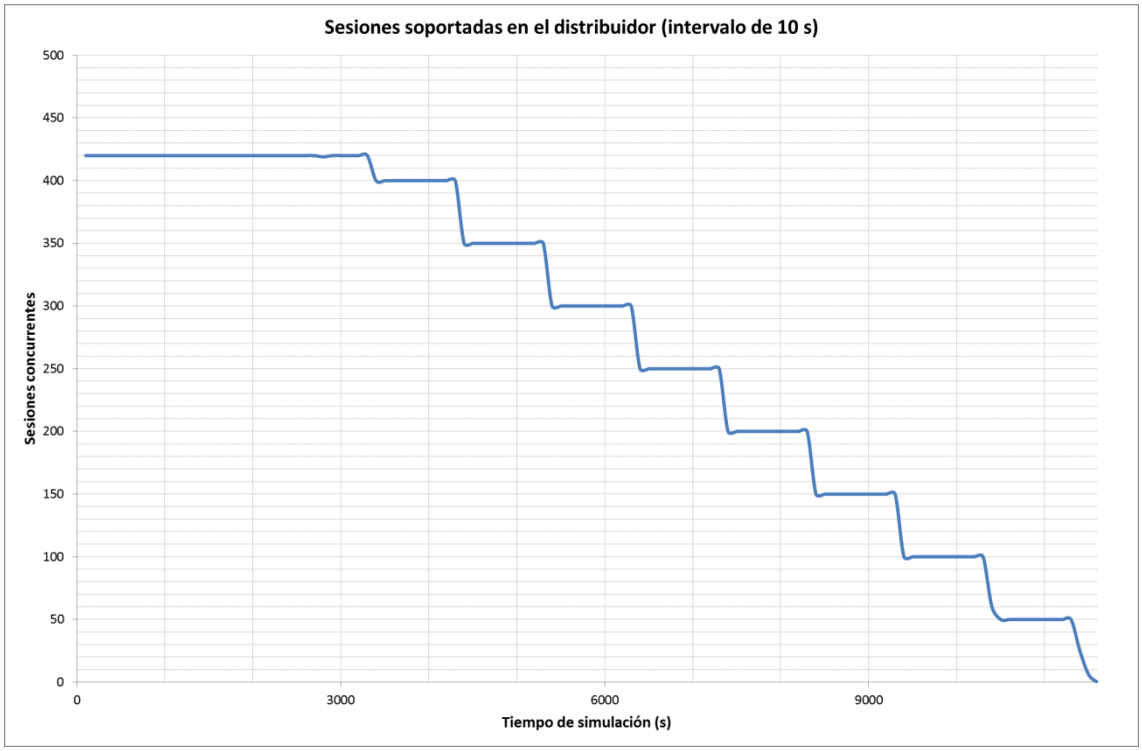


Gráfico 41: Carga soportada durante las pruebas de ajuste de Umin

En el caso de las pruebas realizadas, se han hecho con una carga decreciente con mesetas estacionarias, de forma que se ejercita el valor del parámetro $U_{mín}$ en el peor de los casos de actuación del algoritmo. Tal y como se muestra en el Gráfico 41, se comienza en una situación de sobresolicitación y se continúa bajando hasta liberar los recursos del clúster (nótese que en la figura, se muestran las sesiones soportadas y en la parte de sobresolicitación hay control de admisión, pues se está solicitando la misma carga que en el Gráfico 35, pero en orden inverso).

Tener rebotes es malo porque hace que el clúster tenga un conjunto de nodos excesivamente fluctuante y, además de las implicaciones en el deterioro del hardware por una alta frecuencia de ciclos de encendido y apagado, hace que la capacidad efectiva del clúster sea muy cambiante, por lo que es recomendable evitarlos. Sin embargo, evitar los rebotes, no es tan capital como en el caso de los rechazos de sesión ($U_{mín}$ no es responsable de rechazos de sesión, sólo de mejorar

[16] Un tiempo corto no puede superar la duración de una sesión, esto es, la resolución temporal que el algoritmo tiene para tomar decisiones.

la eficiencia y, posiblemente, si está en un valor demasiado alto, de introducir rebotes de encendido).

Un rechazo de sesión, además de perjudicar el servicio, indica una saturación y un incremento del tiempo de respuesta porque se está tardando demasiado en encender un nodo. Un rebote, simplemente[17] afecta a la eficiencia energética del servicio.

U_{min}	Rebotes
0,45	0
0,55	0
0,65	0
0,75	0
0,80	0
0,85	1

Tabla 5: Comparación de número de rebotes para varios valores de Umin

En el caso de las pruebas realizadas, con la configuración de $U_{mín} = 0{,}85$ se empiezan a notar rebotes, por lo que con una configuración de 0,8 se tiene un muy buen compromiso entre eficiencia energética y estabilidad en la operación del algoritmo.

7.4 Pruebas en entornos heterogéneos

Las pruebas en entornos heterogéneos tienen como finalidad comprobar el funcionamiento del algoritmo en las situaciones más genéricas y donde las decisiones tienen un impacto grande en la eficiencia del clúster, así como en la calidad de servicio proporcionada a los clientes.

7.4.1 Diseño experimental

En este conjunto de pruebas se han configurado todos los recursos de computación disponibles de la forma siguiente:

- El nodo distribuidor, mediante un nodo de tipo Intel.
- El nodo inyector, mediante un nodo de tipo AMD.
- Tres nodos trabajadores de baja capacidad, mediante tres nodos AMD.

[17] En este caso, tiene una importancia capital, pero es una implicación interna que no afecta a la percepción del servicio por los usuarios.

- Tres nodos trabajadores de alta capacidad y eficiencia energética, mediante tres nodos Intel.

Más allá de la configuración experimental, lo interesante es el diseño de las pruebas para comprobar la influencia de los parámetros en estas configuraciones. De esta forma se van a realizar las siguientes pruebas, teniendo en cuenta la configuración obtenida en los experimentos anteriores:

$U_{max} = 0,85$	$U_{max} = 0,90$	$U_{max} = 0,95$	$U_{max} = 1,00$
$U_{min} = 0,80$	$U_{min} = 0,85$	$U_{min} = 0,90$	$U_{min} = 0,95$
$U_{min} = 0,75$	$U_{min} = 0,80$	$U_{min} = 0,85$	$U_{min} = 0,90$
$U_{min} = 0,70$	$U_{min} = 0,75$	$U_{min} = 0,80$	$U_{min} = 0,85$

Tabla 6: Resumen de configuración de pruebas en clústeres heterogéneos

En total se van a realizar 12 experimentos, con las configuraciones definidas en la Tabla 6, de forma que se ejerciten, además de los valores previamente determinados, los valores límite a ellos para ver hasta qué punto la influencia de los parámetros es diferente en el caso de configuraciones de clústeres heterogéneas.

El objetivo principal, más allá de seleccionar un valor para los parámetros, dada la dependencia de la configuración del servicio y del hardware que tiene, es comprobar la dependencia y la influencia que el valor de estos tienen en la degradación de la calidad de servicio conforme la utilización del clúster se incrementa.

7.4.2 Estimación del consumo en línea

Uno de los problemas principales a resolver para la validación del algoritmo es la estimación del consumo energético del clúster y, además, hacerlo de forma dinámica. No sólo es interesante conocer el consumo energético del clúster, sino que también es conveniente conocer la eficiencia energética del mismo, parámetro que deberá ser optimizado.

En primer lugar, es conveniente diferenciar los términos potencia (medida en vatios, W) y energía (medida julios, J). La primera hace referencia a la tasa de utilización de la energía, mientras que la segunda es la energía consumida en total.

Al utilizar polímetros para calibrar y obtener modelos para los nodos, se han obtenido modelos de potencia, esto es, que muestran la tasa de utilización de la energía. Por tanto, para obtener métricas adecuadas de eficiencia y consumo, es necesario, convertir estas tasas en medidas de

energía. Este paso es clave para una correcta evaluación del algoritmo en entornos heterogéneos, puesto que la eficiencia energética de cada uno de los nodos es diferente y el decidir entre el encendido de uno u otro depende en gran medida de este factor.

En este sentido, es más apropiado utilizar un métrica de energía, en lugar de potencia, para calcular el coste energético de cada sesión y, de esa forma, tener una estimación directamente medible en términos monetarios de la eficiencia.

La conversión y la medición se va a realizar directamente en el distribuidor, de forma que se estimará de forma dinámica en cada intervalo de captura de información de *QoS*, de duración T_{QoS}, en base a los modelos de potencia y a las sesiones soportadas por el nodo.

Se calculará un consumo medio $\overline{W}$ en base al consumo en el instante de captura anterior y el actual, de forma que se obtenga una buena aproximación de la potencia (en W) consumida durante el período de captura mediante un modelo de regresión lineal[18].

Una vez obtenido el consumo (en términos de potencia), se convertirá a una medida energética muy utilizada para la facturación, y equivalente a los julios, el vatio-hora (Wh), esto es:

$$E_{QoS}\,[Wh] = \overline{W} \cdot \frac{T_{QoS}[seg]}{3600}$$

Teniendo el consumo en términos de energía y no de potencia permite tener una idea del volumen de facturación de la electricidad consumida por el clúster y obtener métricas de eficiencia, denotada por *e*, de forma efectiva, para caracterizar de forma adecuada los equipos del clúster y el servicio en sí, por ejemplo:

$$e\,[Wh/sesión] = \frac{E\,[Wh]}{sesiones\ procesadas}$$

El objetivo será minimizar la métrica de eficiencia energética.

[18] Esto hace que a menor tiempo de intervalo de captura (mayor frecuencia), la precisión de las estimaciones mejore, pero incrementa los recursos necesarios para la autogestión del distribuidor.

7.4.3 Influencia de los parámetros U_{max} y U_{min}

La prueba de validación de $U_{máx}$ y $U_{mín}$ tiene como objetivo calibrar los valores y, sobre todo, comprobar la influencia que tienen sobre el consumo y la calidad de servicio. Para las pruebas se van a usar la configuración descrita en la sección 7.4.1 y la carga descrita en el Gráfico 42.

Es importante tener en cuenta que, pese a que se van a usar como parámetros iniciales para las pruebas los valores deducidos en las pruebas realizadas para los clústeres homogéneos (ver sección 7.3), debido a que la configuración del clúster es diferente y, asimismo, la carga es mucho más amplia (al disponer de muchos más recursos).

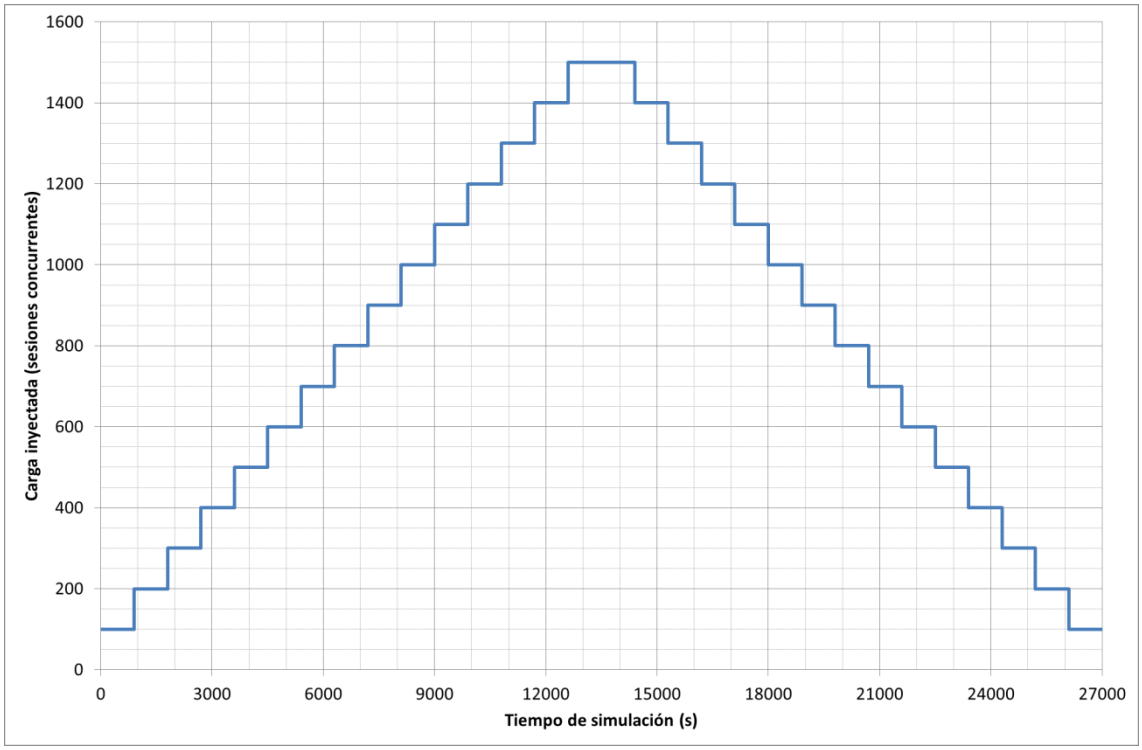


Gráfico 42: Carga inyectada en la prueba de validación de clúster heterogéneos

Para la elección de carga, se ha escogido un ciclo diario de trabajo de un servicio general, como podría ser un servidor empresarial, con incrementos en la mañana, un largo estacionario durante la jornada laboral y descensos en las tardes.

Normalmente, este es un régimen de trabajo bastante estudiado y permite recoger un entorno de incremento y decremento de carga, además de un largo período de sobresolicitación. No solo permite modelar servicios empresariales, puesto que muchos ciclos de carga de otros tipos de servicio se comportan de forma similar (aunque con diferente pendiente en los incrementos y descensos de carga).

En principio, este período se ha acortado, ya que lo interesante es estudiar el algoritmo en momentos de cambio de carga (que es cuando se toman decisiones) más que en períodos estacionarios. En cualquier caso, se cuenta con un estacionario lo suficientemente grande como para poder estudiar con solvencia el comportamiento del algoritmo en este tipo de mesetas de carga.

7.4.3.1 *Resultados para* U_{max} = 0,95 *y* U_{min} = 0,8

Debido a que estos fueron los valores de los parámetros deducidos en las pruebas en entornos homogéneos, van a ser los parámetros estudiados al inicio en el análisis en entornos heterogéneos, sin perjuicio del estudio de otros valores para comprobar la influencia de los mismos en el consumo, rendimiento y calidad de servicio.

En el Gráfico 43 se muestran los resultados del experimento, de un duración total de 7,5 horas; con la carga descrita en el Gráfico 42. Los datos proceden de la captura en línea en el distribuidor de carga, capturada con un sistema de ventana deslizante como el descrito en la sección 9.3.2. La ventana es de 100 segundos para estas pruebas.

En primer lugar se muestra la evolución del percentil-90 del tiempo de respuesta (R) del servicio. Como se puede comprobar, se mantiene la garantía del SLA (establecido en 2 segundos) en todo momento.

Se observan picos en el tiempo de respuesta en los momentos de sobresolicitación, mientras se encienden nuevos nodos y en los momentos de sobre carga en el estacionario principal de 1600 sesiones.

Tal y como se puede deducir comparando los datos con los resultados de rechazos de sesiones, éstos son los momentos en los que se producen rechazos de sesiones. De esta forma, teniendo en cuenta que se producen rechazos de sesión en los momentos transitorios de encendido de nodos, es síntoma de que el parámetro $U_{máx}$ debe ser reducido, pues se está tardando demasiado tiempo en encender los nodos y, por tanto, la capacidad no llega a estar disponible en el momento en el que se necesita (como se observó anteriormente, para este perfil de carga).

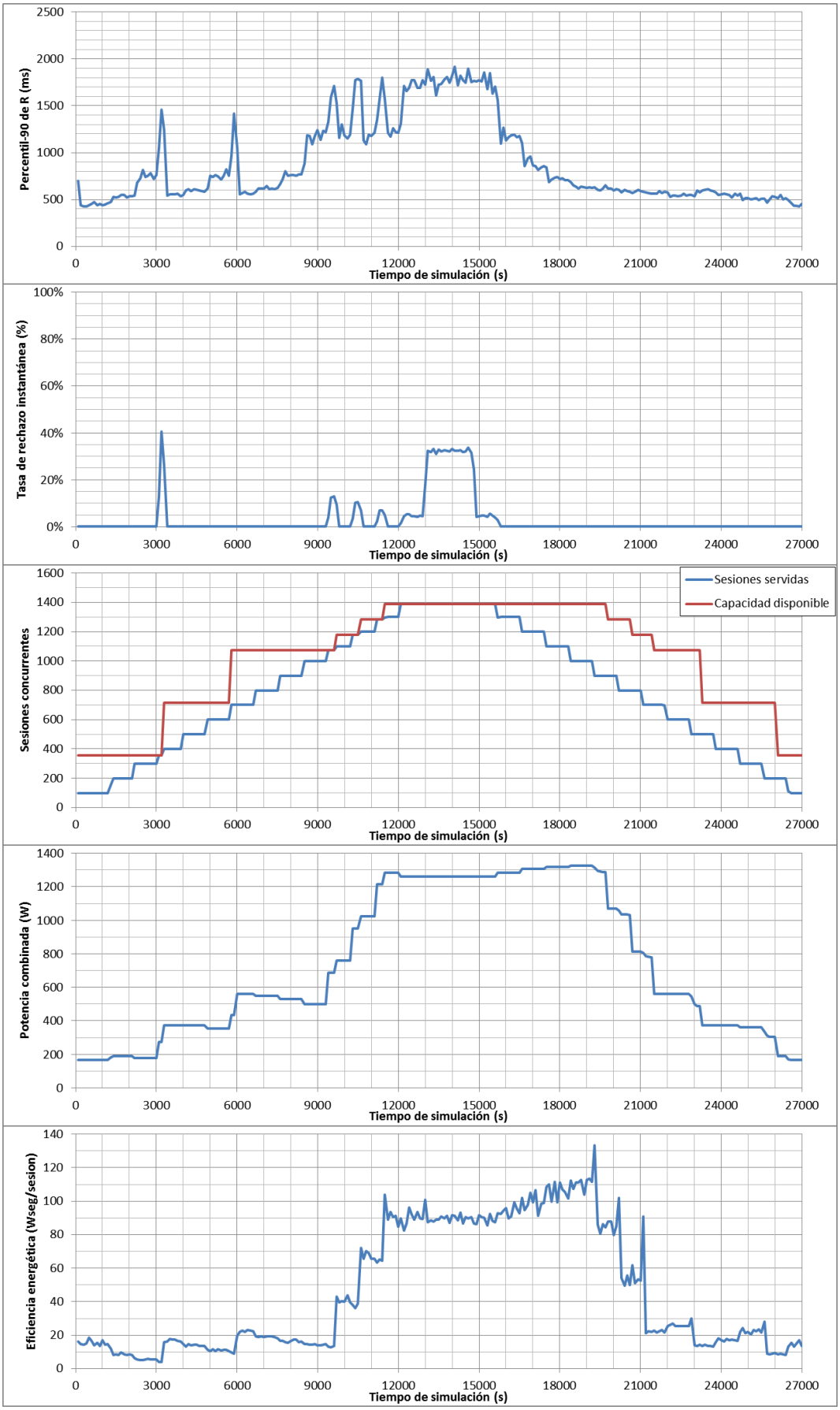


Gráfico 43: Experimento en clúster heterogéneo para U_{max} = 0,95 y U_{min} = 0,8

Otra forma de observar este efecto es haciendo uso de las curvas de carga, donde puede observarse cómo existen momentos, que coinciden con los momentos de rechazo, en los que la cantidad de sesiones servidas alcanza la capacidad disponible, indicando saturación del servicio.

En general, pueden definirse hasta tres curvas de carga diferentes:

- **Carga inyectada:** carga que el inyector de carga está solicitando, puede ser mayor o menor que la capacidad del servicio. Está representada en estas pruebas en el Gráfico 42.
- **Capacidad del servicio:** cantidad de sesiones que el servicio soporta. Depende de la cantidad de nodos disponibles y de la capacidad individual de cada uno. Hay que hacer notar que existen dos intervalos de tiempo que hay que tener en cuenta hasta que la capacidad está disponible: el intervalo de tiempo intrínseco al algoritmo necesario para que decida solicitar más capacidad encendiendo un nodo (definido por $U_{máx}$) y el tiempo de encendido de los nodos.
- **Carga servida:** sesiones activas de forma efectiva en el servicio. Siempre ha de ser menor o igual que la capacidad del servicio y menor o igual que la carga inyectada, en el segundo caso, de ser menor que la carga inyectada, se produce control de admisión y, por tanto parte de las sesiones inyectadas se rechazan, indicando que el servicio está saturado.

En los datos representados en el Gráfico 43 se muestran las dos últimas curvas.

En lo relativo al consumo energético, se muestra la evolución de la potencia instantánea, variable, al aplicarse técnicas de DVS antes de la saturación. Se observa, además, como el consumo de nodos es prácticamente el mismo (los saltos de consumo son similares para cada tipo de nodo), mientras que la capacidad aportada al servicio es claramente mayor en el caso de los nodos Intel.

Esto tiene un impacto importante en la eficiencia energética del clúster. Este gráfico, que indica la eficiencia del servicio medida en $Wseg/sesión$ (esto es, es una medida de coste y el objetivo es su minimización), muestra dos puntos importantes: primero, como era de esperar, el coste de las sesiones en los nodos Intel es menor que en los nodos AMD. En segundo lugar, se muestra como, conforme se incrementa la carga (sin encender nuevos nodos), la eficiencia energética se incrementa, decreciendo el coste por sesión.

Adicionalmente, un punto interesante de esta gráfica son los picos producidos en los intervalos de tiempo transitorio mientras se apagan nodos. Este comportamiento es coherente, pues en esos nodos se está reduciendo rápidamente la carga, de forma que la eficiencia de éstos esa decreciendo rápidamente también.

Por último, este experimento tuvo un coste energético de 5540 Wh, con una cantidad de rechazos del 6,16% y con una cantidad de sesiones por encima del SLA del 2,84% del total de sesiones servidas (luego, se está cumpliendo el SLA, con un percentil 97 en lugar de percentil 90).

7.4.3.2 *Resultados para* U_{max} = 0,95 *y* U_{min} = 0,85

Uno de los efectos que también se observaban en los resultados mostrados en el Gráfico 43 es, durante los períodos de descarga, el tiempo que pasa entre que la carga se reduce y el algoritmo decide reducir la capacidad disponible para ahorrar energía.

En este sentido, en la prueba anterior el algoritmo permanece mucho tiempo con más capacidad de la necesaria en los momentos de descarga y, mediante el incremento de $U_{mín}$ se puede adelantar el tiempo de apagado, reduciendo el consumo en la segunda parte del experimento.

Tal y como se demuestra en el Gráfico 44, incrementar el umbral de apagado en 0,05 permite adelantar el encendido en unos 1000 segundos en cada nodo, de forma que el consumo del experimento se reduce a 5361 Wh (unos 200 Wh) manteniendo una calidad de servicio similar (6,03% de rechazos y 2,88% de sesiones sobre el SLA).

7.4.3.3 *Resultados para* U_{max} = 0,95 *y* U_{min} = 0,9

Continuando con la evolución del parámetro $U_{mín}$, se puede incrementar en 0,05 para llegar a un valor de 0,9. En este caso, y como se observa en los resultados del Gráfico 45, el comportamiento en el apagado es similar, aunque el primero de los nodos AMD se apaga unos 1000 segundos antes.

Esto permite reducir el consumo de la prueba hasta los 5299 Wh, teniendo un rechazo del 6,17% y una cantidad de sesiones sobre el SLA del 2,81%.

Es importante destacar cómo es más improbable que existan rechazos de sesión en los momentos de reducción de carga, ya que se están utilizando umbrales de post apagado, tal y como se explicó en la sección 6.3. Sin embargo, sí podrían producirse rebotes, aunque en estas pruebas no se ha detectado ninguno.

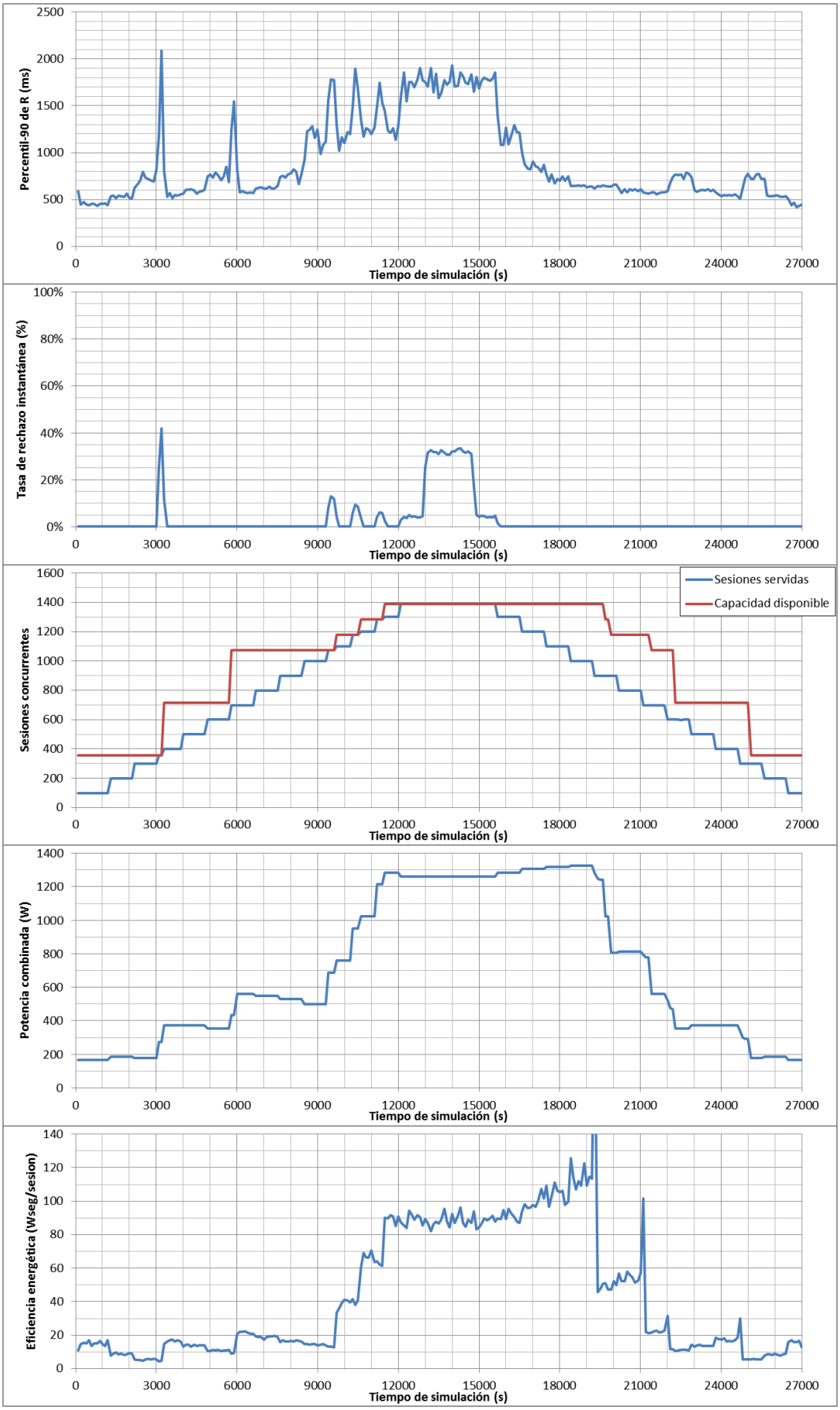


Gráfico 44: Experimento en clúster heterogéneo para Umax = 0,95 y Umin = 0,85

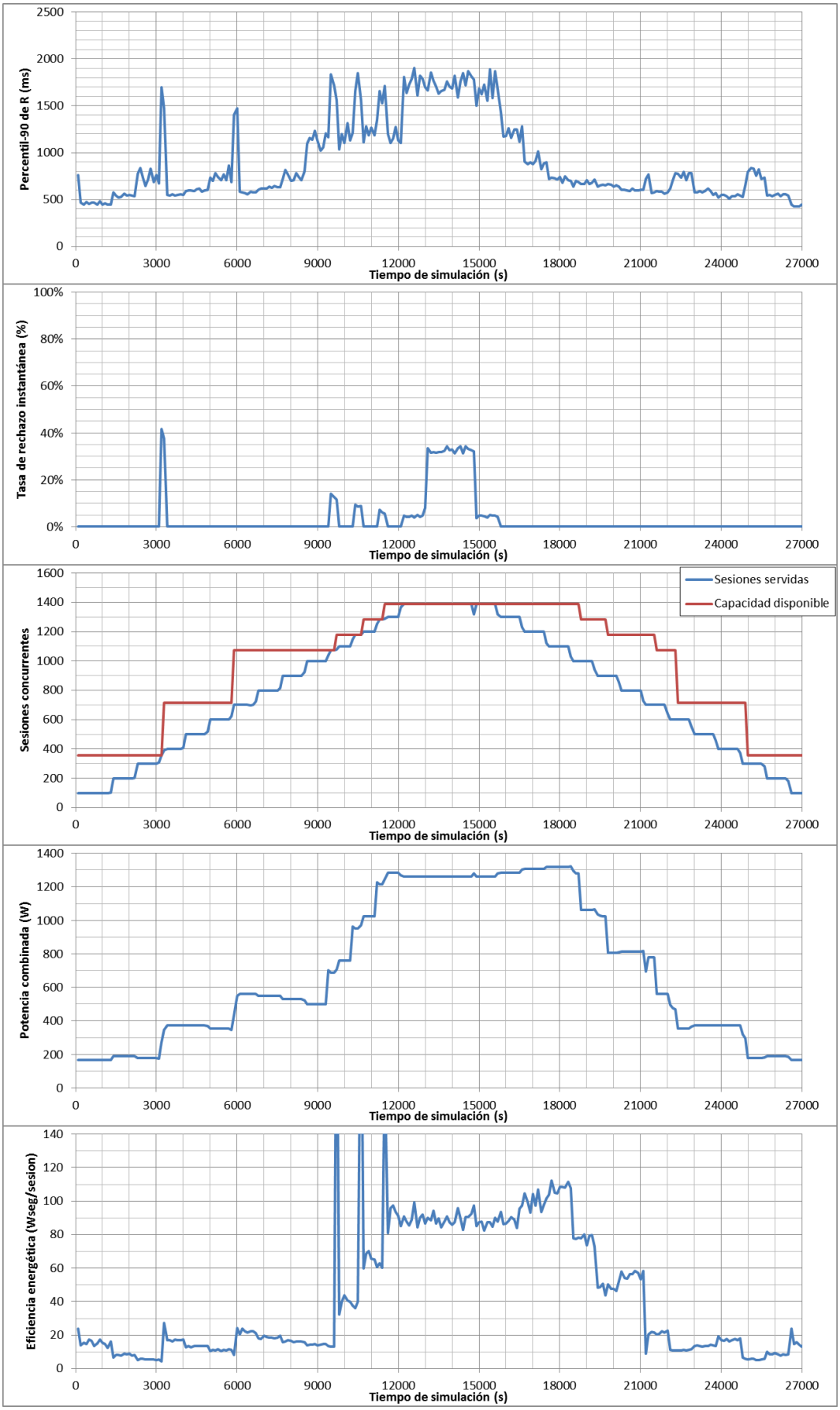


Gráfico 45: Experimento en clúster heterogéneo para Umax = 0,95 y Umin = 0,9

7.4.3.4 *Resultados para* U_{max} = 0,9 *y* U_{min} = 0,75

Para esta prueba se ha reducido el valor del umbral de encendido para adelantar el encendido de los nodos en situaciones de carga creciente. De esta forma, se reduce la tasa de rechazo de sesiones en los transitorios de encendido de nodos, pero se incrementa ligeramente el consumo.

Observando los resultados del Gráfico 46 puede verse cómo efectivamente casi se han eliminado todos los rechazos en los transitorios y cómo los nodos se encienden con mayor antelación que en las pruebas anteriores.

Un hecho interesante que se produce en esta prueba es el apagado de dos nodos de forma simultánea (hacia el segundo 20 500 de simulación), correspondientes a los dos nodos AMD.

Éstos son seleccionados para el apagado debido a la menor eficiencia energética que tienen y son seleccionados para el apagado en un período de tiempo muy corto (exactamente, entre el cierre de 3 sesiones). Esto es posible por la no existencia del bloqueo de apagado cuando un nodo ya está apagándose, lo que permite responder de forma rápida ante bajadas drásticas de carga.

En este caso, la carga no baja de forma muy drástica, pero el umbral de apagado se ha establecido a un valor muy bajo; 0,75, lo que hace que el efecto que se produce al alargar mucho el apagado de los nodos tenga las mismas consecuencias que una bajada drástica de carga.

En esta prueba se observan también los picos de ineficiencia energética, tal y como se esperaba y de forma coherente con las anteriores. El consumo se incrementó hasta los 5868 Wh, pero debido al adelantamiento del encendido de los nodos, la proporción de rechazo de sesiones se redujo al 5,67% y el número de sesiones sobre el SLA al 2,43%.

Como puede observarse, tanto en este experimento como en los anteriores, parece existir una relación entre la energía consumida en la operación del clúster y la calidad de servicio ofrecida a los clientes (ver sección 7.4.4 para más detalles).

Esta relación es esperable, y deseable, pues permite cuantificar la degradación de la calidad de servicio conforme se reduce el consumo energético.

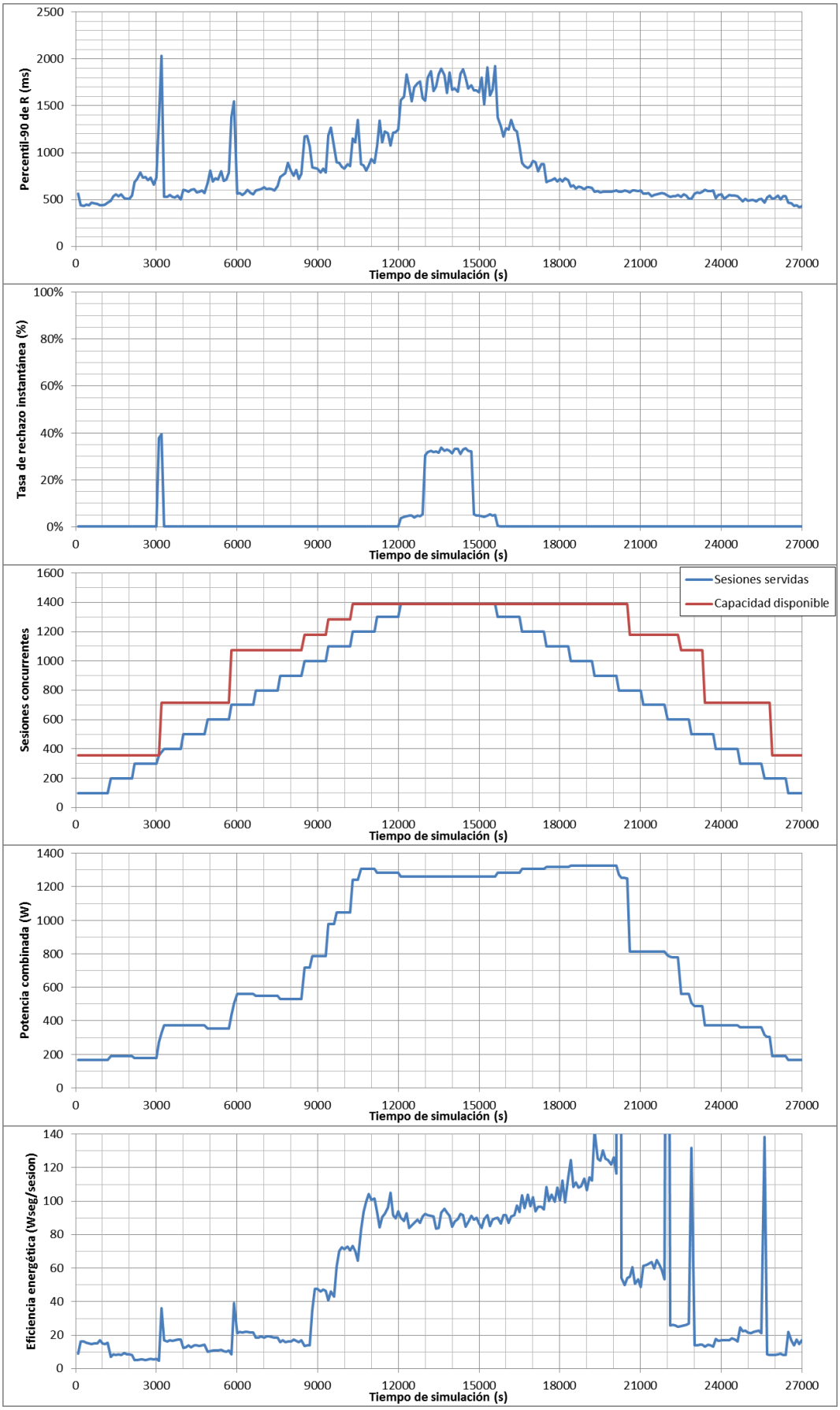


Gráfico 46: Experimento en clúster heterogéneo para Umax = 0,9 y Umin = 0,75

7.4.3.5 *Resultados para* U_{max} = 0,9 *y* U_{min} = 0,8

Tal y como se observa en el Gráfico 47, incrementar el umbral de apagado en este caso permite ajustar un poco más el consumo, aunque no de forma sustancial, pues los nodos se apagan prácticamente en el mismo momento que en configuraciones anteriores.

7.4.3.6 *Resultados para* U_{max} = 0,9 *y* U_{min} = 0,85

Es incrementando el valor del umbral de apagado hasta 0,85 cuando se puede observar el efecto, de nuevo, de un adelantamiento del apagado de los nodos. En el Gráfico 48 puede observarse cómo los nodos se apagan del orden de 1000 segundos antes que en la configuración anterior y, de hecho, los dos primeros nodos AMD se apagan de forma simultánea.

En estos experimentos cabe destacar cómo la eficiencia energética combinada del clúster depende de tres aspectos fundamentales:

- **La eficiencia de cada nodo:** de forma trivial, la eficiencia global dependerá de la eficiencia de cada uno de los nodos involucrados en la gestión. Concretamente, existen dos puntos a revisar con atención en este aspecto:
    - **El consumo básico (*idle power*):** determina el coste energético que tiene un nodo por el propio hecho de estar encendido. De esta forma, es conveniente minimizar el número de nodos encendidos para concentrar la carga.
    - **La proporcionalidad en la evolución del consumo:** en el caso de los nodos menos eficientes, la diferencia entre el consumo básico y el consumo de pico es notablemente inferior a la diferencia hallada en los nodos más eficientes. Es más, el grado de linealidad en el incremento del consumo conforme se incrementa la carga es también clave para mantener un alto grado de eficiencia.
- **La cantidad de nodos encendidos:** a su vez, la cantidad de nodos encendidos es importante, pues multiplica el efecto del consumo básico y, si dichos nodos tienen diferentes curvas de potencia, la eficiencia global dependerá de la combinación de las mismas.

Tal y como se puede deducir de las anteriores afirmaciones, el coste energético de cada sesión no es el mismo en todos los rangos de operación del nodo (sólo lo sería en caso de crecimiento perfectamente lineal en la curva de potencia). De esta forma, aunque en términos generales, cuanta más carga tiene un nodo, más eficiente es (debido al efecto del estancamiento del

consumo de pico en un máximo), en regímenes intermedios de utilización para clústeres heterogéneos, asignar una sesión a un nodo u otro puede tener implicaciones en el consumo global del servicio (ver sección 8).

7.4.3.7 *Resultados para* U_{max} = 1,0 *y* U_{min} = 0,85

Una vez explorados los efectos y el comportamiento del algoritmo con los valores iniciales y con valores inferiores para los umbrales de control. Es conveniente observar de forma empírica cómo se comporta el algoritmo para valores altos de los parámetros. En este caso, se está probando con el valor de $U_{máx}$ más alto posible y una combinación de valores de $U_{mín}$ hasta el máximo (con saltos de 0,05).

Utilizar un valor de umbral de encendido tan alto como 1,0 implica que los nuevos nodos van a ser encendidos (iniciado el encendido) cuando el clúster esté al máximo de su capacidad, de forma que no se deja nada de capacidad de respaldo para permitir el crecimiento mientras los nodos se encienden. Esto significa que el rechazo de sesiones es seguro en estos transitorios, pero que la eficiencia en la parte ascendente del experimento va a ser máxima.

Tal y como se observa en el Gráfico 49, en la parte ascendente del experimento, la capacidad disponible del clúster presenta el mejor ajuste posible a la capacidad solicitada de forma que el rechazo de sesiones es mínimo.

Como se puede deducir, la eficiencia se maximiza (el coste de cada sesión es mínimo) ya que se alarga el servicio con los nodos más eficientes. No obstante, la calidad de servicio se ve perjudicada, con percentiles de respuesta más altos (en promedio), aunque no se supera el *SLA*. Y, sobre todo, la tasa de rechazo en los transitorios se dispara.

Es conveniente, asimismo, reseñar cómo existe otro aspecto a comentar relativo a la parte descendente de carga, y son los picos que se producen en el percentil cuando se apagan los nodos.

En esta medición son más visibles y son debidos a la concentración de carga que se produce al apagar uno de los nodos, que hace que los nodos restantes en el conjunto de nodos encendidos pasen a operar en un régimen de carga más alto, incrementando su tiempo de respuesta.

Estos picos nunca van a superar el umbral del *SLA*, pues, al no haber migración de carga, nunca se aceptará una sesión que haga pasar la utilización de un nodo por encima de 1,0.

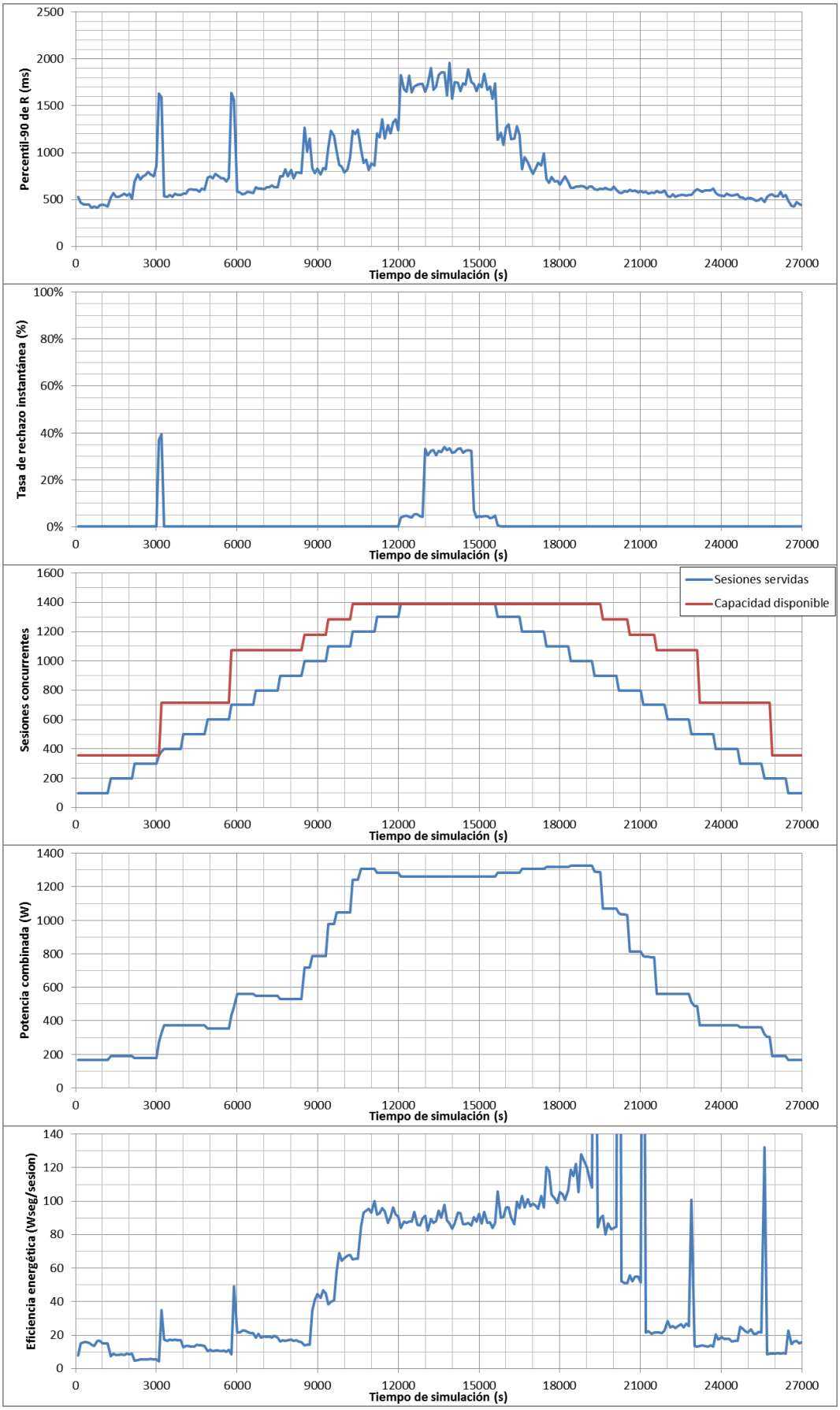


Gráfico 47: Experimento en clúster heterogéneo para Umax = 0,9 y Umin = 0,8

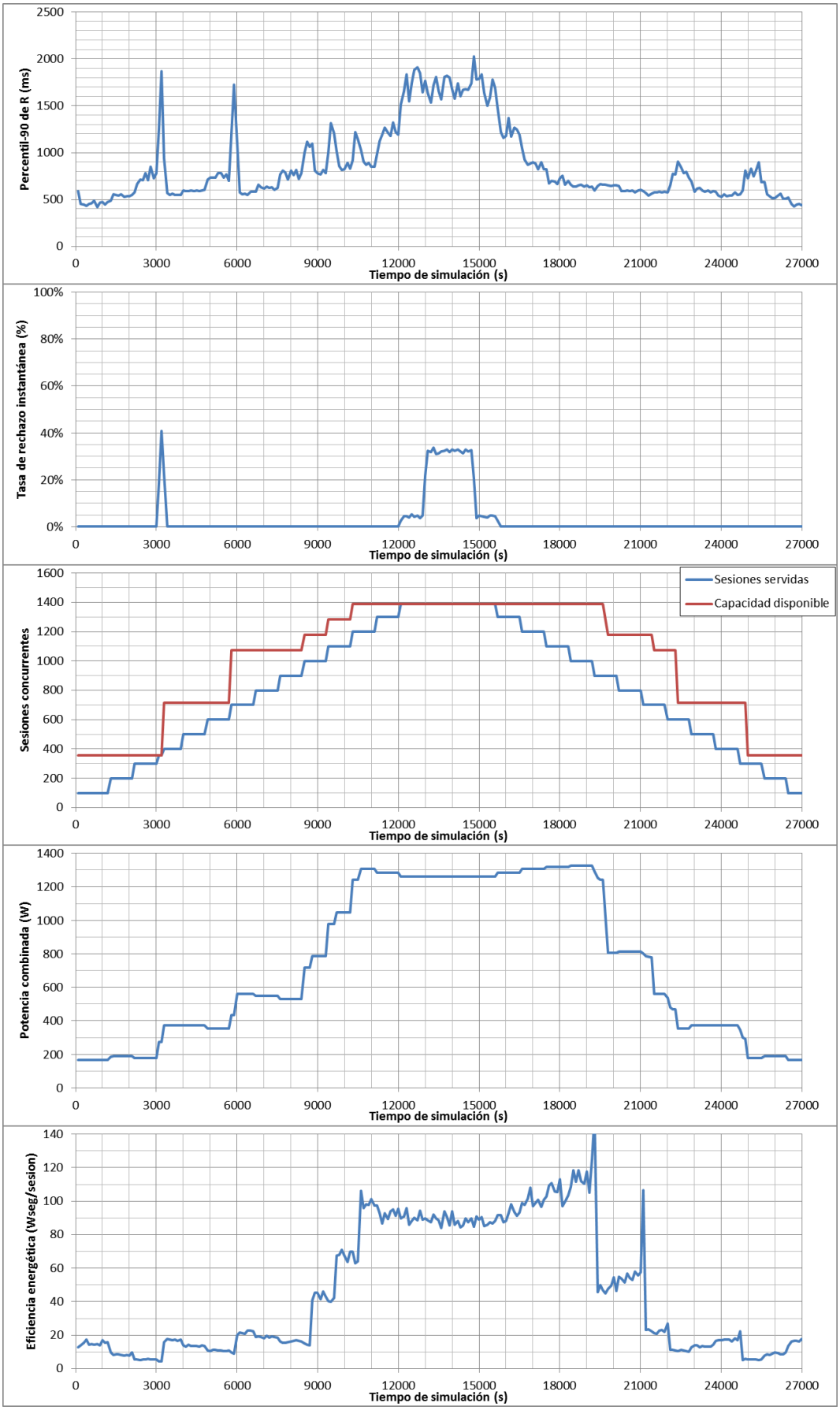


Gráfico 48: Experimento en clúster heterogéneo para Umax = 0,9 y Umin = 0,85

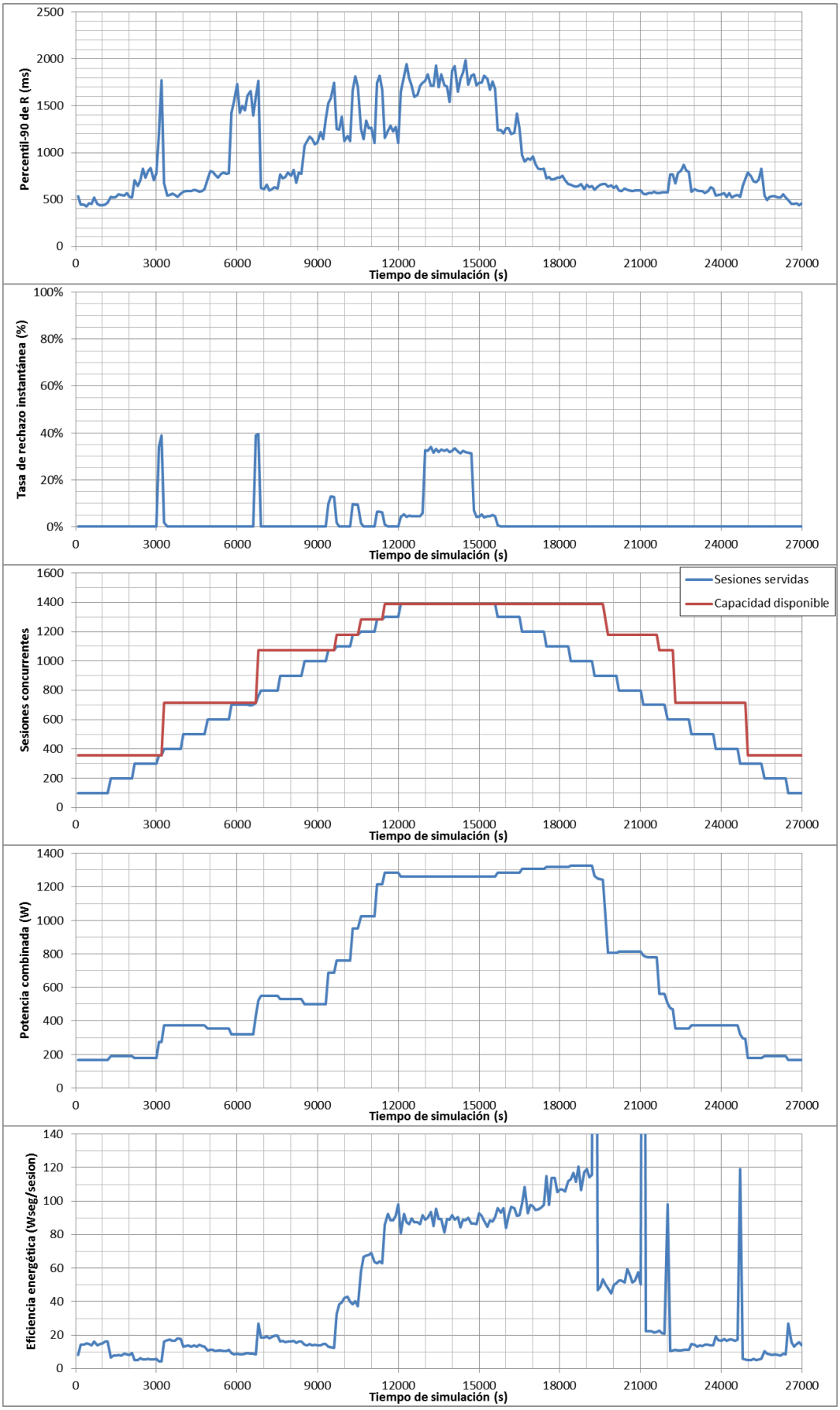


Gráfico 49: Experimento en clúster heterogéneo para Umax = 1,0 y Umin = 0,85

7.4.3.8 *Resultados para* U_{max} = 1,0 *y* U_{min} = 0,9

Al incrementar el valor del umbral de apagado, se observa cómo, al igual que en casos anteriores, se adelanta el apagado.

En el Gráfico 50 puede observarse, aproximadamente en el segundo 21 500 de simulación cómo parece producirse un régimen de rechazo de sesiones al apagarse el primero de los nodos Intel. Ésta es la única situación en la que podría producirse un rechazo de sesión en zonas de descenso de carga, y se da cuando, en situaciones de descenso se produce un régimen estacionario de carga o un ligero incremento y el valor de $U_{máx}$ es demasiado alto.

En este sentido, hay que destacar dos factores que acentúan este comportamiento: el tiempo de apagado y la capacidad del nodo a apagar.

Si el tiempo de apagado es muy largo, el clúster toma decisiones de apagado con mucho tiempo de antelación. Refinamientos como el *rescate* de nodos en descarga permiten reducir este efecto, eliminando rebotes hasta un cierto intervalo de tiempo (siempre depende de la relación existente entre el tiempo de apagado y la duración de la sesión, que marca la resolución temporal discreta que el algoritmo tiene para decidir).

Si la capacidad del nodo es muy alta, como en el caso de este nodo Intel, el apagado va a cambiar drásticamente la capacidad del clúster, y por tanto, las utilizaciones de los nodos aún encendidos. Si el valor del umbral de encendido es razonable, se producirá un rebote (en este caso, deseado) con suficiente antelación para poder recibir carga antes del agotamiento de la capacidad restante.

Si el umbral de encendido es demasiado alto, además del rebote para adaptarse a la nueva carga, se producirán rechazos de sesión en el transitorio. Es por esto que ambos valores han de ser lo más altos posible, pero mantener una relación adecuada.

7.4.3.9 *Resultados para* U_{max} = 1,0 *y* U_{min} = 0,95

Este experimento, cuyos resultados se muestran en el Gráfico 51, es el experimento con los valores de umbrales más altos posibles y, como cabría esperar, con menor consumo combinado ($5016\,Wh$). Asimismo, es el experimento donde la calidad de servicio es peor, con tasas de rechazo más altas (superiores al 6,5%) y sesiones sobre el SLA (más del 3%).

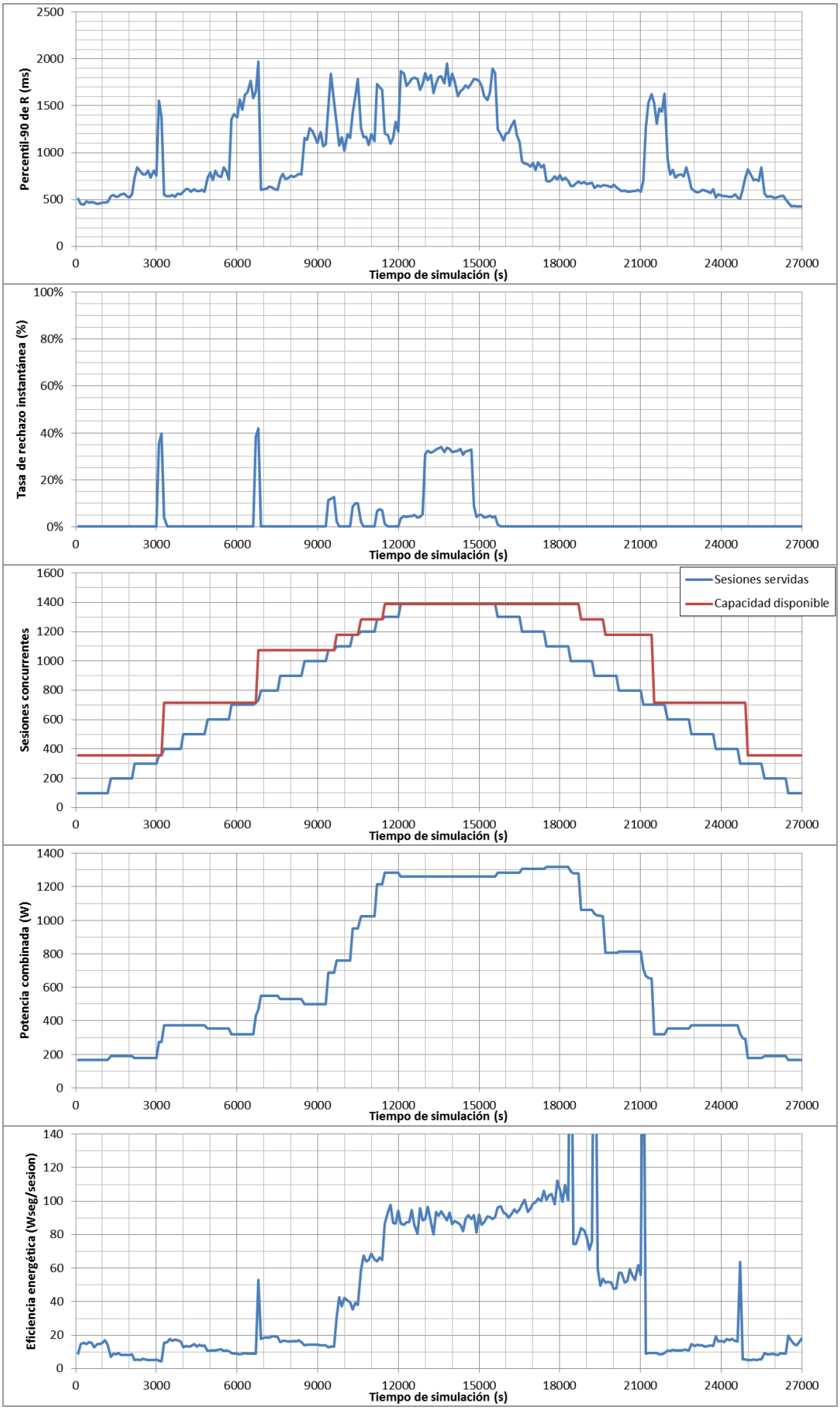


Gráfico 50: Experimento en clúster heterogéneo para Umax = 1,0 y Umin = 0,9

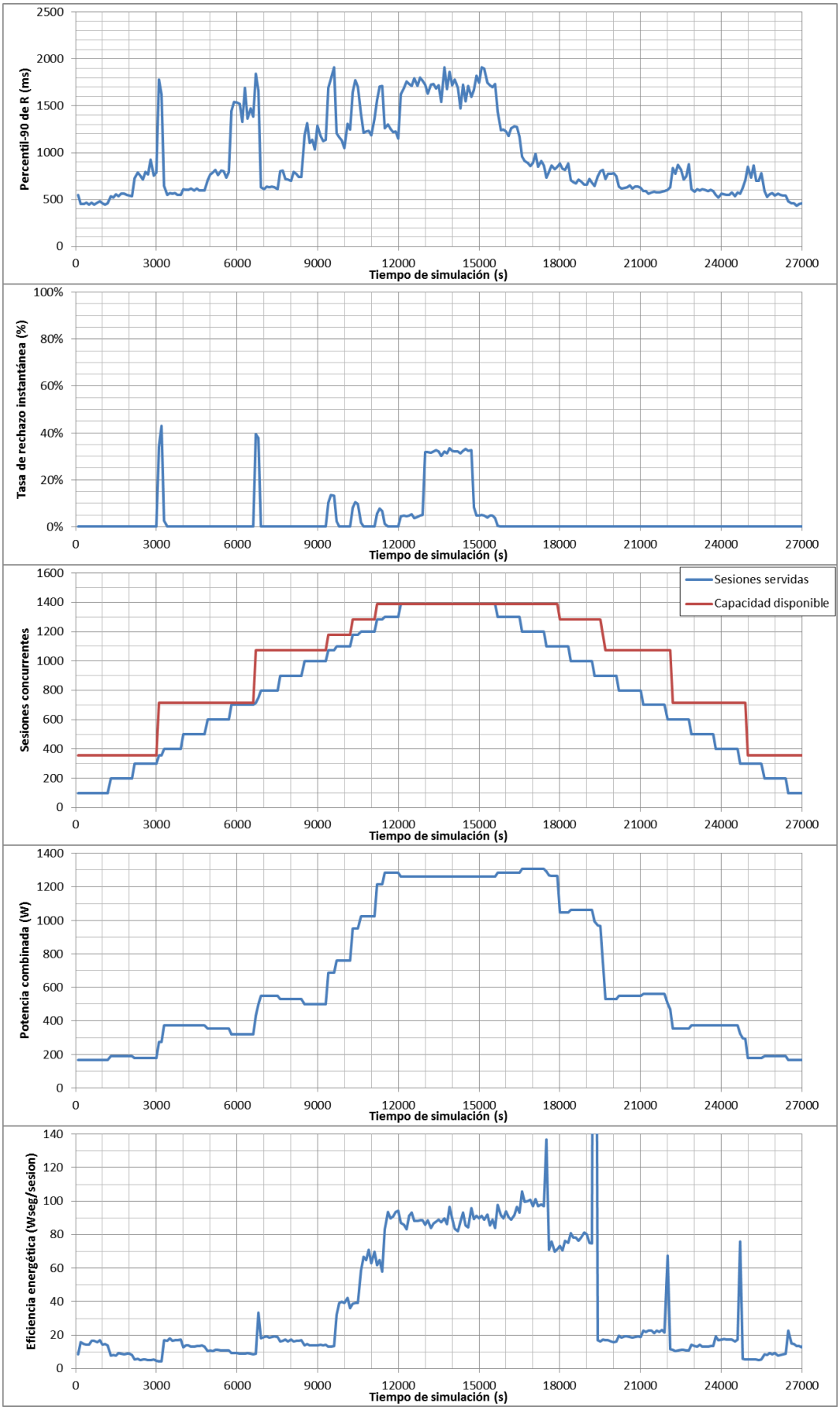


Gráfico 51: Experimento en clúster heterogéneo para Umax = 1,0 y Umin = 0,95

7.4.3.10 *Resultados para* U_{max} = 0,85 *y* U_{min} = 0,7

Con objetivo de continuar la exploración de los valores de los umbrales, va a reducirse el valor del umbral de encendido a 0,85; de tal forma que se pueda ver la influencia con respecto a dicho umbral en valores bajos del mismo.

Fundamentalmente, el efecto conseguido se centra en minimizar el ratio de rechazos adelantando el encendido de los nodos. Adicionalmente, se logra mejorar la calidad de servicio, minimizando, como era de esperar, el número de rechazos y el tiempo promedio de respuesta.

7.4.3.11 *Resultados para* U_{max} = 0,85 *y* U_{min} = 0,75

Incrementando el umbral de apagado se consigue adelantar el apagado de los nodos, de forma que se reduce el consumo, de 5975 a 5914 Wh.

7.4.3.12 *Resultados para* U_{max} = 0,85 *y* U_{min} = 0,8

Incrementando aún más el umbral de apagado permite reducir el consumo ligeramente, pero llegado a un punto el ahorro empieza a ser menor a cada paso de incremento de los umbrales (sobre todo con un umbral de apagado tan bajo).

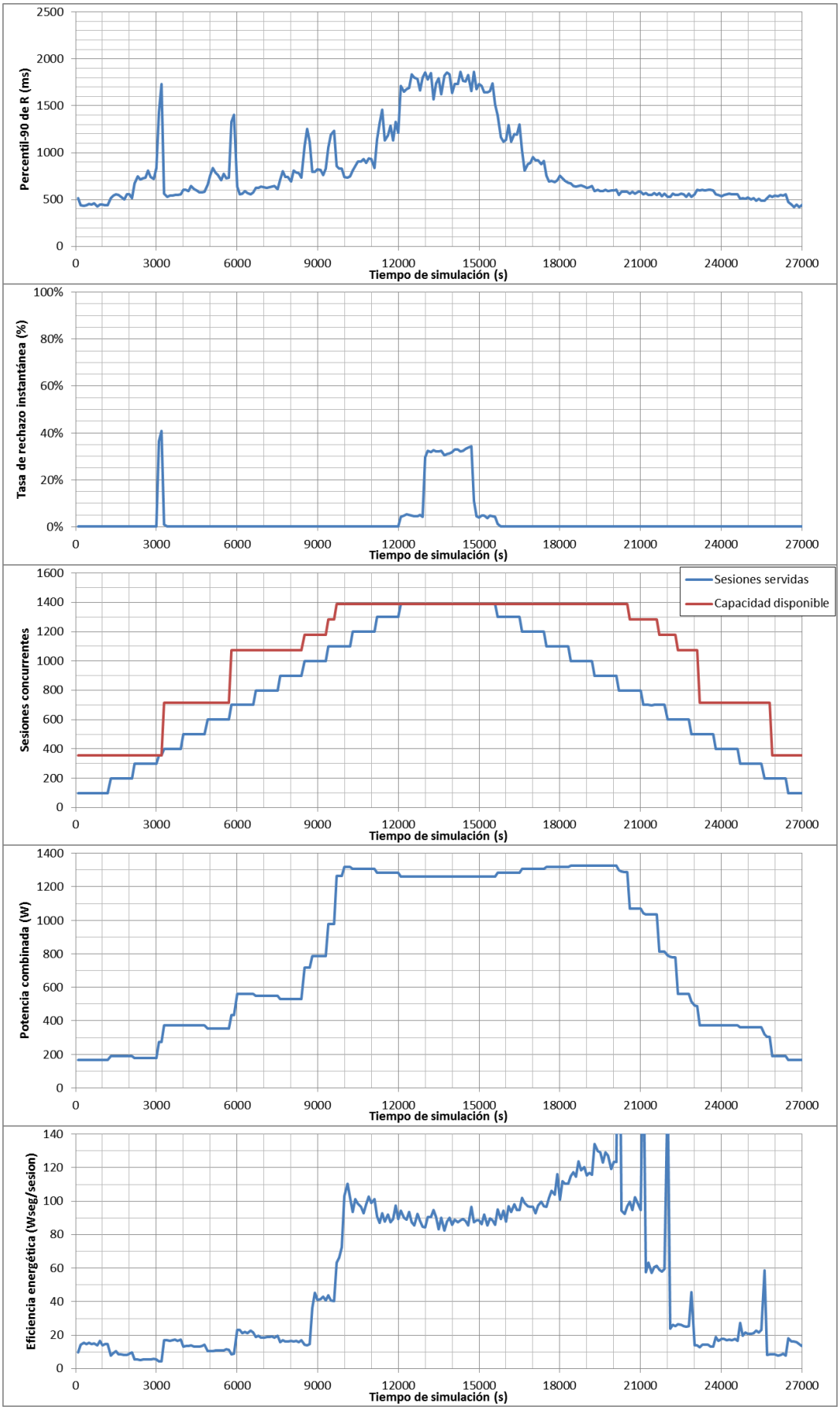


Gráfico 52: Experimento en clúster heterogéneo para Umax = 0,85 y Umin = 0,7

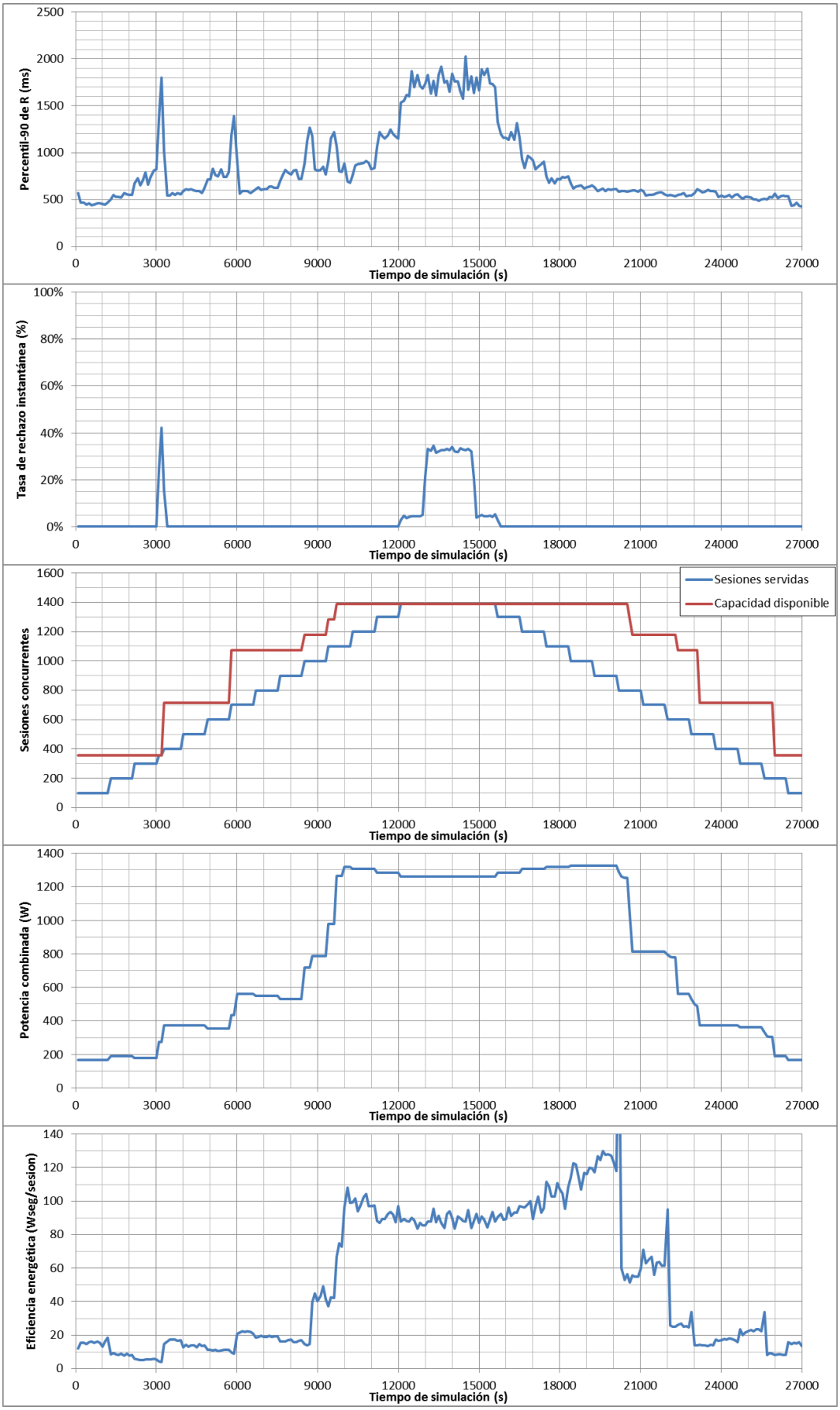


Gráfico 53: Experimento en clúster heterogéneo para Umax = 0,85 y Umin = 0,75

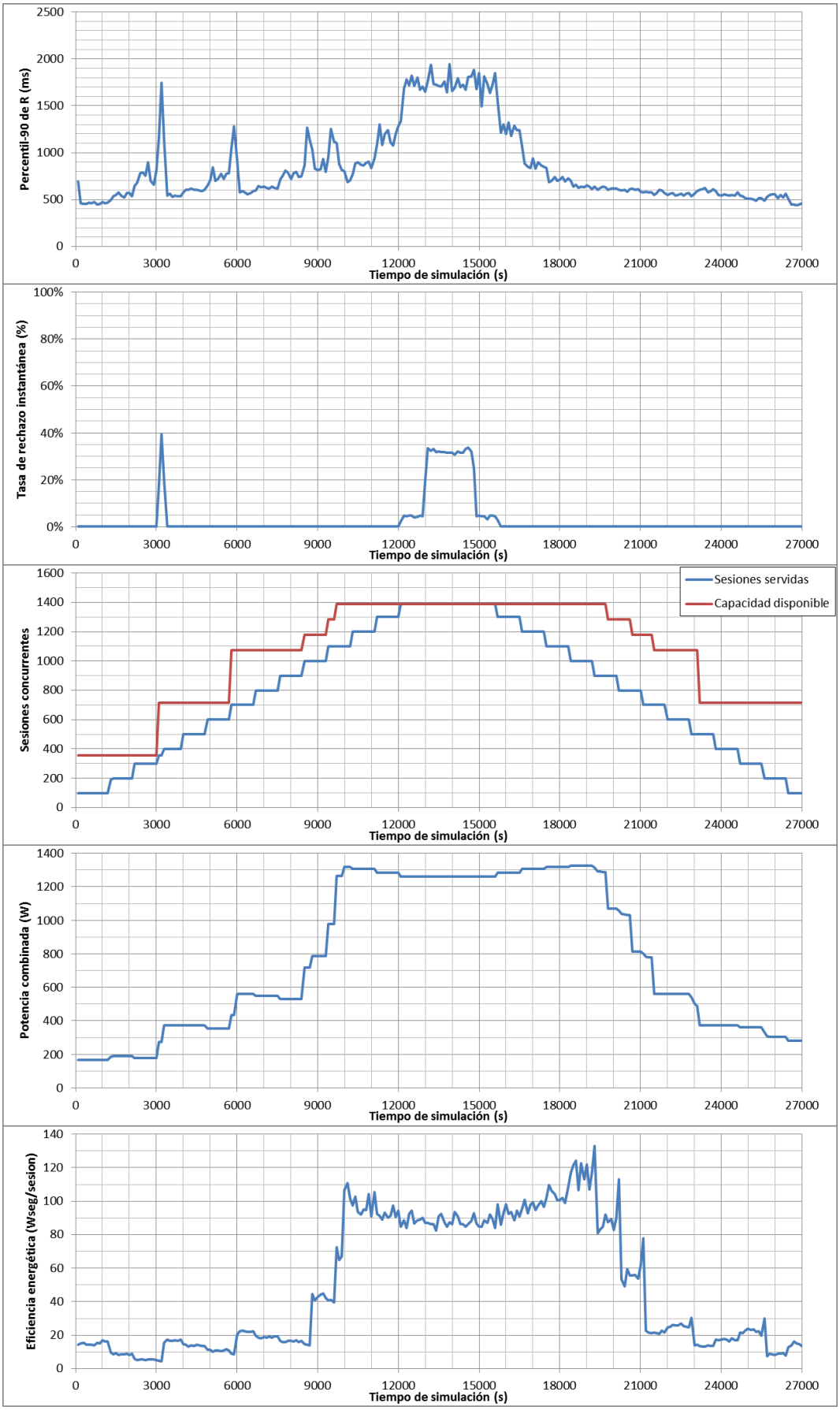


Gráfico 54: Experimento en clúster heterogéneo para Umax = 0,85 y Umin = 0,8

7.4.4 Relación entre umbrales, calidad de servicio y energía

Si bien se ha realizado un análisis bastante exhaustivo de los resultados y los puntos más destacados de las pruebas de validación del algoritmo y se ha podido apreciar ciertos resultados globales relativos a la configuración del algoritmo en base a sus dos parámetros fundamentales, a continuación se presenta un resumen de los resultados, destacando las relaciones entre calidad de servicio, energía y el valor de los parámetros.

7.4.4.1 Degradación de la calidad de servicio con el ahorro energético

Un resultado esperable es que se produzca cierta degradación de calidad de servicio ofrecida conforme se reduce la energía que se está consumiendo, de forma que, a su vez, se reducen los recursos disponibles.

En el Gráfico 55 puede observarse la primera de las relaciones entre la calidad de servicio, en términos de rechazos de sesiones, con respecto a la energía consumida por cada uno de los experimentos, además de una curva de tendencia cuadrática.

Todos los gráficos de esta sección siguen el siguiente convenio de colores y marcadores para cada uno de los resultados: los puntos marcados con un cuadrado representan las configuraciones con un valor de $U_{mín}$ más bajo para cada valor de $U_{máx}$, los puntos con un círculo un valor intermedio y los punto con un triángulo un valor más alto de $U_{mín}$.

Cada nivel de U_máx viene representado por un color, siendo el azul para el valor 1,0, el rojo para 0,95, el verde para 0,9 y finalmente el morado para 0,85.

En función de qué resultado se esté observando, este convenio permite relacionar los datos de diferentes gráficos con distinta dimensionalidad.

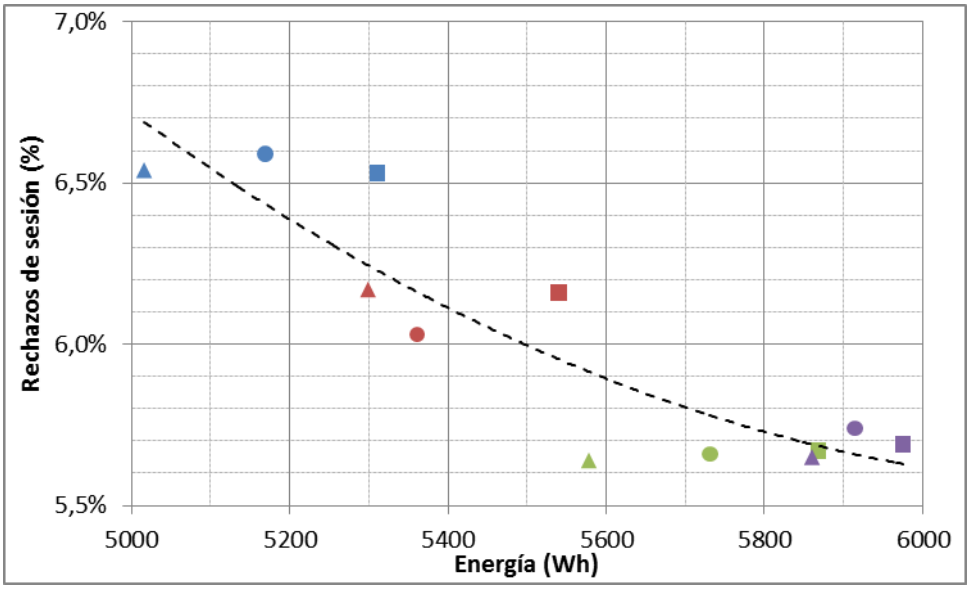


Gráfico 55: Relación entre el rechazo de sesiones y la energía

Este gráfico permite resumir de forma clara cómo a mayor consumo de energía, la calidad de servicio mejora, reduciendo el número de rechazos. Además, en el caso de ciertas parejas de configuraciones, permitiría escoger configuraciones concretas que presentan mejor tasa de rechazos para menor cantidad de energía (esto es altamente dependiente de la carga y, adicionalmente, de la propia varianza intrínseca de la experimentación).

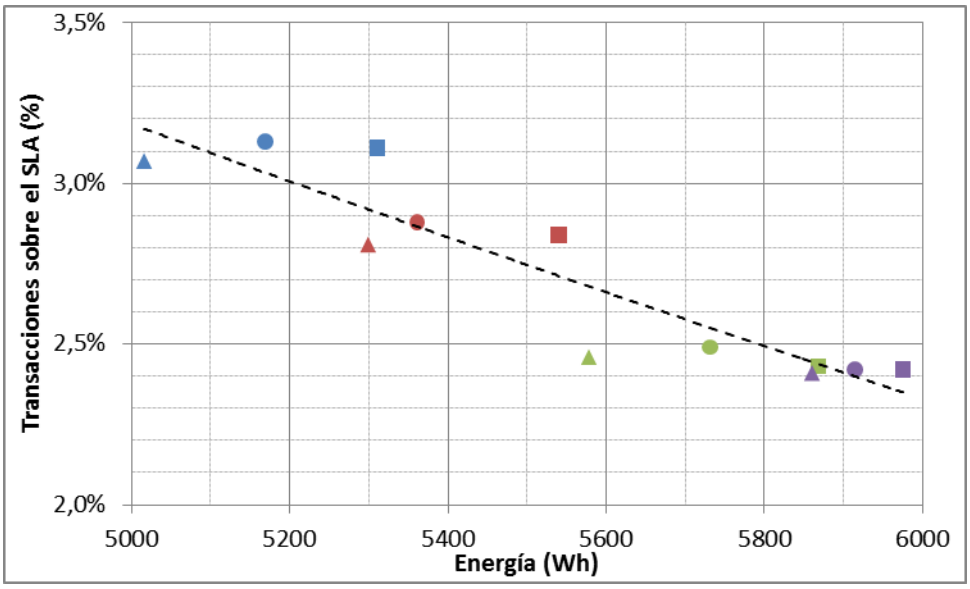


Gráfico 56: Relación entre las transacciones por encima del SLA y la energía

Exactamente el mismo análisis se ha realizado para el caso de la otra métrica importante de calidad de servicio, el número de transacciones por encima del *SLA*. En este caso se observa una

tendencia muy similar a la anterior, que ratifica las asunciones teóricas realizadas en secciones anteriores.

7.4.4.2 Mejora del consumo energético con el incremento de los umbrales

Tal y como se avanzaba en la sección 4.3.4, un mayor valor para el umbral de encendido permite reducir el consumo retardando el encendido de los nodos, mientras que un incremento del umbral de apagado permite ajustarlo más, adelantando el apagado.

El Gráfico 57 muestra, para cada par de valores de $U_{máx}$ y $U_{mín}$ (cada uno de los experimentos realizados) el consumo energético del mismo.

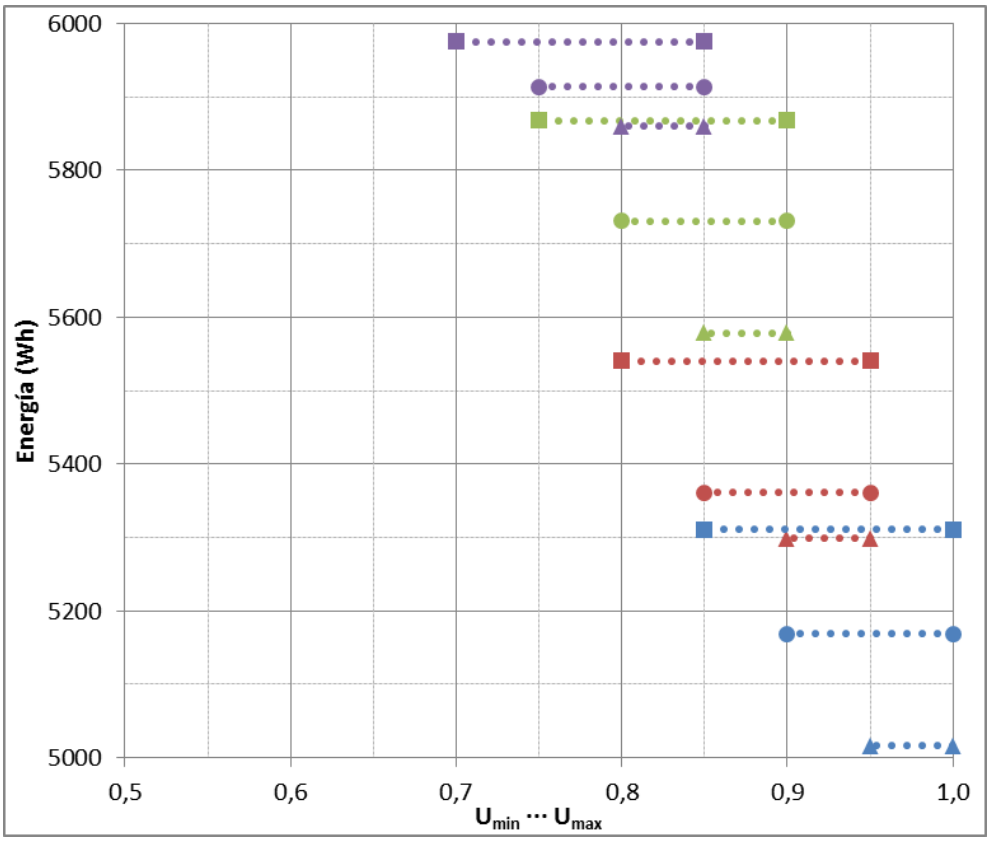


Gráfico 57: Relación entre los umbrales y el consumo energético

Pueden observarse tres tendencias:

- Conforme se incrementa el umbral de encendido $U_{máx}$, se reduce el consumo.

- Conforme se incrementa el umbral de apagado $U_{mín}$, se reduce el consumo, para un valor de $U_{máx}$ dado.
- En general, cuanto más cercanos están ambos umbrales, y más altos son, el consumo se tiende a minimizar.

7.4.4.3 Calidad de servicio e incremento de umbrales

Finalmente, se analiza la relación existente entre el valor de los umbrales de apagado y encendido y la calidad de servicio ofrecida.

En el caso del Gráfico 58, puede observarse cómo, conforme se incrementan los umbrales, si bien se reduce el consumo, se degrada la calidad de servicio, de forma que las sesiones rechazadas pueden llegar a incrementarse en hasta un punto porcentual.

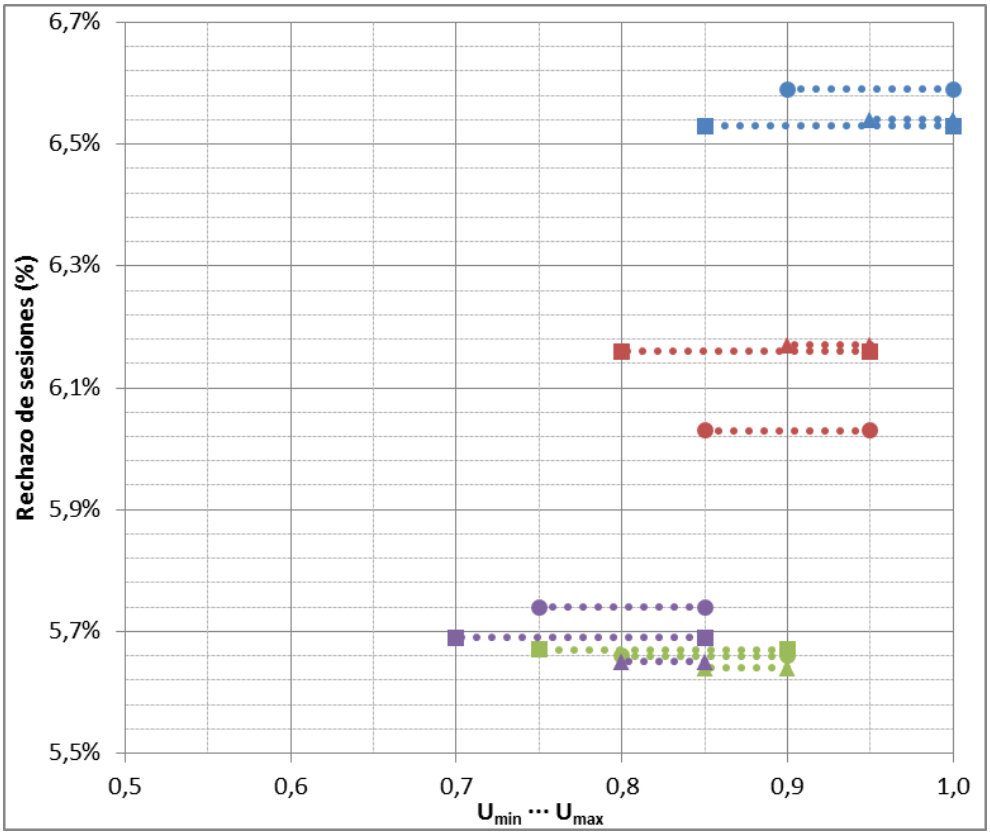


Gráfico 58: Relación entre el valor de los umbrales y el rechazo de sesiones

En el caso del Gráfico 59, puede observarse esta misma relación, pero aplicada al número de transacciones por encima del SLA. En este caso, la relación es más clara.

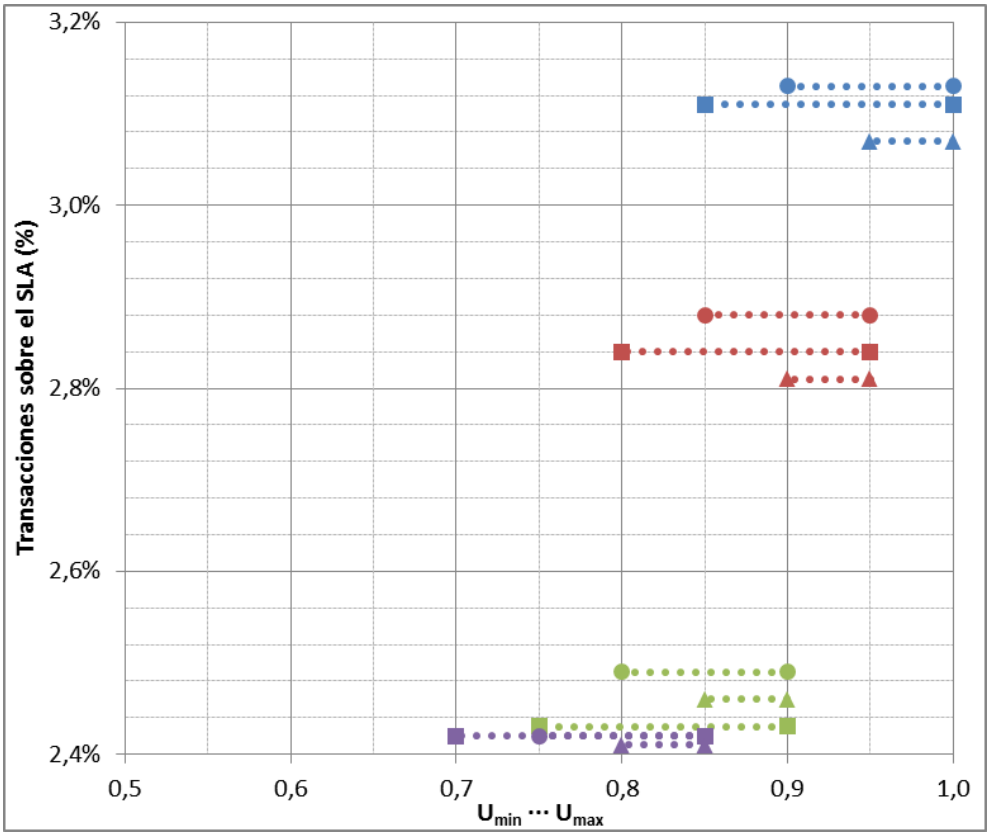


Gráfico 59: Relación entre el valor de los umbrales y las transacciones sobre el SLA

Es importante hacer notar como en el caso de la calidad de servicio, la mayor dependencia se produce respecto al umbral de encendido, de forma que aumentarlo, retrasa el encendido de los nodos, lo que afecta a la calidad de servicio.

Incrementar el umbral de apagado permite ajustar la eficiencia energética, de forma que adelantando el apagado de los nodos se reduce el consumo global del clúster. Esto también tiene impacto en la calidad de servicio, pues reduce el número de nodos disponibles e incrementa en promedio el tiempo de respuesta.

8 CONCLUSIONES Y TRABAJO FUTURO

Tras la evaluación del algoritmo diseñado e implementado se concluye con un análisis final de los resultados y propuestas de evolución en el mismo.

8.1 Aportaciones del algoritmo

Fundamentalmente, la aportación principal del algoritmo presentado en este trabajo es la combinación de métodos de modelado sencillos y directos que facilitan la implementación en servicios existentes con la provisión dinámica de nodos con garantía del *SLA*.

Es fundamental destacar que esta propuesta es la primera en incorporar garantía de calidad de servicio y, a la vez, permitir modelar el servicio con tan solo dos modelos de caja negra que pueden ser estimados por los usuarios de forma sencilla.

Asimismo, mediante la incorporación de los parámetros de apagado y encendido es posible ajustar de forma más detallada el funcionamiento del algoritmo.

En cuanto al ahorro energético logrado, es inútil reducir el análisis a una cifra porcentual de ahorro, pues ésta es altamente dependiente de la carga, de la configuración del clúster, de la capacidad de los nodos y del propio hardware.

En las pruebas realizadas, los ahorros oscilaron entre el 17% y el 85% en determinadas situaciones, por lo que además, la horquilla de valores es amplia.

En cualquier caso, los resultados permiten ser optimistas debido a que se está logrando un ahorro similar a otras alternativas que implementan modelos complejos, pero asegurando la calidad de servicio, lo que es un punto muy importante en servicios de Internet como los que se despliegan hoy en día.

8.2 Limitaciones de la solución aportada

Es necesario destacar la existencia de dos limitaciones importantes en la solución aportada: en primer lugar, la portabilidad del prototipo desarrollado, muy ligado a plataformas Windows, debido a la facilidad que ello proporcionaba en el entorno de experimentación disponible.

En segundo lugar, existen limitaciones en las curvas de potencia medidas de forma experimental, dada la inestabilidad del consumo de los computadores, sobre todo en el caso de los equipos AMD, que redundan en ciertas variaciones en las curvas combinadas de consumo.

Por último, es conveniente reseñar cómo, con una infraestructura de pruebas de mayor tamaño, sería posible validar el algoritmo en escenarios mucho más complejos, como un *site* Grid.

8.3 Trabajo futuro

En las siguientes iteraciones de esta investigación, van a abordarse los siguientes puntos de mejora:

8.3.1 Adaptación de modelos en línea

La implementación de curvas de consumo y respuesta basadas en estimaciones NNLS permite estimar modelos de servicio en línea garantizando las características deseables de los modelos.

La estimación en línea es útil cuando se está en un escenario en el que parte del clúster es un servicio externo, por ejemplo, y el comportamiento no es ciertamente predecible.

8.3.2 Estados intermedios de apagado y *rescate* de nodos

Para mejorar más el comportamiento en los momentos de variación de carga, es imprescindible abordar la gestión de los estados intermedios de apagado, tales como suspensión o hibernación.

En este sentido, será necesario evaluar en primer lugar si el consumo ahorrado y el tiempo de recuperación es aceptable y presenta una mejora significativa.

Adicionalmente, todo el modelo de estados ha de ser revisado y actualizado, puesto que las relaciones de *rescate* y transiciones de energía se incrementan.

8.3.3 Balanceo de carga inteligente

Otra mejora a investigar se centra en el balanceo de carga, donde es conveniente realizarlo teniendo en cuenta, además de factores de utilización, factores de consumo energético, pues, tal y como se observa en las curvas de potencia y eficiencia, no siempre la asignación de sesiones tiene el mismo coste dependiendo del nivel de carga actual.

8.3.4 Priorización y *SLA* dinámico

Por último, y como característica ciertamente experimental, se puede abordar la priorización de carga en servidores y clientes de características diferentes.

Asimismo, y continuando con la estimación dinámica de modelos, es posible abordar la utilización de *SLA* variables en tiempo de ejecución.

9 DESCRIPCIÓN DEL SISTEMA TRANSACCIONAL SINTÉTICO

Se ha decidido implementar un servicio transaccional sintético que permita emular y de reflejar de forma fexible numerosos servicios de Internet, tales como servidores de bases de datos, servidores Web o servidores de aplicaciones de comercio electrónico.

Este prototipo permite, además de implementar el algoritmo diseñado y realizar pruebas sobre el mismo, desplegar el servicio de forma muy rápida y, asimismo, la reconfiguración y conversión en diferentes elementos de servicio de forma sencilla.

Fundamentalmente, se ha implementado este servicio desde cero dada la ausencia de software de emulación de este tipo y el coste de licencias y tiempo de configuración de servicios existentes, como Microsoft SQL Server e Internet Information Services en el caso de las plataformas Microsoft disponibles en el laboratorio de experimentación.

La flexibilidad y rapidez en el desarrollo del prototipo utilizado ha permitido, además, validar todos los aspectos y dependencias de integración del algoritmo en entornos existentes en producción, de forma que se ha diseñado para minimizar el tiempo de integración.

9.1 Perfiles de gestión energética

No todos los sistemas tienen los mismos objetivos energéticos, ni siquiera éstos son constantes a lo largo del tiempo de funcionamiento. Los perfiles energéticos permiten resumir los requerimientos de energía de diversos sistemas arquetípicos.

Fundamentalmente, las mayores diferencias existen entre tres grupos de equipos: estaciones de trabajo, servidores y ordenadores portátiles. Cada uno de los tres tipos anteriores de

computadores tiene un objetivo diferente y unas prestaciones distintas, por lo que es fundamental establecer una política adecuada de gestión energética.

El prototipo del servicio transaccional desarrollado refleja y representa el perfil energético de servidores.

9.2 Gestión energética supra-nodo

A la hora de realizar la gestión energética de todo un clúster se puede distinguir entre dos tipos de gestión bien diferenciados:

- **Gestión intra-nodo:** esta gestión se circunscribe únicamente a un nodo del clúster de forma individual sin importar ni tener información sobre los demás nodos. El objetivo de esta gestión será hacer que el nodo consuma el mínimo de energía para satisfacer la demanda de trabajo que se le solicita de forma individual.
- **Gestión supra-nodo:** el objetivo de esta gestión de gestionar los nodos del clúster de forma combinada para que tenga, el conjunto, un consumo mínimo de energía sin afectar a la calidad del servicio proporcionado. Es importante señalar que el objetivo de esta gestión se limita al estado global de los nodos, sin entrar en gestión de dispositivos de cada uno de ellos.

Mientras que la gestión de la energía intra-nodo está relativamente bien resuelta por los propios sistemas operativos, como se comentaba anteriormente, la gestión supra-nodo no tiene una resolución práctica extendida que tenga en cuenta la calidad del servicio a proporcionar.

El método de un gestor energético supra-nodo que tenga en cuenta la calidad del servicio es, grosso modo, un módulo que se encargue de apagar y encender los diversos nodos que forman parte del sistema. Modularmente, tiene una estructura similar a la comentada anteriormente:

- **Monitor:** en este caso, ha de ser un monitor distribuido que tenga un proceso en cada nodo que informe a un colector de información centralizado. Habrá que disponer de un protocolo que permita la comunicación entre el colector y los procesos locales.
- **Actuador:** de cara a activar nodos, deberá ser un proceso centralizado el encargado de utilizar mecanismos como *Wake On LAN* para activar nodos.
- **Gestor energético:** dirige todo el sistema y ha de tener conexión con el módulo de calidad de servicio que decida cuándo necesita de más potencia de proceso o puede prescindir de parte de la misma.

La gestión del servicio será supra-nodo, dejando la gestión del *DVS* a los propios módulos de gestión de energía de los sistemas operativos de cada nodo. En el ámbito del servidor por el momento el uso de DVS centralizado no proporciona ganancias suficientes como para complicar la gestión y la estrategia en esta primera iteración del proceso de validación y desarrollo.

9.3 Prototipo de servicio transaccional

El objetivo de este prototipo es modelar un servicio transaccional sintético para realizar pruebas del gestor de energía en un clúster.

Este servidor se basa en tres partes fundamentales:

- **Inyector de carga:** permite modelar conjuntos de peticiones y cargas sobre el sistema.
- **Front-end:** es la parte del servidor que se encarga de recibir, mantener y repartir entre los nodos de cálculo las diversas sesiones transaccionales que recibe el servicio. Este módulo encapsula todo el clúster ofreciendo el servicio a los clientes como si de un solo equipo se tratara.
- **Nodos de back-end:** son los nodos de cálculo encargados de procesar las transacciones según se las reparta el front-end.

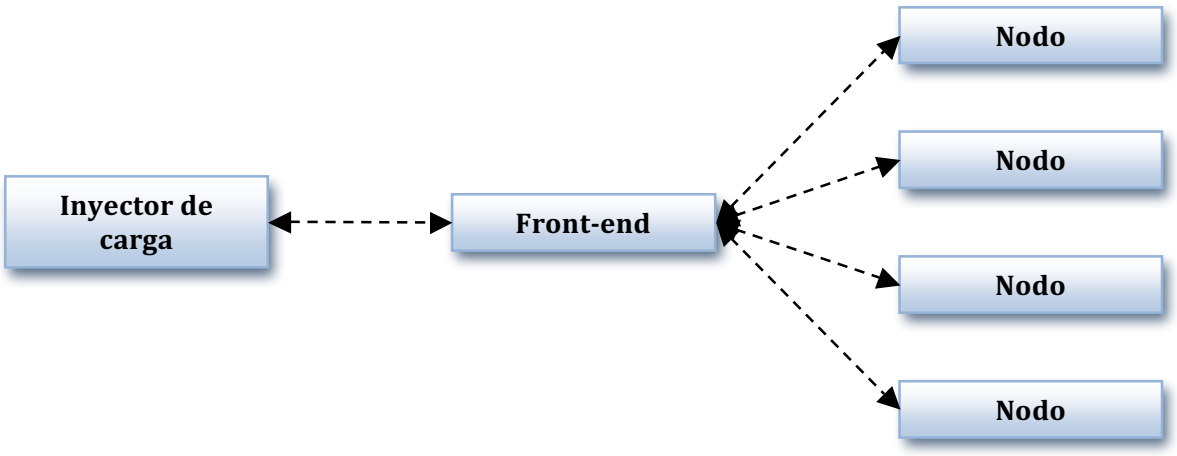


Figura 14: Diagrama del prototipo del servicio

La Figura 26 representa un esquema del servicio que modela este prototipo:

9.3.1 Modelado de sesiones

El modelado de sesiones del servidor transaccional puede enfocarse desde el punto de emular sesiones *B2B* (*Business-to-Business*) o *B2C* (*Business-to-Customer*).

Fundamentalmente, la diferencia entre ambos enfoques se encuentra en el tiempo de reflexión existente entre cada una de las peticiones, mucho mayor (y variable) en entornos *B2C*.

De cara al modelado de la sesión, pueden utilizarse las propias conexiones *TCP* y su ciclo de vida como una sesión sintética de negocio.

Estas conexiones *TCP* emularán, cada una, una transacción constituida por varias peticiones individuales, que el *front-end* repartirá entre los diferentes nodos de cálculo y éstos resolverán. Por tanto, es tarea del *front-end*, además del reparto de las peticiones individuales, el mantenimiento de los estados de las sesiones abiertas.

El número de peticiones que formará una sesión será una variable aleatoria que se describirá por una distribución de probabilidad aleatoria determinada.

El cómo se repartan estas peticiones marcará notablemente el desempeño del servicio: pueden repartirse en forma *round-robin* o en función de la carga de cada nodo.

Otra cuestión a destacar es la posible heterogeneidad de los nodos de cálculo, que hace conveniente distinguir la calidad de servicio y capacidad que proporciona cada nodo por separado, además de influir en gran medida en qué nodos provisionar cuando sea necesario.

La Figura 15 representa un diagrama de secuencia del proceso de una sesión aceptada.

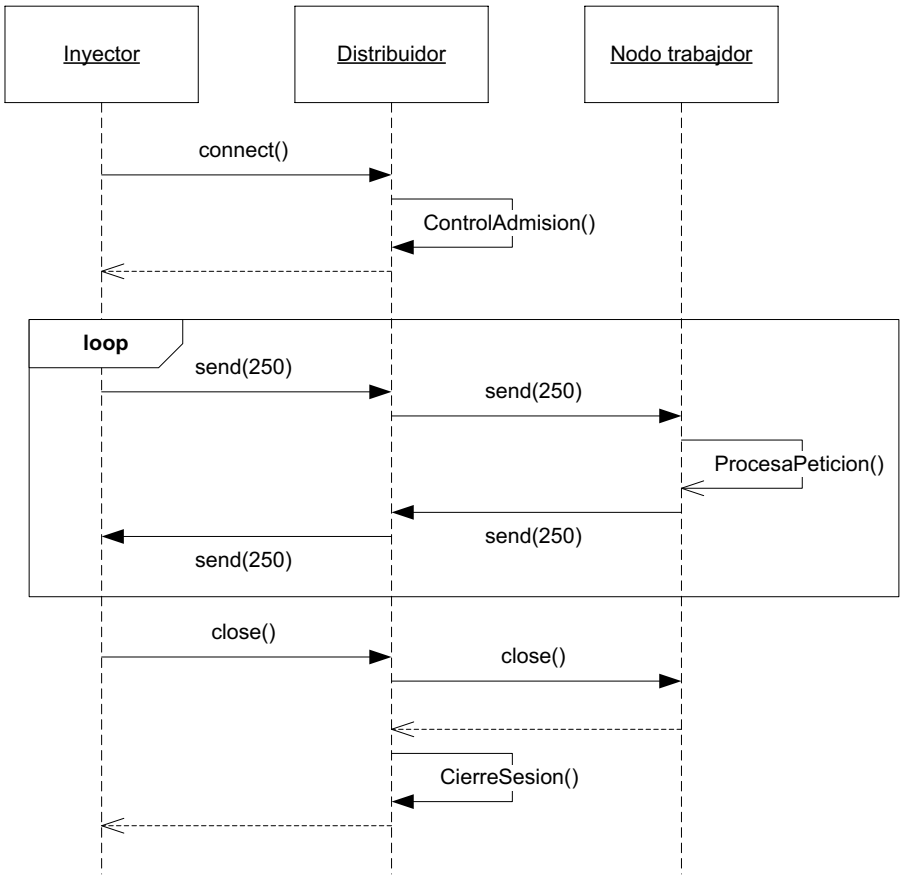


Figura 15: Diagrama de secuencia del manejo de una sesión

9.3.2 Captura y seguimiento de calidad de servicio en línea

El *front-end* o distribuidor de carga ha de ser consciente de la duración de cada una de las sesiones para determinar la calidad del servicio que está proporcionando y, en base a ella, solicitar el gestor energético la actuación sobre los nodos. Para ello se usará un buffer circular para poder tomar percentiles de calidad de servicio periódicamente (normalmente, con intervalo del orden de 10 segundos).

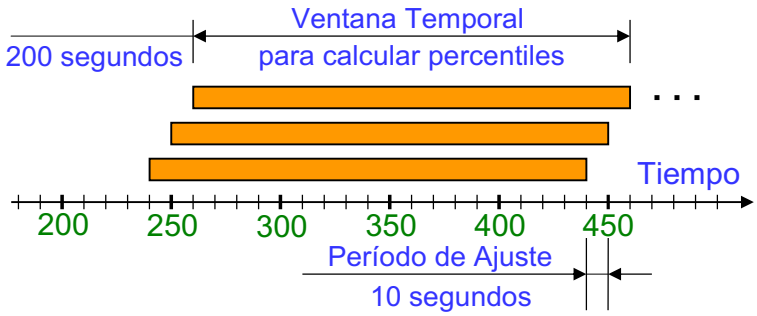


Figura 16: Diagrama del método de ventana para seguimiento de QoS

Como se explica en la Figura 16, se capturan las informaciones de todas las transacciones soportadas y, cada un intervalo predefinido, se accede a información de las transacciones de los últimos segundos. Normalmente, el tamaño de las ventanas será al menos un orden de magnitud mayor que el intervalo de ajuste.

Esto permite reducir el ruido de los datos y eliminar de forma sencilla *outliers*, además de permitir gestionar de forma sencilla los datos con diferente nivel de usuarios en el registro de datos.

En base a estos datos se ha implementado un *tracker* de *QoS* en el nodo distribuidor que permite validar que el balanceo de carga y la utilización de los nodos.

9.3.3 Especificación del servidor del *back-end*

El servidor que se ejecuta en cada uno de los nodos (back-end) es el encargado de procesar cada una de las peticiones que los clientes (emulados por el inyector) de cara realicen al sistema.

En la implementación realizada, cada una de estas peticiones que compone una sesión es idempotente e independiente a las demás. De esta forma, el concepto de sesión no llega hasta este punto y el *back-end* sólo procesará peticiones independientes, que el distribuidor de carga asignará a cada uno de los nodos en función de la planificación que determine.

El cálculo de las peticiones se emula a continuación mediante ejecución de instrucciones, lecturas y escrituras de disco en base a una distribución aleatoria de tipo uniforme limitada por los parámetros de lanzamiento del servidor indicados por el usuario.

La configuración de estos parámetros permite emular diferentes tipos de servicios, tales como un servidor Web, un servidor de base de datos o un sistema experto que realiza cálculos específicos.

Estos parámetros permiten además emular nodos de diferentes prestaciones cuando se cuenta con un clúster homogéneo para realizar las pruebas.

En el ámbito más detallado de la implementación, el servidor cuenta con un *pool* de hilos que atenderán las peticiones para dotar al servicio de un grado de paralelismo adecuado.

9.3.3.1 Generación de carga

Uno de los problemas más importantes para emular un servicio transaccional con un software sintético como el implementado es la generación de una carga realista.

En una máquina de altas prestaciones, correr un software que ejecuta instrucciones que caben de sobre en la caché (incluso en las de más alto nivel) y datos que no suponen un acceso intensivo a memoria e, incluso, a memoria virtual, hace que la emulación no represente fielmente la carga computacional de un servicio real.

Para modelar cualquier tipo de servicio y permitir generar una cantidad adecuada de carga por sesión, se ha definido cada una de las peticiones por los siguientes parámetros:

- **Consumo de CPU:** en este caso se modela por el número de instrucciones a ejecutar en cada ciclo de procesamiento de sesión.
- **Consumo de memoria principal:** permite configurar la cantidad de memoria que se necesita para procesar una petición.
- **Consumo de disco:** tanto de escritura como de lectura.
- **Consumo de red:** se define en base a la configuración del tamaño de la petición y la respuesta transmitida por la red.

En base a estos parámetros se puede modelar cualquier tipo de servicio transaccional, bastando con realizar las mediciones adecuadas para configurar el servicio sintético, que realiza el siguiente procesamiento para tratar cada una de las peticiones:

1. Recibir petición.
2. Reservar memoria principal
3. Leer datos de disco.

4. Realizar la computación adecuada.
5. Escribir datos al disco.
6. Enviar la respuesta al cliente.

9.3.4 Especificación del *front-end* o distribuidor de carga

El modulo distribuidor de carga (o *front-end* del servicio) es la parte del servicio a la que los clientes tienen acceso para realizar sus peticiones.

En este caso, un cliente realiza sesiones de servicio que se modelan con el ciclo de vida de una conexión *TCP*. Mientras la conexión esté abierta, la sesión está activa.

Para planificar la repartición de sesiones entre los nodos de cálculo que forman el clúster, se han tomado las siguientes decisiones de diseño:

- **Planificación a nivel de sesión:** todas las peticiones de cada sesión se asignarán a un nodo concreto del clúster. Esto permite una mayor flexibilidad y simplicidad a la hora de introducir la componente de gestión energética para encender y, sobre todo, apagar nodos de cálculo sin corromper peticiones. Por el contrario, reduce el nivel de granularidad con que asigna peticiones a los nodos y puede hacer que en determinadas condiciones el reparto de carga sea menos equitativo.
- **Aceptación de sesiones en paralelo:** el servicio permite atender diversas sesiones en paralelo mediante la implementación multihilo del prototipo. El distribuidor cuenta con un pool de hilos, configurado en el momento del lanzamiento de la aplicación, que atiende las sesiones en paralelo. La cola de sesiones entrantes se modela utilizando la cola de aceptación de conexiones de la librería de sockets del sistema operativo, dado que la sesión de ha modelado como una conexión TCP.
- **Asignación de peticiones en serie:** cada una de las peticiones que forma la sesión se procesará en serie y no se contempla paralelismo a nivel de sesión o *pipelining* a la hora de atender a cada uno de los clientes individualmente.

Adicionalmente, el servidor debe mantener el estado de cada una de las sesiones abiertas (mediante el mantenimiento de las sesiones *TCP*) y el estado de cada uno de los nodos.

Mantener el estado de cada uno de los nodos es crucial para repartir sesiones basándose en la carga de cada nodo (y en sus capacidades) y para permitir la integración de una política de gestión energética adecuada (ver sección 3.2).

En la implementación realizada, la admisión de sesiones supone la asignación de un nodo de cálculo en función de los parámetros de carga que presente el sistema, parámetros que se irán actualizando dinámicamente conforme el sistema avance en su funcionamiento.

La selección del hilo de servicio y la asignación de la sesión recibida a uno de los nodos de cálculo se hace de forma centralizada al aceptar la sesión, de forma que es posible ampliar la gestión con categorización de clientes y rechazo de sesiones de forma sencilla.

9.3.4.1 Mantenimiento de información relativa a los nodos

Una de las responsabilidades del distribuidor de carga es mantener información relativa a los nodos de cálculo que forman el clúster para planificar sesiones y realizar la gestión energética cuando el módulo de gestión energética se integre en el mismo.

Para satisfacer las anteriores funcionalidades, se mantiene la siguiente información sobre cada uno de los nodos:

- **Información de localización:** permite al distribuidor saber cómo llegar al nodo para asignarle peticiones para procesar (en general, con la dirección IP y el puerto de servicio será suficiente).
- **Número de sesiones asignadas:** permite realizar el reparto de carga entre los nodos de cálculo.
- **Estado energético:** debido a que ciertas configuraciones de energía hacen que un nodo no pueda recibir sesiones (apagado y apagándose, ya no se pueden recibir nuevas sesiones, pues el nodo va a cerrarse), es necesario conocer el estado de cada nodo.
- **Curvas de respuesta y potencia:** permiten realizar la estimación de las utilizaciones y de la potencia y tomar decisiones al algoritmo de optimización.
- **Tiempos de encendido y apagado:** permite realizar bloqueos y establecer temporizadores.
- **Capacidad y utilización instantánea:** permiten conocer el estado instantáneo del nodo y ser comparados con el umbral de post-apagado y de encendido.

9.3.4.2 Relaciones para la gestión energética

Para la gestión física de la energía se ha implementado un sistema basado en *WoL* y *WMI*. El primero para el encendido y el segundo para el apagado.

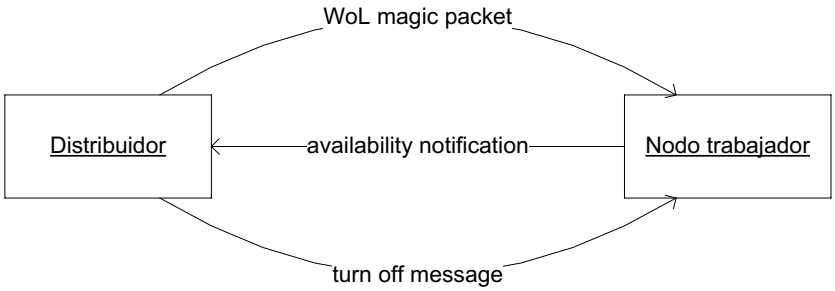


Figura 17: Diagrama de relaciones energéticas del prototipo

En el caso del apagado, basado en *WMI*, para simplificar la gestión en el distribuidor no se ha implementado *WMI Remoting*, y es el nodo trabajador quien se apaga a sí mismo tras recibir un paquete personalizado por la red en un hilo de espera de notificaciones. Cuando el nodo arranca, envía una notificación personalizada al distribuidor, que cuenta con un hilo de espera y registro.

Aunque viola la arquitectura definida en la sección 4.4, introduciendo más puntos de integración, permite simplificar el despliegue en el clúster utilizado para pruebas.

9.3.5 Especificación del inyector de carga

El inyector de carga es una parte fundamental del servicio pues permite definir modelos de carga de diferentes características de cara a las pruebas del servicio. Además, es el encargado del modelado de las sesiones, determinando el tipo de servicio en función de las características de las sesiones emuladas.

El inyector de carga permite emular diversos clientes en paralelo y configurar tiempos de arranque (para permitir estabilizar los recursos del servidor y tener medidas más precisas) y de medición.

La implementación de esta emulación de clientes se basa en un pool de hilos que lanzan sesiones concurrentes en función de los parámetros que definen una sesión: tiempo de reflexión entre sesiones y número medio de peticiones por sesión.

El número medio de peticiones por sesión se genera conforme a una distribución exponencial de media seleccionada por el usuario y permite, junto con la configuración del tiempo de reflexión entre sesiones, emular diferentes tipos de servicios, tales como un servicio bancario o de tienda en línea.

Hay que tener en cuenta que se debe de seleccionar un número de peticiones por sesión acorde al tiempo de medición seleccionado y a las prestaciones del servidor.

Para la medición, se ha instrumentado el inyector de forma que permita tomar tiempos de respuesta del servicio para valorar la calidad de servicio y obtener curvas de rendimiento y caracterización del funcionamiento del servidor.

9.3.5.1 Generación de curvas de carga personalizadas

De cara a la realización de emulaciones de servicios transaccionales, una funcionalidad muy interesante y útil es la posibilidad de generar cargas de trabajo variables en el inyector de carga.

Un ejemplo muy claro de una carga variable se da en un servicio que de soporte a terminales punto de venta de un centro comercial. En las horas de apertura de los centros, la carga será mayor que por las noches, cuando éste está cerrado.

Parece que estas diferentes cargas se pueden emular mediante el encadenamiento de simulaciones con diferentes parámetros, sin embargo:

- Aparecen transitorios entre las simulaciones que distorsionan los resultados.
- No se pueden hacer cambios de carga paulatinos (por ejemplo, generar un incremento lineal de carga).

Por tanto, debe ser el inyector de carga quien modifique dinámicamente la carga que está generando durante la emulación.

Una forma directa de emular la carga es modificar el número de usuarios que están accediendo concurrentemente al servicio, que en el inyector viene emulado por el número de hilos *activos* en cada momento.

El concepto de hilo activo es importante de cara a la implementación de la generación de curvas de carga personalizadas, pues se relaciona directamente con la activación y desactivación de hilos en tiempo de ejecución. De esta forma, el inyector irá redimensionando el pool de hilos

activos en cada momento para adecuarse a los objetivos de carga de la emulación en dicho instante.

9.3.5.1.1 Temporización de eventos de cambio de carga

En primer lugar, es conveniente definir cuándo se van a producir los eventos que alteren la carga del sistema para permitir emular las curvas objetivo.

- **Temporización discreta:** se establece un temporizador que expira a intervalos iguales de tiempo.
- **Temporización orientada a eventos:** en este caso, la temporización va dirigida por eventos, que sucederán en intervalos de tiempo variable.

El uso de temporización discreta permite simplificar la gestión, pero introduce una sobrecarga inútil en el caso del manejo de regímenes estacionarios de carga de larga duración.

9.3.5.1.2 Descripción de objetivos de carga

A continuación, es necesario establecer una forma para describir los objetivos de carga que se establecerán para cada instante de la emulación. Existen dos formas fundamentales de abordar el problema:

- **Descripción continua:** se describe la curva de carga mediante una función matemática que responde en cada momento a la carga objetivo.
- **Descripción discreta:** se describen los instantes de cambio de carga, indicando el instante de tiempo donde se produce el cambio de carga y el valor de la nueva carga.

La primera alternativa es la que mayor precisión aporta, además de ser compacta en la descripción. Esta alternativa, combinada con una temporización discreta permite describir curvas complejas de forma sencilla.

Sin embargo, si van a emularse regímenes estacionarios y variaciones sencillas de carga, una descripción discreta combinada con una temporización orientada a eventos simplifica la gestión y reduce la sobrecarga de la temporización discreta.

9.3.5.1.3 Arquitectura de control de la temporización y el cambio de carga

Sin duda, la generación de curvas de carga personalizadas requiere del manejo de temporización, sea cual sea la forma de descripción.

Para controlar la temporización y realizar los cambios de carga, teniendo en cuenta la arquitectura multihilo del inyector de carga, pueden considerarse dos aproximaciones a la solución:

- **Usar una arquitectura centralizada:** en este modelo, un hilo sería el responsable de mantener los instantes de cambio y de activar y desactivar los hilos que emulan las sesiones.
- **Usar una arquitectura distribuida:** por el contrario, en este modelo, el pool de hilos que emulan los usuarios concurrentes se auto-gestionaría en la activación y desactivación de hilos. Un hilo debería llevar control de los tiempos.

Ambas alternativas se comportan muy bien cuando los cambios de carga son de incremento, pues la activación de los hilos es instantánea. Sin embargo, de cara a una reducción de carga, pueden producirse efectos de "amortiguamiento" debido al tiempo que tardan en finalizar las sesiones.

Al desactivar un hilo, éste debe de concluir la sesión que está gestionando antes de pasar a estado inactivo. Esto supone que los primeros hilos que hay que seleccionar para desactivar son los que primero vayan a terminar la sesión activa para minimizar el efecto de amortiguamiento y alcanzar el objetivo de carga lo antes posible.

En el modelo centralizado esto supone predecir cuál de los hilos va a ser el primero en concluir su sesión, para seleccionarlo como hilo a desactivar. Se pueden utilizar numerosos algoritmos para, basándose en el tiempo de sesión transcurrido y el tiempo medio, predecir cuál de los hilos va a terminar en primer lugar.

Sin embargo, el modelo distribuido permite resolver este problema de una forma mucho más elegante y sencilla. La aproximación al problema es la inversa de la anterior: en lugar de predecir qué hilo va a terminar primero la sesión activa, que sea el propio hilo el que determine si es el primero o no.

En este modelo, cada hilo es responsable (distribuidamente) de ajustar la carga actual al objetivo en el instante dado. Ello hace necesario poder determinar en cualquier momento la siguiente información:

- El objetivo de carga para el momento actual.
- La carga que se está generando en este instante.

Como se puede deducir, esta información puede ser descrita por dos variables de tipo entero, lo que da sencillez al método. El mantenimiento de la carga objetivo es responsabilidad del hilo encargado de la temporización, que la actualizará conforme al modelo de temporización y descripción seleccionado.

En el instante en el que acaba una sesión, el hilo que la está gestionando siempre es el primero en terminar una sesión. Aunque esto pueda parecer evidente, es lo que permite asegurar que este modelo ajusta la carga lo más rápido posible a la curva objetivo, minimizando el efecto de amortiguamiento.

Sin embargo, surge un nuevo problema cuando la modificación del objetivo carga hace que haya que incrementar la carga actual, es decir, en el momento de la terminación de una de las sesiones, uno de los hilos que modelan un cliente debe activar uno o más hilos que se encuentran suspendidos. El problema consiste en determinar cuántos de los hilos activar.

Si se decide activar todos los hilos necesarios para alcanzar el objetivo de carga se producirán incrementos de carga muy rápidos que impedirán modelar curvas de carga suaves, por el contrario, si se decide activar de uno en uno, se tardará demasiado tiempo en alcanzar el objetivo de carga marcado.

De esta forma, se ha implementado un *factor de amortiguación*, que permite configurar el parámetro de la rapidez del incremento de carga, de forma que el prototipo sea flexible de cara a realizar emulaciones de diferente naturaleza.

Si se configura el parámetro al valor 1, se activarán todos los hilos necesarios para alcanzar el objetivo de carga, permitiendo realizar incrementos bruscos de carga. El valor mínimo es 0, de forma que se hagan incrementos de un hilo, valor adecuado en el caso de definir emulaciones con pendientes suaves.

9.3.6 Características generales de la implementación del servicio

Además de las características específicas de cada módulo del servicio, cabe destacar los siguientes detalles comunes de la implementación:

- Todo el código fuente es conforme al estándar *ANSI C99*.

- Tanto el distribuidor de carga como el servidor *back-end* utilizan una implementación basada en un pool de hilos que actúan sobre un socket *TCP*.
- Se ha usado temporización orientada a eventos y descripción discreta en el inyector de carga. Para la generación de curvas personalizadas, se ha implementado el modelo distribuido.

9.4 Prototipo del gestor de energía

El gestor de energía se encargará de monitorizar la carga de los diferentes nodos para tener una idea global de la carga total del clúster.

Las funciones que, a grandes rasgos, debe proporcionar el gestor son las que siguen:

- Mantener información sobre la topología del clúster, es decir, sobre sus nodos (tipo, número, características, nombre…).
- Asignar estados a los nodos del clúster.
- Enviar órdenes de apagado y/o encendido a los nodos, actualizando la topología y la monitorización del clúster.

Como se definió anteriormente, no es necesario mantener un seguimiento de la utilización de los dispositivos de cada nodo, pues todo está recogido en el modelo de calidad de servicio y en la utilización de los nodos.

En el caso de la implementación del algoritmo diseñado, éste se ha integrado en el código del distribuidor de carga, de manera que sólo dos puntos de acceso son necesarios para su operación: al realizar el control de admisión y al cerrar una sesión.

9.5 Resumen de especificación y diseño

Tras la exposición de diferentes opciones de diseño e implementación, se ofrece un resumen de las decisiones tomadas para el prototipo, así como de los requisitos de software para el mismo. De esta forma, esta sección puede considerarse un manual de desarrollo en caso de desear extender el software.

9.5.1 Requisitos del prototipo

Antes de comenzar con la exposición de requisitos del sistema, es conveniente indicar cómo el objetivo de dicho prototipo es satisfacerlos de forma que su desarrollo sea rápido y permita dedicar más recursos temporales a las pruebas y validación del propio algoritmo.

9.5.1.1 Requisitos del inyector de carga

A continuación figura la lista de requisitos funcionales y no funcionales para el inyector de carga. De esta forma esta pieza de software:

R.I.1. Ha de ser capaz de emular el comportamiento de un número variable de clientes de forma concurrente.

R.I.2. Debe ser capaz de generar diferentes niveles de carga, definidos mediante una función de carga, para cada una de las emulaciones ejecutadas.

R.I.3. Cada una de los tiempos de reflexión ha de configurarse mediante una función de distribución exponencial, de forma que puedan emularse servicios *B2C* y *B2B*.

R.I.4. Debe permitir configurarse en cada ejecución de forma diferente, modificando el número de clientes, tiempo de prueba y parámetros estadísticos que definen la emulación.

R.I.5. Debe utilizar el formato de sesiones *TCP* definido en la sección 9.3.1.

R.I.6. Ha de almacenar información de log relativa a la simulación ejecutada, de forma que se almacenen los tiempos de respuesta para cada transacción y se emitan unos resultados resumidos al finalizar la ejecución.

R.I.7. El inyector ha de ser capaz de tratar los rechazos de sesión y registrarlos como información de log.

R.I.8. El orden de concurrencia que debe soportar se situa en el intervalo de 1000 – 5000 sesiones concurrentes.

9.5.1.2 Requisitos del distribuidor de carga

De forma similar al inyector, los requisitos para el distribuidor de carga son:

R.R.1. Ha de ser capaz de servir sesiones a clientes de forma concurrente en el mismo puerto de servicio.

R.R.2. Cada sesión es idempotente de las demás, no existiendo priorización.

R.R.3. El servicio debe ser transparente a la estrucutra interna del clúster de computación.

R.R.4. El distribuidor de carga deberá almacenar la información relativa los puntos definidos en la sección 9.3.4.1.

R.R.5. El distribuidor de carga debe de realizar un log de todas y cada una de las sesiones procesadas.

R.R.6. Adicionalmente, se deben registrar métricas de calidad de servicio y su evolución durante la prueba utilizando un método de ventana con tamaño e intervalo configurable.

R.R.7. Estas métricas de calidad de servicio deben incluir una estimación de la potencia instantántea y la energía consumida (en W y Wh respectivamente).

R.R.8. Las sesiones tendrán el formato definido anteriormente y se basarán en el ciclo de vida de una conexión *TCP*.

R.R.9. Todas las respuestas pasarán por el distribuidor de carga.

R.R.10. El orden de concurrencia que debe soportar se situa en el intervalo de 1000 – 5000 sesiones concurrentes.

R.R.11. El distribuidor deberá implementar el algoritmo de gestión de energía diseñado en el resto del proyecto[19].

R.R.12. La planificación del algoritmo será a nivel de sesión.

R.R.13. El distribuidor de carga ha de ser capaz de enviar mensajes de encedido usando el protocolo WoL, tal y como se detalla en la sección 9.3.4.2.

[19] Nótese como este requisito es extremadamente genérico, pero permite el diseño y validación incremental del algoritmo conforme se desarrolla el prototipo.

R.R.14. El distribuidor de carga ha de ser capaz de enviar mensajes de apagado usando un protocolo propio o basado en *WMI-Remoting*.

9.5.1.3 *Requisitos del servidor de cálculo o* back-end

Finalmente, se enumeran los requisitos del servidor de *back-end*:

R.S.1. El servidor back-end deberá poder servir sesiones de forma idempotente y concurrente.

R.S.2. Permitirá configurar los parámetros indicados en la sección 9.3.3.1 con la mayor granularidad que tenga sentido para cada uno de ellos.

R.S.3. El orden de ejecución de las acciones de procesamiento asociadas a una sesión será el definido en la sección 9.3.3.1.

R.S.4. El servidor ha de ser capaz de notificar su disponibilidad tras ecenderse.

R.S.5. El orden de concurrencia que debe soportar se situa en el intervalo de 100 – 1000 sesiones concurrentes.

9.5.2 Metodología de análisis y diseño

En lo relativo al diseño, aparte lo ya comentado en las secciones anteriores, se ha implementado mediante el paradigma de programación estructurada, de forma que se simplifica la programación en un entorno de prototipos.

Se ha optado por realizar un desarrollo incremental, integrando el propio análisis y diseño en pequeñas fases en cada iteración de desarrollo de código. Cada una de las iteraciones de desarrollo se ha basado en un modelo ágil, donde se discutía de forma informal mediante un *brainstorming* en el equipo del proyecto las opciones y posteriormente se diseñaba la opción más conveniente en tiempo y recursos, para posteriormente implementarla y probarla y validarla con el algoritmo.

Debido a que el algoritmo se ha diseñado también de forma incremental, este modelo es ideal para la integración constante y pruebas de validación continuas.

9.5.3 Tecnologías de implementación

Todo el código está basado en el lenguaje de progamación C99, sobre el API Win32. Es necesario indicar cómo hay una alta dependencia con dicho API, lo que resta la portabilidad del código. No obstante, esto permite obtener buen rendimiento en las máquinas del laboratorio de pruebas y realizar configuraciones avanzadas en los sistemas operativos Windows donde se ha ejecutado la experimentación.

Para la emulación de clientes se ha optado por desplegar un *pool* de hilos que emulan cada uno un cliente y que permite la generación de carga de forma dinámica.

9.5.4 Planificación y desarrollo de pruebas de software

Las pruebas del software se han integrado en el desarrollo y validación del algoritmo, tal y como se ha explicado en detalle en la sección 7.

Continuando con la metodología de desarrollo incremental, para las pruebas se ha realizado de la misma forma, con fases de validación de software y del algoritmo en cada iteración del desarrollo.

9.6 Manual de usuario

En esta sección se resume de forma práctica el modo de operación de los tres componentes de software, para aquellos que deseen ejecutar pruebas con ellos.

9.6.1 Manual del inyector

Normalmente, el inyector es el último componente en ser lanzado, de forma que, una vez ejecutado el servicio, se procede a emular clientes y a tomar medidas de rendimiento. Se trata de una aplicación de consola con los siguientes parámetros:

```
DIR_SERVIDOR        - dirección de la maquina con el servicio (normalemente, el distribuidor de
                      carga., aunque puede medirse directamente el servidor SSIF)
```

```
PUERTO              - puerto de servicio del servidor
NUM_CLIENTES        - numero de clientes a emular (numero de hilos concurrentes)
TIEMPO_REFLEXION    - tiempo medio de reflexion (en segundos, distribución exponencial)
INTERVALO_ARRANQUE  - tiempo (en segundos) del transitorio de arranque (puede ser 0)
TIEMPO_MEDICION     - tiempo (en segundos) de la medición
FICHERO_VOLCADO     - nombre para el fichero de volcado del log
PETICIONES          - media del numero de peticiones por sesión (distribución exponencial)
TAM_PETICION        - tamaño de la peticion y respuesta (en bytes, ha de ser igual en los tres
                      servicios)

FICHERO_CARGA (opc) - fichero con las curvas de carga (ver especificación en la sección 9.6.1.1)
AMORTIGUACION (opc) - factor de amortiguacion para el incremento de carga. Por omision se toma 0,5
```

9.6.1.1 Especificación de carga

Para especificar la carga se dispone del fichero de configuración de carga, proporcionado de forma opcional al inyector, con el formato de una tupla de dos valores separados por un espacio en cada línea.

Esta tupla define un intervalo de carga, donde el primer valor determina el número de segundos a mantenerlo desde el establecimiento del mismo y el segundo, el número de clientes objetivo.

Se trata de un número de clientes objetivo, pues una vez establecido dicho objetivo, en función del factor de amortiguación establecido, éste se alcanzará en mayor o menor tiempo.

9.6.2 Manual del distribuidor

El distribuidor de carga se lanza en una máquina destinada a tal efecto, o dónde está corriendo otro servidor de cálculo (nótese el impacto en el rendimiento que esto tiene). Los parámetros para el lanzamiento del distribuidor son los que siguen:

```
PUERTO              - puerto de servicio para atender peticiones
POOL_HILOS          - tamano del pool de hilos de servicio (recomendable que sea mayor que el
                      número de clientes a emular en el inyector)
FICHERO_CONF        - fichero de configuración (ver especificación en la sección 9.6.2.1)
INTERVALO_QOS       - intervalo de captura de informacion QoS (segundos)
TAM_QOS             - limite al espacio de captura de informacion QoS (normalmente, el número de
                      hilos multiplicado por 10 y por el tamaño del intervalo de QoS)
TAM_PETICION        - tamano de la peticion y respuesta (en bytes, ha de ser igual en los tres
                      servicios)
LOG                 - prefijo del nombre de los ficheros de log (se generan tres ficheros con
```

```
el mismo prefijo configurable y sufijo determinado por tipo de log)
```

9.6.2.1 Especificación de servidores de cálculo

Para especificar los servidores y los parámetros de cada uno de los servidores se utiliza un fichero de configuración en formato INI, con la sintaxis que sigue (los indicadores entre ‹› son marcas de tokens y los marcados con {} comentarios):

```
[globals]
uon = <flotante, 0.0 – 1.0>
uoff = <flotante, 0.0 – uon>
<servidor>+

<servidor> :=
        [<id único, entero, 0..>]
        hostname = <nombre dns o dirección IP>
        puerto = <0..65534>
        mac = <dirección MAC, 6 octetos hexadecimales>
        estado = <código de estado, ver sección 4.3.1>
        respuesta0 = <flotante>    {coeficiente de grado 0 para el tiempo respuesta}
        respuesta1 = <flotante>    {coeficiente de grado 1 para el tiempo respuesta}
        respuesta2 = <flotante>    {coeficiente de grado 2 para el tiempo respuesta}
        respuesta3 = <flotante>    {coeficiente de grado 3 para el tiempo respuesta}
        potencia3 = <flotante>     {coeficiente de grado 3 para el tiempo potencia}
        potencia2 = <flotante>     {coeficiente de grado 2 para el tiempo potencia}
        potencia1 = <flotante>     {coeficiente de grado 1 para el tiempo potencia}
        potencia0 = <flotante>     {coeficiente de grado 0 para el tiempo potencia}
        sla = <flotante, > 0.0>
        timeoutenc = <flotante, > 0.0>  {timeout de apagado para el nodo, segundos}
        timeoutapa = <flotante, > 0.0>  {timeout de apagado para el nodo, segundos}
```

Por ejemplo, un fichero de configuración válido podría ser el siguiente:

```
        [globals]
        uon = 0.9
        uoff = 0.7

        [0]
        hostname = 192.168.1.10
        puerto = 7000
        mac = D8:D3:85:22:A5:98
        estado = 2
        respuesta0 = 159.460572
        respuesta1 = 0
        respuesta2 = 0.041908
        respuesta3 = 0.00236
```

```
potencia3 = 0.000006
potencia2 = -0.0046
potencia1 = 1.09
potencia0 = 115.89
sla = 2000
timeoutenc = 120
timeoutapa = 120

[1]
hostname = 192.168.1.11
puerto = 7000
mac = D8:D3:85:22:A5:98
estado = 2
respuesta0 = 159.460572
respuesta1 = 0
respuesta2 = 0.041908
respuesta3 = 0.00236
potencia3 = 0.000006
potencia2 = -0.0046
potencia1 = 1.09
potencia0 = 115.89
sla = 2000
timeoutenc = 120
timeoutapa = 120
```

9.6.3 Manual del servidor de procesamiento

El servidor de procesamiento puede ser configurado de la siguiente forma:

```
USO_CPU             - consumo de CPU en bloques de 10K instrucciones
LECT_DISCO          - lectura de disco por peticion (KiB)
ESCR_DISCO          - lectura de disco por peticion (KiB)
USO_MEMORIA         - tamano de la reserva de memoria (KiB)
MEDIA_VISITAS       - numero medio de visitas por petición (distribución exponencial)
NUM_HILOS           - tamano del pool de hilos de servicio
DIRECTORIO          - directorio con las carpetas Lect y Escr
PUERTO              - puerto de servicio
TAM_PETICION        - tamano del mensaje de peticion y respuesta (bytes)
NUM_FICHEROS        - numero de ficheros de lectura en Lect
TAM_FICHERO         - tamano de los ficheros de lectura (KiB)
PUERTO_NOTIFICACION - puerto de notificacion de disponibilidad del distribuidor (normalmente, el
                      puerto de servicio del distribuidor + 1)

-c (opc)            - desactiva el cache del sistema de ficheros
-v (opc)            - activa el modo de traza
```

9.6.4 Consideraciones prácticas

El factor de amortiguación en la generación de carga, de cara a obtener curvas suaves, ha de establecerse a un valor pequeño, del orden de 0,01. Aunque este valor depende de la configuración de la carga y del tamaño de los saltos.

El distribuidor de carga, cuando soporta muchos clientes despliega muchos sockets ($2 \cdot clientes + 1$), por lo que es recomendable revisar los ajustes del sistema operativo para comprobar si están disponibles tantas conexiones. Todos los sockets tienen la configuración de *linger* adecuada, pero una configuración de sesión con pocas transacciones o de muy poca duración puede incrementar la tasa de creación de conexiones por encima de los límites del sistema. En este caso, es recomendable reducir el tiempo del temporizador TIME_WAIT.

Desactivar la caché del sistema de ficheros en el servidor incrementa mucho (dos órdenes de magnitud el uso del disco).

9.7 Limitaciones del prototipo desarrollado

Fundamentalmente, las limitaciones están relacionadas con la optimización y escalabilidad del software. El prototipo usa una aproximación de fuerza bruta basada en navegación para calcular los umbrales y seleccionar los nodos y estados en los que se debe operar.

En una implementación final es necesario cambiar el modelo a un sistema de gestión de conjuntos, de forma que pueda manejar miles de nodos de forma simple y rápida.

9.8 Obtención del código fuente del prototipo

Para la obtención del código fuente del prototipo, pese a que un *snapshot* a fecha de impresión de este documento del mismo se encuentra impreso como anexo y adjunto en el CD que acompaña a esta memoria, es recomendable acceder al mismo usando el repositorio SVN donde está alojado, dado que el proyecto que continuará siendo utilizado y mejorado. Para acceder, puede usarse la siguiente URI de esquema `svn`:

```
svn://rigel.atc.uniovi.es/srv/subversion/pfcs/joaquin/10power/
```

10 GESTIÓN DEL PROYECTO

En esta sección se resumen los aspectos de gestión del proyecto, tales como el presupuesto del desarrollo y la planificación del mismo.

10.1 Presupuesto de ejecución

El presupuesto de licitación del proyecto es el que sigue:

Tabla de precios			
ID	**Concepto**	**Precio**	**Unidad**
P.II	Ingeniero informático	40,00	€/hora

Tabla de medidas			
ID	**Concepto**	**Cantidad**	**Unidad**
Fase: Planificación y lanzamiento			
M.P.1	Planificación y lanzamiento	160	horas-persona
Fase: Revisión del estado del arte			
M.A.1	Análisis de alternativas	160	horas-persona
M.A.2	Revisión de resultados	320	horas-persona
Fase: Implementación básica			
M.IB.1	Análisis inicial	160	horas-persona
M.IB.2	Implementación inicial	240	horas-persona
M.IB.3	Validación inicial	80	horas-persona
Fase: Diseño del algoritmo			
M.D.1	Diseño básico	480	horas-persona
M.D.2	Diseño de aspectos avanzados	320	horas-persona
M.D.4	Validación teórica	160	horas-persona
Fase: Implementación de funciones avanzadas			
M.IA.1	Análisis de funciones	160	horas-persona
M.IA.2	Implementación de funciones	160	horas-persona

M.IA.3	Validación	80	horas-persona
Fase: Validación			
M.V.1	Diseño experimental	80	horas-persona
M.V.2	Pruebas homogéneos	80	horas-persona
M.V.3	Pruebas heterogéneos	160	horas-persona
M.V.4	Análisis de resultados	80	horas-persona
Fase: Revisión			
M.R.1	Ejecución de pruebas	80	horas-persona
		2960	**horas-persona**

Capítulo 1: Desarrollo de software				
ID	**Concepto**	**Cantidad**	**Precio**	**Importe**
1.1	Fase: Planificación	160	40,00	6.400,00
1.2	Fase: Revisión de alternativas	480	40,00	19.200,00
1.3	Fase: Implementación básica	480	40,00	19.200,00
1.4	Fase: Diseño del algoritmo	960	40,00	38.400,00
1.5	Fase: Implementación avz.	400	40,00	16.000,00
1.6	Fase: Validación	400	40,00	16.000,00
1.7	Fase: Revisión	80	40,00	3.200,00
			Subtotal	**118.400,00 €**

Capítulo 2: Equipos y materiales				
ID	**Concepto**	**Cantidad**	**Precio**	**Importe**
2.1	Clúster AMD	4	1.800,00	7.200,00
2.2	Clúster Intel	4	2.300,00	9.200,00
2.3	PC de escritorio	1	1.000,00	1.000,00
			Subtotal	**17.400,00 €**

RESUMEN DEL PRESUPUESTO			
ID	**Capítulo**		**Importe**
R.1	Capítulo 1: Desarrollo de software	-	118.400,00
R.2	Capítulo 2: Equipos y materiales	-	17.400,00
R.3	Gastos generales	19%	25.802,00
R.4	Beneficio industrial	6,00%	8.148,00
R.5	IVA	18,00%	30.555,00
P.L	**Presupuesto de licitación**		**200.305,00 €**

Asciende el presente presupuesto de licitación por proyecto a dos cientos mil tres cientos cinco euros. En Oviedo, a 22 de Marzo de 2011, fdo. el/la responsable.

10.2 Planificación y seguimiento

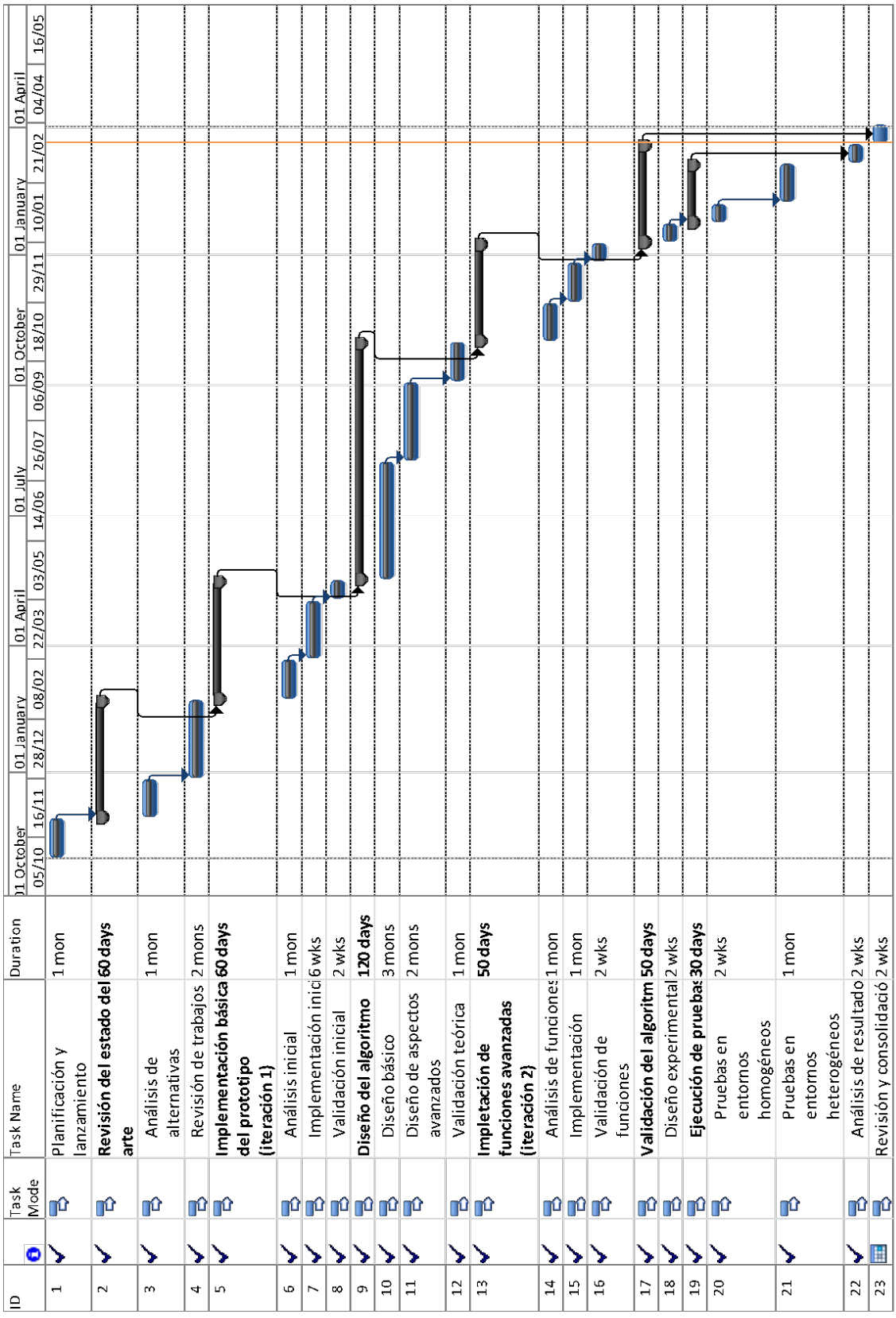


Figura 18: Estado de la planificación del proyecto a 22 de Marzo de 2011

11 CARACTERÍSTICAS DE LOS EQUIPOS DE PRUEBAS

El servidor utilizado para las pruebas de todo el desarrollo es un clúster con las siguientes características técnicas. En primer lugar, se han utilizado unos nodos con procesador Intel de última generación (basados en la microarquitectura Nehalem) y con alto grado de paralelismo. También cuentan con una gestión de energía moderna:

Número de nodos:	4
Tipo del nodo:	HP ProLiant DL380 G6
Número de procesadores/nodo:	2
Número de núcleos/nodo:	8
Número de hilos/nodo:	16
Tipo de procesador:	Intel Xeon L5520
Frecuencia:	2,26 GHz
Caché por procesador:	8 MiB (L3)
Tecnología de fabricación:	45 nm
Cantidad de transistores:	731 M (263 mm^2)
Consumo energético:	60 W (TDP)
Memoria RAM por nodo:	16 GiB
Tipo de memoria:	DDR3-1066
Almacenamiento:	SAS
Conectividad LAN:	Gigabit Ethernet (1000 Mpbs)

Tabla 7: Resumen de características del clúster de pruebas (nodos Intel)

Adicionalmente, y con propósito de probar el sistema bajo condiciones de heterogeneidad, se han utilizado unos nodos con procesador AMD que cuentan con una menor potencia de cálculo y una gestión energética más primitiva:

Número de nodos:	4
Tipo del nodo:	HP ProLiant DL145 G2
Número de procesadores/nodo:	2
Número de núcleos/nodo:	2
Número de hilos/nodo:	4
Tipo de procesador:	AMD Opteron 265 [AMD06]
Frecuencia:	1,80 GHz

Caché por procesador:	1+1 MiB (L2)
Tecnología de fabricación:	90 nm
Cantidad de transistores:	233 M (199 mm^2)
Consumo energético:	95 W (TDP)
Memoria RAM por nodo:	2 GiB
Tipo de memoria:	DDR-400
Almacenamiento:	SATA
Conectividad LAN:	Gigabit Ethernet (1000 Mpbs)

Tabla 8: Resumen de características del clúster de pruebas (nodos AMD)

12 GLOSARIO

Business-to-Business (B2B): servicio transaccional pensado para ser utilizado entre empresas. El interlocutor esperado es otro servidor B2B de otra organización que realiza paquetes de trasacciones de forma continua, por esto, esta clase de servicios se caracterizan por carecer de tiempo de reflexión o ser éste muy reducido.

Business-to-Customer (B2C): servicio transaccional orientado al uso parte de los consumidores finales. Fundamentalmente, la diferencia principal con los servicios B2B es la existencia de un tiempo de reflexión muy elevado.

Calidad de Servicio (QoS): tecnologías que garantizan la transmisión de datos en un cierto tiempo límite. En términos de servicios hace referencia a la provisión del servicio dentro de un tiempo razonable según el tipo y prioridad del cliente.

Capacidad: máxima cantidad de clientes simultáneos que soporta un computador o, en su caso, un clúster de forma combinada, de forma que se garantiza el cumplimiento de un determinado acuerdo de nivel de servicio (*SLA*, *Service Level Agreement*).

Carga inyectada: cantidad de sesiones emuladas desde el inyector de carga en los experimentos de carga. Esta cantidad puede ser mayor que la capacidad del servicio, en cuyo caso, para garantizar el cumplimiento del SLA, ha de rechazarse cierta cantidad de la misma.

Carga soportada o efectiva: cantidad de sesiones soportadas en el servicio en un instante dado. Si esta carga es menor a la inyectada, se están produciendo rechazos, debido a que se está solicitando más cantidad de carga que la capacidad del servicio, por tanto la carga efectiva ha de ser siempre menor o igual que dicha capacidad.

Clúster: conjunto de ordenadores construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora.

Distribuidor de carga: (ver Repartidor de carga) software que se engarga de recibir las peticiones de apertura de sesión de los clientes externos y, generalmente, las transacciones que

componen cada sesión y asignarlas a cada nodo de servicio que hará el procesamiento. Normalmente este software se ejecuta en una máquina dedicada a tal efecto. La respuesta a cada transacción puede pasar por este nodo o, cada nodo de cálculo, responder directamente al cliente.

Histéresis: en el análisis de algoritmos como el presentado, la histéresis es el retardo que tiene el mismo a los estímulos que los deben hacer variar. Por ejemplo, el algoritmo presentado, en casos de incremento de carga, toma decisiones hasta que uno de los umbrales se supera, introduciendo un retardo en la toma de decisiones efectiva.

Repartidor de carga: en este contexto, es sinónimo de Distribuidor de carga.

Sesión: en este contexto, una sesión es un conjunto de tareas solicitadas por un cliente en el servicio. Está constituda por un conjunto de una o más transacciones, ejecutadas en orden y secuencialmente.

Sistema transaccional: sistema informático que procesa información en unidades de trabajo denominadas transacciones. Cada transacción debe dejar el sistema en un estado consistente, para lo que debe tolerar fallos y ser capaz de recuperarse de los que se produzcan en mitad de la ejecución de una transacción.

***SLA* (*Service Level Agreement*):** acuerdo entre el prestador del servicio y el cliente por el que se definen las características del servicio a proveer. Además de restricciones de rendimiento mínimo, pueden incluirse cláusulas de todo tipo, como garantía de requisitos funcionales.

Transacción: unidad mínima de procesamiento parte de una sesión. Comienza con una petición por parte del cliente y termina con una respuesta desde el servicio, incluyendo el procesamiento entre ambas. Esta respuesta puede pasar a través del distribuidor de carga o proceder directamente desde el nodo de cálculo adecuado.

13 Referencias

[Abts10] Energy proportional datacenter networks. Abts, D., Marty, M.R., Wells, P.M., Klausler, P. and Liu, H. Proceedings of the 37th annual international symposium on Computer architecture. 338—347 (2010)

[ACPI09] Advanced Configuration and Power Interface Specification. Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation. 2009. 2009. Revision 4.0.

[Aikebaier09] Distributed Cluster Architecture for Increasing Energy Efficiency in Cluster Systems. 2009. International Conference on Parallel Processing Workshops. pp. 470-477. 1530-2016

[AMD06] AMD Opteron Processor Power and Thermal Data Sheet. Advanced Micro Devices, 2006. Publication #30417.

[Amirijoo07] Toward adaptive control of QoS-importance decoupled real-time systems. Amirijoo, M., Brännström, P., Hansson, J., Gunnarsson, S. and Son, S.H. IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (2007)

[Andrew09] Optimal speed scaling under arbitrary power functions. Andrew, L.L.H., Wierman, A. and Tang, A. ACM SIGMETRICS Performance Evaluation Review, 37(2). 39—41 (2009)

[Bai07] Measurement and modelling of the effective sleep time interval for dynamic power management of PCs. B ai, Y.W. and Tsai, C.H. IEEE Transition 551. 30—121 (2007)

[Barroso03] Web search for a planet: The Google cluster architecture. Barroso, Luiz André, Dean, Jeffrey and Hölzle, Urs. 2003. IEEE Micro. 23(2), pp. 22-28.

[Barroso05] The price of performance. Barroso, Luiz André, 2005. ACM Queue, 3(7). pp. 48-53. 1542-7730.

[Barroso07] The case for energy-proportional computing. Barroso, Luiz André, Holzle, Urs. IEEE Computer, 12(40). 2007. pp. 33-37. 0018-9162.

[Behle08] IBM EnergyScale for POWER6 Processor-Based Systems. Behle, Brad, et al. 2008. Systems and Technology Group, IBM Corporation. Somers, New York : s.n., 2008.

[Berard07] Windows Server Power Management. Berard, Stephen. 2007. Los Angeles : Microsoft Corporation, 2007. WinHEC.

[Bertini07] Statistical QoS guarantee and energy-efficiency in web server clusters. Bertini, L., Leite, JCB and Mossé, D. 19th Euromicro Conference on Real-Time Systems, 2007. ECRTS'07. 83—92 (2007)

[Bertini10a] Power optimization for dynamic configuration in heterogeneous web server clusters. Bertini, Luciano, Leite, Julius C.B., Mossé, Daniel. 2010. Journal of Systems and Software, 83(4). pp. 585-598. 0164-1212.

[Bertini10b] Power and performance control of soft real-time web server clusters. Bertini, Luciano, Leite, Julius C.B., Mossé, Daniel. 2010. Information Processing Letters. 0020-0190.

[Bianchini04] Power and Energy Management for Server Systems. Bianchini, Ricardo and Rajamony, Ram. 2004. 11, Los Alamitos, U.S. : IEEE Computer Society, Noviembre 2004, Computer, Vol. 37, pp. 68-74. 0018-9162.

[Blanquicet08] Managing energy use in a network with a new SNMP Power State MIB. Blanquicet, Francisco and Christensen, Ken. 2008. Montreal : s.n., 2008. 33rd IEEE Conference on Local Computer Networks. pp. 509-511. 978-1-4244-2412-2.

[Bohrer01] Energy Conservation for Servers. Bohrer, P., Cohn, D., Elnozahy, E., Keller, T., Kistler, M., Lefurgy, C., Rajamony, R., Rawson, F. and Hensbergen, EV. IEEE Workshop on Power Management for Real-time and Embedded Systems (2001)

[Cai06] Joint Power Management of Memory and Disk Under Performance Constraints. Cai, L. and Pettis, N. and Lu, Y. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 25(12). 2697 (2006)

[Carrera03] Conserving Disk Energy in Network Servers. Carrera, E.V. and Pinheiro, E. and Bianchini, R. Proceedings of the 17th annual international conference on Supercomputing. 97 (2003)

[Case90] A Simple Network Management Protocol (SNMP). Case, Jeffrey D., et al. 1990. Network Working Group, Internet Engineering Task Force. s.l. : RFC Editor, 1990. Request For Comments. 1157.

[Chase01a] Balance of Power: Energy Management for Server Clusters. Chase, Jeffrey S. and Doyle, Ronald P. 2001. s.l. : IEEE Computer Society, 2001. Eighth Workshop on Hot Topics in Operating Systems. pp. 165-. 0-7695-1040-X.

[Chase01b] Managing energy and server resources in hosting centers. Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A.M. and Doyle, R.P. Proceedings of the eighteenth ACM symposium on Operating systems principles. 103—116 (2001)

[Chedid02] Survey on power management techniques for energy efficient computer systems. Chedid, W. and Yu, C. Department of Eletrical and Computer Engineering, Cleveland State University, 2121.

[ChenD07] Nonnegativity Constraints in Numerical Analysis. Chen, Donghui and Plemmons, Robert J. Symp on the Birth of Numerical Analysis, Leuven, Belgium, October 2007.

[ChenK06] Local Temperature Control in Data Center Cooling: Part II, Statistical Analysis. Chen, K., Bash, C.E., Auslander, D.M. and Patel, C.D. HP Laboratories Palo Alto (2006)

[ChenY05] Managing server energy and operational costs in hosting centers. Chen, Yiyu, Das, Amitayu, Qin, Wubi, Sivasubramaniam, Anand, Wang, Qian, Gautam, Natarajan. 2005. ACM SIGMETRICS Performance Evaluation Review, 33(1). pp. 303-314. 1595930221.

[Chung99a] Dynamic power management using adaptive learning tree. Chung, E.Y., Benini, L. and De Micheli, G. Published by the IEEE Computer Society. 1092—3152 (1999)

[Chung99b] Dynamic Power Management for non-stationary service requests. Chung, E.Y., Benini, L., Bogliolo, A. and De Micheli, G. Proceedings of the conference on Design, automation and test in Europe (1999)

[Chung02] Dynamic power management for nonstationary service requests. Chung, E.Y., Benini, L., Bogliolo, A., Lu, Y.H. and De Micheli, G. IEEE Transactions on Computers. 1345—1361 (2002)

[Das08] Autonomic Multi-Agent Management of Power and Performance in Data Centers. Das, R., Kephart, J.O., Lefurgy, C., Tesauro, G., Levine, D.W. and Chan, H. Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track. 107—114 (2008)

[DTMF09] Common Informaton Model (CIM) Infrastructure. Distributed Management Task Force. 2009. Standard. Version 2.5. DSP0004.

[Economou06] Full-system power analysis and modeling for server environments. Economou, D., Rivoire, S., Kozyrakis, C. and Ranganathan, P. Workshop on Modeling, Benchmarking, and Simulation (MoBS) (2006)

[Elnozahy03a] Energy-efficient server clusters. Elnozahy, Mootaz, Kistler, Michael, Rajamony, Ramakrishnan. 2003. Proceedings of Power-Aware Computer Systems. pp. 179-197. Springer.

[Elnozahy03b] Energy conservation policies for web servers. Elnozahy, M., Kistler, M. and Rajamony, R. Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems, 4 (2003)

[Freeh07] Analyzing the energy-time trade-off in high-performance computing applications. Freeh, V.W., Lowenthal, D.K., Pan, F., Kappiah, N., Springer, R., Rountree, B.L. and Femal, M.E. IEEE Transactions on Parallel and Distributed Systems, 18(6). 835—848 (2007)

[Gandhi10] Optimality analysis of energy-performance trade-off for server farm management. Gandhi, A., Gupta, V., Harchol-Balter, M. and Kozuch, M.A. Elsevier Performance Evaluation (2010)

[Garcia09] Self-Adjustment Strategy for Models used in Autonomic Transactional Systems. García, Daniel F., Valledor, Pablo, Entrialgo, Joaquín, Medrano, Ramón, et al. Proceedings of the 9th WSEAS International Conference on Applied Informatics and Communications (AIC '09). 2009.

[Garcia10] A self-managing strategy for balancing response time and power consumption in heterogeneous server clusters. Garcia, D.F. and Entrialgo, J. and Garcia, J. and Garcia, M. International Conference On Electronics and Information Engineering (ICEIE), 1 (2010)

[Golding96] Idleness is not sloth. Golding, R., Bosch, P., Staelin, C., Sullivan, T. and Wilkes, J. Proceedings of the USENIX 1995 Technical Conference, 17 (1995)

[Gurumurthi02] Using complete machine simulation for software power estimation: The softwatt approach. Gurumurthi, S., Sivasubramaniam, A., Irwin, M.J., Vijaykrishnan, N. and Kandemir, M. Proceedings. Eighth International Symposium on High-Performance Computer Architecture. 141—150 (2002)

[Gurumurthi03] DRPM: dynamic speed control for power management in server class disks. Gurumurthi, S., Sivasubramaniam, A., Kandemir, M. and Franke, H. ublished by the IEEE Computer Society (2003)

[Hamacher03] Organización de Computadores. Hamacher, Carl, Vranesic, Zvonko y Zaky, Safwat. 2003. Quinta edición. : McGraw Hill, 2003. 84-481-3951-8.

[Heath02] Application-supported device management for energy and performance. Heath, T., Pinheiro, E. and Bianchini, R. Proceedings of the 2nd international conference on Power-aware computer systems. 157—178 (2002)

[Heath05] Energy conservation in heterogeneous server clusters. Heath, Taliver, Diniz, Bruno, Carrera, Enrique V., Meira Jr., Wagner, Bianchini, Ricardo. 2005. Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming. pp. 186-195. 1595930809.

[Heo07] Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. Heo, J., Henriksson, D., Liu, X. and Abdelzaher, T. 28th IEEE International Real-Time Systems Symposium. 227—238 (2007)

[Hong98] On-line scheduling of hard real-time tasks on variable voltage processor. Hong, I., Potkonjak, M. and Srivastava, M.B. Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design. 653—656 (1998)

[Hong99] Power optimization of variable-voltage core-based systems. Hong, I. and Kirovski, D. and Qu, G. and Potkonjak, M. and Srivastava, M.B. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 18(12). 1702—1714 (1999, published 2002)

[Horn01] Autonomic computing: IBM's perspective on the state of information technology. Horn, Paul. IBM Corporation, 2001. 15. pp. 1-39.

[Horvath07a] Enhancing energy efficiency in multi-tier web server clusters via prioritization. Horvath, T. and Skadron, K. and Abdelzaher, T. Urbana (51) (2007)

[Horvath07b] Dynamic voltage scaling in multitier web servers with end-to-end delay control. Horvath, T. and Abdelzaher, T. and Skadron, K. and Liu, X. IEEE Transactions on Computers (2007)

[Hsu03] The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. Hsu, C.H. and Kremer, U. ACM SIGPLAN Notices, 38(5). 38—48 (2003)

[Huang09] Energy-Efficient Cluster Computing via AccurateWorkload Characterization. Huang, S. and Feng, W. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID'09. 68—75 (2009)

[Hwang97] A predictive system shutdown method for energy saving of event-driven computation. Hwang, C.H. and Wu, A.C.H. ACM Transactions on Design Automation of Electronic Systems (TODAES), 5(2). 226—241 (1997, published 2000)

[Intel96] Advanced Power Management BIOS Interface Specification. Intel Corporation, Microsoft Corporation. 1996. Revision 1.2.

[Intel09a] ACPI Component Architecture Programmer Reference. Open Source Technology Center, Intel Corporation. 2009. Revision 1.25.

[Intel09b] Intelligent Platform Management Interface. Intel Corporation. 2009. Second Edition.

[Irani02] Competitive analysis of dynamic power management strategies for systems with multiple power savings states. Irani, S. and Gupta, R. and Shukla, S. Proceedings of the conference on Design, automation and test in Europe. 117 (2002)

[Ishihara98] Voltage scheduling problem for dynamically variable voltage processors. Ishihara, T. and Yasuura, H. Proceedings. 1998 International Symposium on Low Power Electronics and Design. 197—202 (1998, published 2005)

[ITU94] Telephone network and ISDN quality of service, network management and traffic engineering. ITU-T recommendation E.800. International Telecommunication Union, group 2. August 1994.

[Jung08] Stochastic modeling of a thermally-managed multi-core system. Jung, H. and Rong, P. and Pedram, M. Proceedings of the 45th annual Design Automation Conference. 728—733 (2008)

[Kandasamy06] Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters. Kandasamy, N. and Wang, M. and Guez, A. and Kam, M. Autonomic Computing, 2006. International Conference on ICAC'06. IEEE. 165—174 (2006)

[Kephart03] The vision of autonomic computing. Kephart, Jeffrey O., Chess, David M., 2003. IEEE Computer, 36(1). pp. 41-50. 0018-9162.

[Khargharia07] Autonomic power and performance management for computing systems. Khargharia, B. and Hariri, S. and Yousif, M.S. Cluster Computing, 11(2). 167—181 (2007)

[Koomey07] Estimating total power consumption by servers in the U.S. and the World. Koomey, Jonathan G. 2007. Advanced Micro Devices, Inc. Oakland : Analytics Press, 2007.

[Koomey08] Worldwide electricity used in data centers. Koomey, J.G. Environmental Research Letters, 3 (2008)

[Kumar09] M-Channels and M-Brokers: New Abstractions for Co-ordinated Management in Virtualized Systems. Kumar, S. and Nathuji, R. and Schwan, K. and Talwar, V. and Ranganathan, P. Proceedings of the Workshop on Managed Many-Core Systems (MMCS) (2009)

[Kusic09] Power and performance management of virtualized computing environments via lookahead control. Kusic, Dara, Kephart, Jeffrey O., Hanson, James E., Kandasamy, Nagarajan, Jiang, Guofei. 2009. Cluster Computing, 12(1). pp. 1-15. 1386-7857.

[Kwa02] PCI Express Architecture Power Management. Kwa, Seh and Cohen, Debra T. Intel Corporation. 2002. White Paper.

[Lawson95] Solving Least Squares Problems. Lawson, Charles L., Hanson, Richard J. Society for Industrial and Applied Mathematics (SIAM), 1995.

[LeeH05] On improving performance and conserving power in cluster-based Web servers. Lee, H.K. and Vageesan, G. and Kim, E.J. 2005 IEEE International Conference on Web Services, 2005. ICWS 2005. (2005)

[LeeS00] Run-time Voltage Hopping for Low-power Real-time Systems. Lee, S. and Sakurai, T. Proceedings of the 37th Annual Design Automation Conference. 806—809 (2000)

[Lefurgy03] Energy management for commercial servers. Lefurgy, C. and Rajamani, K. and Rawson, F. and Felter, W. and Kistler, M. and Keller, T.W. IEEE Computer, 36(12). 39—48 (2003)

[Leverich09] Power management of datacenter workloads using per-core power gating. Leverich, J. and Monchiero, M. and Talwar, V. and Ranganathan, P. and Kozyrakis, C. Computer Architecture Letters, 8(2). 48—51 (2009)

[Lu99] Adaptive hard disk power management on personal computers. Lu, Y.H. and De Micheli, G. Proceedings. Ninth Great Lakes Symposium on VLSI, 1999. 50—53 (1999, published 2002)

[Lu01] Comparing system level power management policies. Lu, Y.H. and De Micheli, G. IEEE Design & Test of Computers, 18(2). 10—19 (2001)

[Lu02] Power-aware operating systems for interactive systems. Lu, Y.H. and Benini, L. and De Micheli, G. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 10(2). 119—134 (2002)

[Mandagere07] GreenStor: Application-Aided Energy-Efficient Storage. Mandagere, N. and Diehl, J. and Du, D. Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies. 16—29 (2007)

[Marshall06a] ACPI in Windows Vista. Marshall, Allen. 2006. Taipei : Microsoft Corporation, 2006. WinHEC.

[Mastroleon05] Automatic power management schemes for Internet servers and data centers. Mastroleon, L. and Bambos, N. and Kozyrakis, C. and Economou, D. IEEE Global Telecommunications Conference, 2005. GLOBECOM'05, 2(5) (2006)

[Matthew09] Memory miser: Improving main memory energy efficiency in servers. Tolentino, M.E. and Turner, J. and Cameron, K.W. IEEE Transactions on Computers, 58(3). 336—350 (2009)

[Marshall06b] PCI Express in Depth for Windows Vista and Beyond. Marshall, Allen and Mamtani, Vinod. 2006. Taipei : s.n., 2006. WinHEC.

[McCloghrie91] Management Information Base for Network Management of TCP/IP-based internets: MIB-II. McCloghrie, Keith and Rose, Marshall T. 1991. Network Working Group, Internet Engineering Task Force. s.l. : RFC Editor, 1991. Request For Comments. 1213.

[Microsoft06] Active State Power Management in Windows Vista. Microsoft Corporation. 2006.

[Microsoft07] ACPI Driver Interface in Windows Vista. Microsoft Corporation. 2007.

[Microsoft09] Using PowerCfg to Evaluate System Energy Efficiency. Microsoft Corporation. 2009.

[Nakamura92] Métodos numéricos aplicados con software. Nakamura, Soichiro. 1992. Prentice-Hall. 968-880-263-8.

[Nathuji07] Exploiting platform heterogeneity for power efficient data centers. Nathuji, R. and Isci, C. and Gorbatov, E. Fourth International Conference on Autonomic Computing, 2007. ICAC'07, 7 (2007)

[Pakbaznia07] Minimizing Data Center Cooling and Server Power Costs. Pakbaznia, E. Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design. 145—150 (2009)

[Pedram10] Power and Performance Modeling in a Virtualized Server System. Pedram, M. and Hwang, I. (2010)

[Petrucci09] A framework for dynamic adaptation of power-aware server clusters. Petrucci, V. and Loques, O. and Mossé, D. Proceedings of the 2009 ACM symposium on Applied Computing. 1034—1039 (2009)

[Petrucci10] A dynamic optimization model for power and performance management of virtualized clusters. Petrucci, V. and Loques, O. and Mossé, D. Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. 225-233 (2010)

[Pettis09] A homogeneous architecture for power policy integration in operating systems. Pettis, N.E. and Lu, Y.H. IEEE Transactions on Computers. 945—955 (2008)

[Pinheiro01] Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. Pinheiro, Eduardo, et al. 2001. 2001. Workshop on Compilers and Operating Systems for Low Power.

[Pinheiro03] Dynamic cluster reconfiguration for power and performance. Pinheiro, E. and Bianchini, R. and Carrera, E. and Heath, T. Compilers and operating systems for low power (2003)

[Qiu99] Stochastic modeling of a power-managed system-construction and optimization. Qiu, Q. and Qu, Q. and Pedram, M. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systemsm 20(10). 1200—1217 (1999, published 2002)

[Qiu00] Dynamic power management of complex systems using generalized stochastic Petri nets. Qiu, Q. and Wu, Q. and Pedram, M. Proceedings of the 37th Annual Design Automation Conference. 352—356 (2000)

[Raghavendra07] N o power struggles: Coordinated multi-level power management for the data center. Raghavendra, R. and Ranganathan, P. and Talwar, V. and Wang, Z. and Zhu, X. ACM SIGPLAN Notices, 43(3). 48—59 (2007, published 2008)

[Rajamani08] Power management for computer systems and datacenters. Rajamani, K. and Lefurgy, C. and Ghiasi, S. and Rubio, J.C. and Hanson, H. and Keller, T. International Symposium on Low Power Electronics and Design (ISLPED) (2008)

[Ramanathan00] System level online power management algorithms. Ramanathan, D. and Gupta, R. Design, Automation and Test in Europe Conference and Exhibition. 606—611 (2000, published 2002)

[Ranganathan10] Recipe for efficiency: principles of power-aware computing. Ranganathan, P. Communications of the ACM, 53(4). 60—67 (2010)

[Ren04] Hierarchical adaptive dynamic power management. Ren, Z. and Krogh, B.H. and Marculescu, R. IEEE Transactions on Computers. 409—420 (2004, published 2005)

[Rivoire07] Models and metrics to enable energy-efficiency optimizations. Rivoire, S. and Shah, M.A. and Ranganatban, P. and Kozyrakis, C. and Meza, J. Computer, 40(12). 39—48 (2007)

[Rohou99] Dynamically managing processor temperature and power. Rohou, E. and Smith, M.D. In 2nd Workshop on Feedback-Directed Optimization (1999)

[Rong05] Hierarchical power management with application to scheduling. Rong, P. and Pedram, M. Proceedings of the 2005 international symposium on Low power electronics and design. 269—274 (2005)

[Rusu06] Energy-efficient real-time heterogeneous server clusters. Energy-efficient real-time heterogeneous server clusters. Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium. 418—428 (2006)

[Schlatter05] Every Joule is Precious. Schlatter, C.E. Duke University. Milly Watt Project.

[Shah07] UltraSPARC T2: A Highly-Threaded, Power-Efficient, SPARC SOC. Shah, Manish, et al. 2007. Jeju : IEEE Asian, 2007. Solid-State Circuits Conference. ASSCC '07. pp. 22-25. 978-1-4244-1360-7.

[SharmaRK03] Balance of power: Dynamic thermal management for internet data centers. Sharma, R.K. and Bash, C.E. and Patel, C.D. and Friedrich, R.J. and Chase, J.S. IEEE Internet Computing, 9(1). 42—49 (2003, published in 2005)

[SharmaV03] Power-aware QoS Management in Web Servers. Sharma, V. and Thomas, A. and Abdelzaher, T. and Skadron, K. and Lu, Z. 24th IEEE Real-Time Systems Symposium. 63—72 (2003, published 2005)

[Simunic01] Event-driven power management. Simunic, T. and Benini, L. and Glynn, P. and De Micheli, G. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 20(7). 840—857 (2001)

[Simunic02a] Managing Power Consumption in Networks on Chips. Simunic, T. and Boyd, S.P. and Glynn, P. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 12(1). 96—107 (2004)

[Simunic02b] Dynamic management of power consumption. Simunic, T. Power aware computing. 101—125 (2002)

[Solari03] The Complete PCI Express Reference. Solari, Edward and Congdon, Brad. 2003. Intel Press, 2003.

[Steigerwald08] Creating Energy-Efficient Software. Steigerwald, Bob, et al. 2008. Intel Corporation. 2008.

[Stemen06] Power Management in Windows Vista. Stemen, Pat. 2006. Taipei : Microsoft Corporation, 2006. WinHEC.

[Tang08] Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. Tang, Q. and Gupta, S.K.S. and Varsamopoulos, G. IEEE Transactions on Parallel and Distributed Systems. 1458—1472 (2008)

[Tsai07] Online web cluster capacity estimation and its application to energy conservation. Tsai, C.H. and Shin, K.G. and Reumann, J. and Singhal, S. IEEE Transactions on Parallel and Distributed Systems. 932—945 (2007)

[Tsai08] Measurement and modelling for the transition penalty and the effective sleep time interval for the dynamic power measurement of PCs. Tsai, C.H. and Bai, Y.W. and Lin, M.B. and Cheng, Y.S. Proceedings of the 27th IASTED International Conference on Modelling, Identification and Control. 66—71 (2008)

[Vahdat00] Every joule is precious: The case for revisiting operating system design for energy efficiency. Vahdat, A. and Lebeck, A. and Ellis, C.S. Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system. 31—36 (2000)

[Valledor09] Técnicas de autoajuste de modelos de respuesta temporal de servidores transaccionales aplicadas al control de la calidad de servicio. Valledor, Pablo, García, Daniel F. 2009. Departamento de Informática, Universidad de Oviedo.

[WangJ08] eRAID: Conserving energy in conventional disk-based RAID system. Wang, J. and Zhu, H. and Li, D. IEEE Transactions on Computers, 57(3). 359—374 (2008)

[WangL09] Efficient power management of heterogeneous soft real-time clusters. Wang, L. and Lu, Y. Real-Time Systems Symposium. 323—332 (2009)

[WangX08] Cluster-level feedback power control for performance optimization. Wang, X. and Chen, M. IEEE 14th International Symposium on High Performance Computer Architecture- 101—110 (2008)

[WangX09] Co-con: Coordinated control of power and application performance for virtualized server clusters. Wang, X. and Wang, Y. 17th International Workshop on Quality of Service. 1—9 (2009)

[Weiser94] Scheduling for reduced CPU energy. Weiser, M. and Welch, B. and Demers, A. and Shenker, S. Mobile Computing. 449—471 (114, published 1996)

[Xue07] An energy-efficient management mechanism for large-scale server clusters. Xue, Z. and Dong, X. and Ma, S. and Fan, S. and Mei, Y. The 2nd IEEE Asia-Pacific Service Computing Conference. 509—516 (2007)

[Yao95] A scheduling model for reduced CPU energy. Yao, F. and Demers, A. and Shenker, S. 36th Annual Symposium on Foundations of Computer Science. 374—382 (2002)

[Zhong07] Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. Zhong, X. and Xu, C.Z. IEEE Transactions on computers. 358—372 (2007)

[Zhu04] Reducing energy consumption of disk storage using power-aware cache management. Zhu, Q. and David, F.M. and Devaraj, C.F. and Li, Z. and Zhou, Y. and Cao, P. (2004)

A DOCUMENTACIÓN ADJUNTA

En el CD-ROM adjunto pueden encontrarse los siguientes recursos:

`Codigo/` Copia del último snapshot del código fuente para Microsoft Visual Studio 2010.

`Documentacion/` Copia de esta documentación en formatos para Microsoft Word y PDF. Además de documentos adicionales, como gráficos y hojas de cálculo.

`Experimentos/` Copia de los resultados más destacados de la validación del proyecto y hojas de cálculo de resumen.

`Referencias/` Copia de todos los documentos referenciados cuya licencia permite ser redistribuidos en un contexto académico.

B CÓDIGO FUENTE DEL PROTOTIPO

13.1 Librería de funciones de uso común

Esta librería presenta dependencias con la librería GNU Scientific Library en su versión 1.13. El código fuente, liberado bajo licencia GPL (ver texto de la licencia más adelante) está disponible en su sitio Web.

13.1.1 Comun.h

```
#ifndef __COMUN_LIB__
#define __COMUN_LIB__

/* Fichero: Comun.h
 * Autor: Ramón Medrano Llamas
 * Versión: 1.11
 * Descripción: Definciones e inclusiones para la librería de componentes
 *              compartidos por los módulos del servicio transaccional.
 */

/* La aplicacion no es compatible con caracteres unicode (WCHAR) por el momento. */
#undef UNICODE // Para Win32.
#undef _UNICODE // Para stdlib.

/* Inclusiones para todos los módulos. Incrementa un poco el tiempo de preprocesamiento
   pero, ya que windows.h es tan grande, no se nota mucho y mejora la legibilidad.
   La libreria GSL la incluimos donde nos haga falta (se incluye por modulos). */
#include <assert.h>
#include <errno.h>
#include <float.h>
#include <limits.h>
#include <math.h>
#include <search.h>
#include <signal.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <winsock2.h> // No se puede incluir antes de windows.h
#include <ws2tcpip.h>
#include <windows.h>
```

```
#include <shlwapi.h>  // Debe ir despues de windows.h

/* Definiciones de macros. */
#define KBYTES 1024
#define MBYTES (KBYTES*KBYTES)
#define DOUBLE_MAX DBL_MAX
#define MAX_CTIME_STRING 26
#define SOCK_TIMEOUT 5 // segundos

/* Compatibilidad de WinSock con versiones anteriores de Windows. */
#ifndef MSG_WAITALL
   #define MSG_WAITALL 0x0
#endif

/* Definición de tipos para coherencia con Win32. */
typedef USHORT UINT16;
typedef UINT16 * PUINT16;
typedef DOUBLE * PDOUBLE;
typedef CONST PDOUBLE PCDOUBLE;
typedef CONST PVOID PCVOID;
typedef SOCKET * PSOCKET;
typedef FILE * PFILE;

/* Tipos propios para las funciones matemáticas. */
typedef struct _CURVAPOLINOMICA {
   UINT16 grado;
   PDOUBLE coeficientes;
} CURVAPOLINOMICA, * PCURVAPOLINOMICA;

/* Inicialización y liberación de la librería. */
BOOL InicializarLibreria(UINT32 semilla);
BOOL LiberarLibreria();

/* Funciones para la generación de distribuciones de probabilidad. */
DOUBLE DistribucionUniforme(DOUBLE limiteInferior, DOUBLE limiteSuperior);
DOUBLE DistribucionExponencial(DOUBLE media);

/* Funciones para el manejo de los contadores de rendimiento de Windows. */
DOUBLE TicksPorMilisegundo();
DOUBLE MilisegundosTranscurridos(LARGE_INTEGER ini, LARGE_INTEGER fin);
DOUBLE TiempoAbsoluto(); // En milisegundos

/* Funciones de manejo de errores y logging. */
VOID Debug(PCSTR mensaje, ...);
VOID Error(PCSTR mensaje, ...);
VOID ErrorExit(PCSTR mensaje);

/* Funciones de configuración y manejo de sockets. */
BOOL ConfiguraSocket(SOCKET sock);
BOOL SocketBroadcast(SOCKET sock);
BOOL ResuelveNombre(PCSTR nombre, PIN_ADDR direccion);
BOOL CierraSocket(SOCKET sock);
BOOL DireccionUDPNuestra(PSOCKADDR_IN origen);

/* Funciones matemáticas generales. */
UINT64 SiguienteID();
DOUBLE BuscarImagen(PCURVAPOLINOMICA curva, DOUBLE x);
DOUBLE BuscarRaiz(PCURVAPOLINOMICA curva, DOUBLE e);
DOUBLE CalcularPercentil(PDOUBLE vector, SIZE_T len, DOUBLE k);

/* Manejadores de curvas polinomicas de cualquier grado. */
PCURVAPOLINOMICA CrearCurvaPolinomica(UINT16 grado);
PCURVAPOLINOMICA CopiarCurvaPolinomica(PCURVAPOLINOMICA curva);
VOID EliminarCurvaPolinomica(PCURVAPOLINOMICA curva);
PCURVAPOLINOMICA CompactarCurvaPolinomica(PCURVAPOLINOMICA curva);

/* Funciones de ordenación. */
VOID Ordenar(PDOUBLE v, SIZE_T len);

/*** FUNCIONES PRIVADAS. NO DEBERIAN NECESITAR LLAMARSE. ***/

/* Funciones de manejo de la librería criptográfica de Windows. Permiten obtener
   proveedores sencillos de cara a la generación de números aleatorios. */
BOOL InicializarCriptografia(UINT32 semilla);
```

```
BOOL LiberarCriptografia();

/* Funciones para la inicialización de contadores de rendimiento y estructuras
   auxiliares requeridas para la implementación. */
BOOL InicializarContadores();
BOOL LiberarContadores();

/* Funciones para la inicialización de ficheros y secciones críticas del módulo
   de logging y manejo de errores. */
BOOL InicializarLogging();
BOOL LiberarLogging();

/* Funciones para la inicialización de variables para el configurador de sockets. */
BOOL InicializarSockets();
BOOL LiberarSockets();

/* Funciones para la inicialización de variables para el módulo de matemáticas. */
BOOL InicializarMath();
BOOL LiberarMath();

/* Funciones de busqueda de raices usando GSL. Ojo: modifican la curva que se pasa como
   parametro. No hacen copia para acelerar el proceso con una sola copia y compactacion
   en el dispatcher a las librerias GSL. */
DOUBLE BuscarRaizGrado2(PCURVAPOLINOMICA curva, DOUBLE e);
DOUBLE BuscarRaizGrado3(PCURVAPOLINOMICA curva, DOUBLE e);
DOUBLE BuscarRaizGenerica(PCURVAPOLINOMICA curva, DOUBLE e);

/* Funciones internas de manejo de curvas. */
BOOL InicializarCurvaPolinomica(PCURVAPOLINOMICA curva, UINT16 grado);
VOID LiberarCurvaPolinomica(PCURVAPOLINOMICA curva);

/* Funciones para la obtención de números aleatorios con gran período. */
BOOL GenerarUIntAleatorio(PUINT32 aleatorio);
/* Solo usar esta funcion por motivos de rendmiento. */
BOOL GenerarDoubleAleatorio(PDOUBLE aleatorio); // [0.0; 1.0)

#endif
```

13.1.2 Aleatorios.c

```
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include "Comun.h"

/* Fichero: Aleatorios.c
 * Autor: Ramón Medrano Llamas
 * Versión: 2.0
 * Descripción: Implementación de funciones de generación de números aleatorios
 *              mediante las librería de generación de aleatorios basada en GSL.
 *              GSL es thread-safe, por lo que deberia de ser mas rapido.
 */

/* Estado del Mersenne Twister. */
static gsl_rng * r;

BOOL InicializarCriptografia(UINT32 semilla)
{
   /* El generador gsl_rng_ranlxs0 es el mas rapido de todos los que son
      estadisticamente validados. */
   if ((r = gsl_rng_alloc(gsl_rng_ranlxs0)) == NULL)
      return FALSE;
   gsl_rng_set(r, (unsigned long)semilla);
   return TRUE;
}

BOOL LiberarCriptografia()
{
   gsl_rng_free(r);
```

```
   return TRUE;
}

/* La interfaz de estass funciones es fea, pero permite conservar compatibilidad
   hacia atras.*/
BOOL GenerarUIntAleatorio(PUINT32 aleatorio)
{
   *aleatorio = gsl_rng_get(r);

   return TRUE;
}

BOOL GenerarDoubleAleatorio(PDOUBLE aleatorio)
{
   *aleatorio = gsl_rng_uniform(r);

   return TRUE;
}

DOUBLE DistribucionUniforme(DOUBLE limiteInferior, DOUBLE limiteSuperior)
{
   DOUBLE num;

   if (!GenerarDoubleAleatorio(&num))
      return 0.0;

   num = limiteInferior + (num * (limiteSuperior - limiteInferior));

   assert((num >= limiteInferior) && (num < limiteSuperior));

   return num;
}

DOUBLE DistribucionExponencial(DOUBLE media)
{
   return gsl_ran_exponential(r, media);
}
```

13.1.3 Comun.c

```
#include "Comun.h"

/* Fichero: Comun.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.1
 * Descripción: Implementación de funciones de manejo general de la
 *              librería de utilidades.
 */

static CRITICAL_SECTION liberacionCritialSection; // Evita liberaciones en paralelo.
static BOOL inicializado = FALSE;

BOOL InicializarLibreria(UINT32 semilla)
{
   BOOL ret;

   InitializeCriticalSection(&liberacionCritialSection);

   ret = InicializarCriptografia(semilla);
   ret = ret && InicializarContadores();
   ret = ret && InicializarLogging();
   ret = ret && InicializarSockets();
   ret = ret && InicializarMath();
   inicializado = ret;

   assert(ret);

   return ret;
```

```
}

BOOL LiberarLibreria()
{
   BOOL ret;

   if (!inicializado)
      return FALSE;

   EnterCriticalSection(&liberacionCritialSection);

   ret = LiberarCriptografia();
   ret = ret && LiberarContadores();
   ret = ret && LiberarLogging();
   ret = ret && LiberarSockets();
   ret = ret && LiberarMath();

   LeaveCriticalSection(&liberacionCritialSection);

   assert(ret);

   return ret;
}
```

13.1.4 Contadores.c

```
#include "Comun.h"

/* Fichero: Contadores.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.1
 * Descripción: Implementación de funciones de manejo de los contadores de
 *              rendimiento de Windows para métricas temporales.
 */

/* Variable para la conversión de ticks a milisegundos. */
static DOUBLE ticksPorMilisegundo;

/* Tiempo de inicio de la aplicación para calcular tiempos absolutos. No está
   protegida por una sección crítica pues sólo se hará acceso de lectura. */
static LARGE_INTEGER ticksInicio;

BOOL InicializarContadores()
{
   LARGE_INTEGER frecuencia;

   /* Inicializamos los ticks por milisegundo para esta máquina. */
   if (!QueryPerformanceFrequency(&frecuencia))
      return FALSE;

   ticksPorMilisegundo = (DOUBLE)frecuencia.QuadPart / 1000.0;

   /* Marcamos los ticks al comienzo, necesarios para los tiempos absolutos. */
   if (!QueryPerformanceCounter(&ticksInicio))
      return FALSE;

   return TRUE;
}

BOOL LiberarContadores()
{
   return TRUE;
}

DOUBLE TicksPorMilisegundo()
{
   /* Evita hacer muchas llamadas al API. */
   return ticksPorMilisegundo;
```

```
}

DOUBLE MilisegundosTranscurridos(LARGE_INTEGER ini, LARGE_INTEGER fin)
{
   LARGE_INTEGER diferencia;
   DOUBLE milis;

   assert(fin.QuadPart >= ini.QuadPart);

   diferencia.QuadPart = fin.QuadPart - ini.QuadPart;
   milis = (DOUBLE)diferencia.LowPart / ticksPorMilisegundo;

   if (diferencia.HighPart != 0)
      milis += ldexp((DOUBLE)diferencia.HighPart, 32) / ticksPorMilisegundo;

   return milis;
}

DOUBLE TiempoAbsoluto()
{
   LARGE_INTEGER ticksAhora;

   if (!QueryPerformanceCounter(&ticksAhora))
      return 0.0;

   return MilisegundosTranscurridos(ticksInicio, ticksAhora);
}
```

13.1.5 Logging.c

```
#include "Comun.h"

/* Fichero: Logging.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.1
 * Descripción: Implementación de funciones de manejo de loggers y
 *              emisión de errores.
 */

/* Protege el acceso a stderr para enviar errores. */
static CRITICAL_SECTION stderrCriticalSection;

/* Si se llama a ErrorExit(), solo un thread puede estar en ella. */
static CRITICAL_SECTION errorExitCriticalSection;

BOOL InicializarLogging()
{
   InitializeCriticalSection(&stderrCriticalSection);
   InitializeCriticalSection(&errorExitCriticalSection);
   return TRUE;
}

BOOL LiberarLogging()
{
   DeleteCriticalSection(&stderrCriticalSection);
   DeleteCriticalSection(&errorExitCriticalSection);
   return TRUE;
}

VOID Debug(PCSTR mensaje, ...)
{
#if (defined(_DEBUG_) || defined(_DEBUG) || defined(DEBUG))
   va_list argptr;

   assert(mensaje);

   /* Cogemos el rango variable de argumentos. */
   va_start(argptr, mensaje);
```

```
   EnterCriticalSection(&stderrCriticalSection);
   fprintf(stdout, "[DBG] ");
   vfprintf_s(stdout, mensaje, argptr);
   fprintf(stdout, "\n");
   LeaveCriticalSection(&stderrCriticalSection);

   /* Liberamos los argumentos. */
   va_end(argptr);
#endif
}

VOID Error(PCSTR mensaje, ...)
{
   va_list argptr;

   assert(mensaje);

   /* Cogemos el rango variable de argumentos. */
   va_start(argptr, mensaje);

   EnterCriticalSection(&stderrCriticalSection);
   vfprintf_s(stderr, mensaje, argptr);
   fprintf(stderr, "\n");
   LeaveCriticalSection(&stderrCriticalSection);

   /* Liberamos los argumentos. */
   va_end(argptr);
}

VOID ErrorExit(PCSTR mensaje)
{
   EnterCriticalSection(&errorExitCriticalSection); // Nunca se sale.
   Error(mensaje);
   LiberarLibreria();
   exit(EXIT_FAILURE);
}
```

13.1.6 Math.c

```
#include "Comun.h"

/* Fichero: Math.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.1
 * Descripción: Implementación de funciones matemáticas generales para
 *              el manejo de polinomios, raíces y curvas.
 */

static UINT64 id; // Permite generar ID unicos.
static CRITICAL_SECTION idCriticalSection; // Paralelismo en el id.

BOOL InicializarMath()
{
   id = 0;
   InitializeCriticalSection(&idCriticalSection);
   return TRUE;
}

BOOL LiberarMath()
{
   DeleteCriticalSection(&idCriticalSection);
   return TRUE;
}

UINT64 SiguienteID()
{
   UINT64 ret;
```

```
   EnterCriticalSection(&idCriticalSection);
   ret = id++;
   LeaveCriticalSection(&idCriticalSection);

   return ret;
}

PCURVAPOLINOMICA CrearCurvaPolinomica(UINT16 grado)
{
   PCURVAPOLINOMICA ret;

   ret = (PCURVAPOLINOMICA)malloc(sizeof(CURVAPOLINOMICA));

   if (ret == NULL)
      return NULL;

   if (!InicializarCurvaPolinomica(ret, grado)) {
      free(ret);
      return NULL;
   }

   return ret;
}

PCURVAPOLINOMICA CopiarCurvaPolinomica(PCURVAPOLINOMICA curva)
{
   PCURVAPOLINOMICA ret;

   ret = CrearCurvaPolinomica(curva->grado);

   if (ret == NULL)
      return NULL;

   memcpy(ret->coeficientes, curva->coeficientes, sizeof(DOUBLE) * (curva->grado + 1));

   return ret;
}

VOID EliminarCurvaPolinomica(PCURVAPOLINOMICA curva)
{
   if (curva == NULL)
      return;

   LiberarCurvaPolinomica(curva);

   free(curva);
}

BOOL InicializarCurvaPolinomica(PCURVAPOLINOMICA curva, UINT16 grado)
{
   UINT16 i;

   curva->grado = grado;

   curva->coeficientes = (PDOUBLE)malloc(sizeof(DOUBLE) * ((SIZE_T)grado + 1));

   if (curva->coeficientes == NULL) {
      curva->coeficientes = NULL;
      curva->grado = 0;
      return FALSE;
   }

   for (i = 0; i <= grado; ++i)
      curva->coeficientes[i] = 0.0;

   return TRUE;
}

VOID LiberarCurvaPolinomica(PCURVAPOLINOMICA curva)
{
   if (curva == NULL)
      return;
```

```
   if (curva->coeficientes == NULL)
      return;

   free(curva->coeficientes);
}

PCURVAPOLINOMICA CompactarCurvaPolinomica(PCURVAPOLINOMICA curva)
{
   UINT32 i;
   PCURVAPOLINOMICA nuevaCurva;

   for (i = curva->grado; i >= 0; --i) {
      if (curva->coeficientes[i] != 0.0) {
         break;
      }
   }

   nuevaCurva = CrearCurvaPolinomica((UINT16)i);

   memcpy(nuevaCurva->coeficientes, curva->coeficientes, sizeof(DOUBLE) * (i + 1));
   nuevaCurva->grado  = (UINT16)i;

   return nuevaCurva;
}
```

13.1.7 Math2.c

```
#include <gsl/gsl_errno.h>
#include <gsl/gsl_poly.h>
#include <gsl/gsl_sort.h>
#include <gsl/gsl_statistics.h>
#include "Comun.h"

/* Fichero: Math2.c
 * Autor: Ramón Medrano Llamas
 * Versión: 2.0
 * Descripción: Implementación de funciones matemáticas generales para
 *              el manejo de polinomios, raíces y curvas con GSL.
 */

DOUBLE CalcularPercentil(PDOUBLE vector, SIZE_T len, DOUBLE k)
{
   /* Ojo, con el valor de k. */
   if (k < 0.0 || k > 1.0 || len < 0)
      return 0.0;

   gsl_sort(vector, 1, len);

   return gsl_stats_quantile_from_sorted_data(vector, 1, len, k);
}

DOUBLE BuscarImagen(PCURVAPOLINOMICA curva, DOUBLE x)
{
   UINT32 i;
   DOUBLE imagen = 0.0;

   /* Hay problemas al enlazar con GSL con esta funcion, que usa el
      metodo de Horner para ser mas rapido. */
   //return gsl_poly_eval(curva->coeficientes, curva->grado + 1, x);

   for (i = 0; i <= curva->grado; ++i) {
      imagen += curva->coeficientes[i] * pow(x, (DOUBLE)i);
   }

   return imagen;
}

DOUBLE BuscarRaiz(PCURVAPOLINOMICA curva, DOUBLE e)
```

```
{
   DOUBLE raiz;
   PCURVAPOLINOMICA curvaCompacta;
   #if (defined(_DEBUG_) || defined(_DEBUG) || defined(DEBUG))
   DOUBLE t0 = TiempoAbsoluto();
   #endif

   /* Compactamos la curva. */
   curvaCompacta = CompactarCurvaPolinomica(curva);

   /* En funcion del grado de la curva, llamamos a una u otra para ser mas eficientes. */
   switch (curvaCompacta->grado) {
      case 0:
         Error("Infinitas raices o ninguna (curva de grado 0)");
         raiz = -1.0;
         break;
      case 1:
         raiz = (e - curva->coeficientes[1]) / curva->coeficientes[0];
         break;
      case 2:
         raiz = BuscarRaizGrado2(curvaCompacta, e);
         break;
      case 3:
         raiz = BuscarRaizGrado3(curvaCompacta, e);
         break;
      default:
         raiz = BuscarRaizGenerica(curvaCompacta, e);
   }

   /* Validacion propia de las caracteristicas de las curvas a procesar. */
   assert(raiz > 0.0); // En caso de error (-1.0), nos avisa :)

   #if (defined(_DEBUG_) || defined(_DEBUG) || defined(DEBUG))
   Debug("Curva grado %d (compactada a %d), computada en %lf ms", curva->grado, curvaCompacta->grado,
TiempoAbsoluto() - t0);
   #endif

   /* Limpieza. */
   EliminarCurvaPolinomica(curvaCompacta);

   return raiz;
}

DOUBLE BuscarRaizGrado2(PCURVAPOLINOMICA curva, DOUBLE e)
{
   DOUBLE x0, x1;
   INT32 raices;

   x0 = x1 = 0.0;
   raices = gsl_poly_solve_quadratic(curva->coeficientes[2], curva->coeficientes[1], curva-
>coeficientes[0] - e, &x0, &x1);

   switch(raices) {
      case 1:
         /* Validacion propia de las caracteristicas de las curvas a procesar. */
         assert(x0 > 0.0);
         return x0;
      case 2:
         assert(x1 > 0.0);
         return (x0 > 0.0)?x0:x1;
      default:
         Error("La curva cuadratica no tiene raices reales");
         return -1.0;
   }
}

DOUBLE BuscarRaizGrado3(PCURVAPOLINOMICA curva, DOUBLE e)
{
   INT32 i;
   DOUBLE x0, x1, x2;
   DOUBLE raiz;

   /* Transformamos los coeficientes para hacer la operacion. */
   curva->coeficientes[0] -= e;
```

```
   /* Dividimos los coeficientes. Hacer esto con intrinsic SSE seria lo optimo. */
   for (i = 0; i < curva->grado; ++i) {
      curva->coeficientes[i] /= curva->coeficientes[curva->grado];
   }
   curva->coeficientes[curva->grado] = 1.0;

   /* Llamamos a GSL. */
   i = gsl_poly_solve_cubic(curva->coeficientes[2], curva->coeficientes[1], curva->coeficientes[0],
&x0, &x1, &x2);

   /* Buscamos la raiz que nos conviene. */
   switch(i) {
      case 1:
         /* Validacion propia de las caracteristicas de las curvas a procesar. */
         assert(x0 > 0.0);
         raiz = x0;
         break;
      case 3:
         assert(x2 > 0.0);
         raiz = (x0>0.0)?x0:((x1>0.0)?x1:x2);
         break;
      default:
         Error("La curva cubica no tiene raices reales");
         raiz = -1.0;
   }

   assert(raiz > 0.0);
   return raiz;
}

DOUBLE BuscarRaizGenerica(PCURVAPOLINOMICA curva, DOUBLE e)
{
   UINT32 i;
   PDOUBLE z;
   DOUBLE min;
   gsl_poly_complex_workspace * wspc;

   /* Sanity check para el coeficiente de mayor grado. */
   if (curva->coeficientes[curva->grado] == 0.0) {
      Error("Coeficiente no valido para la curva");
      return -1.0;
   }

   if ((z = (PDOUBLE)malloc(sizeof(DOUBLE) * curva->grado * 2)) == NULL) {
      Error("No se pudo reservar memoria para la resolucion del polinomio");
      return -1.0;
   }

   if ((wspc = gsl_poly_complex_workspace_alloc(curva->grado + 1)) == NULL) {
      Error("No se pudo generar el espacio de trabajo para la resolucion del polinomio");
      free(z);
      return -1.0;
   }

   /* Modificamos el coeficiente con el parametro e */
   curva->coeficientes[0] -= e;

   if (gsl_poly_complex_solve(curva->coeficientes, curva->grado + 1, wspc, z) != GSL_SUCCESS) {
      Error("No se pudo calcular la reduccion QR para el polinomio");
      gsl_poly_complex_workspace_free(wspc);
      free(z);
      return -1.0;
   }

   /* Debemos retornar la raiz mayor que 0 real mas pequeña. */
   min = DOUBLE_MAX;
   for (i = 0; i < curva->grado; ++i) {
      if (z[2*i+1] == 0.0 && z[2*i] >= 0.0 && z[2*i] < min) {
         min = z[2*i];
      }
   }

   gsl_poly_complex_workspace_free(wspc);
```

```
   free(z);

   assert(min > 0.0);
   return min;
}
```

13.1.8 Quicksort.c

```
#include "Comun.h"

/* Fichero: Quicksort.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.0
 * Descripción: Implementación de funciones de ordenación para vectores de
 *              elementos de coma flotante. Se utiliza fundamentalmente para
 *              el cálculo de percentiles, aunque se ofrece como función pública.
 */

/* No usamos tipos Win32 para evitar problemas con el calificador const y los punteros
   a función en qsort_s(). El contexto no se utiliza. */
#pragma warning (disable : 4100)
__forceinline int _cdecl Compara(void * contexto, const void * a, const void * b)
{
   DOUBLE r;

   r = *((PCDOUBLE)a) - *((PCDOUBLE)b);

   if (r < 0.0)
      return -1;
   else if (r > 0.0)
      return 1;
   else
      return 0;
}

VOID Ordenar(PDOUBLE v, SIZE_T len)
{
   qsort_s((PVOID)v, len, sizeof(DOUBLE), Compara, NULL);
}
```

13.1.9 Sockets.c

```
#include "Comun.h"

/* Fichero: Sockets.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.1
 * Descripción: Implementación de funciones de configuración de sockets.
 */

static WSADATA wsaData;

BOOL InicializarSockets()
{
   /* Incializacion de las funciones de sockets. */
   return WSAStartup(MAKEWORD(2, 2), &wsaData) == 0;
}

BOOL LiberarSockets()
{
   return WSACleanup() == 0;
}
```

```
BOOL ConfiguraSocket(SOCKET sock)
{
   LINGER lingerData;

   lingerData.l_onoff = 1;
   lingerData.l_linger = SOCK_TIMEOUT;

   if (setsockopt(sock, SOL_SOCKET, SO_LINGER, (PCHAR)&lingerData, sizeof(LINGER)) == SOCKET_ERROR)
      return FALSE;

   return TRUE;
}

BOOL SocketBroadcast(SOCKET sock)
{
        INT32 uno;

        uno = 1;
        /* Convertimos el socket en broadcast. */
   if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (PSTR)&uno, sizeof(INT32)) == SOCKET_ERROR)
      return FALSE;

   return TRUE;
}

BOOL ResuelveNombre(PCSTR nombre, PIN_ADDR direccion)
{
   ADDRINFO hints;
   ADDRINFO * direcciones;

   /* Inicializar los hints, indicando qué soportamos. */
   direcciones = NULL;
   ZeroMemory(&hints, sizeof(ADDRINFO));
   hints.ai_family = AF_INET;
   hints.ai_socktype = SOCK_STREAM; // Siempre usamos TCP.
   hints.ai_protocol = IPPROTO_TCP; // Siempre usamos TCP.

   /* Llamamos a la función de resolución. gethostbyname() está desaconsejada. */
   if (getaddrinfo(nombre, NULL, &hints, &direcciones) != 0)
      return FALSE;

   /* Obtenemos la primera dirección devuelta por el resolutor DNS. */
   if (direcciones != NULL)
      *direccion = ((PSOCKADDR_IN)direcciones->ai_addr)->sin_addr;
   else
      return FALSE;

   /* Liberamos la lista de nombres. */
   freeaddrinfo(direcciones);

   return TRUE;
}

BOOL CierraSocket(SOCKET sock)
{
   BOOL ret;

   ret = shutdown(sock, SD_BOTH) != SOCKET_ERROR;
   ret = ret && closesocket(sock) != SOCKET_ERROR;

   return ret;
}

/* Comprueba si una dirección de origen de un paquete se corresponde con una de las direcciones
   IP de la maquina local. Útil para descartar datagramas UDP broadcast que provienen de nosotros
   mismos. */
BOOL DireccionUDPNuestra(PSOCKADDR_IN origen)
{
   CHAR hostname[256]; /* Con 256 siempre bastara. */
   ADDRINFO hints;
   ADDRINFO * direcciones, * ptr;

   /* Inicializar los hints, indicando qué soportamos. */
```

```
   direcciones = NULL;
   ZeroMemory(&hints, sizeof(ADDRINFO));
   hints.ai_family = AF_INET;
   hints.ai_socktype = SOCK_DGRAM; // En este caso, esta pensado para UDP.
   hints.ai_protocol = IPPROTO_UDP; // En este caso, esta pensado para UDP.

   /* Buscamos nuestro nombre de host. */
   if (gethostname(hostname, sizeof(hostname)) == SOCKET_ERROR) {
      return FALSE;
   }

   /* Llamamos a la función de resolución. gethostbyname() está desaconsejada. */
   if (getaddrinfo(hostname, NULL, &hints, &direcciones) != 0)
      return FALSE;

   if (direcciones == NULL)
      return FALSE;

   /* Itereamos por la lista de direcciones, comprobando si es alguna de las nuestras. */
   for (ptr = direcciones; ptr != NULL; ptr = ptr->ai_next) {
      if (((PSOCKADDR_IN)ptr->ai_addr)->sin_addr.s_addr == origen->sin_addr.s_addr) {
         /* Si la IP es nuestra, indicamos que asi lo es. */
         freeaddrinfo(direcciones);
         return TRUE;
      }
   }

   /* Liberamos la lista de nombres. */
   freeaddrinfo(direcciones);

   return FALSE; /* Ninguna era la adecuada. */
}
```

13.2 Estimador NNLS

13.2.1 Estimador.c

```
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include "nnls.h"

/* Fichero: Estimador.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.0
 * Descripción: Código que permite invocar al estimador NNLS para obtener
 *              estimaciones de curvas de tercer grado no negativas. Se trata
 *              un programa sin dependencias Win32, solo a la libreria estandar
 *              de C y a NNLS.lib.
 */

int main(int argc, char ** argv)
{
   double * a = NULL;
   double * b = NULL;
   double * x = NULL;
   double error = 0.0;
   int grado = 0;
   int num_datos = 0;
   FILE * fichero = NULL;
   errno_t err;
   int i;
```

```
    /* Comprobamos los argmentos. */
    if (argc != 3) {
       fprintf(stderr, "Uso: %s <grado> <fichero>\n", argv[0]);
       exit(1);
    }

    /* Parseamos los argumentos. */
    if (sscanf_s(argv[1], "%d", &grado) != 1 || grado <= 0) {
       fprintf(stderr, "El grado proporcionado no es valido (%d)\n", grado);
       exit(2);
    }

    if ((err = fopen_s(&fichero, argv[2], "r")) != 0) {
       fprintf(stderr, "No se ha podido abrir el fichero con los datos a ajustar (%s, %d)\n", argv[2],
err);
       exit(3);
    }

    /* Leemos cuantos datos tenemos desde la primera linea del fichero. */
    if (fscanf_s(fichero, "%d", &num_datos) != 1) {
       fprintf(stderr, "No se ha podido leer el numero de elementos del fichero\n");
       fclose(fichero);
       exit(4);
    }

    /* Reservamos memoria para el procesamiento. */
    if ((a = (double *)malloc(num_datos * sizeof(double))) == NULL) {
       fprintf(stderr, "No se ha podido reservar memoria (a)\n");
       fclose(fichero);
       exit(5);
    }

    if ((b = (double *)malloc(num_datos * sizeof(double))) == NULL) {
       fprintf(stderr, "No se ha podido reservar memoria (b)\n");
       free(a);
       fclose(fichero);
       exit(6);
    }

    if ((x = (double *)malloc((grado + 1) * sizeof(double))) == NULL) {
       fprintf(stderr, "No se ha podido reservar memoria (x)\n");
       free(a);
       free(b);
       fclose(fichero);
       exit(7);
    }

    /* Leemos los datos del fichero. */
    for (i = 0; i < num_datos; ++i) {
       if (fscanf_s(fichero, "%lf\t%lf", &(a[i]), &(b[i])) != 2) {
          fprintf(stderr, "No se ha podido leer la linea %d. Formato no valido\n", i + 1);
          free(a);
          free(b);
          free(x);
          fclose(fichero);
          exit(8);
       }
    }

    /* Lanzamos el NNLS sobre los datos. */
    memset(x, 0, sizeof(double) * (grado + 1));
    if ((error = nnls(a, b, num_datos, x, grado)) == -1) {
       fprintf(stderr, "Error al procesar los datos con NNLS\n");
       free(a);
       free(b);
       free(x);
       fclose(fichero);
       exit(9);
    }

    printf("=====================================\n");
    printf("Estimacion NNLS de polinomio de grado %d\n", grado);
    printf("=====================================\n");
    printf("Numero de datos = %d\n", num_datos);
```

```
   printf("ECM = %.2lf\n", error);
   printf("Coeficientes (grado creciente):\n");

   for (i = 0; i < grado + 1; ++i) {
      printf("\tcoef. grado %d = %.13lf\n", i, x[i]);
   }

   printf("=====================================");

   free(a);
   free(b);
   free(x);
   fclose(fichero);

   return 0;
}
```

13.3 Inyector de carga

13.3.1 Inyector.h

```
#ifndef __INYECTOR_H__
#define __INYECTOR_H__

/* Fichero: Inyector.h
 * Autor: Ramón Medrano Llamas
 * Versión: 1.5
 * Descripción: Definiciones para el modulo inyector de carga.
 */

#include "Comun.h"

/* Tipo para la informacion de los hilos clientes. */
typedef struct _HILOCLIENTE {
   HANDLE handleHilo;
   /* Indica si el hilo está activo o no, para que otros hilos lo activen. */
   BOOL activo;
   /* Informacion de instrumentacion. */
   UINT64 sesionesValidas;
   UINT64 petcionesValidas;
   DOUBLE tiempoRespuestaMedio;
   DOUBLE tiempoSesionMedio;
} HILOCLIENTE, * PHILOCLIENTE;

/* Funciones auxiliares definidas en Auxiliares.c */
VOID MostrarUso(PCSTR ejecutable);

/* Funciones de control de carga definidas en GestorCarga.c */
INT32 ObjetivoInicial(PCSTR pathFicheroCarga); // Devuelve -2 si no hay fichero y -1 en error.
DWORD WINAPI GestorCarga(LPVOID parametro);

/* Funciones principales definidas en Main.c */
VOID SimularSesion(UINT16 numHilo);
DWORD WINAPI Usuario(LPVOID parametro);
INT32 main(INT32 argc, PCHAR argv[]);

#endif
```

13.3.2 Auxiliares.c

```
#include "Inyector.h"

/* Fichero: Auxiliares.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.4
 * Descripción: Funciones de utilidad para el modulo inyector de carga.
 */

VOID MostrarUso(PCSTR ejecutable)
{
   /* No bloqueamos este uso de stderr porque se hará en un contexto mono-hilo. */
   printf("\nUso: %s DIR_SERVIDOR PUERTO NUM_CLIENTES TIEMPO_REFLEXION INTERVALO_ARRANQUE
TIEMPO_MEDICION FICHERO_VOLCADO PETICIONES TAM_PETICION [ FICHERO_CARGA AMORTIGUACION ]\n",
ejecutable);
   printf("\n    DIR_SERVIDOR        - direccion de la maquina con el servicio.\n");
   printf("    PUERTO              - puerto de servicio\n");
   printf("    NUM_CLIENTES        - numero de clientes a simular (numero de hilos)\n");
   printf("    TIEMPO_REFLEXION    - tiempo medio de reflexion (en segundos)\n");
   printf("    INTERVALO_ARRANQUE  - tiempo (en segundos) del transitorio\n");
   printf("    TIEMPO_MEDICION     - tiempo (en segundos) de la medida\n");
   printf("    FICHERO_VOLCADO     - nombre para el fichero de volcado\n");
   printf("    PETICIONES          - media del numero de peticiones por sesion\n");
   printf("    TAM_PETICION        - tamano de la peticion y respuesta (en bytes)\n\n");
   printf("    FICHERO_CARGA (opc) - fichero con las curvas de carga\n");
   printf("    AMORTIGUACION (opc) - factor de amortiguacion para el incremento de\n");
   printf("                          carga. Por omision se toma 0,5.\n");
}
```

13.3.3 GestorCarga.c

```
#include "Inyector.h"

/* Fichero: GestorCarga.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.2
 * Descripción: Implementación del modulo que controla la generación de curvas
 *              de carga de forma dinámica.
 */

extern UINT32 cargaActual;
extern UINT32 objetivoCarga;
extern CRITICAL_SECTION cargaActualCriticalSection;
extern HANDLE handleGestorCarga;
extern UINT64 peticionesIntervalo;

INT32 ObjetivoInicial(PCSTR pathFicheroCarga)
{
   PFILE fichero;
   INT32 intervalo, nuevaCarga;

   if (pathFicheroCarga == NULL) {
      return -1;
   }

   if ((fichero = fopen(pathFicheroCarga, "r")) == NULL) {
      if (errno == ENOENT)
         return -1;
      else
         return -2;
   }

   fscanf(fichero, "%d %d", &intervalo, &nuevaCarga);
```

```
   fflush(fichero);
   fclose(fichero);

   return nuevaCarga;
}

DWORD WINAPI GestorCarga(LPVOID p)
{
   PFILE fichero;
   UINT32 nuevaCarga, intervalo;
   HANDLE temporizador;
   LARGE_INTEGER liIntervalo;
   CHAR hora[MAX_CTIME_STRING];
   time_t secs;
   UINT64 peticiones;

   if (p == NULL) {
      printf("AVISO: No se generara carga dinamica\n");
      ExitThread(EXIT_FAILURE);
   }

   if ((temporizador = CreateWaitableTimer(NULL, TRUE, "temporizador_inyector")) == NULL) {
      Error("No se pudo crear el temporizador.");
      ExitThread(EXIT_FAILURE);
   }

   if ((fichero = fopen((PCSTR)p, "r")) == NULL) {
      if (errno == ENOENT)
         printf("AVISO: No se generara carga dinamica\n");
      else
         Error("No se pudo abrir el fichero de carga.");
      CloseHandle(temporizador);
      ExitThread(EXIT_FAILURE);
   }

   /* Mientras haya eventos en el fichero de curvas de carga, leemos uno. */
   while (!feof(fichero) && !ferror(fichero)) {
      /* Leemos la entrada correspondiente en el fichero de carga. */
      fscanf(fichero, "%d %d", &intervalo, &nuevaCarga);

      /* Establecemos el nivel de carga que indica el fichero. */
      EnterCriticalSection(&cargaActualCriticalSection);
      objetivoCarga = nuevaCarga;
      time(&secs);
      LeaveCriticalSection(&cargaActualCriticalSection);

      ctime_s(hora, MAX_CTIME_STRING, &secs);
      printf("[%.2lf] Establecido nuevo objetivo de carga de %d usuarios. Hora = %s", TiempoAbsoluto(),
objetivoCarga, hora);

      /* Esperamos tantos segundos como nos indique el  evento. */
      printf("[%.2lf] Proximo cambio de carga en %d segundos.\n", TiempoAbsoluto(), intervalo);
      liIntervalo.QuadPart = ((LONGLONG)intervalo) * -10000000; // Grupos de 100 nanosegundos.
Relativo.

      if (SetWaitableTimer(temporizador, &liIntervalo, 0, NULL, NULL, 0) == FALSE) {
         Error("No se pudo configurar el temporizador");
         ExitThread(EXIT_FAILURE);
      }

      /* Espera por el intervalo. */
      WaitForSingleObject(temporizador, INFINITE);

      /* Al terminar el periodo de carga, obtenemos la productividad del periodo.. */
      EnterCriticalSection(&cargaActualCriticalSection);
      peticiones = peticionesIntervalo;
      peticionesIntervalo = 0;
      LeaveCriticalSection(&cargaActualCriticalSection);

      /* Mostramos las productividad intra-periodo. */
      if (intervalo != 0)
         printf("[%.2lf] Productividad del periodo con %d usuarios = %.4lf peticiones/s\n",
TiempoAbsoluto(), nuevaCarga, ((DOUBLE)peticiones) / ((DOUBLE)intervalo));
      else
```

```
         printf("[%.2lf] No ha lugar a calcular la productividad para %d usuarios (intervalo 0).\n",
TiempoAbsoluto(), nuevaCarga);
   }

   fflush(fichero);
   fclose(fichero);
   CloseHandle(temporizador);

   printf("[%.2lf] Finalizada la generacion dinamica de carga.\n", TiempoAbsoluto());

   return EXIT_SUCCESS;
}
```

13.3.4 Inyector.c

```
#include "Inyector.h"

/* Fichero: Inyector.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.5
 * Descripción: Implementación del modulo inyector de carga.
 */

/* Variable con la direccion del servidor. */
IN_ADDR direccionServidor;
UINT16 puertoServidor;

/* Numero de hilos (clientes) de la simulacion. */
UINT16 numHilos;

/* Variable global con la hora de inicio de la prueba, para sincronizar con
   los datos del perfmon. */
time_t instanteInicioHora;
DOUBLE horaInicio, horaFinal;
CHAR cadenaHoraInicio[MAX_CTIME_STRING], cadenaHoraFinal[MAX_CTIME_STRING];

/* Tamaño de la petición (en bytes). */
SIZE_T tamanoPeticion;

/* Variable global con el limite máximo de peticiones por sesion. */
UINT32 mediaPeticionesPorSesion;

/* Variable global con la media del tiempo de reflexion (milisegundos). */
DOUBLE mediaTiempoReflexion;

/* Tiempo del transitorio de arranque y de medición en milisegundos. */
DOUBLE tiempoTransitorio;
DOUBLE tiempoMedicion;

/* Tiempo lmite de la medicion en milisegundos:
   tiempoLimite = tiempoTransitorio + tiempoMedicion */
DOUBLE tiempoLimite;

/* Informacion de los hilos cliente para permitir la activacion dinamica. */
PHILOCLIENTE informacionHilo;

/* Factor de amortiguacion para lanzar nuevos hilos. */
DOUBLE factorAmortiguacion;

/* Fichero de volcado de datos. */
PFILE ficheroVolcado;
CRITICAL_SECTION volcadoCriticalSection;

/* Variables que permiten realizar el control dinámico de la generación de carga
   mediante la activación dinámica de hilos y su desactivación. Todo se maneja
   mediante dos variables:
     - Número actual de hilos, incializada en el main() al número de hilos (todos
       empiezan activos).
```

```
      - Objetivo actual de hilos, actualizada por el hilo GestorCarga en funcion
        de los objetivos y la temporizacion definidas en los ficheros de carga. */
BOOL generandoCarga;
UINT32 cargaActual;
UINT32 objetivoCarga;
CRITICAL_SECTION cargaActualCriticalSection; // Mutex que protege la variable cargaActual.
HANDLE handleGestorCarga; // Identificador del hilo gestor carga.
UINT64 peticionesIntervalo; // Peticiones realizadas en un intervalo (permite calcular
                            // productividades inter-intervalo de carga, o sea, por nivel
                            // de carga inyectada).

/* Funcion que realiza las peticiones al servidor, se conectara pedira una
   cadena y recibira el resultado un numero determinado de veces, constituyendo
   asi una sesion de servicio.
   Recibe como parametro el numero de hilo para acceder a la estructura de
   datos y el tiempo de reflexion que se ha de esperar tras terminar.
   Devuelve el tiempo en milisegundos de final de la sesion, contando desde el
   inicio del experimento. Esto permite ver si paramos o no de hacer peticiones
   en el hilo que simula el cliente cuando este tiempo se salga del limite de
   medicion establecido por el usuario.
   Antes de terminar y si la sesion es valida (esta dentro de los limites de
   tiempo establecidos) se almacena sus datos de instrumentacion en la estructura
   definida a tal efecto. */
VOID SimularSesion(UINT16 numHilo)
{
   PBYTE buffer;
   SOCKADDR_IN direccion;
   SOCKET sock;
   DOUBLE tiempoInicioSesion, tiempoFinalSesion, tiempoSesion;
   DOUBLE tiempoInicioPeticion, tiempoFinalPeticion;
   UINT32 peticiones, i;
   PDOUBLE tiemposRespuesta;
   DOUBLE acumulador;

   /* Inicializar estrucutra de direccion. */
   direccion.sin_family = AF_INET;
   direccion.sin_addr.s_addr = direccionServidor.s_addr; // Ya la tenemos resuelta.
   direccion.sin_port = htons(puertoServidor);

   /* Crear socket. */
   if ((sock = socket(PF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
      Error("Error al crear el socket");
      return;
   }

   /* Configurar el linger. */
   if (!ConfiguraSocket(sock)) {
      Error("No se pudo configurar el socket");
      /* Continuamos, no es crítico. */
   }

   /* Calculamos la cantidad de peticiones a realizar. */
   peticiones = (UINT32)ceil(DistribucionExponencial((DOUBLE)mediaPeticionesPorSesion));
   tiemposRespuesta = (PDOUBLE)malloc(sizeof(DOUBLE) * peticiones);
   buffer = (PBYTE)malloc(tamanoPeticion);

   /* Es aqui donde obtenemos el contador de ticks actuales. No operaremos con el hasta
      el final para evitar introducir mas instrucciones en el codigo a medir. */
   tiempoInicioSesion = TiempoAbsoluto();

   /* Conectarse. */
   if (connect(sock, (PSOCKADDR)&direccion, sizeof(SOCKADDR)) == SOCKET_ERROR) {
      Error("Error al conectar al servidor (%d)\n", WSAGetLastError());
      free(tiemposRespuesta);
      free(buffer);
      return;
   }

   /* Hacer las peticiones de la sesión. */
   for (i = 0; i < peticiones; ++i) {

      /* Contamos el tiempo de la petición. */
      tiempoInicioPeticion = TiempoAbsoluto();
```

```
      /* Enviar (enviamos todo de una vez). */
      if (send(sock, (PCSTR)buffer, (INT32)tamanoPeticion, 0) == SOCKET_ERROR) {
         if (!(WSAGetLastError() == WSAECONNABORTED))
            Error("No se ha enviado correctamente la peticion (%d)\n", WSAGetLastError());

         CierraSocket(sock);

         /* Nos dormimos lo estipulado por el tiempo de reflexión doblado, evita saturar el
distribuidor. */
         Sleep((DWORD)DistribucionExponencial(2.0 * mediaTiempoReflexion));

         /* Liberación de recursos. */
         free(tiemposRespuesta);
         free(buffer);
         return;
      }

      /* Recibir (recibimos todo de una vez). */
      if (recv(sock, (PSTR)buffer, (INT32)tamanoPeticion, MSG_WAITALL) == SOCKET_ERROR) {
         if (!(WSAGetLastError() == WSAECONNRESET))
            Error("No se ha recibido correctamente la respuesta (%d)\n", WSAGetLastError());

         CierraSocket(sock);

         /* Nos dormimos lo estipulado por el tiempo de reflexión. */
         Sleep((DWORD)DistribucionExponencial(2.0 * mediaTiempoReflexion));

         /* Liberación de recursos. */
         free(tiemposRespuesta);
         free(buffer);
         return;
      }

      /* Contamos el tiempo. */
      tiempoFinalPeticion = TiempoAbsoluto();

      /* Nos dormimos. */
      Sleep((DWORD)DistribucionExponencial(mediaTiempoReflexion));

      /* Guardamos el tiempo de respuesta. */
      tiemposRespuesta[i] = tiempoFinalPeticion - tiempoInicioPeticion;

      /* Apuntamos la petición recién completada para el cálculo de productividad del período. */
      EnterCriticalSection(&cargaActualCriticalSection);
      peticionesIntervalo++;
      LeaveCriticalSection(&cargaActualCriticalSection);
   }

   /* Cerrar socket para concluir la sesión. */
   CierraSocket(sock);

   /* Ahora medimos el tiempo final de la sesión */
   tiempoFinalSesion = TiempoAbsoluto();

   /* Procesamos estos tiempos para saber si la sesion completa es valida... */
   if (tiempoInicioSesion > tiempoTransitorio && tiempoFinalSesion < tiempoLimite) {
      /* ... en cuyo caso, la anotamos en nuestra lista de datos.
         Escribimos los tiempos de respuesta al fichero. */
      EnterCriticalSection(&volcadoCriticalSection);
      acumulador = 0.0;
      for (i = 0; i < peticiones; ++i) {
         /* El timestamp tiene como objeto poder ordenar los datos. */
         fprintf(ficheroVolcado, "%d;%lf;%d;%lf\n", numHilo, tiemposRespuesta[i], cargaActual,
tiempoInicioSesion);
         /* Aprovechamos para acumular la cuenta. */
         acumulador += tiemposRespuesta[i];
      }
      fflush(ficheroVolcado);
      LeaveCriticalSection(&volcadoCriticalSection);

      /* Calculamos el tiempo de sesion. */
      tiempoSesion = tiempoFinalSesion - tiempoInicioSesion;

      /* Actualizamos el conteo de peticiones. */
```

```
      informacionHilo[numHilo].petcionesValidas = informacionHilo[numHilo].petcionesValidas +
peticiones;
      /* Actualizamos el conteo de sesiones. */
      informacionHilo[numHilo].sesionesValidas = informacionHilo[numHilo].sesionesValidas + 1;
      /* Actualizamos la media de tiempo de respuesta. */
      informacionHilo[numHilo].tiempoRespuestaMedio =
         (informacionHilo[numHilo].tiempoRespuestaMedio * informacionHilo[numHilo].petcionesValidas +
acumulador)
         / informacionHilo[numHilo].petcionesValidas;
      /* Actualizamos la media del tiempo de sesion. */
      informacionHilo[numHilo].tiempoSesionMedio =
         (informacionHilo[numHilo].tiempoSesionMedio * informacionHilo[numHilo].sesionesValidas +
tiempoSesion)
         / informacionHilo[numHilo].sesionesValidas;
   }

   free(tiemposRespuesta);
   free(buffer);
}

/* Funcion que simula un usuario del servidor. Se ejecutara en un hilo y
   recibe como parametro el numero de hilo del que se trata.
   Simulara tantas sesiones como pueda en el tiempo de medicion establecido. */
DWORD WINAPI Usuario(LPVOID parametro)
{
   UINT16 numHilo;
   UINT32 diferenciaCarga, i, j;
   DOUBLE hora;

   numHilo = *((PUINT16)parametro);
   free(parametro);

   /* Suspender el hilo al principio si supera la carga marcada. El hilo 0 siempre
      estará activo.*/
   EnterCriticalSection(&cargaActualCriticalSection);
   if (generandoCarga && numHilo != 0 && numHilo >= objetivoCarga) { // Seguro para objetivos iniciales
diferentes a 0.
      Debug("El hilo %d se va a suspender al inicio (cargaActual = %d, objetivoCarga = %d)", numHilo,
cargaActual, objetivoCarga);
      informacionHilo[numHilo].activo = FALSE;
      LeaveCriticalSection(&cargaActualCriticalSection);
      SuspendThread(informacionHilo[numHilo].handleHilo);
      /* Usar goto, aunque sucio, permite mejorar la robustez cuando los hilos
         son despertados por terceros. */
      goto loop;
   } else {
      Debug("El hilo %d se activa al inicio (cargaActual = %d, objetivoCarga = %d)", numHilo,
cargaActual, objetivoCarga);
      informacionHilo[numHilo].activo = TRUE;
      ++cargaActual;
      Debug("El hilo %d se ha activado al inicio (cargaActual = %d, objetivoCarga = %d)", numHilo,
cargaActual, objetivoCarga);
      LeaveCriticalSection(&cargaActualCriticalSection);
      goto loop;
   }

loop:
   hora = TiempoAbsoluto();
   while (hora < horaFinal) {
      /* Hacer peticion al servidor recogiendo el tiempo de cuando empezo. */
      SimularSesion(numHilo);

      /* Hemos de comprobar el objetivo de carga actual y la carga actual. */
      if (generandoCarga) {
         EnterCriticalSection(&cargaActualCriticalSection);
         if (cargaActual > objetivoCarga) {
            /* Actualizamos la informacion de carga. */
            informacionHilo[numHilo].activo = FALSE;
            --cargaActual;
            /* Liberamos los recursos antes de dormir. */
            Debug("El hilo %d se va a suspender por reduccion de carga (cargaActual = %d, objetivoCarga
= %d)", numHilo, cargaActual, objetivoCarga);
            LeaveCriticalSection(&cargaActualCriticalSection);
            /* Nos dormimos con SuspendThread. */
```

```
            SuspendThread(informacionHilo[numHilo].handleHilo);
            /* Usar goto, aunque sucio, permite dar mas robustez cuando los hilos son
               despertados por tercerlos hilos. */
            goto exit;
         } else if (cargaActual < objetivoCarga) {
            /* Habra que despertar tantos como queden por despertar, además de un factor.
               de multiplicacion. */
            diferenciaCarga = (UINT32)ceil(((DOUBLE)(objetivoCarga - cargaActual)) *
factorAmortiguacion);
            Debug("Despertando %d hilos desde el hilo %d (cargaActual = %d, objetivoCarga = %d)",
diferenciaCarga, numHilo, cargaActual, objetivoCarga);

            for (i = 0, j = 0; i < numHilos && j < diferenciaCarga; ++i) {
               if (informacionHilo[i].activo == FALSE) {
                  informacionHilo[i].activo = TRUE;
                  /* Nos aseguramos de que el hilo despierta. */
                  while (ResumeThread(informacionHilo[i].handleHilo) > 1);
                  ++cargaActual;
                  ++j;
               }
            }

            /* Aviso de que no hay bastantes hilos. */
            if (j < diferenciaCarga) {
               Debug("No hay suficientes hilos en el pool configurado");
            }

            /* Liberamos los recursos */
            Debug("Despertados %d hilos desde el hilo %d (cargaActual = %d, objetivoCarga = %d)",
diferenciaCarga, numHilo, cargaActual, objetivoCarga);
            LeaveCriticalSection(&cargaActualCriticalSection);
            goto exit;
         } else {
            /* Si son iguales, liberamos los mutexes par evitar interbloqueos. */
            Debug("El hilo %d no realizara ninguna accion (cargaActual = %d, objetivoCarga = %d)",
numHilo, cargaActual, objetivoCarga);
            LeaveCriticalSection(&cargaActualCriticalSection);
            goto exit;
         }
      }

exit:
      /* Verificamos la hora para terminar la simuacion. */
      hora = TiempoAbsoluto();
   }

   /* Antes de terminar, debemos de despertar todos los hilos dormidos para que terminen su ejecución,
al
      estar fuera de hora, ninguna de sus sesiones será válida. */
   for (i = 0; i < numHilos; ++i)
      while (ResumeThread(informacionHilo[i].handleHilo) > 1);

   return 0;
}

INT32 main(INT32 argc, PCHAR argv[])
{
   UINT32 i;
   PUINT16 param;
   UINT64 totalSesiones, totalPeticiones;
   DOUBLE mediaTiemposRespuesta, mediaTiemposSesion;
   time_t tmp;

   /* Cargamos la librería de utilidades comunes. */
   if (!InicializarLibreria(_time32(NULL))) {
      fprintf(stderr, "Error fatal.\n");
      exit(EXIT_FAILURE);
   }

   /* Validacion de los parametros recibidos por consola.
       - Direccion del servidor.
       - Numero de clientes.
       - Tiempo medio de reflexión.
       - Intervalos de arranque y medicion.
```

```
        - Nombre del fichero de salida de datos.
        - Numero medio de peticiones por sesion. */
    if (argc != 10 && argc != 12) {
       MostrarUso(argv[0]);
       exit(EXIT_FAILURE);
    }

    puertoServidor = (UINT16)strtol(argv[2], NULL, 10);
    numHilos = (UINT16)strtol(argv[3], NULL, 10);
    mediaTiempoReflexion = strtod(argv[4], NULL) * 1000.0;
    tiempoTransitorio = strtod(argv[5], NULL) * 1000.0;
    tiempoMedicion = strtod(argv[6], NULL) * 1000.0;
    mediaPeticionesPorSesion = strtol(argv[8], NULL, 10);
    tamanoPeticion = (SIZE_T)strtol(argv[9], NULL, 10);
    peticionesIntervalo = 0;

    if (argc == 12) {
       generandoCarga = TRUE;
       factorAmortiguacion = strtod(argv[11], NULL);
       printf("[ControlCarga] Factor de amortiguacion establecido en %.2lf\n", factorAmortiguacion);

       if (factorAmortiguacion <= 0.0 || factorAmortiguacion > 1.0)
          ErrorExit("Factor de amortiguacion no valido: 0 < factor <= 1");
    } else {
       generandoCarga = FALSE;
       factorAmortiguacion = 1.0;
    }

    if (mediaTiempoReflexion < 0.0 || tiempoTransitorio < 0.0 || tiempoMedicion <= 0.0)
       ErrorExit("Parametros de tiempos indicados no validos");

    /* Inicializamos los mecanismos de sincronización. */
    InitializeCriticalSection(&volcadoCriticalSection);
    InitializeCriticalSection(&cargaActualCriticalSection);

    /* Resolvemos el nombre del servidor. */
    if (!ResuelveNombre(argv[1], &direccionServidor))
       ErrorExit("Direccion del servidor no valida");

    /* Inicializacion de los parametros de carga dinamica. */
    if (generandoCarga) {
       cargaActual = 0;  // Todos los hilos empezaran parados.
       objetivoCarga = ObjetivoInicial(argv[10]); // Buscamos la carga objetivo.
       printf("[ControlCarga] El objetivo de carga inicial se ha establecido en %d usuarios.\n",
objetivoCarga);

       if (objetivoCarga < 0)
          ErrorExit("No se puede leer el fichero de carga dinamica");
    } else {
       cargaActual = 0;
       /* Asi todos los hilos despertaran al principio y punto. */
       objetivoCarga = numHilos;
    }

    /* Abrimos los recursos de instrumentacion. */
    if ((ficheroVolcado = fopen(argv[7], "w")) == NULL)
       Error("No se pudo crear el fichero de volcado");

    fprintf(ficheroVolcado, "ID hilo;R;Carga inyectada;Timestamp\n");

    /* Inicializacion de los espacios de memoria. */
    informacionHilo = (PHILOCLIENTE)malloc(sizeof(HILOCLIENTE) * numHilos);
    if (informacionHilo == NULL)
       ErrorExit("No se pudo reservar memoria para los hilos");

    /* Calculamos el tiempo limite en ms de la medicion. */
    tiempoLimite = tiempoTransitorio + tiempoMedicion;

    /* Calculamos las horas para sincronizarse con perfmon. */
    time(&instanteInicioHora);
    horaInicio = TiempoAbsoluto() + tiempoTransitorio;
    horaFinal = horaInicio + tiempoMedicion;

    tmp = instanteInicioHora + (time_t)(tiempoTransitorio / 1000.0);
```

```
   ctime_s(cadenaHoraInicio, MAX_CTIME_STRING, &tmp);
   cadenaHoraInicio[24] = '\0';
   tmp = instanteInicioHora + (time_t)(tiempoMedicion / 1000.0);
   ctime_s(cadenaHoraFinal, MAX_CTIME_STRING, &tmp);
   cadenaHoraFinal[24] = '\0';

   printf("[%.2lf] Hoas de inicio y fin de medicion: %s - %s\n", TiempoAbsoluto(), cadenaHoraInicio,
cadenaHoraFinal);

   /* Lanzamiento de los hilos-cliente. Establecemos el objetivo de carga a 0 para que se
      lancen los hilos suspendidos. */
   for (i = 0; i < numHilos; ++i) {
      param = (PUINT16)malloc(sizeof(UINT16));
      *param = (UINT16)i;
      informacionHilo[i].activo = FALSE;
      informacionHilo[i].sesionesValidas = 0;
      informacionHilo[i].petcionesValidas = 0;
      informacionHilo[i].tiempoRespuestaMedio = 0.0;
      informacionHilo[i].tiempoSesionMedio = 0.0;
      informacionHilo[i].handleHilo = CreateThread(NULL, 0, Usuario, param, 0, NULL);

      if (informacionHilo[i].handleHilo == NULL)
         ErrorExit("Error al lanzar los hilos trabajadores");
   }

   /* Lanzamos el hilo gestor de carga, ahora que esta todo en funcionamiento. */
   if ((handleGestorCarga = CreateThread(NULL, 0, GestorCarga, (LPVOID)argv[10], 0, NULL)) == NULL)
      ErrorExit("Error al lanzar el hilo de gestion de carga");

   /* Espera por los hilos-cliente a que terminen. */
   // Esperar por el hilo de carga puede quitar coherencia a definir una duración de simulación.
   //WaitForSingleObject(handleGestorCarga, INFINITE);
   for (i = 0; i < numHilos; ++i)
      WaitForSingleObject(informacionHilo[i].handleHilo, INFINITE);

   /* Cálculo de las estádisticas de la inyección. */
   mediaTiemposRespuesta = mediaTiemposSesion = 0.0;
   totalSesiones = totalPeticiones = 0;
   for (i = 0; i < numHilos; ++i) {
      totalSesiones += informacionHilo[i].sesionesValidas;
      totalPeticiones += informacionHilo[i].petcionesValidas;
      mediaTiemposRespuesta += informacionHilo[i].tiempoRespuestaMedio;
      mediaTiemposSesion += informacionHilo[i].tiempoSesionMedio;
   }

   printf("\nMedicion completada. Resultados\n");
   printf("==============================\n");
   printf("  Hora de inicio: %s\n", cadenaHoraInicio);
   printf("  Hora de final: %s\n", cadenaHoraFinal);
   printf("  Tiempo de medicion: %.4lf s\n\n", tiempoMedicion / 1000.0);
   printf("  Total de sesiones: %d\n", totalSesiones);
   printf("  Total de peticiones: %d\n", totalPeticiones);
   printf("  Media de peticiones por sesison: %.4lf\n\n", ((DOUBLE)totalPeticiones) / totalSesiones);
   printf("  Media del tiempo de respuesta: %.4lf\n", mediaTiemposRespuesta / numHilos);
   printf("  Media del tiempo de sesion: %.4lf\n\n", mediaTiemposSesion / numHilos);
   printf("  Productividad total: %.4lf peticiones/s\n", ((DOUBLE)totalPeticiones) / (tiempoMedicion /
1000.0));

   fflush(ficheroVolcado);
   fclose(ficheroVolcado);
   free(informacionHilo);
   DeleteCriticalSection(&volcadoCriticalSection);
   DeleteCriticalSection(&cargaActualCriticalSection);
   LiberarLibreria();

   return EXIT_SUCCESS;
}
```

13.4 Librería NNLS

La siguiente librería se adjunta en formato de código fuente, pues ha sido adaptada al lenguaje C desde la implementación en FORTRAN de Lawson y Hanson. El código está fuertemente heredado de dicha implementación.

13.4.1 nnls.h

```
#ifndef __LIB_NNLS_H__
#define __LIB_NNLS_H__

/* Fichero: nnls.h
 * Autor: Ramon Medrano Llamas
 * Versión: 1.0
 * Descripción: Definciones e inclusiones para el módulo de calculo de curvas estimadas
 *              mediante los algoritmos LS y NNLS.
 *              Permite ser incluida en cualquier programa en C/C++ y derivados sin
 *              ninguna dependencia mas alla de la libreria estandar.
 *
 * The original version of this code was developed by
 * Charles L. Lawson and Richard J. Hanson at Jet Propulsion Laboratory
 * 1973 JUN 15, and published in the book
 * "SOLVING LEAST SQUARES PROBLEMS", Prentice-Hall, 1974.
 * Revised FEB 1995 to accompany reprinting of the book by SIAM.
 */

/* Dependencias de compilacion:
 *   Libreria estandar de C. Puede ser necesario en ciertos sistemas, como GNU/Linux
 *   o Solaris compilar con la opcion -lm (para enlazar con la libreria matematica).
 *   De esta libreria solo se usan las funciones fabs() y sqrt().
 */
#include <math.h>
#include <stdlib.h>

#ifndef max
#define max(x,y) (((x)<(y))?(y):(x))
#endif
/* Macro para devolver el signo del valor en forma de double. */
#define fsign(x,y) (((y)<(0.0))?(fabs(x)*(-1)):(fabs(x)))

/* Funcion: nnls()
 * Version: 1.0
 * Descripcion: Esta funcion calcula un vector de soluciones, x, para el problema restringido
 *              de minimos cuadrados siguiente:
 *                Ax ~= b, x >= 0
 *              A y b son los vectores para realizar la regresión restringida. Se produce un
 *              vector de soluciones correspondientes a los coeficientes de los elementos de un
 *              polinomio de grado "grado". Es necesario proporcionar un espacio de memoria para
 *              "grado + 1" elementos en el apuntador x. Los vectores a y b contienen "muestras"
 *              pares (x,y) con los que se hace la interpolación.
 *
 * Funciones invocadas: nnls_core()
 */
double nnls(double *a, double *b, int muestras, double *x, int grado);

/* LAS SIGUIENTES FUNCIONES SON DE USO INTERNO, AUNQUE PUEDEN SER DE UTILIDAD. */

/* Funcion: nnls_core()
 * Version: 1.0
 * Descripcion: Esta funcion calcula un vector de soluciones, x, para el problema restringido
 *              de minimos cuadrados siguiente:
```

```
 *                Ax ~= b, x >= 0
 *              A es una matriz dada de dimension MxN y b es un m-vector. Este problema siempre
 *              tiene solucion, pero sera no unica si el rango de A es inferior a N.
 *
 * Funciones invocadas: h12(), g1(), g2(), diff()
 */
int nnls_core(double *a, int mda, int m, int n, double *b, double *x, double *rnorm, double *w, double
*z, int *index, int *mode);

/* Funcion: n12()
 * Version: 1.0
 * Descripcion: Esta funcion implementa los algoritmos H1 y H2. Dado un m-vector v y los enteros
 *              lp y l1, calcula un m-vector u y un numero s tal que la matriz ortogonal
 *              simetrica MxM (Householder) Q satisface Qv = w. Opcionalmente, esta matriz
 *              puede ser multiplicada por un conjunto de m-vectores NCV.
 *
 * Funciones invocadas: -
 */
int h12(int mode, int lpivot, int l1, int m, double *u, int iue, double *up, double *c, int ice, int
icv, int ncv);

/* Funcion: g1()
 * Version: 1.0
 * Descripcion: Esta funcion, dados los numeros x1 y x2, calcula los datos que definen
 *              la matriz de ortogonal de rotacion G.
 *
 * Funciones invocadas: -
 */
void g1(double x1, double x2, double *c, double *s, double *r);

/* Funcion: g2()
 * Version: 1.0
 * Descripcion: Esta funcion calcula el producto matriz-vector, aplicando la rotacion
 *              calculada en g1().
 *
 * Funciones invocadas: -
 */
void g2(double c, double s, double *z1, double *z2);

/* Funcion: diff()
 * Version: 1.0
 * Descripcion: Esta funcion calcula la diferencia entre dos numeros en funcion de la
 *              precision de la maquina en la que se ejecuta.
 *
 * Funciones invocadas: -
 */
double diff(double x, double y);

#endif
```

13.4.2 diff.c

```
#include "nnls.h"

double diff(double x, double y)
{
   return x - y;
}
```

13.4.3 g.c

```
#include "nnls.h"
```

```
void g1(double x1, double x2, double *c, double *s, double *r)
{
   double xr, yr;

   if (fabs(x1) > fabs(x2)) {
      xr = x2 / x1;
      yr = sqrt(xr * xr + 1.0);
      *c = fsign(1.0 / yr, x1);
      *s = *c * xr;
      *r = fabs(x1) * yr;
      return;
   }

   if (x2 != 0.0) {
      xr = x1 / x2;
      yr = sqrt(xr * xr + 1.0);
      *s = fsign(1.0 / yr, x2);
      *c = *s * xr;
      *r = fabs(x2) * yr;
      return;
   }

   *r = 0.0;
   *s = 1.0;
   *c = 0.0;
}

void g2(double c, double s, double *z1, double *z2)
{
   double xr;

   xr = c * *z1 + s * *z2;
   *z2 = - s * *z1 + c * *z2;
   *z1 = xr;
}
```

13.4.4 h12.c

```
#include "nnls.h"

int h12(int mode, int lpivot, int l1, int m, double *u, int iue, double *up, double *c, int ice, int
icv, int ncv)
{
   int i, j, u_offset, i2, i3, i4, incr;
   double cl, clinv, sm, tmp, b;

   /*** AJUSTE DE PARAMETROS ***/
   u_offset = iue + 1;
   u -= u_offset;
   c--;

   if (0 >= lpivot || lpivot >= l1 || l1 > m)
      return -1;

   cl = fabs(u[lpivot * iue + 1]);

   if (mode == 2)
      goto L60;

   /*** CONSTRUIR LA TRANSFORMACION ***/
   for (j = l1; j <= m; j++)
      cl = max(fabs(u[j * iue + 1]), cl);

   if (cl <= 0.0)
      return -1;

   clinv = 1.0 / cl;
```

```
   sm = u[lpivot * iue + 1] * clinv;
   sm = sm * sm;

   for (j = l1; j <= m; j++) {
      tmp = u[j * iue + 1] * clinv;
      sm += tmp * tmp;
   }

   cl = cl * sqrt(sm);

   if (u[lpivot * iue + 1] > 0.0)
      cl = -cl;

   *up = u[lpivot * iue + 1] - cl;
   u[lpivot * iue + 1] = cl;
   goto L70;

   /*** APLICACION DE LA TRANSFORMACION ***/
L60:
   if (cl <= 0.0)
      return -1;

L70:
   if (ncv <= 0)
      return 0;

   b = *up * u[lpivot * iue + 1];

   if (b >= 0.0)
      return -1;

   b = 1.0 / b;
   i2 = 1 - icv + ice * (lpivot - 1);
   incr = ice * (l1 - lpivot);

   for (j = 1; j <= ncv; j++) {
      i2 += icv;
      i3 = i2 + incr;
      i4 = i3;
      sm = c[i2] * *up;

      for (i = l1; i <= m; i++) {
         sm += c[i3] * u[i * iue + 1];
         i3 += ice;
      }

      if (sm == 0.0)
         continue;

      sm = sm * b;
      c[i2] += sm * *up;

      for (i = l1; i <= m; i++) {
         c[i4] += sm * u[i * iue + 1];
         i4 += ice;
      }
   }

   return 0;
}
```

13.4.5 nnls.c

```
#include "nnls.h"

int nnls_core(double *a, int mda, int m, int n, double *b, double *x, double *rnorm, double *w, double
*z, int *index, int *mode)
```

```
{
   int i, ii, j, jj, l, a_offset, iter, itmax, iz, jz, izmax, iz1, iz2, nsetp, npp1, rtnkey, ip;
   double cc, ss, sm, wmax, alpha, asave, up, dummy, unorm, ftmp, ztest, t;

   a_offset = mda + 1;
   a -= a_offset;
   --b;
   --x;
   --w;
   --z;
   --index;

   *mode = 1;
   if (m <= 0 || n <= 0) {
      *mode = 2;
      return -1;
   }
   iter = 0;
   itmax = n * 3;
   izmax = 0;
   jj = 0;

   /* INICIALIZAR LOS ARRAYS index[] Y x[] */
   for (i = 1; i <= n; ++i) {
      x[i] = 0.0;
      index[i] = i;
   }

   iz2 = n;
   iz1 = 1;
   nsetp = 0;
   npp1 = 1;

   /*******  EL BUCLE PRINCIPAL EMPIEZA AQUI  *******/
L30:
   /* SALIR SI TODOS LOS COEFICIENTES ESTAN EN LA SOLUCION
      O SI m COLUMNAS DE a HAN SIDO TRIANGULARIZADAS. */
   if (iz1 > iz2 || nsetp >= m)
      goto L350;

   /* CALCULAR LAS COMPONENTES DEL GRADIENTE NEGATIVO DE w[] */
   for (iz = iz1; iz <= iz2; ++iz) {
      j = index[iz];
      sm = 0.0;
      for (l = npp1; l <= m; ++l)
         sm += a[l + j * mda] * b[l];
      w[j] = sm;
   }

   /* BUSCAR EL POSITIVO MAS GRANDE EN w[] */
L60:
   wmax = 0.0;
   for (iz = iz1; iz <= iz2; ++iz) {
      j = index[iz];
      if (w[j] > wmax) {
         wmax = w[j];
         izmax = iz;
      }
   }

   /* SI wmax <= 0 IR A LA TERMINACION
      ESTO INDICA LA SATISFACCION DE LAS CONDICIONES DE KUHN-TUCKER. */
   if (wmax <= 0.)
      goto L350;

   iz = izmax;
   j = index[iz];

   /* EL SIGNO DE w[j] PERMITE MOVER j AL CONJUNTO P.
      COMENZAR LA TRANSFORMACION Y COMPROBAR LA DIAGONAL PARA EVITAR
    DEPENDENCIA LINEAL */
   asave = a[npp1 + j * mda];
   h12(1, npp1, npp1 + 1, m, &a[j * mda + 1], 1, &up, &dummy, 1, 1, 0);
   unorm = 0.0;
```

```
   if (nsetp != 0)
      for (l = 1; l <= nsetp; ++l) {
         ftmp = a[l + j * mda];
         unorm += ftmp * ftmp;
      }
   unorm = sqrt(unorm);
   ftmp = unorm + fabs(a[npp1 + j * mda]) * 0.01;
   if (diff(ftmp, unorm) > 0.0) {
      /* LA COLUMNA j ES SUFICIENTEMENTE INDEPENDIENTE. COPIAR b[] EN z[]
         Y ACTUALIZAR z[]. RESOLVER EL Z-test */
      for (l = 1; l <= m; ++l)
         z[l] = b[l];
      h12(2, npp1, npp1 + 1, m, &a[j * mda + 1], 1, &up, &z[1], 1, 1, 1);
      ztest = z[npp1] / a[npp1 + j * mda];
      /* COMPROBAR SI EL Z-test ES POSITIVO */
      if (ztest > 0.0)
         goto L140;
   }

   /* DESCARTAR j COMO CANDIDATO A MOVER DESDE Z A P.
      RESTAURAR a[NPP1,j] Y PONER w[j] A 0. VOLVER A CALCULAR EL GRADIENTE
    NEGATIVO OTRA VEZ */
   a[npp1 + j * mda] = asave;
   w[j] = 0.0;
   goto L60;

   /* EL INDICE j HA SIDO SELECCIONADO PARA SER MOVIDO DE Z A P. ACTUALIZAR P,
      LOS INDICES, APLICAR LA TRANSFORMACION DE HOUSEHOLDER A LAS COLUMNAS EN Z
    Y ESTABLECER A 0 LA SUBDIAGONAL EN LA COLUMNA j*/
L140:
   for (l = 1; l <= m; ++l)
      b[l] = z[l];

   index[iz] = index[iz1];
   index[iz1] = j;
   ++iz1;
   nsetp = npp1;
   ++npp1;

   if (iz1 <= iz2)
      for (jz = iz1; jz <= iz2; ++jz) {
         jj = index[jz];
         h12(2, nsetp, npp1, m, &a[j * mda + 1], 1, &up, &a[jj * mda + 1], 1, mda, 1);
      }

   if (nsetp != m)
      for (l = npp1; l <= m; ++l)
         a[l + j * mda] = 0.0;
   w[j] = 0.0;

   /* RESOLVER EL SISTEMA TRIANGULAR, GUARDANDO LA SOLUCION TEMPORAL EN z[] */
   rtnkey = 1;
   goto L400;

   /******* EL BUCLE SECUNDARIO EMPIEZA AQUI *******/
L210:
   ++iter;
   if (iter > itmax) {
      *mode = 3;
      goto L350;
   }

   /* COMPROBAR SI LOS NUEVOS COEFICIENTES RESTRINGIDOS SON POSIBLES.
      SI NO, CALCULAR alpha*/
   alpha = 2.0;
   for (ip = 1; ip <= nsetp; ++ip) {
      l = index[ip];
      if (z[ip] <= 0.0) {
         t = -x[l] / (z[ip] - x[l]);
         if (alpha > t) {
            alpha = t;
            jj = ip;
         }
      }
```

```
   }

   /* SI TODOS LOS COEFICIENTES SON POSIBLES, alpha SEGUIRA SIENDO 2. SI ES
      ASI, SALIR DEL BUCLE INTERNO */
   if (alpha == 2.0)
      goto L330;

   /* SI NO SON POSIBLES, alpha ESTARA ENTRE 0 Y 1. INTERPOLAR ENTRE x[] Y z[] */
   for (ip = 1; ip <= nsetp; ++ip) {
      l = index[ip];
      x[l] += alpha * (z[ip] - x[l]);
   }

   /* MODIFICAR a Y b Y LOS INDICES PARA MOVER EL COEFICIENTE i DEL CONJUNTO
      P AL Z */
   i = index[jj];
L260:
   x[i] = 0.0;

   if (jj != nsetp) {
      ++jj;
      for (j = jj; j <= nsetp; ++j) {
         ii = index[j];
         index[j - 1] = ii;
         g1(a[j - 1 + ii * mda], a[j + ii * mda], &cc, &ss, &a[j - 1 + ii * mda]);
         a[j + ii * mda] = 0.0;
         for (l = 1; l <= n; ++l)
            if (l != ii) {

               /* TODO: APLICAR PROCEDIMIENTO g2 */

               ftmp = a[j - 1 + l * mda];
               a[j - 1 + l * mda] = cc * ftmp + ss * a[j + l * mda];
               a[j + l * mda] = -ss * ftmp + cc * a[j + l * mda];
            }

         /* TODO: APLICAR PROCEDIMIENTO g2 */

         ftmp = b[j - 1];
         b[j - 1] = cc * ftmp + ss * b[j];
         b[j] = -ss * ftmp + cc * b[j];
      }
   }

   npp1 = nsetp;
   --nsetp;
   --iz1;
   index[iz1] = i;

   /* COMPROBAR SI LOS COEFICIENTES RESTANTES SON POSIBLES. DEBERIAN DE SERLO
      PUESTO QUE alpha HA SIDO CALCULADO. SI NO LO SON ES POR UN ERROR DE
      PRECISION DE CALCULO, EN ESE CASO SE PONDRAN A CERO Y VOLVERAN AL
      CONJUNTO Z. */
   for (jj = 1; jj <= nsetp; ++jj)
      if (x[index[jj]] <= 0.0)
         goto L260;

   /* COPIAR b[] EN z[] Y VOLVER AL PRINCIPIO. */
   for (i = 1; i <= m; ++i)
      z[i] = b[i];
   rtnkey = 2;
   goto L400;

L320:
   goto L210;
   /*******  FIN DEL BUCLE SECUNDARIO  *******/

L330:
   for (ip = 1; ip <= nsetp; ++ip)
      x[index[ip]] = z[ip];

   /* TODOS LOS COEFICIENTES SON POSITIVOS, VOLVER AL PRINCIPIO. */
   goto L30;
```

```
   /*******  FIN DEL BUCLE PRINCIPAL  *******/

   /* CALCULAR EL RESIDUO */

L350:
   sm = 0.0;
   if (npp1 <= m)
      for (i = npp1; i <= m; ++i)
         sm += b[i] * b[i];
   else
      for (j = 1; j <= n; ++j)
         w[j] = 0.0;

   *rnorm = sqrt(sm);
   return 0;

   /* TODO: EL SIGUIENTE BLOQUE SE USA COMO UNA SUBRUTINA. ES MUY SUCIO. */

L400:
   for (l = 1; l <= nsetp; ++l) {
      ip = nsetp + 1 - l;
      if (l != 1)
         for (ii = 1; ii <= ip; ++ii)
            z[ii] -= a[ii + jj * mda] * z[ip + 1];

      jj = index[ip];
      z[ip] /= a[ip + jj * mda];
   }
   switch ((int)rtnkey) {
      case 1:
         goto L210;
      case 2:
         goto L320;
   }
   return 0;
}
```

13.4.6 vandermonde.c

```
#include "nnls.h"

double nnls(double *a, double *b, int muestras, double *x, int grado)
{
   double * matriz_E, * vector_B, * potencias, * w, * z, norm;
   int * index, i, j, num_elementos, max_potencia, mode, ret;

   /* ¿Cuantas potencias hay que calcular? ¿De que dimensiones son las matrices? */
   num_elementos = grado + 1;
   max_potencia = 2 * grado + 1;

   /* Reserva de espacios de memoria para las matrices. */
   matriz_E = (double *)malloc(sizeof(double) * num_elementos * num_elementos);
   vector_B = (double *)malloc(sizeof(double) * num_elementos);
   potencias = (double *)malloc(sizeof(double) * max_potencia);

   /* Calculamos las potencias de los elementos x que estan en a. */
   potencias[0] = muestras;
   for (i = 1; i < max_potencia; i++) {
      potencias[i] = 0.0;
      for (j = 0; j < muestras; j++) {
         potencias[i] += pow(a[j], i);
      }
   }

   /* La matriz E contiene la matroz de Vandermonde para los puntos de entrada. */
   for (i = 0; i < num_elementos; i++) {
      for (j = 0; j < num_elementos; j++) {
         matriz_E[i * num_elementos + j] = potencias[i + j];
```

```
      }
   }

   /* Calculamos el vector B. */
   for (i = 0; i < num_elementos; i++) {
      vector_B[i] = 0.0;
      for (j = 0; j < muestras; j++) {
         vector_B[i] += pow(a[j], i) * b[j];
      }
   }

   /* Invocamos el NNLS reservando memoria temporal. */
   w = (double *)malloc(sizeof(double) * num_elementos);
   z = (double *)malloc(sizeof(double) * num_elementos);
   index = (int *)malloc(sizeof(int) * num_elementos);
   ret = nnls_core(matriz_E, num_elementos, num_elementos, num_elementos, vector_B, x, &norm, w, z,
index, &mode);

   /* Liberamos los espacios de memoria temporales. */
   free(matriz_E);
   free(vector_B);
   free(potencias);
   free(w);
   free(z);
   free(index);

   /* Control de errores. */
   if (ret != 0)
      return -1;

   /* Devolvemos el ECM. */
   return (pow(norm, 2) / muestras);
}
```

13.5 Repartidor

13.5.1 Repartidor.h

```
#ifndef __REPARTIDOR_H__
#define __REPARTIDOR_H__

/* Fichero: Repartidor.h
 * Autor: Ramón Medrano Llamas
 * Versión: 2.1
 * Descripción: Definciones e inclusiones para el módulo repartidor de carga.
 */

/* Inclusiones. */
#include "Comun.h"

/* Macros. */
#define Siguiente(x,s) (((x)+1)%(s))
#define Anterior(x,s) ((((x)-1)<0)?((s)-1):((x)-1))
/* DESACTIVA EL APAGADO/ENCENDIDO FISICO */
#define _NO_APAGADO_FISICO_

/* Definición de tipos. */
typedef enum _ESTADOENERGETICO {
   /* Estados energeticos de los nodos. */
   apagado = 0,
   encendiendose = 1,
   encendido = 2,
   descargandose = 3,
```

```
   apagandose = 4,
   broken = -1
} ESTADOENERGETICO;

typedef enum _EVENTO {
   /* Posibles eventos que hacen cambiar al algoritmo. */
   nueva_sesion,
   fin_sesion,
   nodo_encendiendose,
   nodo_encendido,
   nodo_descargandose,
   nodo_descargado,
   nodo_apagandose,
   nodo_apagado,
   nodo_broken
} EVENTO;

typedef struct _NODO {
   /* Informacion relativa a un NODO. */
   UINT16 identificador;
   CHAR hostname[NI_MAXHOST];
   IN_ADDR direccion;
   UINT16 puerto;
   UINT8 mac[6];
   /* Informacion para la gestion energetica. */
   UINT32 sesiones;
   ESTADOENERGETICO estado;
   DOUBLE sla;
   PCURVAPOLINOMICA curvaRespuesta;
   PCURVAPOLINOMICA curvaPotencia; // La eficiencia es tambien una curva.
   DOUBLE consumo; // Consumo instantaneo anterior.
   DOUBLE energia; // Energia anterior.
   DOUBLE eficiencia; // Whora / sesion.
   UINT64 sesiones_servidas; // Numero de sesiones servidas.
   /* Información de estado de NODO. */
   DOUBLE modificador; // Factor de modificacion de capacidad.
   UINT32 capacidad; // Sesiones adimisibles.
   DOUBLE utilizacion; // Relativa al SLA.
   /* Temporizadores asociados al nodo. */
   DOUBLE timeoutApagado;
   HANDLE temporizadorEncendido;
   DOUBLE timeoutEncendido;
   HANDLE temporizadorApagado;
} NODO, * PNODO;

/* Informacion de una sesion. */
typedef struct _SESION {
   /* Identificador único de sesión. */
   UINT64 id;
   /* Para relacionarse con el cliente */
   SOCKET socket;
   SOCKADDR_IN cliente;
   /* Para relacionarse con el servidor (accediendo a nodosServidor) */
   INT32 nodo;
} SESION, * PSESION;

/* Unidad de información que almacenamos de cada petición para calcular la
   calidad de servicio que estamos ofreciendo. */
typedef struct _ENTRADAQOS {
   DOUBLE respuesta;
   time_t timestamp;
} ENTRADAQOS, * PENTRADAQOS;

/* Informacion de QoS para cada sesion. */
typedef struct _INFOQOS {
   /* Cola circular de registro de peticiones. */
   INT64 indice;
   INT64 tamano;
   DOUBLE intervaloQoS; // Almacenado en ms.
   /* Las entradas las reservamos dinamicamente para permitir varios tamaños. */
   PENTRADAQOS info;
   /* Información del hilo gestor de la QoS. */
   HANDLE threadQoS;
   CRITICAL_SECTION seccionCritica;
```

```
   /* Información estadística general para este periodo. */
   UINT64 sesiones;
   UINT64 rechazos;
   UINT64 peticiones;
} INFOQOS, * PINFOQOS;

/* Declaracion de funciones matematicas para el cálculo de eficiencias y
   utilizaciones de nodos. */
VOID ActualizarUtilizacion(PNODO nodo);
VOID ActualizarCapacidad(PNODO nodo);
DOUBLE CalcularPotenciaApagando(PNODO nodo);
DOUBLE CalcularPotenciaEncendiendo(PNODO nodo);

/* Declaración de funciones de parseo del fichero de configuración. */
BOOL ParseaFicheroConfiguracion(PCSTR fichero);

/* Declaración de funciones auxiliares. */
VOID InicializarLoggers(PCSTR cabecera);
VOID EmitirLog(PCSTR mensaje, ...);
VOID EmitirLogExcel(PSESION sesion, EVENTO evt);
INT32 HacerPeticionServidor(PSTR buffer, SIZE_T len, SOCKET sock);
VOID CerrarServicio(INT32 senal);
VOID InicializarSocketServicio(PSOCKET sock, UINT16 puerto);
VOID Limpiar();
BOOL ParseaDireccionMAC(PCSTR str, PUINT8 bin);

/* Declaración de funciones de manejo de la cola circular. */
VOID InicializarCola(PINFOQOS cola, UINT32 tamano, DOUBLE intervaloQoS);
VOID LiberarCola(PINFOQOS cola);

/* Declaracion de funciones de manejo de la QoS. */
VOID RegistrarPeticion(DOUBLE respuesta);
DWORD WINAPI HiloRegistroQoS(LPVOID p);

/* Declaracion de funciones de gestion energetica de nodos. */
VOID EncenderNodo(PSESION sesion);
VOID ApagarNodo(PSESION sesion);
VOID ApagarNodoFisicamente(PSESION sesion);

/* Declaración de funciones de manejo de WoL y demás. */
BOOL DespiertaNodo(PNODO nodo);
DWORD WINAPI HiloEsperaNotificaciones(LPVOID p);
BOOL EnviaMensajeApagado(PNODO nodo);

/* Funciones para el control del timing de los estados. */
VOID CALLBACK TimerApagadoExpirado(LPVOID p, BOOLEAN fired);
VOID CALLBACK TimerEncendidoExpirado(LPVOID p, BOOLEAN fired);

/* Declaración de funciones del distribuidor. */
INT32 ControlAdmision(PSESION sesion); // Control de admision.
VOID CerrarSesion(PSESION sesion); // Control del salida.
VOID ServirSesion(PSESION sesion);
DWORD WINAPI HiloServicioSesion(LPVOID p);
INT32 main(INT32 argc, PCHAR argv[]);

#endif
```

13.5.2 Auxiliares.c

```
#include "Repartidor.h"

/* Fichero: Auxiliares.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.6
 * Descripción: Funciones de ayuda para implementar el repartidor de carga. Principalmente
 *              son piezas de código extraidas del módulo principal para mejorar su
 *              legibilidad.
 */
```

```
extern PNODO nodosServidor; // Cada nodo tendra su propio mutex.
extern CRITICAL_SECTION nodosCriticalSection; // Bloquea la estructura al completo.
extern UINT16 numeroNodosServidor;
extern PFILE logger;
extern PFILE loggerExcel;
extern CRITICAL_SECTION loggerCriticalSection;
extern CRITICAL_SECTION loggerExcelCriticalSection;
extern SOCKET socketServidor;
extern CRITICAL_SECTION socketCriticalSection;
extern PSESION sesiones; // Tiene que ser dinamica entre ejecuciones.
extern INFOQOS qos; // Toda la informacion de QoS.
extern UINT16 tamanoPoolHilos;
extern PHANDLE hndsPoolHilos;
extern DOUBLE energiaTotal; // Energia consumida en total.
extern UINT64 transaccionesTotales; // Transacciones procesadas en total.
extern UINT64 sesionesTotales; // Sesiones totales.
extern UINT64 transaccionesMalas; // Transacciones por encima del SLA.
extern UINT64 sesionesRechazadas; // Sesiones rechazadas.
extern CRITICAL_SECTION datosTotalesCriticalSection;

VOID InicializarLoggers(PCSTR cabecera)
{
   CHAR nombreLog[MAX_PATH];

   /* Inicializar las secciones críticas que protejen los ficheros. */
   InitializeCriticalSection(&loggerCriticalSection);
   InitializeCriticalSection(&loggerExcelCriticalSection);

   /* Inicializar ficheros de log. */
   sprintf_s(nombreLog, MAX_PATH, "%s.log", cabecera);
   if ((logger = fopen(nombreLog, "w")) == NULL) {
      Error("AVISO: No se puede almacenar la informacion de log");
   }

   sprintf_s(nombreLog, MAX_PATH, "%s_excel.log", cabecera);
   if ((loggerExcel = fopen(nombreLog, "w")) == NULL) {
      Error("AVISO: No se puede almacenar la informacion de log para Excel");
   } else {
      fprintf(loggerExcel, "Timestamp;Evento;SLA;Capacidades;Utilizaciones;Estados;Decision;ID
sesion\n");
      fflush(loggerExcel);
   }
}

/* OJO: Hay que llamar a esta función desde un entorno protegido de la estructura de
   los nodos. No está protegido porque podría llamarse desde dentro de una sección
   crítica de nodos y provocar un auto-interbloqueo. */
VOID EmitirLogExcel(PSESION sesion, EVENTO evt)
{
   UINT32 i;

   if (loggerExcel == NULL)
      return;

   EnterCriticalSection(&loggerExcelCriticalSection);

   /* Tiempo y tipo de evento. */
   fprintf(loggerExcel, "%.2lf;%d;", TiempoAbsoluto(), evt);

   /* SLAs de los nodos. */
   for (i = 0; i < numeroNodosServidor; ++i) {
      fprintf(loggerExcel, "%lf;", nodosServidor[i].sla);
   }

   /* Capacidades de los nodos. */
   for (i = 0; i < numeroNodosServidor; ++i) {
      fprintf(loggerExcel, "%d;", nodosServidor[i].capacidad);
   }

   /* Utilizaciones de los nodos. */
   for (i = 0; i < numeroNodosServidor; ++i) {
      fprintf(loggerExcel, "%lf;", nodosServidor[i].utilizacion);
   }
```

```
   /* Estados de los nodos. */
   for (i = 0; i < numeroNodosServidor; ++i) {
      fprintf(loggerExcel, "%d;", nodosServidor[i].estado);
   }

   /* Información miscelánea de la sesión. */
   fprintf(loggerExcel, "%d;%lu\n", sesion->nodo, sesion->id);

   /* Volcado y cierre. */
   fflush(loggerExcel);
   LeaveCriticalSection(&loggerExcelCriticalSection);
}

VOID EmitirLog(PCSTR mensaje, ...)
{
   va_list argptr;

   if (logger == NULL)
      return;

   /* Mensaje convertido con el número variable de argumentos. */
   va_start(argptr, mensaje);

   EnterCriticalSection(&loggerCriticalSection);
   /* Escribimos todo. */
   fprintf(logger, "[%.2lf] ", TiempoAbsoluto());
   vfprintf_s(logger, mensaje, argptr);
   fprintf(logger, "\n");
   fflush(logger);
   LeaveCriticalSection(&loggerCriticalSection);

   /* Liberación de las cosas. */
   va_end(argptr);
}

INT32 HacerPeticionServidor(PSTR buffer, SIZE_T len, SOCKET sock)
{
   /* Enviar (enviamos todo de una vez). */
   if (send(sock, buffer, (INT32)len, 0) == SOCKET_ERROR) {
      return -1;
   }

   /* Recibir (recibimos todo de una vez). */
   if (recv(sock, buffer, (INT32)len, MSG_WAITALL) == SOCKET_ERROR) {
      return -1;
   }

   return 0;
}

VOID CerrarServicio(INT32 senal)
{
   /* Limpieza de  recursos y cierre del servicio. */
   if (senal == SIGINT) {
      EnterCriticalSection(&datosTotalesCriticalSection);
      printf("Fin del servicio.\n");
      printf("Energia total = %lf Wh\n", energiaTotal);
      printf("Transacciones totales = %lu\n", transaccionesTotales);
      printf("Transacciones sobre SLA = %lu\n", transaccionesMalas);
      printf("SesionesTotales = %lu\n", sesionesTotales);
      printf("Sesiones rechazadas = %lu\n", sesionesRechazadas);
      LeaveCriticalSection(&datosTotalesCriticalSection);
      Limpiar();
      exit(EXIT_SUCCESS);
   }
}

VOID InicializarSocketServicio(PSOCKET sock, UINT16 puerto)
{
   SOCKADDR_IN direccion;

   /* Configuración de la dirección. */
   direccion.sin_family = AF_INET;
```

```
   direccion.sin_addr.s_addr = htonl(INADDR_ANY);
   direccion.sin_port = htons(puerto);

   if ((*sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) == INVALID_SOCKET)
      ErrorExit("No se ha podido crear el socket servidor");

   /* Configuramos las opciones del socket. */
   if (!ConfiguraSocket(*sock))
      ErrorExit("No se ha podido configurar el socket");

   if (bind(*sock, (PSOCKADDR)&direccion, sizeof(SOCKADDR_IN)) == SOCKET_ERROR)
      ErrorExit("No se ha podido asignar el socket al puerto TCP de escucha");

   if (listen(*sock, SOMAXCONN) == SOCKET_ERROR)
      ErrorExit("No se ha podido poner el socket en modo escucha");
}

VOID Limpiar()
{
   UINT32 i;

   EmitirLog("Terminando programa");

   /* Suspendemos todos los hilos (esta funcion se ejecuta en un hilo propio en Win32). */
   for (i = 0; i < tamanoPoolHilos; ++i) {
      SuspendThread(hndsPoolHilos[i]);
   }

   /* Cerrar socket de servicio. */
   CierraSocket(socketServidor);

   /* Liberar memoria */
   if (sesiones != NULL)
      free(sesiones);

   if (hndsPoolHilos !=  NULL)
      free(hndsPoolHilos);

   /* Cerrar el log. */
   if (logger != NULL) {
      fflush(logger);
      fclose(logger);
   }

   if (loggerExcel != NULL) {
      fflush(loggerExcel);
      fclose(loggerExcel);
   }

   /* Liberar mutex del socket. */
   DeleteCriticalSection(&socketCriticalSection);

   /* Liberar mutex de los nodos. */
   DeleteCriticalSection(&nodosCriticalSection);

   /* Liberar el mutex del logger. */
   DeleteCriticalSection(&loggerCriticalSection);
   DeleteCriticalSection(&loggerExcelCriticalSection);

   /* Liberar el mutex de los datos totales. */
   DeleteCriticalSection(&datosTotalesCriticalSection);

   /* Librerar la cola de captura de información de QoS. */
   LiberarCola(&qos);

   /* Cerrar y liberar la librería. */
   LiberarLibreria();
}

BOOL ParseaDireccionMAC(PCSTR str, PUINT8 bin)
{
   if (sscanf_s(str, "%02x:%02x:%02x:%02x:%02x:%02x", &bin[0], &bin[1], &bin[2], &bin[3], &bin[4],
&bin[5]) != 6)
      return FALSE;
```

```
   return TRUE;
}
```

13.5.3 ColaCircular.c

```
#include "Repartidor.h"

/* Fichero: ColaCircular.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.1
 * Descripción: Implementa funciones de manejo de la cola circular para
 *              el almacenamiento de la información de QoS.
 */

VOID InicializarCola(PINFOQOS cola, UINT32 tamano, DOUBLE intervaloQoS)
{
   UINT32 i;

   /* Inicializaciones generales. */
   cola->indice = 0;
   cola->tamano = tamano;
   cola->sesiones = 0;
   cola->rechazos = 0;
   cola->peticiones = 0;
   cola->intervaloQoS = intervaloQoS * 1000.0;

   /* Inicialización del almacenamiento de los datos. */
   cola->info = (PENTRADAQOS)malloc(sizeof(ENTRADAQOS) * tamano);

   for (i = 0; i < tamano; ++i) {
      cola->info[i].respuesta = 0.0;
      cola->info[i].timestamp = (time_t)0;
   }

   /* Inicialización relativa al thread de QoS. */
   cola->threadQoS = NULL;
   InitializeCriticalSection(&cola->seccionCritica);
}

VOID LiberarCola(PINFOQOS cola)
{
   DeleteCriticalSection(&cola->seccionCritica);

   if (cola->info != NULL)
      free(cola->info);

   if (cola->threadQoS != NULL)
      CloseHandle(cola->threadQoS);
}
```

13.5.4 Energia.c

```
#include "Repartidor.h"

/* Fichero: Energia.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.2
 * Descripción: Implementa funciones de gestion energetica (encendido y apagado de
 *              nodos). ATENCION: Las funciones siguientes acceden a la estructura de
 *              informacion de los nodos sin proteccion para evitar interbloqueos por
 *              reentrada al ser llamadas por los gestores de sesiones. Hay que recordar
```

```
 *              llamarlas solo en entornos protegidos por mutex_nodos.
 */

extern PNODO nodosServidor; // Cada nodo tendra su propio mutex.
extern CRITICAL_SECTION nodosCriticalSection; // Bloquea la estructura al completo.
extern UINT16 numeroNodosServidor;

VOID EncenderNodo(PSESION sesion)
{
   UINT32 i;
   INT32 seleccionado;
   DOUBLE minPotencia, potencia;

   /* Hay que buscar un nodo en la lista de nodos para ponerlo en estado encendido y
      que acepte sesiones. De hecho, solo el mas eficiente de los apagados nos sirve. */
   minPotencia = DOUBLE_MAX;
   seleccionado = -1;

   /* Por estabilidad, si hay algun nodo encendiendose, no podemos encender otro (bloqueo
      de estados). */
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == encendiendose) {
         return;
      }
   }

   /* Es prioritario rescatar algun nodo en descarga, ya que es instantaneo. */
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == descargandose) {
         potencia = CalcularPotenciaEncendiendo(&nodosServidor[i]);
         // Si el nodo es igual de eficiente, escogemos el más cargado.
         if (potencia < minPotencia || (potencia == minPotencia && nodosServidor[i].utilizacion >
nodosServidor[seleccionado].utilizacion)) {
            seleccionado = i;
            minPotencia = potencia;
         }
      }
   }

   /* Si el nodo escogido esta en descarga, bastara con cambiarlo de estado. */
   if (seleccionado != -1) {
      assert(nodosServidor[seleccionado].estado == descargandose);

      /* Cambiamos el estado para que vuelva a recibir sesiones. No hay que cambiar
         conteo de sesiones. */
      nodosServidor[seleccionado].estado = encendido;
      EmitirLog("Se ha decidido sacar de descarga al nodo %d", seleccionado);
      return;
   }

   /* Si no hemos encontrado ninguno en descarga, buscamos entre los apagados. */
   if (seleccionado == -1) {
      minPotencia = DOUBLE_MAX;
      for (i = 0; i < numeroNodosServidor; ++i) {
         if (nodosServidor[i].estado == apagado) {
            potencia = CalcularPotenciaEncendiendo(&nodosServidor[i]);
            if (potencia < minPotencia) {
               seleccionado = i;
               minPotencia = potencia;
            }
         }
      }
   }

   /* Si hemos seleccionado algun nodo para encender, encendamoslo. */
   if (seleccionado != -1) {
      EmitirLogExcel(sesion, nodo_encendiendose);
      EmitirLog("Se ha decidido encender el nodo %d", seleccionado);

      /* Enviamos el mensaje de encendido del nodo seleccionado. */
      if (!DespiertaNodo(&nodosServidor[seleccionado])) {
         Error("Error al enviar un paquete magico WoL");
         return;
      }
```

```
      /* Arrancamos el temporizador de encendido, por si el nodo estuviera roto. */
      if (nodosServidor[seleccionado].temporizadorEncendido != NULL) {
         Error("Ya habia un timer de encendido activado");
         return;
      }

      if (CreateTimerQueueTimer(&nodosServidor[seleccionado].temporizadorEncendido,
                                NULL,
                                (WAITORTIMERCALLBACK)TimerEncendidoExpirado,
                                (PVOID)&nodosServidor[seleccionado],
                                (DWORD)nodosServidor[seleccionado].timeoutEncendido,
                                (DWORD)0,
                                WT_EXECUTEINTIMERTHREAD) == 0) {
         Error("No se pudo arrancar el temporizador de encendido");
         return;
      }

      /* Cambiamos el estado del nodo en la lista. */
      nodosServidor[seleccionado].estado = encendiendose;
      nodosServidor[seleccionado].sesiones = 0;
   }
}

VOID ApagarNodo(PSESION sesion)
{
   UINT32 i, encendidos;
   INT32 seleccionado;
   DOUBLE minPotencia, potencia;

   minPotencia = DOUBLE_MAX;
   seleccionado = -1;
   encendidos = 0; // Siempre hay que dejar un nodo encendido al menos.
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == encendido) {
         ++encendidos;
         potencia = CalcularPotenciaApagando(&nodosServidor[i]);
         if (potencia < minPotencia) {
            seleccionado = i;
            minPotencia = potencia;
         }
      }
   }

   /* Procedemos a descargar el nodo, antes de apagarlo, si tenemos mas de uno encendido. */
   if (encendidos > 1 && seleccionado != -1) {
      /* Este nodo ya no aceptara nuevas sesiones. Cuando las que tiene lleguen a cero, el hilo
         gestor de apagado le enviara la señal para apagarse. */
      EmitirLogExcel(sesion, nodo_descargandose);
      nodosServidor[seleccionado].estado = descargandose;
      EmitirLog("El nodo %d ha empezado a descargarse (M = %lf)", seleccionado,
nodosServidor[seleccionado].utilizacion);
   }
}

VOID ApagarNodoFisicamente(PSESION sesion)
{
   /* Ya no hay sesiones en el nodo, apagamos el nodo fisicamente. */
   EmitirLogExcel(sesion, nodo_apagandose);
   EmitirLog("El nodo %d va a ser apagado", nodosServidor[sesion->nodo].identificador);

   /* Enviamos el mensaje de apagado. */
   if (!EnviaMensajeApagado(&nodosServidor[sesion->nodo])) {
      Error("No se ha podido enviar la senal de apagado");
      return;
   }

   /* Arrancamos el temporizador de apagado. */
   if (nodosServidor[sesion->nodo].temporizadorApagado != NULL) {
      Error("Ya habia un timer de apagado activado");
      return;
   }

   if (CreateTimerQueueTimer(&nodosServidor[sesion->nodo].temporizadorApagado,
```

```
                              NULL,
                              (WAITORTIMERCALLBACK)TimerApagadoExpirado,
                              (PVOID)&nodosServidor[sesion->nodo],
                              (DWORD)nodosServidor[sesion->nodo].timeoutApagado,
                              (DWORD)0,
                              WT_EXECUTEINTIMERTHREAD) == 0) {
      Error("No se pudo arrancar el temporizador de apagado");
      return;
   }

   /* Cambiamos el estado del nodo en la lista. */
   nodosServidor[sesion->nodo].estado = apagandose;
   nodosServidor[sesion->nodo].sesiones = 0;
}

VOID CALLBACK TimerEncendidoExpirado(LPVOID p, BOOLEAN fired)
{
   if (fired) {
      /* Si espira un temporizador de encendido marcamos el nodo como roto y eliminamos el
temporizador.
         Vamos a acceder a la seccion critica porque esto se ejecutara en un hilo del pool de kernel.
*/
      EnterCriticalSection(&nodosCriticalSection);

      #if defined(_NO_APAGADO_FISICO_)
      ((PNODO)p)->estado = encendido;
      #elif
      #pragma message("Se ha activado el apagado fisico en el Repartidor")
      ((PNODO)p)->estado = broken;
      #endif
      ((PNODO)p)->sesiones = 0;
      ((PNODO)p)->temporizadorEncendido = NULL;

      LeaveCriticalSection(&nodosCriticalSection);
   }
}

VOID CALLBACK TimerApagadoExpirado(LPVOID p, BOOLEAN fired)
{
   if (fired) {
      /* Si espira un temporizador de apagado marcamos el nodo como roto y eliminamos el temporizador.
         Vamos a acceder a la seccion critica porque esto se ejecutara en un hilo del pool de kernel.
*/
      EnterCriticalSection(&nodosCriticalSection);

      ((PNODO)p)->estado = apagado;
      ((PNODO)p)->sesiones = 0;
      ((PNODO)p)->temporizadorEncendido = NULL;

      LeaveCriticalSection(&nodosCriticalSection);
   }
}
```

13.5.5 Math.c

```
#include "Repartidor.h"

/* Fichero: Math.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.3
 * Descripción: Implementa funciones de apoyo matematico para la generacion de
 *              numeros aleatorios, evaluacion de polinomios y demas.
 */

extern PNODO nodosServidor; // Cada nodo tendra su propio mutex.
extern CRITICAL_SECTION nodosCriticalSection; // Bloquea la estructura al completo.
extern UINT16 numeroNodosServidor;
```

```
VOID ActualizarUtilizacion(PNODO nodo)
{
   ActualizarCapacidad(nodo);

   nodo->utilizacion = ((DOUBLE)nodo->sesiones) / ((DOUBLE)nodo->capacidad);
}

VOID ActualizarCapacidad(PNODO nodo)
{
   nodo->capacidad = (UINT32)ceil(BuscarRaiz(nodo->curvaRespuesta, nodo->sla) * nodo->modificador);
}

/* Indica la eficiencia global del clúster que se tendría si se apagara el nodo indicado. La
   eficiencia se indica en usuarios por vatio, de forma que hay que maximizarla. */
DOUBLE CalcularPotenciaApagando(PNODO nodo)
{
   UINT32 i;
   DOUBLE potencia;
   UINT64 sesionesTotal;
   UINT64 capacidadTotal;
   DOUBLE utilizacionEsperada;
   DOUBLE sesiones;

   /* Calculamos el numero total de sesiones que estamos soportando y, ademas, la capacidad
      total que tiene el cluster en este momento. */
   capacidadTotal = sesionesTotal = 0;
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == encendido) {
         sesionesTotal += nodosServidor[i].sesiones;
         capacidadTotal += nodosServidor[i].capacidad;
      }
   }

   /* Ahora calculamos la utilizacion esperada que tendriamos si apagamos el nodo que nos indican
      de forma que se llegara a ella en un equilibrado instantaneo de la carga *actual*. */
   utilizacionEsperada = ((DOUBLE)sesionesTotal) / ((DOUBLE)(capacidadTotal - nodo->capacidad));

   /* Con la utilizacion esperada, miramos el consumo de cada nodo, calculando cuantas sesiones
      suponen en cada uno ese nivel de utilizacion. No tenemos en cuenta el nodo a apagar. */
   potencia = 0.0;
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == encendido && nodosServidor[i].identificador != nodo-
>identificador) {
         /* Miramos el nivel de utilizacion esperado en sesiones. */
         sesiones = utilizacionEsperada * nodosServidor[i].capacidad;

         /* Calculamos el consumo esperado del nodo a ese nivel. */
         potencia += BuscarImagen(nodosServidor[i].curvaPotencia, sesiones);
      }
   }

   /* Ya tenemos la potencia calculada para la nueva configuracion tentativa. */
   return potencia;
}

DOUBLE CalcularPotenciaEncendiendo(PNODO nodo)
{
   UINT32 i;
   DOUBLE potencia;
   UINT64 sesionesTotal;
   UINT64 capacidadTotal;
   DOUBLE utilizacionEsperada;
   DOUBLE sesiones;

   /* Calculamos el numero total de sesiones que estamos soportando y, ademas, la capacidad
      total que tiene el cluster en este momento. */
   capacidadTotal = sesionesTotal = 0;
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == encendido) {
         sesionesTotal += nodosServidor[i].sesiones;
         capacidadTotal += nodosServidor[i].capacidad;
      }
   }
```

```
   /* Ahora calculamos la utilizacion esperada que tendriamos si encendemos el nodo que nos indican
      de forma que se llegara a ella en un equilibrado instantaneo de la carga *actual*. */
   utilizacionEsperada = ((DOUBLE)sesionesTotal) / ((DOUBLE)(capacidadTotal + nodo->capacidad));

   /* Con la utilizacion esperada, miramos el consumo de cada nodo, calculando cuantas sesiones
      suponen en cada uno ese nivel de utilizacion. */
   potencia = 0.0;
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == encendido) {
         /* Miramos el nivel de utilizacion esperado en sesiones. */
         sesiones = utilizacionEsperada * nodosServidor[i].capacidad;

         /* Calculamos el consumo esperado del nodo a ese nivel. */
         potencia += BuscarImagen(nodosServidor[i].curvaPotencia, sesiones);
      }
   }

   /* Tenemos que sumar el aporte del nodo a encender. */
   sesiones = utilizacionEsperada * nodo->capacidad;
   potencia += BuscarImagen(nodo->curvaPotencia, sesiones);

   /* Ya tenemos la potencia calculada para la nueva configuracion tentativa. */
   return potencia;
}
```

13.5.6 Parser.c

(Esta pieza de código presenta un dependencia con la librería iniParser en su versión 3.0).

```
#include "Repartidor.h"
#include "iniparser.h"

/* Fichero: Parser.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.1
 * Descripción: Implementa el parser del fichero de configuración en formato INI
 *              usando las funciones proporcionadas por Win32.
 */

#define TMP_BUFFER 32

extern PNODO nodosServidor; // Cada nodo tendra su propio mutex.
extern CRITICAL_SECTION nodosCriticalSection; // Bloquea la estructura al completo.
extern UINT16 numeroNodosServidor;

/* Umbrales de funcionamiento del distribuidor. */
extern DOUBLE Uon; // Umbral de encendido.
extern DOUBLE Uoff; // Umbral de apagado.

BOOL ParseaFicheroConfiguracion(PCSTR fichero)
{
   BOOL ret;
   UINT32 i;
   CHAR tmp[TMP_BUFFER];
   PCHAR seccion;
   dictionary * diccionario;

   ret = TRUE;

   /* La librería iniParser no es thread-safe, por lo que debemos usarla dentro
      de una seccion crítica, aprovechamos la propia de los nodos. */
   EnterCriticalSection(&nodosCriticalSection);

   /* Cargamos el fichero de configuración. */
   if ((diccionario = iniparser_load(fichero)) == NULL) {
      Error("No se ha podido abrir el fichero de configuracion");
```

```
        LeaveCriticalSection(&nodosCriticalSection);
        return FALSE;
    }

    /* Cargamos las configuraciones globales. */
    Uon = iniparser_getdouble(diccionario, "globals:uon", Uon);
    Uoff = iniparser_getdouble(diccionario, "globals:uoff", Uoff);

    /* Leemos el número de secciones que tenemos para procesar (sera el numero de
       nodos a utilizar, descontando la globals). */
    if ((numeroNodosServidor = (UINT16)iniparser_getnsec(diccionario) - 1) == -1) {
        Error("No se ha podido leer el fichero de configuracion");
        numeroNodosServidor = 0;
        iniparser_freedict(diccionario);
        LeaveCriticalSection(&nodosCriticalSection);
        return FALSE;
    }

    /* Reservamos memoria para los nodos. */
    if ((nodosServidor = (PNODO)malloc(sizeof(NODO) * numeroNodosServidor)) == NULL) {
        Error("No se ha podido reservar memoria para los nodos");
        iniparser_freedict(diccionario);
        LeaveCriticalSection(&nodosCriticalSection);
        return FALSE;
    }

    /* Para cada sección, generamos y rellenamos un objeto de NODO. */
    for (i = 0; i < numeroNodosServidor; ++i) {
        /* Obtenemos el nombre de la seccion y, por tanto, el ID del nodo. */
        if ((seccion = iniparser_getsecname(diccionario, i + 1)) == NULL) {
            Error("No se ha podido interpretar el fichero de configuracion");
            ret = FALSE;
            continue;
        }
        nodosServidor[i].identificador = (UINT16)strtol(seccion, NULL, 10);

        /* Cargamos cada par valor clave en la estructura del nodo. */
        sprintf_s(tmp, TMP_BUFFER, "%d:hostname", nodosServidor[i].identificador);
        strcpy_s(nodosServidor[i].hostname, NI_MAXHOST, iniparser_getstring(diccionario, tmp,
"localhost"));

        sprintf_s(tmp, TMP_BUFFER, "%d:puerto", nodosServidor[i].identificador);
        nodosServidor[i].puerto = (UINT16)iniparser_getint(diccionario, tmp, 0);

        sprintf_s(tmp, TMP_BUFFER, "%d:estado", nodosServidor[i].identificador);
        nodosServidor[i].estado = (ESTADOENERGETICO)iniparser_getint(diccionario, tmp, -1);

        if ((nodosServidor[i].curvaRespuesta = CrearCurvaPolinomica(3)) == NULL) {
            Error("No se ha podido resevar memoria para la curva de respuesta");
            ret = FALSE;
            continue;
        }

        nodosServidor[i].curvaRespuesta->grado = 3;

        sprintf_s(tmp, TMP_BUFFER, "%d:respuesta0", nodosServidor[i].identificador);
        nodosServidor[i].curvaRespuesta->coeficientes[0] = iniparser_getdouble(diccionario, tmp, 0.0);

        sprintf_s(tmp, TMP_BUFFER, "%d:respuesta1", nodosServidor[i].identificador);
        nodosServidor[i].curvaRespuesta->coeficientes[1] = iniparser_getdouble(diccionario, tmp, 0.0);

        sprintf_s(tmp, TMP_BUFFER, "%d:respuesta2", nodosServidor[i].identificador);
        nodosServidor[i].curvaRespuesta->coeficientes[2] = iniparser_getdouble(diccionario, tmp, 0.0);

        sprintf_s(tmp, TMP_BUFFER, "%d:respuesta3", nodosServidor[i].identificador);
        nodosServidor[i].curvaRespuesta->coeficientes[3] = iniparser_getdouble(diccionario, tmp, 0.0);

        if ((nodosServidor[i].curvaPotencia = CrearCurvaPolinomica(3)) == NULL) {
            Error("No se ha podido resevar memoria para la curva de potencia");
            ret = FALSE;
            continue;
        }

        nodosServidor[i].curvaPotencia->grado = 3;
```

```
      sprintf_s(tmp, TMP_BUFFER, "%d:potencia0", nodosServidor[i].identificador);
      nodosServidor[i].curvaPotencia->coeficientes[0] = iniparser_getdouble(diccionario, tmp, 0.0);

      sprintf_s(tmp, TMP_BUFFER, "%d:potencia1", nodosServidor[i].identificador);
      nodosServidor[i].curvaPotencia->coeficientes[1] = iniparser_getdouble(diccionario, tmp, 0.0);

      sprintf_s(tmp, TMP_BUFFER, "%d:potencia2", nodosServidor[i].identificador);
      nodosServidor[i].curvaPotencia->coeficientes[2] = iniparser_getdouble(diccionario, tmp, 0.0);

      sprintf_s(tmp, TMP_BUFFER, "%d:potencia3", nodosServidor[i].identificador);
      nodosServidor[i].curvaPotencia->coeficientes[3] = iniparser_getdouble(diccionario, tmp, 0.0);

      sprintf_s(tmp, TMP_BUFFER, "%d:sla", nodosServidor[i].identificador);
      nodosServidor[i].sla = iniparser_getdouble(diccionario, tmp, 0.0);

      sprintf_s(tmp, TMP_BUFFER, "%d:modificador", nodosServidor[i].identificador);
      nodosServidor[i].modificador = iniparser_getdouble(diccionario, tmp, 1.0);

      sprintf_s(tmp, TMP_BUFFER, "%d:timeoutenc", nodosServidor[i].identificador);
      nodosServidor[i].timeoutEncendido = iniparser_getdouble(diccionario, tmp, 0.0) * 1000.0;

      sprintf_s(tmp, TMP_BUFFER, "%d:timeoutapa", nodosServidor[i].identificador);
      nodosServidor[i].timeoutApagado = iniparser_getdouble(diccionario, tmp, 0.0) * 1000.0;

      sprintf_s(tmp, TMP_BUFFER, "%d:mac", nodosServidor[i].identificador);

      if (!ParseaDireccionMAC(iniparser_getstring(diccionario, tmp, "FF:FF:FF:FF:FF:FF"),
nodosServidor[i].mac)) {
         Error("No se ha podido interpretar la direccion MAC");
         ret = FALSE;
         continue;
      }

      /* Resolvemos el nombre del servidor. */
      if (!ResuelveNombre(nodosServidor[i].hostname, &nodosServidor[i].direccion)) {
         Error("No se ha podido resolver el nombre %s", nodosServidor[i].hostname);
         ret = FALSE;
      }

      /* Inicializaciones varias. */
      nodosServidor[i].sesiones = 0;
      ActualizarUtilizacion(&nodosServidor[i]);
      nodosServidor[i].temporizadorApagado = NULL;
      nodosServidor[i].temporizadorEncendido = NULL;
      /* Inicializamos el consumo basico. */
      nodosServidor[i].consumo = BuscarImagen(nodosServidor[i].curvaPotencia, 0);
      nodosServidor[i].energia = 0.0;
      nodosServidor[i].sesiones_servidas = 0;
   }

   /* Liberamos el diccionario. */
   iniparser_freedict(diccionario);

   /* Fin de la sección crítica. */
   LeaveCriticalSection(&nodosCriticalSection);

   return ret;
}
```

13.5.7 QoS.c

```
#include "Repartidor.h"

/* Fichero: QoS.c
 * Autor: Ramón Medrano Llamas
 * Versión: 2.0
 * Descripción: Funciones de manejo y monitorizacion de la calidad de serivicio.
```

```
 */

extern PNODO nodosServidor; // Cada nodo tendra su propio mutex.
extern CRITICAL_SECTION nodosCriticalSection; // Bloquea la estructura al completo.
extern UINT16 numeroNodosServidor;
extern INFOQOS qos; // Toda la informacion de QoS.
extern DOUBLE energiaTotal; // Energia consumida en total.
extern CRITICAL_SECTION datosTotalesCriticalSection;

VOID RegistrarPeticion(DOUBLE respuesta)
{
   /* Permiso para usar la estructura, para compartirla por los hilos de sesion. */
   EnterCriticalSection(&qos.seccionCritica);

   /* Insertamos e incrementamos. No nos importa comprobar si sobreescribimos, si lo hacemos, la
      información tendrá una marca de tiempo más nueva que la que había. */
   qos.info[qos.indice].respuesta = respuesta;
   time(&qos.info[qos.indice].timestamp);
   ++qos.peticiones;

   qos.indice = Siguiente(qos.indice, qos.tamano);

   LeaveCriticalSection(&qos.seccionCritica);
}

#pragma warning ( disable : 4100 )
DWORD WINAPI HiloRegistroQoS(LPVOID p)
{
   HANDLE temporizador;
   PFILE fichero;
   LARGE_INTEGER liIntervalo;
   PDOUBLE local;
   UINT32 i;
   INT64 j;
   DOUBLE percentil;
   DOUBLE pctjeRechazo;
   DOUBLE productividad;
   time_t tiempoLimite;
   UINT64 sesiones;
   CHAR nombre[MAX_PATH];
   DOUBLE consumo;
   UINT64 capacidad;

   /* Reservar memoria local. */
   local = (PDOUBLE)malloc(sizeof(DOUBLE) * (SIZE_T)qos.tamano);

   /* Cambiar intervalo. */
   liIntervalo.QuadPart = ((INT64)qos.intervaloQoS) * -10000; // De grupos de nanosegundos a
milisegundos.

   /* Este hilo monitoriza la calidad de servicio, guardando cada cierto tiempo el percentil-90 de
   los tiempos de respuesta de las peticiones de los ultimos segundos. */

   /* Creamos el temporizador. */
   if ((temporizador = CreateWaitableTimer(NULL, TRUE, "temporizador_repartidor")) == NULL) {
      Error("No se pudo crear el temporizador.");
      free(local);
      ExitThread(1);
   }

   /* Creamos el fichero de salida. */
   sprintf_s(nombre, MAX_PATH, "%s.qos", p);
   if ((fichero = fopen(nombre, "w")) == NULL) {
      Error("No se pudo abrir el fichero de registro de QoS.");
      CloseHandle(temporizador);
      free(local);
      ExitThread(2);
   }

   fprintf(fichero, "Hora;Percentil-90;Productividad;Numero de
marcas;Rechazos;Sesiones;Capacidad;Energias;Eficiencias;Potencias;\r\n");

   /* Bucle infinito del hilo. */
   #pragma warning ( disable : 4127 )
```

```
   while (TRUE) {
      #pragma warning ( default : 4127 )
      /* Esperamos por el temporizador. */
      if (SetWaitableTimer(temporizador, &liIntervalo, 0, NULL, NULL, 0) == 0)
         Error("No se pudo configurar el temporizador");
      WaitForSingleObject(temporizador, INFINITE);

      /* Calculamos cuantas entradas necesitamos sacar, las sacamos y devolvemos el control de
         la estructura. */
      i = 0;
      time(&tiempoLimite);
      tiempoLimite -= (time_t)(qos.intervaloQoS / 1000);

      /* Entramos en la sección crítica. */
      EnterCriticalSection(&qos.seccionCritica); // Vamos a leer el valor de entrada.
      j = Anterior(qos.indice, qos.tamano);

      while (j != qos.indice) {
         /* Si esta en intervalo, nos lo quedamos. Y lo agregamos a la copia local. */
         if (qos.info[j].timestamp > tiempoLimite) {
            local[i++] = qos.info[j].respuesta;
            j = Anterior(j, qos.tamano);
         } else { // Si es demasiado antiguo, terminamos el bucle (los datos estan ordenados
temporalmente).
            break;
         }
      }

      /* Calculamos el ratio de rechazo. */
      if (qos.sesiones != 0) {
         pctjeRechazo = ((DOUBLE)qos.rechazos) / ((DOUBLE)qos.sesiones);
      } else {
         EmitirLog("Aviso: no hay registro de sesiones en este periodo");
         pctjeRechazo = 0.0;
      }

      /* Calculamos la productividad del periodo. */
      productividad = ((DOUBLE)qos.peticiones) / (qos.intervaloQoS / 1000.0);

      /* Ponemos a cero para contar solo entre periodos y tener una métrica pseudo-instantánea. */
      qos.rechazos = qos.sesiones = qos.peticiones = 0;

      /* Calculamos y actualizamos el consumo de cada nodo en este periodo. */
      capacidad = sesiones = 0; // Sesiones abiertas en este momento (qos.sesiones es la cuenta de
sesiones atendidas).
      for (j = 0; j < numeroNodosServidor; ++j) {
         if (nodosServidor[j].estado == apagado || nodosServidor[j].estado == broken) {
            consumo = 0.0;
         } else {
            consumo = BuscarImagen(nodosServidor[j].curvaPotencia, nodosServidor[j].sesiones);
            if (nodosServidor[j].estado == encendido)
               capacidad += nodosServidor[j].capacidad;
         }

         nodosServidor[j].energia = (((consumo + nodosServidor[j].consumo) / 2) * qos.intervaloQoS) /
3600.0 / 1000.0;
         nodosServidor[j].consumo = consumo;

         if (nodosServidor[j].sesiones_servidas > 0)
            nodosServidor[j].eficiencia = (nodosServidor[j].energia * 3600.0) /
nodosServidor[j].sesiones_servidas;
         else
            nodosServidor[j].eficiencia = 0.0;

         nodosServidor[j].sesiones_servidas = 0; // Historico de sesiones procesadas.
         sesiones += nodosServidor[j].sesiones; // Carga actual.

         EnterCriticalSection(&datosTotalesCriticalSection);
         energiaTotal += nodosServidor[j].energia;
         LeaveCriticalSection(&datosTotalesCriticalSection);
      }

      LeaveCriticalSection(&qos.seccionCritica);
```

```
      /* Hay situaciones de baja carga en las que no hay datos disponibles. */
      if (i == 0)
         continue;

      /* Hallamos el percentil de los datos calculados. */
      percentil = CalcularPercentil(local, i, 0.9);

      /* Grabamos a fichero junto con el timestamp. */
      fprintf(fichero, "%.2lf;%lf;%lf;%d;%lf;%d;%d;", TiempoAbsoluto(), percentil, productividad, i,
pctjeRechazo, sesiones, capacidad);
      fflush(fichero);
      for (j = 0; j < numeroNodosServidor; ++j)
         fprintf(fichero, "%lf;", nodosServidor[j].energia);
      fflush(fichero);
      for (j = 0; j < numeroNodosServidor; ++j)
         fprintf(fichero, "%lf;", nodosServidor[j].eficiencia);
      fflush(fichero);
      for (j = 0; j < numeroNodosServidor; ++j)
         fprintf(fichero, "%lf;", nodosServidor[j].consumo);
      fprintf(fichero, "\r\n");
      fflush(fichero);
      EmitirLog("Informacion de QoS almacenada (%lf) con %d marcas", percentil, i);
   }

   /* No habria que llegar aqui, pero por si acaso y para evitar warnings. */
   fclose(fichero);
   CloseHandle(temporizador);
   free(local);

   return 0;
}
#pragma warning ( default : 4100 )
```

13.5.8 Repartidor.c

```
#include "Repartidor.h"

/* Fichero: Repartidor.c
 * Autor: Ramón Medrano Llamas
 * Versión: 2.0
 * Descripción: Implementa el módulo principal del repartidor de carga para
 *              el simulador del servicio transaccional para pruebas. Este módulo
 *              reparte peticiones a deversos servidores SSIF previamente
 *              configurados.
 */

/* Variables globales. Todas se definen aquí y en los otros módulos son extern. */
PNODO nodosServidor; // Cada NODO tendra su propio mutex.
CRITICAL_SECTION nodosCriticalSection; // Bloquea la estructura al completo.
UINT16 numeroNodosServidor;
UINT16 tamanoPoolHilos;
SIZE_T tamanoPeticion;
PHANDLE hndsPoolHilos; // Lista de los handlers de los hilos.
HANDLE hndHiloNotificaciones;
PFILE logger;
PFILE loggerExcel;
CRITICAL_SECTION loggerCriticalSection;
CRITICAL_SECTION loggerExcelCriticalSection;
UINT16 puertoServidor;
SOCKET socketServidor;
CRITICAL_SECTION socketCriticalSection;
PSESION sesiones; // Tiene que ser dinamica entre ejecuciones.
INFOQOS qos; // Toda la informacion de QoS.

/* Datos para el conteo de sesiones totales. */
DOUBLE energiaTotal; // Energia consumida en total.
UINT64 transaccionesTotales; // Transacciones procesadas en total.
UINT64 sesionesTotales; // Sesiones totales.
```

```
UINT64 transaccionesMalas; // Transacciones por encima del SLA.
UINT64 sesionesRechazadas; // Sesiones rechazadas.
CRITICAL_SECTION datosTotalesCriticalSection;

/* Umbrales de funcionamiento del distribuidor. */
DOUBLE Uon = 0.9; // Umbral de encendido por defecto.
DOUBLE Uoff = 0.8; // Umbral de apagado por defecto.

INT32 ControlAdmision(PSESION sesion)
{
   UINT32 i;
   INT32 seleccionado;
   DOUBLE minCap;
   BOOL encender;

   /* Escogemos en qué NODO atender la petición en función de la política. Si la
      sesión se rechazara, se retornará -1 al terminar la función. Es importante
      hacer notar cómo ahora se bloquea todo la estructura de la información de los
      nodos de forma completa. Este decisor se invoca desde varios hilos simultáneamente
      y, de cara a las actualizaciones de utilización y cálculo de límites se necesita
      tener toda la información de forma atómica.
      Hay tres pasos que realizar de cara a la admisión de un nuevo cliente:
         1. Actualizar la informacion de la utilización instantánea de los nodos.
         2. Seleccionar el NODO que tenga menor utilización utilizada como NODO de sesión.
            En este paso, si todos los nodos estuvieran con una utilización por encima de 1.0
            en función del SLA, se rechará la sesión.
         3. Si todos los nodos tienen una utilización por encima del umbral, se encenderá un
            NODO del pool de nodos apagados. Si hay algún NODO en estado de descarga, se
            sacara de ese estado a encedido de forma instantánea. */

   /* Bloqueamos la estructura completamente. */
   EnterCriticalSection(&nodosCriticalSection);

   /* PASO 1. Obligatorio. Recalcular y actualizar las utilizaciones relativas al SLA de los nodos. */
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == encendido)
         ActualizarUtilizacion(&(nodosServidor[i]));
   }

   /* PASOS 2 y 3. En vista de las utilizaciones, seleccionar un NODO y determinar si hay que
      encender uno nuevo. A continuacion, encenderlo si es necesario. */
   seleccionado = -1;
   minCap = 1.0; // La politica no garantiza que no se pase de 1.0. Sin embargo, si la utilización de
un
   // NODO fuera mayor a 1.0 nunca se escogeria.
   for (i = 0; i < numeroNodosServidor; ++i) {
      if (nodosServidor[i].estado == encendido && nodosServidor[i].utilizacion < minCap) { // Esto es,
el minimo y menor que 1.0.
         seleccionado = i;
         minCap = nodosServidor[i].utilizacion;
      }
      // Si todos los nodos estuvieran con una utilización igual o superior a 1.0, seleccionado seria -
1.
   }

   /* Indicamos la seleccion del NODO en el logger para Excel. */
   sesion->nodo = seleccionado;
   EmitirLogExcel(sesion, nueva_sesion);

   /* PASO 4. Decidir si encendemos uno nuevo NODO. Habra que encenderlo si las utilizaciones superan
los
      umbrales tras actualizar las utilizaciones. */
   if (seleccionado != -1) {
      /* Actualizar las utilizaciones, solo del NODO seleccionado. */
      ++(nodosServidor[seleccionado].sesiones); // Toda la estructura esta bloqueada.
      ++nodosServidor[seleccionado].sesiones_servidas;
      ActualizarUtilizacion(&(nodosServidor[seleccionado]));
      /* Decidir si encender el nuevo NODO. Todos los nodos han de superar el umbral. */
      encender = TRUE;
      for (i = 0; i < numeroNodosServidor; ++i) {
         /* Es mas facil buscar por un NODO que no supere el umbral, y en ese caso, no encender. */
         if (nodosServidor[i].estado == encendido && nodosServidor[i].utilizacion < Uon) {
            encender = FALSE;
            break;
```

```
            }
         }
         /* Si no hay ninguno por debajo, encendemos. */
         if (encender) {
            EncenderNodo(sesion);
         }
      }

      /* Desbloqueamos la estructura de los nodos. */
      LeaveCriticalSection(&nodosCriticalSection);

      /* Retornamos el descriptor del NODO o -1 si hay error o rechazo de la sesión. */
      return seleccionado;
   }

   VOID CerrarSesion(PSESION sesion)
   {
      UINT32 i;
      BOOL apagar;
      DOUBLE Udin;
      UINT32 encendidas;

      /* Al terminar una sesion, hay que reconfigurar posiblemente el cluster apagando un NODO.
         Es necesario actualizar las utilizaciones del NODO que se terminado la sesion y comprobar
         el estado de los mismos con respecto a los umbrales. */

      /* Bloqueamos la estructura completamente. */
      EnterCriticalSection(&nodosCriticalSection);

      /* Emitir log desde el entorno protegido. */
      EmitirLogExcel(sesion, fin_sesion);

      /* Actualizar las utilizaciones, solo del NODO seleccionado. */
      --nodosServidor[sesion->nodo].sesiones; // Toda la estructura esta bloqueada.
      ActualizarUtilizacion(&(nodosServidor[sesion->nodo]));

      /* Aprovechamos para apagar el NODO en caso de que se este descargando y esta haya sido la
         ultima sesion que estaba soportando. */
      if ((nodosServidor[sesion->nodo].estado == descargandose) && (nodosServidor[sesion->nodo].sesiones
   == 0)) {
         EmitirLogExcel(sesion, nodo_descargado);
         EmitirLog("El nodo %d ha sido vaciado y se va a apagar fisicamente", sesion->nodo);
         ApagarNodoFisicamente(sesion);
      }

      /* Calculamos el umbral de apagado dinamico en funcion de las maquinas que haya encendidas. */
      encendidas = 0;
      for (i = 0; i < numeroNodosServidor; ++i) {
         if (nodosServidor[i].estado == encendido) {
            ++encendidas;
         }
      }

      /* OJO: este el umbral dinámico de post-apagado. */
      Udin = (((DOUBLE)encendidas - 1.0) / (DOUBLE)encendidas) * Uoff;

      /* Decidir si apagar uno de los nodos. Todos los nodos han de ser inferiores el umbral. */
      apagar = TRUE;
      for (i = 0; i < numeroNodosServidor; ++i) {
         if (nodosServidor[i].estado == encendido && nodosServidor[i].utilizacion > Udin) {
            apagar = FALSE;
            break;
         }
      }

      if (apagar) {
         ApagarNodo(sesion);
      }

      /* Desbloqueamos la estructura de los nodos. */
      LeaveCriticalSection(&nodosCriticalSection);
   }

   VOID ServirSesion(PSESION sesion)
```

```
{
   PSTR buffer;
   UINT32 ret;
   SOCKADDR_IN direccion;
   SOCKET sock;
   DOUBLE inicio, fin;

   buffer = (PSTR)malloc(sizeof(CHAR) * tamanoPeticion);

   /* Una sesión será un conjunto de peticiones del cliente. */
   /* Crear socket para conectar al SSIF. */
   if ((sock = socket(PF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
      Error("Hubo un error al crear un socket para la sesion %d (%d)\n", sesion->id,
WSAGetLastError());
      free(buffer);
      return;
   }

   /* Inicializar estrucutra de direccion. */
   direccion.sin_family = AF_INET;
   memcpy(&direccion.sin_addr, &nodosServidor[sesion->nodo].direccion, sizeof(IN_ADDR));
   direccion.sin_port = htons(nodosServidor[sesion->nodo].puerto);

   /* Configuración del tiempo de TIME_WAIT mediante el uso de sockets bloqueantes. */
   if (!ConfiguraSocket(sock)) {
      Error("AVISO: No se ha podido configurar el socket de sesion.");
   }

   /* Conectarse al servidor SSIF. */
   if (connect(sock, (PSOCKADDR)&direccion, sizeof(SOCKADDR_IN)) == SOCKET_ERROR) {
      Error("Hubo un error al conectar al SSIF en la sesion %d (%d)\n", sesion->id, WSAGetLastError());
      free(buffer);
      return;
   }

   /* La llamada a recv ira retornando el numero de bytes enviados. Si devuelve 0 es que el cliente ha
      cerrado la conexion y, por tanto la sesion ha concluido. Además, comprobamos la ausencia de
errores.
      La lista de errores puede encontrarse en: http://msdn.microsoft.com/en-us/library/ms740668.aspx
*/
   while ((ret = recv(sesion->socket, buffer, (INT32)tamanoPeticion, MSG_WAITALL)) > 0 && ret !=
SOCKET_ERROR) {
      /* Obtener informacion del tiempo de sesion. */
      inicio = TiempoAbsoluto();
      /* Una vez escogido el NODO, hacemos la peticion. Si todo es correcto, se devuelve 0. */
      if (HacerPeticionServidor(buffer, tamanoPeticion, sock) != 0) {
         Error("Hubo un error al hacer una peticion al nodo %d (%d)\n", sesion->nodo,
WSAGetLastError());
         break;
      }
      /* Respondemos al cliente. */
      if (send(sesion->socket, buffer, (INT32)tamanoPeticion, 0) == SOCKET_ERROR)  {
         Error("Hubo un error al hacer una peticion al nodo %d (%d)\n", sesion->nodo,
WSAGetLastError());
         break;
      }
      /* Calcular y registrar el tiempo de sesion para el traking de QoS. */
      fin = TiempoAbsoluto();
      RegistrarPeticion(fin - inicio);
      /* Contamos los datos totales (los datos de QoS son por periodo de captura). */
      EnterCriticalSection(&datosTotalesCriticalSection);
      ++transaccionesTotales;
      if ((fin - inicio) > (nodosServidor[sesion->nodo].sla))
         ++transaccionesMalas;
      LeaveCriticalSection(&datosTotalesCriticalSection);
   }

   /* Terminamos con esta sesion. */
   CierraSocket(sock);
   CierraSocket(sesion->socket);
   free(buffer);
}

DWORD WINAPI HiloServicioSesion(LPVOID p)
```

```
{
   SOCKET sock;
   SIZE_T tamDireccion;
   PSESION sesion;

   sesion = (PSESION)p;

   /* Bucle infinito de servicio de sesiones de clientes. */
   #pragma warning ( disable : 4127 )
   while (TRUE) {
      /* Espera a que llegue una solicitud de inicio de la sesion. */
      tamDireccion = sizeof(SOCKADDR_IN);
      EnterCriticalSection(&socketCriticalSection);
      sock = accept(socketServidor, (SOCKADDR *)&sesion->cliente, (PINT32)&tamDireccion);
      LeaveCriticalSection(&socketCriticalSection);

      /* Contamos los datos totales (los datos de QoS son por periodo de captura). */
      EnterCriticalSection(&datosTotalesCriticalSection);
      ++sesionesTotales;
      LeaveCriticalSection(&datosTotalesCriticalSection);

      /* Guardamos el socket en una memoria especial para el hilo y
         escogemos el NODO en base a las politicas de reparticion. */
      sesion->id = SiguienteID();
      sesion->socket = sock;
      sesion->nodo = ControlAdmision(sesion); // Aqui esta toda la politica de admision.

      /* El conteo de rechazos es una metrica para saber si hay control de admisión. */
      EnterCriticalSection(&qos.seccionCritica);
      ++qos.sesiones;
      if (sesion->nodo == -1) {
         ++qos.rechazos;
         /* Contamos los datos totales (los datos de QoS son por periodo de captura). */
         EnterCriticalSection(&datosTotalesCriticalSection);
         ++sesionesRechazadas;
         LeaveCriticalSection(&datosTotalesCriticalSection);
      }
      LeaveCriticalSection(&qos.seccionCritica);

      /* Puede que se rechaze la sesión al no haber nodos libres (el cluster esta lleno). En ese caso,
         rechazamos la sesion. */
      if (sesion->nodo == -1) {
         CierraSocket(sock);
         continue;
      }

      /* Servimos la sesion recien abierta. */
      ServirSesion(sesion);

      /* Tras terminar la sesion (bien correctamente, erroneamente, o por cancelacion) hay que lanzar
         siempre el control de salida para reconfigurar el cluster y actualizar las utilizaciones. */
      CerrarSesion(sesion);
   }

   return 0;
}

INT32 main(INT32 argc, PCHAR argv[])
{
   UINT32 i;

   /* Instalación de la señal de cierre. */
   #pragma warning ( disable : 4306 )
   if (signal(SIGINT, CerrarServicio) == SIG_ERR)
      ErrorExit("No se pudo instalar el manejador de señal.");
   #pragma warning ( default : 4306 )

   /* Inicializamos la librería común en un contexto mono-hilo. */
   if (!InicializarLibreria(_time32(NULL))) {
      fprintf(stderr, "Error fatal.\n");
      exit(EXIT_FAILURE);
   }

   /* Validación de parámetros. */
```

```
   if (argc != 8) {
      printf("\nUso: %s PUERTO POOL_HILOS FICHERO_CONF INTERVALO_QOS TAM_QOS TAM_PETICION LOG\n",
argv[0]);
      printf("\n    PUERTO              - puerto de servicio\n");
      printf("    POOL_HILOS          - tamano del pool de hilos de servicio\n");
      printf("    FICHERO_CONF        - fichero de configuracion\n");
      printf("    INTERVALO_QOS       - intervalo de captura de informacion QoS (segundos)\n");
      printf("    TAM_QOS             - limite al espacio de captura de informacion QoS\n");
      printf("    TAM_PETICION        - tamano del mensaje de peticion y respuesta (bytes)\n");
      printf("    LOG                 - cabecera del nombre de los ficheros de log.\n");
      exit(EXIT_FAILURE);
   }

   /* Inicializacion de datos totales */
   energiaTotal = 0.0;
   transaccionesTotales = sesionesTotales = transaccionesMalas = sesionesRechazadas = 0;
   InitializeCriticalSection(&datosTotalesCriticalSection);

   puertoServidor = (UINT16)strtol(argv[1], NULL, 10);
   tamanoPoolHilos = (UINT16)strtol(argv[2], NULL, 10);
   tamanoPeticion = (SIZE_T)strtol(argv[6], NULL, 10);

   /* Inicializamos las secciones críticas. */
   InitializeCriticalSection(&nodosCriticalSection);
   InitializeCriticalSection(&socketCriticalSection);

   /* Abrimos ficheros de logging. */
   InicializarLoggers(argv[7]);

   /* Parseamos el fichero de configuración para inicializar los nodos de computacion. */
   if (!ParseaFicheroConfiguracion(argv[3])) {
      ErrorExit("No se puede leer el fichero de configuracion de la estrategia");
   }

   /* Inicializacion del registro de calidad de servicio. */
   InicializarCola(&qos, (UINT32)strtol(argv[5], NULL, 10), strtod(argv[4], NULL));
   if ((qos.threadQoS = CreateThread(NULL, 0, HiloRegistroQoS, argv[7], 0, NULL)) == NULL) {
      ErrorExit("No se puede crear el hilo de monitorizacion de QoS.");
   }

   /* Inicializamos el socket de escucha. */
   InicializarSocketServicio(&socketServidor, puertoServidor);

   /* Inicializacion de las estructuras de las sesiones. */
   sesiones = (PSESION)malloc(sizeof(SESION) * tamanoPoolHilos);

   /* Lanzamiento de los hilos de servicio. */
   hndsPoolHilos = (PHANDLE)malloc(sizeof(HANDLE) * tamanoPoolHilos);

   for (i = 0; i < tamanoPoolHilos; ++i) {
      hndsPoolHilos[i] = CreateThread(NULL, 0, HiloServicioSesion, (LPVOID)&sesiones[i], 0, NULL);
      if (hndsPoolHilos[i] == NULL)
         ErrorExit("Error al crear los hilos de servicio.");
   }

   /* Lanzamos el hilo de recepcion de notificaciones de encendido y apagado. */
   if ((hndHiloNotificaciones = CreateThread(NULL, 0, HiloEsperaNotificaciones, NULL, 0, NULL)) ==
NULL) {
      ErrorExit("Error al crear el hilo de notificaciones");
   }

   /* Terminamos para no acabar con el resto de hilos. */
   ExitThread(0);
}
```

13.5.9 WoL.c

```
#include "Repartidor.h"
```

```
/* Fichero: WoL.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.0
 * Descripción: Implementa funciones para despertar nodos en base al protocolo WoL.
 */

#define WOL_SIZE 102

extern UINT16 puertoServidor;
extern PNODO nodosServidor; // Cada nodo tendra su propio mutex.
extern CRITICAL_SECTION nodosCriticalSection; // Bloquea la estructura al completo.
extern UINT16 numeroNodosServidor;

#pragma warning ( disable : 4100 )
DWORD WINAPI HiloEsperaNotificaciones(LPVOID p)
{
   SOCKET s;
   SOCKADDR_IN direccion, ssif;
   CHAR buffer[16];
   INT32 len;
   UINT32 i;

   /* Escuchamos en el puerto de servicio + 1 por datagramas UDP con las señales de
      apagado del distribuidor, que nos llegarán con el mensahe "APAGADO". */
   if ((s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) == INVALID_SOCKET) {
      Error("No se puede crear un socket UDP");
      ExitThread(EXIT_FAILURE);
   }

   direccion.sin_family = AF_INET;
   direccion.sin_port = htons(puertoServidor + 1);
   direccion.sin_addr.s_addr = htonl(INADDR_ANY);

   if (bind(s, (PSOCKADDR)&direccion, sizeof(SOCKADDR_IN)) == SOCKET_ERROR) {
      Error("No se puede asignar el puerto %d", puertoServidor + 1);
      CierraSocket(s);
      ExitThread(EXIT_FAILURE);
   }

   /* Nos bloqueamos hasta que nos llege la señal, que validamos comprobando que pone
      "DISPONIBLE" en el payload. */
   #pragma warning ( disable : 4127 )
   while (TRUE) {
      #pragma warning ( default : 4127 )

      do {
         len = sizeof(SOCKADDR_IN);
         while (recvfrom(s, buffer, sizeof(buffer), 0, (PSOCKADDR)&ssif, &len) == SOCKET_ERROR) {
            Error("No se ha podido recibir el mensaje");
         }
      } while (DireccionUDPNuestra(&ssif));

      buffer[10] = '\0'; // Sanity check.
      if (strcmp("DISPONIBLE", buffer) == 0) {
         /* Buscamos que nodo es el que se ha activado en base a la direccion que nos deja. */
         EnterCriticalSection(&nodosCriticalSection);

         for (i = 0; i < numeroNodosServidor; ++i) {
            if (nodosServidor[i].direccion.s_addr == ssif.sin_addr.s_addr && nodosServidor[i].puerto ==
ntohs(ssif.sin_port)) {
               /* Este es el nodo. Cambiamos su estado a encendido finalmente. */
               nodosServidor[i].estado = encendido;
               nodosServidor[i].sesiones = 0;
               ActualizarUtilizacion(&nodosServidor[i]);

               /* Cancelamos el temporizador que haya abierto para el encendido. */
               if (nodosServidor[i].temporizadorEncendido != NULL) {
                  if (DeleteTimerQueueTimer(NULL, nodosServidor[i].temporizadorEncendido, NULL) == 0) {
                     if (GetLastError() != ERROR_IO_PENDING) {
                        Error("No se pudo borrar el temporizador de encendido");
                     }
                  }
               }
```

```
                break;
            }
         }

         LeaveCriticalSection(&nodosCriticalSection);
      }
   }

   CierraSocket(s);

   return 0;
}

BOOL DespiertaNodo(PNODO nodo)
{
   #if !defined(_NO_APAGADO_FISICO_)
   UINT32 i, j;
   SOCKET sock;
   SOCKADDR_IN direccion;
   UINT16 puertos[] = {0, 7, 9};
   UINT8 paquete[WOL_SIZE];
   PUINT8 ptr;
   DWORD ret;

   /* Comprobamos el estado del nodo, por coherencia en la estrategia. */
   if (nodo->estado != apagado) {
      Error("El nodo %d no estaba apagado", nodo->identificador);
      return FALSE;
   }

   /* Construimos el paquete magico WoL. Seran seis bytes a 0xFF y 16 veces la
      direccion mac del nodo. */
   ptr = paquete;
   for (i = 0; i < 6; ++i)
      *(ptr++) = (UINT8)0xFF;

   for (j = 0; j < 16; ++j)
      for (i = 0; i < 6; ++i)
         *(ptr++) = (UINT8)nodo->mac[i];

   /* Construimos un socket UDP broadcast para enviar el paquete. Vamos a enviarlo
      cuatro veces a los puertos 0, 7 y 9. */
   if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) == INVALID_SOCKET) {
      Error("No se pudo crear un socket UDP");
      return FALSE;
   }

   if (!SocketBroadcast(sock)) {
      Error("No se pudo crear un socket broadcast");
      CierraSocket(sock);
      return FALSE;
   }

   direccion.sin_family = AF_INET;
   direccion.sin_addr.s_addr = htonl(0xFFFFFFFF);

   /* Buscamos una direccion de broadcast de red, para reducir la inundacion. */
   if (WSAIoctl(sock, SIO_GET_BROADCAST_ADDRESS, NULL, 0, &direccion, sizeof(SOCKADDR_IN), &ret, NULL,
NULL) == SOCKET_ERROR) {
      return FALSE;
   }

   /* Para asegurarnos, enviamos a los tradicionales puertos de WoL y varias veces. */
   for (j = 0; j < 4; ++j) {
      for (i = 0; i < 3; ++i) {
         direccion.sin_port = htons(puertos[i]);
         sendto(sock, (PCSTR)paquete, WOL_SIZE, 0, (PSOCKADDR)&direccion, sizeof(SOCKADDR_IN));
      }
   }

   CierraSocket(sock);

   #endif
```

```
   return TRUE;
}

BOOL EnviaMensajeApagado(PNODO nodo)
{
   #if !defined(_NO_APAGADO_FISICO_)
   SOCKET sock;
   SOCKADDR_IN direccion;

   /* Construimos un socket UDP broadcast para notificar a un cliente que se apage. */
   if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) == INVALID_SOCKET) {
      Error("No se pudo crear un socket UDP");
      return FALSE;
   }

   direccion.sin_family = AF_INET;
   direccion.sin_addr.s_addr = nodo->direccion.s_addr;
   direccion.sin_port = htons(nodo->puerto + 1);

   /* Enviamos la informacion. Contendra un buffer con el texto "APAGADO" */
   if (sendto(sock, "APAGADO", 10, 0, (PSOCKADDR)&direccion, sizeof(SOCKADDR_IN)) == SOCKET_ERROR) {
      Error("No se pudo notificar la disponibilidad del servidor");
   }

   CierraSocket(sock);

   #endif
   return TRUE;
}
```

13.6 SSIF

13.6.1 computa_x64.asm

```
; Fichero: computa_x64.asm
; Autor: Ramón Medrano Llamas
; Versión: 3.0
; Descripción: Implementa el procedimiento ComputaASM() para la arquitectura
;              x64. Contiene exactamtente 1.000 instrucciones.

.code

EXTERN consumoMem:DWORD
rand PROTO c

ComputaASM PROC
  ; Extraemos el parametro.
  mov r9, rcx

  ; Llamamos a la funcion rand().
  call [rand]
  movsxd rdx, eax
  call [rand]
  ; Hacemos el producto de los dos aleatorios.
  imul rax, rdx
  ; Calculamos el modulo con consumoMem.
  xor rdx, rdx
  movsxd rbx, consumoMem
  imul rbx, rbx, 1024d
  sub rbx, 8
  idiv rbx

  ; Traemos de memoria el dato
```

```
  mov rax, [r9+rdx]

  mov rcx, 140d
  bucle:
    ; Operamos con el.
    add rax, 01234567h
    sub rax, 01234567h
    mov rbx, 0123456700000000h
    imul rbx
    xor rdx, rdx
    idiv rbx
    loop bucle

  ; Llamamos a la funcion rand().
  call [rand]
  movsxd rdx, eax
  call [rand]
  ; Hacemos el producto de los dos aleatorios.
  imul rax, rdx
  ; Calculamos el modulo con consumoMem.
  xor rdx, rdx
  movsxd rbx, consumoMem
  imul rbx, rbx, 1024d
  sub rbx, 8
  idiv rbx

  ; Guardamos el dato a memoria.
  mov [r9+rdx], rax

  ret

ComputaASM ENDP

END
```

13.6.2 computa_x86.asm

```
; Fichero: computa_x86.asm
; Autor: Ramón Medrano Llamas
; Versión: 3.0
; Descripción: Implementa el procedimiento ComputaASM() para la arquitectura
;              x86. Contiene exactamtente 1.001 instrucciones.

.686
.MODEL flat,c

EXTERN consumoMem:DWORD
rand PROTO c

.CODE

ComputaASM PROC
  ; Preambulo.
  push ebp
  mov ebp, esp

  ; Llamamos a la funcion rand().
  call [rand]
  mov edx, eax
  call [rand]
  ; Hacemos el producto de los dos aleatorios.
  imul eax, edx
  ; Calculamos el modulo con consumoMem.
  xor edx, edx
  mov ebx, consumoMem
  imul ebx, ebx, 1024d
  sub ebx, 4
  idiv ebx
```

```
  ; Traemos de memoria el dato.
  mov ebx, [ebp+8]
  mov eax, [ebx+edx]

  mov ecx, 138d
  bucle:
    ; Operamos con el.
    add eax, 01234567h
    sub eax, 01230000h
    mov ebx, 01230000h
    imul ebx
    xor edx, edx
    idiv ebx
    loop bucle

  ; Llamamos a la funcion rand().
  call [rand]
  mov edx, eax
  call [rand]
  ; Hacemos el producto de los dos aleatorios.
  imul eax, edx
  ; Calculamos el modulo con consumoMem.
  xor edx, edx
  mov ebx, consumoMem
  imul ebx, ebx, 1024d
  sub ebx, 4
  idiv ebx

  ; Llevamos a memoria el dato.
  mov ebx, [ebp+8]
  mov [ebx+edx], eax

  ; Epilogo.
  mov esp, ebp
  pop ebp
  ret

ComputaASM ENDP

END
```

13.6.3 ssif.c

```
/* Fichero: ssif.c
 * Autor: Ramón Medrano Llamas
 * Versión: 4.17
 * Descripción: Variación del servidor sintético de Sif (SSIF) para que soporte
 *              recibir peticiones en un sólo socket servidor y simplificar el
 *              reparto de sesiones por parte del repartidor de carga.
 *              El servidor crea un pool de hilos de tamaño indicado por consola
 *              comandos al arrancar que responderan a las peticiones en un
 *              puerto también indicado al arranque.
 *              Cada hilo trata las peticiones secuencial y atomicamente, siedo
 *              todas idempotentes entre sí.
 *              Más información en: http://www.atc.uniovi.es/inf_superior/5sif/
 *              Esta version soporta la compilacion como proceso de 64 bits.
 */

/*** INCLUSIONES ***/
#include "Comun.h"

/*** FUNCIONES DE APAGADO DE NODOS ***/
BOOL NotificarDisponibilidad();
DWORD WINAPI HiloEsperaApagado(LPVOID p);
BOOL ApagarNodo();

/*** VARIABLES GLOBALES ***/
```

```
UINT64 consumoCPU;        // Iteraciones de CPU.
SIZE_T consumoLectDisco;  // En KBytes.
SIZE_T consumoEscrDisco;  // En KBytes.
SIZE_T consumoMem;        // En KBytes.
SIZE_T tamanoPeticion;    // En bytes.
UINT32 numMedioVisitas;   // Maximo de visitas.
UINT32 numHilos;          // Numero de cientes.
UINT32 numFicheros;       // Numero de ficheros.
UINT64 tamanoFichero;     // En KBytes.
CHAR dirBase[MAX_PATH];   // Directorio base.
CHAR raizDirBase[4];      // Raiz del disco de ficheros.
BOOL depuracion;          // Flag de traza.
BOOL cache;               // Indica si esta activo el cache o no.
UINT16 puerto;            // Puerto de servicio.
UINT16 puertoNotificacion;// Puerto de notificacion de disponibilidad.
DWORD tamanoSector;       // Para lecturas no cacheadas.

/* Configuracion del API de ficheros de Win32 para acceder con cache o no. */
DWORD flagsLectura;       // Desactiva o no el cache.
DWORD flagsEscritura;     // Desactiva o no el cache y el write-through.
DWORD opcionesLectura;    // Acceso compartido para leer.
DWORD opcionesEscritura;  // Borrar el fichero al cerrar.

/* Variables globales para acceder al socket de escucha de peticiones. */
SOCKET socketServicio;
SOCKADDR_IN dirServidor;
CRITICAL_SECTION socketCriticalSection;

/* Variables para el control del los hilos. */
PHANDLE handleThread;
HANDLE hiloNotificacion;

/*** FUNCIONES DE INICIALIZACION Y LIMPIEZA ***/
VOID Inicializa()
{
   consumoCPU = 0;
   consumoLectDisco = 0;
   consumoEscrDisco = 0;
   consumoMem = 0;
   tamanoPeticion = 0;
   numMedioVisitas = 0;
   numHilos = 0;
   numFicheros = 0;
   tamanoFichero = 0;
   dirBase[0] = '\0';
   depuracion = FALSE;
   cache = TRUE;
   puerto = 0;
   socketServicio = INVALID_SOCKET;
   InitializeCriticalSection(&socketCriticalSection);
   handleThread = NULL;
}

VOID Libera()
{
   DeleteCriticalSection(&socketCriticalSection);
   if (handleThread != NULL)
      free(handleThread);
}

/*** FUNCIONES AUXILIARES ***/
/* Funcion que contiene las instruciones en ensamblador. */
VOID ComputaASM(PBYTE mem);

VOID Lee(HANDLE fichLect, PBYTE mem, SIZE_T bytes)
{
   LONG posFich;
   DWORD leidos;

   /* Calculo del lugar de lectura. */
   posFich = (LONG)floor(DistribucionUniforme(0.0, (DOUBLE)(((((SIZE_T)tamanoFichero) * KBYTES) - 1) -
bytes)));
   assert(posFich >= (LONG)0);
   assert(posFich < (LONG)(tamanoFichero * KBYTES));
```

```
   /* Al ser un fichero sin cache, redondeamos al tamano de sector. */
   assert(tamanoSector > 0);
   posFich = (LONG)(floor((DOUBLE)posFich / (DOUBLE)tamanoSector) * (DOUBLE)tamanoSector);
   assert(posFich >= 0);
   assert(posFich < (LONG)(tamanoFichero * KBYTES));
   assert(posFich % (LONG)tamanoSector == 0);

   /* Establecemos el offset para escribir al medio del fichero. */
   if (posFich > 0)
      if (SetFilePointer(fichLect, posFich, NULL, FILE_BEGIN) == INVALID_SET_FILE_POINTER &&
GetLastError() != NO_ERROR)
         Error("SetFilePointer() = %d", GetLastError());

   if (!ReadFile(fichLect, mem, (DWORD)bytes, &leidos, NULL))
      Error("ReadFile() = %d", GetLastError());
}

VOID Escribe(HANDLE fichEscr, PBYTE mem, SIZE_T bytes)
{
   DWORD escritos;

   /* Escribimos siempre al principio del fichero, para ahorrar espacio de disco. */
   if (!WriteFile(fichEscr, mem, (DWORD)bytes, &escritos, NULL))
      Error("WriteFile() = %d", GetLastError());
}

VOID Computa(PBYTE memoria, SIZE_T bytes)
{
   SIZE_T i;
   UINT64 j;

   /* Rellenamos toda la memoria con enteros. */
   for (i = 0; i < bytes; ++i) {
      memoria[i] = (BYTE)(rand() % 0xFF);
   }

   /* Computamos tantas veces como nos indiquen. */
   for (j = 0; j < consumoCPU; ++j) {
      /* Computamos usando la funcion con 1K instrucciones. */
      ComputaASM(memoria);
   }
}

/*** FUNCIONES DE SERVICIO ***/
DWORD WINAPI HiloPeticion(LPVOID parametro)
{
   SOCKET socketDelCliente;
   SOCKADDR_IN dirCliente;
   INT32 longDirCliente;
   PINT8 peticion;
   UINT32 numFich;
   CHAR pathFichLect[MAX_PATH], pathFichEscr[MAX_PATH];
   HANDLE fichLect, fichEscr;
   PBYTE memoria;
   UINT32 numVisitas;
   UINT16 numHilo;
   UINT32 i, ret;
   UINT32 seed;

   /* Recepcion del parametro. */
   numHilo = *((PUINT16)parametro);
   free(parametro);

   /* Establecimiento de la semilla aleatoria (la funcion en ensamblador usa rand() por
      compatibilidad. */
   seed = 0;
   if (!GenerarUIntAleatorio(&seed))
      Error("AVISO: no se pudo generar una semilla aleatoria para el hilo %d", numHilo);
   srand(seed);

   /* Buffer para ecibir una peticion del cliente (una cadena). */
   peticion = (PINT8)malloc(sizeof(INT8) * tamanoPeticion);
```

```
   /* Path del fichero de escritura, siempre es el mismo. */
   sprintf(pathFichEscr, "%s\\Escr\\f%03d", dirBase, numHilo);

   /* Ahora no tenemos que crear el socket, ya nos viene para poder escuchar.
      Eso si, hay que proteger con un semaforo la llamada a accept para que solo
      un hilo acepte cada peticion. */
   #pragma warning ( disable : 4127 )
   while (TRUE) {
      #pragma warning ( default : 4127 )
      /* Aceptar la conexión, esperando en el mutex del socket. */
      longDirCliente = sizeof(dirCliente);
      EnterCriticalSection(&socketCriticalSection);
      socketDelCliente = accept(socketServicio, (PSOCKADDR)&dirCliente, &longDirCliente);
      LeaveCriticalSection(&socketCriticalSection);

      if (socketDelCliente == INVALID_SOCKET)
         ErrorExit("Error al aceptar una conexión\n");

      /* Configuración del socket a utilizar. */
      if (!ConfiguraSocket(socketDelCliente)) {
         Error("No se pudo configurar el socket");
         CierraSocket(socketDelCliente);
         continue;
      }

      /* Procesaremos peticiones del cliente mientras nos las vaya enviando. */
      while ((ret = recv(socketDelCliente, (PSTR)peticion, (INT32)tamanoPeticion, MSG_WAITALL)) > 0 &&
ret != SOCKET_ERROR) {
         if (depuracion)
            printf("Recibo una petición del cliente %d\n", numHilo);

         /* Apertura de los ficheros de lectura y escritura. */
         if (consumoLectDisco > 0) {
            /* Vamos a hacer un mapeo directo (estilo caché de asociación directa) entre los hilos y
los ficheros
               para repartir mejor las aperturas de ficheros y evitar problemas de violación. Notese
como deja de
               tener sentido tener mas ficheros que hilos. La aleatoriedad en los accesos vendrá dada
por la forma
               en la que se resuelvan los conflictos en el accept() */
            numFich = ((UINT32)numHilo) % numFicheros;
            sprintf(pathFichLect, "%s\\Lect\\f%03d", dirBase, numFich);
            if (depuracion)
               printf("Voy a abrir el fichero %s para lectura\n", pathFichLect);
            fichLect = CreateFile(pathFichLect, GENERIC_READ, opcionesLectura, NULL, OPEN_EXISTING,
flagsLectura, NULL);
            if (fichLect == INVALID_HANDLE_VALUE) {
               if (GetLastError() == ERROR_SHARING_VIOLATION)
                  Error("Demasiados handles abiertos para %s.", pathFichLect);
               else
                  Error("No se pudo abrir el fichero para leer (%d).", GetLastError());
               /* Respondemos para no bloquear la conexion. */
               send(socketDelCliente, (PCSTR)peticion, (INT32)tamanoPeticion, 0);
               continue;
            }
         }

         if (consumoEscrDisco > 0) {
            if (depuracion)
               printf("Voy a abrir el fichero %s para escritura\n", pathFichEscr);
            fichEscr = CreateFile(pathFichEscr, GENERIC_WRITE, opcionesEscritura, NULL, CREATE_ALWAYS,
flagsEscritura, NULL);
            if (fichEscr == INVALID_HANDLE_VALUE) {
               Error("No se pudo abrir el fichero para escribir (%d)", GetLastError());
               if (consumoLectDisco > 0)
                  CloseHandle(fichLect);
               /* Respondemos para no bloquear la conexion. */
               send(socketDelCliente, (PCSTR)peticion, (INT32)tamanoPeticion, 0);
               continue;
            }
         }

         /* Reserva de memoria para leer y escribir datos de disco. VirtualAlloc permite has reservas
forzadas. */
```

```
         memoria = (PBYTE)VirtualAlloc(NULL, consumoMem * KBYTES, MEM_RESERVE | MEM_COMMIT,
PAGE_READWRITE);
         if (memoria == NULL) {
            Error("Fallo al reservar memoria (%d)", GetLastError());
            if (consumoLectDisco > 0)
               CloseHandle(fichLect);
            if (consumoEscrDisco > 0)
               CloseHandle(fichEscr);
            /* Respondemos para no bloquear la conexion. */
            send(socketDelCliente, (PCSTR)peticion, (INT32)tamanoPeticion, 0);
            continue;
         }

         numVisitas = (UINT32)floor(DistribucionExponencial(numMedioVisitas));
         for (i = 0; i < numVisitas; ++i)  {
            if (consumoLectDisco > 0)
               Lee(fichLect, memoria, (SIZE_T)(consumoLectDisco * KBYTES));

            Computa(memoria, (SIZE_T)(consumoMem * KBYTES));

            if (consumoEscrDisco > 0)
               Escribe(fichEscr, memoria, (SIZE_T)(consumoEscrDisco * KBYTES));
         }

         /* Cierre de los ficheros de lectura y escritura. */
         if (consumoLectDisco > 0) {
            CloseHandle(fichLect);
         }

         if (consumoEscrDisco > 0) {
            CloseHandle(fichEscr); // Este cierre implica el borrado del fichero.
         }

         /* Liberacion de la zona de memoria de lectura y escritura de disco. */
         VirtualFree(memoria, 0, MEM_RELEASE);

         /* Devolver la respuesta al cliente (una cadena). */
         if (send(socketDelCliente, (PCSTR)peticion, (INT32)tamanoPeticion, 0) == SOCKET_ERROR) {
            Error("Error en el send (%d)", WSAGetLastError());
            continue;
         }
      }

      /* Cerrar la conexión. */
      CierraSocket(socketDelCliente);
   }

   /* Liberacion de la memoria que contiene la peticion y respuesta */
   free(peticion);

   /* Para evitar el warning. */
   return 0;
}

VOID MostrarUso(PCSTR ejecutable)
{
   printf("\nUso: %s USO_CPU LECT_DISCO ESCR_DISCO USO_MEMORIA MEDIA_VISITAS NUM_HILOS DIRECTORIO
PUERTO TAM_PETICION NUM_FICHEROS TAM_FICHERO PUERTO_NOTIFICACION [ -v ]\n", ejecutable);
   printf("\n    USO_CPU             - consumo de CPU en bloques de 10K instrucciones\n");
   printf("    LECT_DISCO          - lectura de disco por peticion (KiB)\n");
   printf("    ESCR_DISCO          - lectura de disco por peticion (KiB)\n");
   printf("    USO_MEMORIA         - tamano de la reserva de memoria (KiB)\n");
   printf("    MEDIA_VISITAS       - numero medio de visitas por peticion\n");
   printf("    NUM_HILOS           - Tamano del pool de hilos de servicio\n");
   printf("    DIRECTORIO          - directorio con las carpetas Lect y Escr\n");
   printf("    PUERTO              - puerto de servicio\n");
   printf("    TAM_PETICION        - tamano del mensaje de peticion y respuesta (bytes)\n");
   printf("    NUM_FICHEROS        - numero de ficheros de lectura en Lect\n");
   printf("    TAM_FICHERO         - tamano de los ficheros de lectura (KiB)\n\n");
   printf("    PUERTO_NOTIFICACION - puerto de notificacion de disponibilidad\n\n");
   printf("    -c (opc)            - desactiva el cache del sistema de ficheros\n");
   printf("    -v (opc)            - activa el modo de traza\n");
}
```

```
VOID TratarArgumentos(INT32 argc, PCSTR argv[])
{
   UINT32 numValLeidos;

   if (argc != 13 && argc != 14 && argc != 15) {
      Error("Número de argumentos incorrecto");
      MostrarUso(argv[0]);
      exit(EXIT_FAILURE);
   }
   numValLeidos = sscanf(argv[1], "%lu", &consumoCPU);
   numValLeidos += sscanf(argv[2], "%lu", &consumoLectDisco);
   numValLeidos += sscanf(argv[3], "%lu", &consumoEscrDisco);
   numValLeidos += sscanf(argv[4], "%lu", &consumoMem);
   numValLeidos += sscanf(argv[5], "%u", &numMedioVisitas);
   numValLeidos += sscanf(argv[6], "%u", &numHilos);
   numValLeidos += sscanf(argv[7], "%s", dirBase);
   numValLeidos += sscanf(argv[8], "%hu", &puerto);
   numValLeidos += sscanf(argv[9], "%u", &tamanoPeticion);
   numValLeidos += sscanf(argv[10], "%u", &numFicheros);
   numValLeidos += sscanf(argv[11], "%lu", &tamanoFichero);
   numValLeidos += sscanf(argv[12], "%hu", &puertoNotificacion);
   if (argc == 14) {
      if (strcmp(argv[13], "-v") == 0) {
         depuracion = TRUE;
      } else if (strcmp(argv[13], "-c") == 0) {
         cache = FALSE;
      } else {
         Error("Argumento final no valido. Solo puede ser '-v' o '-c'");
         MostrarUso(argv[0]);
         exit(EXIT_FAILURE);
      }
   } else if (argc == 15) {
      if (strcmp(argv[13], "-c") != 0 || strcmp(argv[14], "-v") != 0) {
         Error("Argumentos finales no validos. Solo pueden ser '-c -v'");
         MostrarUso(argv[0]);
         exit(EXIT_FAILURE);
      }
      depuracion = TRUE;
      cache = FALSE;
   }

   if (numValLeidos != 12) {
      fprintf(stderr, "Algún argumento tenía un valor incorrecto\n");
      MostrarUso(argv[0]);
      exit(EXIT_FAILURE);
   }

   /* Obtenemos el path completo del directorio de ficheros. */
   if (!GetFullPathName(argv[7], MAX_PATH, dirBase, NULL)) {
      ErrorExit("No se pudo obtener la ruta absoluta del directorio de ficheros");
   }

   printf("Servidor SSIF. Configuracion:\n");
   printf("============================\n");
   printf("  %-20s %lu\n  %-20s %lu\n  %-20s %lu\n  %-20s %lu\n  %-20s %u\n  %-20s %u\n  %-20s %s\n  %-
20s %hu\n  %-20s %u\n  %-20s %u\n  %-20s %lu\n",
          "consumoCPU:", consumoCPU, "consumoLectDisco:", consumoLectDisco,
          "consumoEscrDisco:", consumoEscrDisco, "consumoMem:", consumoMem,
          "numMedioVisitas:", numMedioVisitas, "numHilos:", numHilos,
          "dirBase:", dirBase, "puerto:", puerto, "tamanoPeticion:", tamanoPeticion,
          "numFicheros:", numFicheros, "tamanoFichero:", tamanoFichero);

   if (numHilos < 1)
      ErrorExit("El numero de hilos tiene que ser mayor que 0");

   if (tamanoPeticion < 1)
      ErrorExit("El tamano de la peticion tiene que ser mayor que 0");

   if (numFicheros < 1)
      ErrorExit("El numero de ficheros tiene que ser mayor que 0");

   if (tamanoFichero < 1)
      ErrorExit("El tamano del fichero tiene que ser mayor que 0");
```

```
   if (consumoCPU < 1)
      ErrorExit("El consumo de CPU tiene que ser mayor que 0");

   if (consumoMem < 1)
      ErrorExit("El consumo de memoria tiene que ser mayor que 0 KiB");

   if (consumoLectDisco > consumoMem)
      ErrorExit("El consumo de lectura de disco tiene que ser menor que el de memoria");

   if (consumoLectDisco > tamanoFichero)
      ErrorExit("El consumo de lectura de disco tiene que ser menor que el tamano de fichero");

   if (consumoEscrDisco > consumoMem)
      ErrorExit("El consumo de escritura de disco tiene que ser menor que el de memoria");

   if (numMedioVisitas < 1)
      ErrorExit("El número de visitas tiene que ser al menos de 1");

   printf("INFO: Usando %d ficheros de lectura de tamaño %d KBs\n", numFicheros, tamanoFichero);
}

VOID IniciarSocketServidor()
{
   /* Creación del socket de conexión. */
   if ((socketServicio = socket(PF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
      ErrorExit("Hubo error al crear el socket");

   /* Asignación de nombre al socket. */
   dirServidor.sin_family = AF_INET;
   dirServidor.sin_port = htons(puerto);
   dirServidor.sin_addr.s_addr = htonl(INADDR_ANY);

   /* Configuracion del socket. */
   if (!ConfiguraSocket(socketServicio)) {
      CierraSocket(socketServicio);
      ErrorExit("Error reconfigurando socket");
   }

   if (bind(socketServicio, (SOCKADDR *)&dirServidor, sizeof(dirServidor)) == SOCKET_ERROR) {
      CierraSocket(socketServicio);
      ErrorExit("Error asignando nombre al socket");
   }

   /* Ponerse a la escucha.*/
   if (listen(socketServicio, SOMAXCONN) == SOCKET_ERROR) {
      CierraSocket(socketServicio);
      ErrorExit("Error poniendose a la escucha");
   }
}

VOID CerrarServicio(INT32 senal)
{
   UINT32 i;

   /* Limpieza de  recursos y cierre del servicio. */
   if (senal == SIGINT) {
      for (i = 0; i < numHilos; ++i)
         SuspendThread(handleThread);

      DeleteCriticalSection(&socketCriticalSection);
      LiberarLibreria();

      free(handleThread);

      exit(EXIT_SUCCESS);
   }
}

BOOL ValidarDirBase()
{
   CHAR dirBaseCanonica[MAX_PATH];

   /* Primero, el path debe ser absoluto, si no no podemos obtener el raiz. */
   if (PathIsRelative(dirBase)) {
```

```
      Error("El path del directorio base ha de ser absoluto");
      return FALSE;
   }

   /* Hacemos el path canonico y 'bonito' */
   if (!PathCanonicalize(dirBaseCanonica, dirBase)) {
      Error("No se pudo hacer canonico el parh de la direccion base (%d", GetLastError());
      return FALSE;
   }

   /* Bonito... y copia. */
   PathMakePretty(dirBaseCanonica);
   strcpy_s(dirBase, MAX_PATH, dirBaseCanonica);

   /* Obtenemos la raiz del path. */
   if (!PathStripToRoot(dirBaseCanonica)) {
      Error("No se pudo obtener la letra de unidad en la direccion base");
      return FALSE;
   }

   /* Copiamos e imprimimos. */
   strcpy_s(raizDirBase, 4, dirBaseCanonica);

   return TRUE;
}

VOID main(INT32 argc, CHAR ** argv)
{
   UINT32 i;
   PUINT16 param;
   DWORD sectores, free, clusteres;

   /* Instalación de la señal de cierre. */
   #pragma warning ( disable : 4306 )
   if (signal(SIGINT, CerrarServicio) == SIG_ERR)
      ErrorExit("No se pudo instalar el manejador de señal.");
   #pragma warning ( default : 4306 )

   if (!InicializarLibreria(_time32(NULL))) {
      fprintf(stderr, "No se ha podido cargar la libreria\n");
      exit(EXIT_FAILURE);
   }

   /* Inicializamos las variables globales. */
   Inicializa();

   /* Para que _tmpnam() no coja el directorio temporal. */
   _putenv("TMP=");

   TratarArgumentos(argc, argv);

   /* Validamos la direccion base. */
   if (!ValidarDirBase()) {
      ErrorExit("La direccion base no es valida");
   }

   /* Obtenemos el tamano del sector en el disco donde esten los ficheros de lectura. */
   if (!GetDiskFreeSpace(raizDirBase, &sectores, &tamanoSector, &free, &clusteres)) {
      ErrorExit("No se ha podido obtener informacion sobre el disco de ficheros");
   }

   assert(tamanoSector > 0);

   printf("INFO: Usando disco %s, bytes por sector: %d\n", raizDirBase, tamanoSector);

   /* Inicialización del uso de sockets. */
   IniciarSocketServidor();

   /* Configuramos el uso del cache o no. */
   flagsLectura = 0;
   flagsEscritura = FILE_FLAG_DELETE_ON_CLOSE;
   opcionesLectura = FILE_SHARE_READ;
   opcionesEscritura = FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE;
```

```
   if (!cache) {
      flagsLectura = flagsLectura | FILE_FLAG_NO_BUFFERING;
      flagsEscritura = flagsEscritura | FILE_FLAG_NO_BUFFERING | FILE_FLAG_WRITE_THROUGH;
      printf("INFO: Se ha deshabilitado el cache del sistema de ficheros.\n");
   }

   /* El primer hilo será el de notificacion de apagado, por si acaso. */
   if ((hiloNotificacion = CreateThread(NULL, 0, HiloEsperaApagado, NULL, 0, NULL)) == NULL) {
      ErrorExit("No se puede crear el hilo de espera de eventos de apagado");
   }

   /* Lanzar Hilos que simulan los clientes. */
   handleThread = (PHANDLE)malloc(sizeof(HANDLE) * numHilos);
   if (handleThread == NULL)
      ErrorExit("Error al reservar memoria para los hilos");

   for (i = 0; i < numHilos; ++i) {
      /* Guardamos el parametro en el heap, ya que los enteros y los punteros
         no tienen el mismo tamaño en x64. */
      param = (PUINT16)malloc(sizeof(UINT16));
      *param = (UINT16)i;
      handleThread[i] = CreateThread(NULL, 0, HiloPeticion, param, 0, NULL);
      if (handleThread[i] == NULL)
         ErrorExit("Hubo error al lanzar el thread");
   }

   /* Ahora que estamos listos, avisamos al distribuidor local de que nos asigne sesiones. */
   if (!NotificarDisponibilidad())
      Error("No se pudo notificar la disponibilidad del servicio");

   /* Terminamos para no acabar con el resto de hilos. */
   ExitThread(0);
}
```

13.6.4 WMI.c

```
#include "Comun.h"

/* Fichero: WMI.c
 * Autor: Ramón Medrano Llamas
 * Versión: 1.0
 * Descripción: Implementa las funciones relativas al apagado y encendido del nodo.
 */

extern UINT16 puerto;
extern UINT16 puertoNotificacion;

BOOL NotificarDisponibilidad();
DWORD WINAPI HiloEsperaApagado(LPVOID p);
BOOL ApagarNodo();

BOOL NotificarDisponibilidad()
{
   SOCKET sock;
   SOCKADDR_IN direccion;
   DWORD ret;

   /* Construimos un socket UDP broadcast para notificar la disponibilidad del servidor. */
   if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) == INVALID_SOCKET) {
      Error("No se pudo crear un socket UDP");
      return FALSE;
   }

   if (!SocketBroadcast(sock)) {
      Error("No se pudo crear un socket broadcast");
      CierraSocket(sock);
      return FALSE;
   }
```

```
   /* Rellenamos con la direccion de broadcast de la red. */
   direccion.sin_family = AF_INET;
   direccion.sin_addr.s_addr = htonl(0xFFFFFFFF);
   direccion.sin_port = htons(puertoNotificacion);

   /* Buscamos una direccion de broadcast de red, para reducir la inundacion. */
   if (WSAIoctl(sock, SIO_GET_BROADCAST_ADDRESS, NULL, 0, &direccion, sizeof(SOCKADDR_IN), &ret, NULL,
NULL) == SOCKET_ERROR) {
      return FALSE;
   }

   /* Enviamos la informacion. Contendra un buffer con el texto "DISPONIBLE" */
   if (sendto(sock, "DISPONIBLE", 10, 0, (PSOCKADDR)&direccion, sizeof(SOCKADDR_IN)) == SOCKET_ERROR) {
      Error("No se pudo notificar la disponibilidad del servidor");
   }

   CierraSocket(sock);
   return TRUE;
}

#pragma warning ( disable : 4100 )
DWORD WINAPI HiloEsperaApagado(LPVOID p)
{
   SOCKET s;
   SOCKADDR_IN direccion, origen;
   CHAR buffer[16];
   INT32 len;

   /* Escuchamos en el puerto de servicio + 1 por datagramas UDP con las señales de
      apagado del distribuidor, que nos llegarán con el mensahe "APAGADO". */
   if ((s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) == INVALID_SOCKET) {
      Error("No se puede crear un socket UDP");
      ExitThread(EXIT_FAILURE);
   }

   direccion.sin_family = AF_INET;
   direccion.sin_port = htons(puerto + 1);
   direccion.sin_addr.s_addr = htonl(INADDR_ANY);

   if (bind(s, (PSOCKADDR)&direccion, sizeof(SOCKADDR_IN)) == SOCKET_ERROR) {
      Error("No se puede asignar el puerto %d", puerto + 1);
      CierraSocket(s);
      ExitThread(EXIT_FAILURE);
   }

   /* Nos bloqueamos hasta que nos llege la señal, que validamos comprobando que pone
      "APAGADO" en el payload. */
   do {
      len = sizeof(SOCKADDR_IN);
      while (recvfrom(s, buffer, sizeof(buffer), 0, (PSOCKADDR)&origen, &len) == SOCKET_ERROR) {
         Error("No se ha podido recibir el mensaje de apagado");
      }

      buffer[7] = '\0'; // Sanity check.
   } while (DireccionUDPNuestra(&origen) || strcmp("APAGADO", buffer) != 0);

   /* Al llegar aqui, un mensaje de otro nodo, con un contenido valido ha llegado. Apagamos. */
   ApagarNodo();

   CierraSocket(s);

   return 0;
}

BOOL ApagarNodo()
{
   HANDLE token;
   TOKEN_PRIVILEGES privilegios;

   /* Obtenemos el token de privilegios. */
   if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &token))
      return FALSE;
```

```
   /* Buscamos el LUID para apagado. */
   LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME, &privilegios.Privileges[0].Luid);

   privilegios.PrivilegeCount = 1;
   privilegios.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

   /* Obtenemos el privilegio de apagado. */
   AdjustTokenPrivileges(token, FALSE, &privilegios, 0, (PTOKEN_PRIVILEGES)NULL, 0);

   if (GetLastError() != ERROR_SUCCESS)
      return FALSE;

   /* Iniciamos el apagado. */
   if (InitiateSystemShutdown(NULL, NULL, 0, TRUE, FALSE) == 0)
      return FALSE;

   /* Eliminamos el token. */
   privilegios.Privileges[0].Attributes = 0;
   AdjustTokenPrivileges(token, FALSE, &privilegios, 0, (PTOKEN_PRIVILEGES)NULL, 0);

   exit(EXIT_SUCCESS);
}
```

C LICENCIAS DE SOFTWARE

Licencia GPL v2.1