Universitat Oberta de Catalunya

Proyecto Fin de Carrera

Ingeniería Informática

Desarrollo e integración de abstracciones software para el gobierno y gestión de energía en sistemas operativos GNU/Linux

Alumno: Javier Muñoz Mellid

Consultor: Dr. Francesc Guim Bernat

A Coruña, junio de 2011

- *a mi madre y a mi abuela*

“Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad”

--Albert Einstein

Agradecimientos

Con gran emoción finalizo este proyecto. A estas alturas del mismo, tengo la oportunidad de compartir mi más sincero agradecimiento con todas aquellas personas que me han apoyado y ayudado a hacerlo posible. Raras veces es posible compartir la emoción de llegar a la meta con tus seres queridos y amigos pero ésta es una de ellas. La experiencia ha sido gratificante y enriquecedora. Ahora una nueva etapa comienza, llena de ilusiones y nuevos proyectos.

En primer lugar me gustaría mostrar mi agradecemiento a todas aquellas personas que directa o indirectamente me han animado a llegar hasta aquí a lo largo de los años. A mis amigos y a mi familia.

También a nivel técnico, para este proyecto, me gustaría agradecer la colaboración y ayuda prestada por Rafael Wysocki, desarrollador y responsable de mantenimiento del subsistema de gestión de energía en el kernel Linux, por sus aclaraciones referentes al nuevo marco de gestión de energía y su interacción con el subsistema PCI; a Greg Kroah Hartman, integrador y responsable de mantenimiento del subsistema de drivers, por la rápida integración del código enviado y a Wu Zhangjin por el testing realizado en hardware en cuanto a los procesos de suspensión e hibernación del sistema una vez aplicados los nuevos parches.

A nivel personal me gustaría agradecer a Cesc, mi consultor de proyecto; la flexibilidad, cercanía y seguimiento con el que hemos trabajado en el proyecto. Su consejo y guía. Gracias Cesc.

A mis compañeros y amigos en Igalia, empresa de Software Libre y proyecto en el que tengo el honor de participar y aprender algo nuevo cada día, por su contagiosa ilusión y optimismo a todos los niveles. A todos ellos, gracias.

A mi amigo Andrés, colega y ciudadano del mundo. Un Leonés que me ha infundido ánimo y alegría. Gracias Andrés.

A mi amigo Carlos. Un soñador y optimista nato, por nuestras innumerables conversaciones y momentos de desarrollo intelectual. Gracias Carlos.

Agradecer a mi amigo Chema, una de esas mentes privilegiadas que no dejan de sorprenderte, sus buenos consejos y su apoyo en los momentos difíciles. Gracias de corazón Chema.

Por último, me gustaría agradecer a mi familia su apoyo incondicional. A mi padre, a mi abuelo y a Carla. A mi tía Mari, por creer en mi hasta el final. Gracias a todos.

A Coruña, junio de 2011

Contenidos

1. Introducción....7
1.1 Contexto....7
1.2 Energía, Potencia y Trabajo....10
1.3 Fuentes de potencia en circuitos actuales....12
1.4 Objetivos principales del proyecto....13
2. Interface ACPI y GNU/Linux....14
2.1 Advanced Configuration and Power Interface Specification (ACPI)....14
2.2 Implementación en GNU/Linux....17
2.3 Experiencia de usuario....20
3. Preservación de energía y API en Linux drivers....22
3.1 Drivers, módulos y hacking....22
3.2 Modelos, interface y lógica....23
3.3 Análisis de un driver tipo: 3Com EtherLink PCI III/XL ethernet....26
3.4 Contribución de parches para la nueva gestión de energía en el driver sm7xx de Linux....29
3.4.1 Soporte actual para el driver gráfico sm7xx en el kernel Linux....29
3.4.2 Desarrollo de mejoras y actualización del código base del driver....29
3.4.3 Contribución e integración de mejoras en el kernel Linux....32
4. Gestión de energía y aplicaciones web....33
4.1 Granularidad y gestión de energía en servidores web....33
4.2 Arquitectura web modificada para gestión de energía....35
4.2.1 El patrón de diseño Modelo-Vista-Controlador....35
4.3.2 Arquitectura propuesta para la gestión de energía en aplicaciones web....38
4.3 Desarrollo de un gobernador para aplicaciones web....40
4.3.1 Instrumentalización dinámica del patrón Modelo-Vista-Controlador en PHP5....40
4.3.2 Implementación y despliegue del gobernador web en Python....42
4.3.3 Infraestructura hardware....44
4.3.4 Software instalado y entorno de desarrollo....45
4.4 Evaluación de un gobernador para aplicaciones web....48
4.4.1 Estrategia de evaluación....48
4.4.2 Evolución temporal en los tiempos de respuesta de un método....49
4.4.3 Evolución temporal en las llamadas a un método....53
4.4.4 Propuesta y evaluación del algoritmo de gobierno....56
4.4.5 Herramientas de evaluación empleadas....60
5. Conclusiones y trabajo futuro....61
6. Referencias....63
Anexo A. Acrónimos....64
Anexo B. Software adjunto....65
Anexo C. Código fuente....66
Código fuente parches driver sm7xx....66
Código fuente aplicación web MVC con soporte de gestión de energía....78
Código fuente del gobernador web....91
Anexo D. Planificación....110
Fases del proyecto....110
Seguimiento y desvío sobre la planificación original....112
Diagrama de Gantt y Calendario....112

Figuras

Figura 1. OSPM/ACPI Global System.....15
Figura 2. Implementación Arquitectura ACPI en GNU/Linux.....18
Figura 3. Estados de Energía Global System y transiciones en ACPI.....19
Figura 4. Patrón de diseño Modelo-Vista-Controlador.....35
Figura 5. Diagrama de secuencia del patrón Modelo-Vista-Controlador.....36
Figura 6. Arquitectura propuesta para gestión de energía en aplicaciones web MVC.....38
Figura 7. Diagrama de secuencia del patrón Modelo-Vista-Controlador modificado.....41
Figura 8. Implementación gobernador web.....43
Figura 9. Tiempos de respuesta en segundos para cada método a mínima y máxima frecuencia.....49
Figura 10. Evolución temporal del tiempo de respuesta en segundos (fact50).....50
Figura 11. Evolución temporal vs diferencial de los tiempos de ejecución (fact50).....51
Figura 12. Número de métodos iniciados vs finalizados por segundo (fact50).....53
Figura 13. Variabilidad en el número de métodos iniciados vs finalizados por segundo (fact50).....54
Figura 14. Diferencial de métodos iniciados vs finalizados por segundo (fact50).....55
Figura 15. Gráficas de evaluación gobernador web (3 usuarios/segundo, 30 segundos, fact50).....59

1. Introducción

1.1 Contexto

$E = M * C^2$ es sin lugar a dudas una de las ecuaciones físicas con mayor reconocimiento e impacto en el denominado mundo desarrollado.

Desde un punto de vista numérico esta ecuación, deducida y compartida a raíz de uno de los ensayos de *Albert Einstein*[1] en 1905, relaciona energía[2] y masa con un factor de conversión igual a la velocidad de la luz al cuadrado. Sin restar protagonismo a este enfoque matemático sin duda una vista física complementaria revela conocimiento no intuitivo sobre aspectos y conceptos que han influído de manera decisiva en el desarrollo y evolución de nuestro mundo en la última centuria.

Así, dicha fórmula nos permite deducir que toda masa conlleva una cierta cantidad de energía aunque se encuentre en reposo e indica la relación cuantitativa entre masa y energía en cualquier proceso en que una se transforme en la otra. Con este conocimiento, por tanto, se puede considerar *E* como la energía liberada cuando una cierta cantidad de masa *M* es desintegrada, o como la energía absorbida para crear esa misma cantidad de masa. En ambos casos, la energía liberada o absorbida es igual a la masa (destruída o creada) multiplicada por su factor de conversión, la constante de la velocidad de la luz al cuadrado, C^2.

Lo anterior revela dos conceptos estructurales en la motivación de este trabajo. La existencia de energía y el uso útil que se hace en la transformación de la misma.

Ciento seis años después de ese ensayo, el mundo actual es el resultado de una evolución física y social continua en el que el consumo dirigido de energía es la locomotora que guía nuestro progreso y desarrollo.

La energía mueve el mundo. "Producirla y consumirla" se convierte en sinónimo de riqueza y bienestar social y como tal su demanda en sociedades desarrolladas es elevada.

En este punto es comprensible entender la energía como un recurso valioso en un mercado global donde transformarla y transportarla allí donde se necesita significa incurrir en costes relevantes, independientemente de su naturaleza o el tipo de almacenamiento que se emplee en su distribución.

La tecnología está presente en todos los aspectos de nuestra vida y ésta a su vez es alimentada por energía. La convergencia entre sistemas de comunicación e información es una realidad a día de hoy. Su madurez conlleva usuarios más exigentes con las características y funcionalidades disponibles en los dispositivos que emplean a diario.

Al mismo tiempo, y en plena consolidación, en la industria *mobile* los usuarios esperan una experiencia cercana a aquello que conocen en el uso diario de sus computadoras personales en el campo profesional o en el hogar. Se espera una experiencia de usuario completa en cuanto a rendimiento, interfaces, almacenamiento, interoperabilidad y conectividad. Así, pantallas táctiles luminosas en color de alta definición, procesadores más rápidos o conectividad permanente forman

1 http://en.wikipedia.org/wiki/Albert_Einstein
2 http://es.wikipedia.org/wiki/Energía

parte del desafío en consumo energético al que se enfrenta esta industria.

Sin duda, el consumo de potencia es el factor limitante no sólo en la industria mobile sino en un escenario mundial con una sociedad global cada vez más peocupada por alcanzar una sostenibilidad real en todos los campos, incluído el energético.

El enfoque actual que se persigue en sistemas físicos de computación es, por tanto, la preservación o conservación de la energía en todos sus elementos constituyentes y niveles de abstracción. Se persigue por tanto reducir el consumo energético en el transcurso de la operación del hardware con la intención de reducir el consumo total de la computadora como un todo.

En esta línea, el presente trabajo estudia y presenta como se realiza esta preservación de la energía en computadoras genéricas que delegan dicha conservación y gobierno en el sistema operativo de las mismas.

A lo largo de las siguientes páginas se estudia como un sistema operativo *GNU/Linux* es responsable de realizar esta actividad recorriendo sus diferentes capas de abstracción hardware y software con foco en la preservación de energía a bajo nivel y el empleo de abstracciones software para su gestión y gobierno.

En concreto, el trabajo aparece estructurado en seis apartados y tres anexos. El apartado actual, el primero de estos seis apartados, comparte las motivaciones del autor para llevar a cabo este trabajo al igual que contextualiza el dominio y establece el punto de partida para el mismo.

El segundo apartado introduce la especificación ACPI, su implementación en *GNU/Linux* y la experiencia de usuario que presentan todos aquellos sistemas basados en *GNU/Linux* en la actualidad. El apartado resalta el rol de ACPI como una pieza central de futuro en gestion de energía. Gran número de ideas y conceptos, así como el propio código fuente de este trabajo, están influídos por ACPI.

El tercer apartado presenta a nivel técnico la preservación de energía que se lleva a cabo en un sistema *GNU/Linux* en base a los modelos existentes y sus interfaces. Un *driver* en producción, y a modo ilustrativo, es analizado en detalle. Este apartado finaliza con una contribución pública de código fuente para uno de los *drivers* del *kernel Linux*. Esta contribución forma parte del trabajo realizado en este apartado.

El apartado cuatro es el apartado más extenso. Es un apartado práctico en el que se propone la implementación de un gobernador web que actúe en base a la carga computacional de una aplicación *web*. El apartado estudia el patrón Modelo-Vista-Controlador con la intención de dotar el Modelo de una interface de gestion energética para su instrumentalización y gobierno. Así, se incluye un prototipo de aplicación *web* y un gobernador *web* que implementan dicha propuesta.

En el apartado cinco se presentan las conclusiones y trabajo futuro derivado del presente proyecto.

En el apartado seís, al igual que en los Anexos, se añade información relacionada y de interés con los apartados anteriores.

El actual proyecto, por tanto, presenta un recorrido exploratorio por la actual arquitectura de gestión de energía en un sistema *GNU/Linux*. Identificando y aplicando aquellas ideas, conceptos y abstracciones software que en opinión del autor son relevantes para una preservación de energía

eficaz en los diferentes niveles de complejidad de un sistema.

1.2 Energía, Potencia y Trabajo

El término energía tiene diversas acepciones y definiciones, relacionadas con la idea de la capacidad para obrar, transformar o poner en movimiento. Así en física, energía se define como la capacidad para realizar un trabajo.

La teoría de la relatividad especial establece una equivalencia entre masa y energía por la cual todos los cuerpos, por el hecho de estar formados de materia, contienen energía; además, pueden poseer energía adicional que se divide conceptualmente en varios tipos según las propiedades del sistema que se consideren.

La energía no es un estado físico real, ni una “sustancia intangible” sino sólo una magnitud escalar que se le asigna al estado de un sistema físico, es decir, la energía es una herramienta o abstracción matemática de una propiedad de los sistemas físicos. Así, a modo de ejemplo, podemos decir que un sistema con energía cinética nula está en reposo.

Al mismo tiempo, las leyes físicas de conservación postulan que durante la evolución temporal de un sistema aislado ciertas magnitudes tienen un valor constante. Puesto que el universo entero constituye un sistema aislado puede aplicársele por tanto diversas leyes de conservación.

Dentro de las leyes de conservación en física clásica centramos nuestro interés en la ley de la conservación de la energía, base para el primer principio de la termodinámica, que afirma que la cantidad total de energía en cualquier sistema aislado sin interacción con ningún otro sistema permanece invariable con el tiempo, aunque dicha energía puede transformarse en otra forma de energía.

Otra forma de entender el alcance de la ley de la conservación de la energía es afirmar que la energía no puede crearse ni destruirse, sólo puede transformarse de una forma a otra o transferirse de un cuerpo a otro; pero en su conjunto permanece estable (o constante).

Es relevante mencionar también que la anterior transformación o transferencia de energía entre cuerpos no se realiza sin pérdidas. La segunda ley de la termodinámica establece, en algunos casos, la imposibilidad de convertir completamente toda la energía de un tipo en otro sin que exista una degradación de la misma imponiendo restricciones para las transferencias de energía que hipotéticamente pudieran llevarse a cabo teniendo en cuenta sólo el primer principio de la termodinámica. Este segundo principio apoya todo su contenido aceptando la existencia de una magnitud física llamada entropía, de tal manera que, para un sistema aislado que no intercambia materia ni energía con su entorno la variación de la entropía siempre debe ser mayor que cero.

Esta variación en la entropía de un sistema en las condiciones anteriores es un proceso irreversible en el que no es posible devolver el sistema al estado termodinámico físico anterior. Sucede entonces que un sistema físico aislado puede cambiar su estado a otro con la misma energía pero con dicha energía en una forma menos aprovechable. Un ejemplo clásico de esto último, es el proceso irreversible de un movimiento con fricción por el cual se convierte energía mecánica en energía térmica. Esa energía térmica no puede convertirse en su totalidad en energía mecánica de nuevo ya que, dado que el proceso opuesto no es espontáneo, es necesario aportar energía extra para que se produzca en el sentido contrario.

Lo anterior fundamenta el hecho de que tanto las máquinas como los procesos desarrollados por el

hombre funcionen con un rendimiento menor al 100%, lo que se traduce en pérdidas de energía dentro de lo que entendemos como una transformación irremediable de la energía.

Esta transformación de energía permite obtener un resultado, perceptible o imperceptible al ser humano, a través de la realización del concepto de trabajo; es decir, podemos entender la energía como la capacidad para realizar trabajo y el trabajo como energía en tránsito con el fin de obtener un resultado determinado. Este enfoque, contemplado entre muchos autores modernos dedicados a temas termodinámicos, considera el trabajo y calor como formas de transmisión de la energía.

En física, trabajo y trabajo útil son empleados a veces de forma indistinta. En un sistema físico sobre el que se desarrolla trabajo la diferencia no es relevante. En estos sistemas se entiende por eficiencia energética la obtención de un resultado minimizando el consumo de energía o, complementariamente, todas aquellas acciones que tienden a reducir el consumo de la misma.

Así, la eficiencia energética debe conducir en un sistema a obtener el mismo resultado anterior, manteniendo o mejorando su calidad, pero con un menor consumo de energía. Por ello, es importante no confudir la eficiencia energética con el ahorro de energía o la reducción del consumo. El servicio prestado por la energía debe mantenerse o mejorarse. Un indicador de eficiencia energética es, por tanto, la relación entre la cantidad de trabajo necesaria para proveeer un servicio y la energía consumida para proveerlo.

Paralelamente, el concepto de potencia está íntimamente relacionado con los conceptos anteriores de energía y trabajo. Así, el concepto de potencia destaca la cantidad de trabajo efectuado por unidad de tiempo.

Es importante destacar por tanto que cuándo hablemos de conservación de la energía se distinga entre la reducción de potencia y la reducción de energía. Energía es una función de integración de la potencia a lo largo del tiempo y, es por esto, que reduciendo la potencia no tiene que significar necesariamente una reducción de energía.

Variables críticas por tanto en la conservación de la energía en un sistema serán la eficiencia energética así como la cantidad de trabajo a desarrollar y la tasa del consumo del mismo.

Mencionar también que las unidades reconocidas por el Sistema Internacional para los conceptos y variables anteriores serán las que utilicemos a lo largo del presente trabajo.

Magnitud	Unidad
Energía	Joule (J)
Trabajo	Joule (J)
Potencia	Watt (W)

1.3 Fuentes de potencia en circuitos actuales

El consumo en un circuito proviene de dos fuentes, el consumo estático y el consumo dinámico.

El consumo estático es el consumo que se produce debido a corrientes de fuga (*leakage*) existentes en los transistores. Es inherente al circuito, incluso cuando el circuito está inactivo. Con el avance de la tecnología este componente de la potencia es cada vez más importante. El valor de este consumo depende de características de la tecnología que se emplea, el número de transistores y la temperatura de funcionamiento del circuito.

La potencia estática $P_{estática}$ se define como el producto del voltaje de la fuente de alimentación V_s por la corriente estática del circuito i_0. Todo esto viene recogido en la siguiente ecuación:

$$P_{estatica} = \sum_{1}^{n} i_0 * V_s \; ; \quad i_0 = i_s \left(e^{\frac{qV_{diodo}}{KT}} - 1 \right)$$

Donde i_S es la corriente inversa de saturación, o corriente de fuga, de los diodos, V_{diodo} es el voltaje del diodo, q es la unidad de carga ($1{,}602 * 10^{-19}$C), K es la constante de Boltzmann ($1{,}38 * 10^{-23}$ J/K) y T es la temperatura.

El consumo dinámico se produce debido a la carga y descarga de la capacidad de los transistores y las conexiones, y depende de la actividad del circuito. Es decir, ocurre únicamente durante las transiciones, cuando las puertas están conmutando. Por lo tanto es proporcional a la frecuencia de conmutación y cuanto mayor sea el número de conmutaciones mayor será el consumo de potencia dinámica. La siguiente ecuación representa la potencia dinámica.

$$P_{dinamica} = a * C * f * V_s^2$$

Donde a es la actividad de conmutación, C es la capacidad en cada nodo que conmuta, f es la frecuencia de reloj y V_s es el valor del potencial de alimentación.

La potencia dinámica tiene dos componentes, como muestra la ecuación: la potencia de conmutación (*crowbar*) y la de carga (*load*). La primera es debida a las corrientes que van desde la fuente de alimentación a tierra cuando el transistor cambia de estado, mientras que la de carga se debe a la corriente necesaria para cargar las capacidades de los elementos conectados a la salida.

$$P_{dinamica} = P_{conmutacin} + P_{carga}$$

1.4 Objetivos principales del proyecto

A continuación se presentan los objetivos principales del proyecto:

- Obtener conocimiento y experiencia en el soporte de gestión de energía en sistemas operativos *GNU/Linux*
- Desarrollar código fuente que permita reducir el consumo energético en el hardware tanto a nivel de *kernel* como a nivel de aplicación
- Configurar y parametrizar hardware gobernado por un sistema operativo *GNU/Linux*
- Ayudar a reducir el impacto de la computación en el planeta mediante la realización y desarrollo de proyectos software personales y profesionales más ecológicos con el medio ambiente

2. Interface ACPI y GNU/Linux

2.1 Advanced Configuration and Power Interface Specification (ACPI)

La especificación de Interface Avanzadada de Configuración y Energía, o *Advanced Configuration and Power Interface* (ACPI), fué desarrollada con el ánimo de establecer interfaces comunes a la industria que permitiesen la gestión de energía y su configuración en dispositivos que delegan esta responsabilidad en el OS. En la industria, esto último es conocido con el acrónimo de OSPM o *Operating System-directed configuration and Power Management*.

ACPI es la consolidación de varios intentos y aproximaciones anteriores que trata de responder a sus mismas necesidades desde un punto de vista menos heterogéneo y más flexible. Así, ACPI agrupa y sustituye rutinas de código localizadas en la BIOS, APM, interfaces de programación de aplicaciones (PNPBIOS APIs, tablas para la especificación multiprocesador (MPS), etc. en una especificación de interface bien definida; aunque extensa y muy compleja. ACPI también proporciona los mecanismos necesarios para realizar una transición ordenada entre el hardware más antiguo y el más reciente en caso de desearlo.

Las interfaces y concepto OSPM que se recogen en la especificación son transversales a las diferentes implementaciones de computadoras genéricas que existen en la industria. Por ejemplo, computadoras personales, estaciones de trabajo, móviles o servidores son implementaciones que pueden beneficiarse de esta especificación OSPM/ACPI. Cabe mencionar, que junto a lo anterior, los OS más difundidos y empleados en el mercado de computadoras; como son *Microsoft Windows*, *GNU/Linux*, *BSD* y *AppleOS* disponen de soporte para ACPI habilitado, en mayor o menor medida, por defecto.

La especificación a su vez explota el concepto de conservación de la energía a través de la transición de dispositivos en estados de bajo consumo, o incluso no consumo, cuando no están realizando trabajo útil.

ACPI, por tanto, describe las interfaces hardware, software y estructuras de datos que, una vez implementados, activan el soporte para realizar OSPM y permite así realizar la gestión de energía desde el propio OS empleando una interface abstracta entre el mismo y el hardware. Es importante destacar que actualmente el soporte inmaduro en cuanto a gestión y configuración de energía se refiere provoca que los principales fabricantes y desarrollos no lleguen a plantearse el uso del mismo como una clara ventaja en el mercado de consumo.

Por otro lado, el hecho de que gran parte de tecnologías anteriores hayan decidido incluir una parte relevante del código de gestión de energía en la BIOS ha provocado que éste alcance un tamaño y complejidad considerables. Esto provoca que el OS no pueda interactuar de forma flexible con los dispositivos físicos, al mimo tiempo que la propia implementación no puede evolucionar de forma natural incorporando las últimas innovaciones y mejoras rápidamene; por encontrarse esta última localizada en un área no fácilmente actualizable desde un punto de vista de desarrollo y mejora continua.

En cuanto a la estructura y arquitectura de ACPI la especificación no sólo define las interfaces a nivel hardware y software así como las estructuras de datos sino también la semántica de dichas

interfaces.

La siguiente figura muestra los componentes software y hardware más relevantes a OSPM/ACPI así como su interrelación.

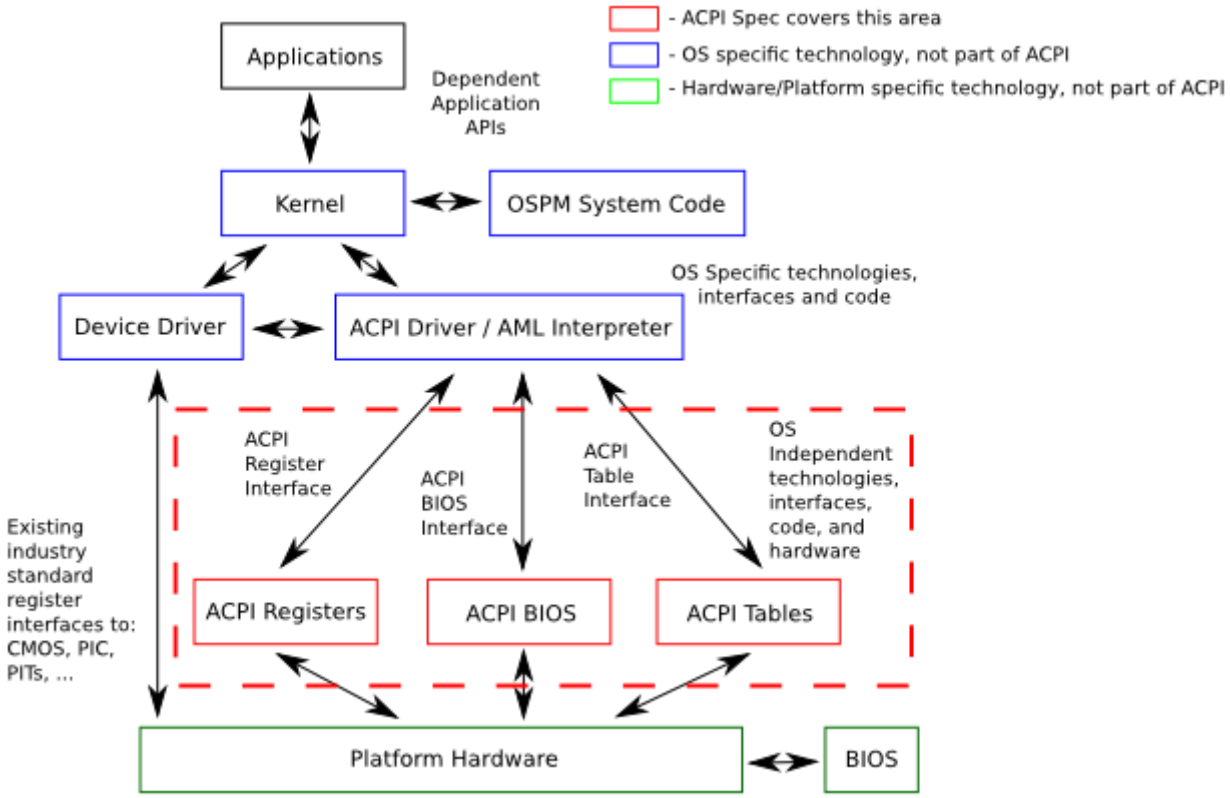


Figura 1. OSPM/ACPI Global System

Como se observa la especificación describe las interfaces entre componentes, los contenidos de las tablas de descripción de sistema ACPI, y la semántica relacionada del resto de componentes.

Las tablas de descripción de sistema ACPI, las cuales describen un hardware de plataforma concreto, son el corazón de la implementación ACPI al mismo tiempo que el *firmware* de sistema ACPI lleva a cabo, entre sus responsabilidades, el suministro de tablas ACPI y no así el suminisro menos flexible de una API nativa.

En ACPI existen tres componentes en tiempo de ejecución:

- *ACPI System Description Tables*. Describen la interface hardware. Las tablas ACPI contienen bloques de definición que pueden hacer uso de un tipo de pseudo-código, cuya interpretación es llevada a cabo por el OS. Así, OSPM emplea un intérprete que ejecuta procedimientos codificados y almacenados en las tablas ACPI; estos procedimientos contienen bloques de definición. El lenguaje de pseudo-código es conocido como *ACPI Machine Language* (AML)
- *ACPI Registers*. La parte restringida de la interface hardware, descrita (al menos en su localización) por las *ACPI System Description Tables*.
- *ACPI System Firmware*. Hace alusión a la porción del *firmware* que es compatible con las especificaciones ACPI. Típicamente este ćodigo es responsable de arrancar (*boot*) la

máquina e implementa interfaces para las operaciones *sleep*, *wake* y *restart*. Las *ACPI Description Tables* son proporcionadas por el *ACPI System Firmware*.

Teniendo en cuenta lo anterior es importante resaltar la necesidad que presenta un OS de ser modificado o alterado para soportar ACPI e implementar OSPM. Los siguientes puntos reflejan las mejoras y elementos necesarios que un OS debe incluir para soportar todas las interfaces definidas en ACPI:

- Usar un mapa de direcciones de sistema con la intención de notificar interfaces
- Encontrar y consumir las *ACPI System Description Tables*
- Interpretar el *ACPI machine language* (AML)
- Enumerar y configurar hardware descritos en el *ACPI Namespace*
- Interface con el *power management timer*
- Interface con la *real-time clock wake alarm*
- Entrar en *ACPI mode* (en sistemas hardware más antiguos)
- Implementar políticas de gestión de energía de dispositivo
- Implementar gestión de recursos de energía
- Implementar estados de energía para el procesador en los manejadores *idle* del planificador
- Controlar estados de rendimiento de procesador y dispositivo
- Implementar el modelo termal ACPI
- Implementar el modelo de programación de evento de ACPI, incluyendo la gestión de interrupciones SCI, eventos fijos, eventos de propósito general, interrupciones del controlador embebido y soporte de dispositivo dinámico.
- Soportar la adquisición y liberación del *Global Lock*
- Usar el registro de *reset* para hacer un *reset* al sistema
- Proporcionar APIs para influir en la política de gestión de energía
- Implementar soporte de *driver* para *ACPI-defined devices*
- Implementar APIs soportando los indicadores del sistema
- Soportar todos los estados de sistema (S1-S5)

Desde un punto de vista funcional, las áreas cubiertas por la especificación son 7: *system power management*, *device power management*, *processor power management*, *Plug and Play*, *handling of system events*, *battery management* y *thermal management*.

Las responsabilidades funcionales de OSPM son por tanto disponer del control de acceso directo y exclusivo sobre las funciones de configuración y gestión de energía del hardware. De este modo, durante la inicialización, OSPM es responsable de gestionar los eventos de configuración generados por el hardware así como de controlar el consumo de energía, rendimiento y estado termal del sistema contando en consideración siempre las preferencias de usuario, peticiones a nivel de aplicación y objetivos de usabilidad y calidad de servicio. Más en detalle las áreas funcionales que permiten a OSPM realizar estas funciones son descritas a continuación:

- *System power management*. ACPI define mecanismos para permitir que la computadora pase a estados de bajo consumo. Esto lo permite realizar a nivel de sistema o a nivel de dispositivo.
- *Device power management*. Las tablas ACPI describen el hardware y sus estados de energía así como permiten poner un dispositivo en diferentes estados de energía.
- *Processor power management*. Mientras el OS está en estado *idle* ACPI permitirá poner el

procesador en estados de bajo consumo.

- *Device and processor performance management*. Mientras el sistema se encuentra activo, OSPM permite realizar transiciones entre estados para dispositivos y procesadores con la intención de obtener el balance deseado entre rendimiento y conservación de la energía.
- *Configuration / Plug and Play*. ACPI especifica información empleada para enumerar y configurar hardware.
- *System Events*. ACPI define mecanismos muy flexibles para encaminar eventos a la lógica de cada chip en hardware.
- *Battery management*. La política de gestión de batería mueve APM BIOS a ACPI. Las baterías compatibles con ACPI disponen de un pequeño interface definido por métodos de control AML.
- *Thermal management*. ACPI soporta gestión termal con la intención de proporcionar un modelo escalable a fabricantes que les permita definir zonas, indicadores y métodos de control para una correcta gestión termal.
- *Embedded controller* / *SMBus controller*. ACPI define un hardware estándar y comunicaciones software a modo de interface entre un OS bus *driver* y un controlador SMBus que permite proporcionar a un fabricante características útiles que un OS y aplicaciones puedan emplear.

2.2 Implementación en GNU/Linux

El soporte de ACPI en el *kernel* gira alrededor de ACPICA (*ACPI Component Architecture*). ACPICA incluye el intérprete AML que implementa la abstracción hardware que se logra a través de ACPI.

Es importante destacar que ACPICA no implementa ninguna política, delegando esa responsabilidad en código específico y dependiente del OS. Así, un sencillo fichero (osl.c) sirve de interface entre ACPICA y todas aquellas funciones dependientes, en este caso, del *kernel Linux*.

En el diagrama de la Figura 1 se aprecian los diferentes bloques de arquitectura de una computadora genérica. A través de un código de colores (verde, rojo y azul) se divide la complejidad de la arquitectura en tres capas o niveles. Así, en la figura existe un nivel de implementación hardware en color verde; un nivel de interface ACPI en color rojo, también localizado en hardware, y empleando tecnología de *mapping* en el espacio de memoria; y un último nivel, en color azul, donde aparecen todos los subsistemas software que instancia el *kernel Linux* y que interactuarán con el mismo y con el nivel anterior (interface ACPI en hardware). Gracias a esto último se da cumplimiento a las peticiones que se realizan desde las aplicaciones. Dichas aplicaciones están situadas en la cima de la pila y son representadas con una caja blanca.

El *kernel Linux*, y desde una perspectiva del procesador, divide la ejecución de su *Instruction Set Architecture* (ISA) en instrucciones protegidas o no protegidas. Esta implementación es dependiente de la arquitectura del procesador. En la arquitectura x86 y compatibles dicha implementación se lleva a cabo empleando 2 modos, de un total de 4 disponibles, denominados *user mode* (ring-3) y *kernel mode* (ring-0). Empleando estos dos modos junto a un enfoque de interrupciones hardware vectorizadas el diseñador del OS en esta arquitectura puede, en ausencia de errores y defectos en la implementación hardware y software, garantizar que las instrucciones que se ejecutan en *supervisor mode* son seguras y no deberían suponer la pérdida en la ejecución de los diferentes caminos lógicos que tienen lugar en el procesador.

En el nivel del *kernel*, tercera capa, los bloques denominados *Button*, *Battery*, *Processor*, *AC*, *Thermal* y *Fan* son *ACPI drivers* que implementan funcionalidades opcionales. Estos bloques son implementados como módulos del *kernel* y contienen políticas relacionadas con los dispositivos y las características específicas que poseen.

En la arquitectura también encontramos un sistema de ficheros virtual (*/proc/acpi*) que sirve de interface entre *user mode* y *kernel mode* para aplicaciones que necesiten consumir la información disponible del sistema de gestión de energía. Algunos ejemplos de este tipo de información pueden ser porcentajes o estados de consumo en los que se encuentren los dispositivos del sistema. Esa interface se complementa en la actualidad con el más reciente sistema de ficheros virtual */sys/power*. Esta interface aparece detallada en el punto siguiente.

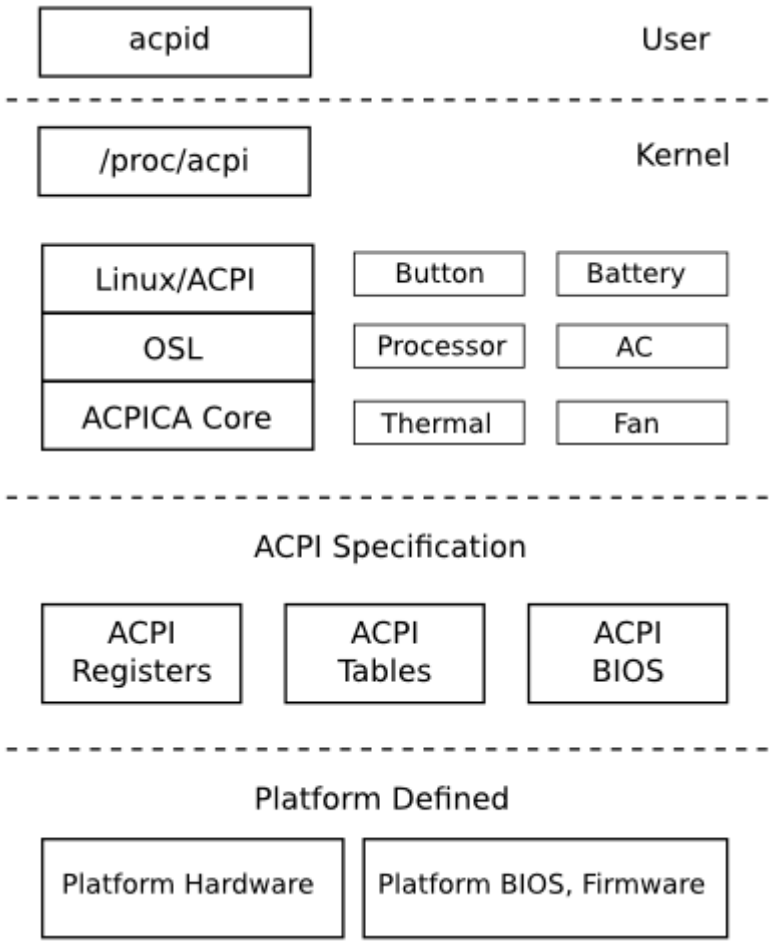


Figura 2. Implementación Arquitectura ACPI en GNU/Linux

Por último, el servidor *acpid* es una aplicación en modo usuario que se ejecuta con privilegios de administrador. Es el encargado de monitorizar eventos y ejecutar todas aquellas acciones que tengan asociadas dichos eventos. Sobre *acpid*, y sus clientes, hablaremos en el punto siguiente en detalle con la intención de exponer en profundidad la experiencia de usuario que ofrece la capa de aplicación de la arquitectura ACPI.

Otro aspecto relevante en la arquitectura es la que se encuentra relacionada con los eventos. La actual implementación es conducida por eventos, de tal forma que la implementación se registra a la

escucha de interrupciones de control de sistema o *System Control Interrupts* (SCI) Esta escucha es la que permite notificar tanto el tipo de evento sucediendo como su ocurrencia a *acpid* en última instancia.

Como se menciona en líneas anteriores, ACPI es una especificación extensa y altamente compleja en la que con frecuencia los desarrolladores tienen que pedir consenso en comunidad para cubrir todos aquellos huecos o ambiguedades técnicas que dificultan progresar en su implementación diaria. Aunque es objetivo de este punto explorar el marco de gestión sin entrar en detalles operativos de implementación, esto es, sin introducir nombres de registros a bajo nivel, valores "mágicos" o secuencias temporales recogidas en la especificación con mayor detalle sí se cree relevante mencionar los tipos de estado más importantes en la misma.

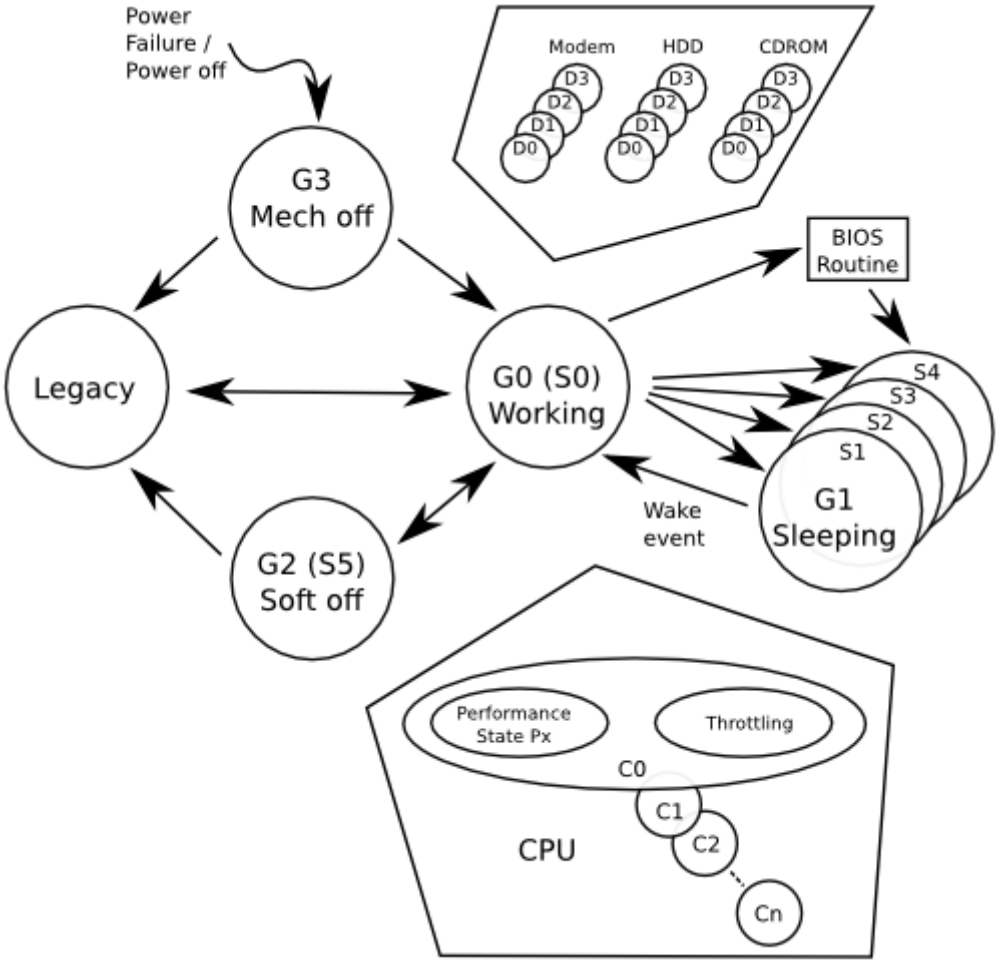


Figura 3. Estados de Energía *Global System* y transiciones en ACPI

ACPI reconoce 5 estados globales o *Global States*, y un número discreto de estados que suceden en alguno de los 5 estados globales mencionados. Una clasificación sigue a continuación:

- *Global States* (G0-G3). *Global States* o Estados Globales. G0 o *working state*, G1 o *sleeping state*, G2 o *soft-off state* y G3 o *mechanical-off state*. Existe también un quinto estado global no considerado dentro de este grupo que es denominado *Legacy*. Este último estado representa el estado del sistema cuando no se encuentra en *ACPI mode*.

- *P-states*. En el contexto de G0 (*Global Working State*) y C0 (*CPU Executing State*) tienen lugar los *P-states* (*Performance states*). Estos estados sirven para modular la frecuencia y el voltaje del procesador ejecutando instrucciones. Son estados muy efectivos y en el dominio de la gestión de energía en el *kernel Linux* tienen un subsistema propio (*cpufreq*). Juegan un papel notable respecto a la técnica de *throttling* en la que el *kernel* trata de obtener un compromiso entre la frecuencia, la potencia y el rendimiento deseado.
- *C-states*. Tienen lugar en el contexto de G0 *Working System State*. C0, uno de los *C-states* (*CPU-state*), hace referencia al estado de ejecución. Estados con un número más alto son preferidos para ahorrar más energía cuando el procesador se encuentra *idle*. Ninguna instrucción se ejecuta en C1, C2 y C3. Mencionar que ACPI remplaza el bucle de *idle* por defecto para poder entrar en C1, C2 y C3.
- *Sleep States*. ACPI dispone de los estados S0-S5. S0 es *non-sleep state*. S1 es *standby*, con el procesador detenido y el *display* apagado. S2 no se está utilizando. S3 es suspender a RAM. S4 es hibernar a disco. S5 es *soft-power off state*.
- *Device States* (D0-D3). ACPI define estados de bajo consumo para los dispositivos: D0-D3. D0 es encendido, D3 es apagado, D1 y D2 son estados intermedios. Estados con números más altos suponen mayor ahorro de energía, menos contexto de dispositivo salvado por el hardware, mayor recuperación de estado por parte del *driver* y una mayor latencia.

2.3 Experiencia de usuario

La experiencia de usuario con el subsistema de gestión de energía implementado en *GNU/Linux* tiene lugar en modo usuario. En concreto, el *kernel* comunica y acepta cualquier información relacionada con el subsistema de gestión de energía a través de una interface bien definida en el sistema de ficheros virtual que por defecto tiene el punto de montaje en */sys*. En concreto en la ruta */sys/power*.

Esta interface es utilizada para interrogar al subsistema de energía, por aplicaciones en modo usuario, en relación al estado actual en el que se encuentra el sistema o solicitar una transición entre estados de energía en el mismo. En detalle la interface aparece documentada en las fuentes del *kernel Linux* (Documentation/power/interface.txt) y en ella destacan las siguientes rutas:

- */sys/power/state* empleada en el control de estado de energía del sistema siendo sus posibles valores 'standby' (Power-On Suspend), 'mem' (Suspend-to-RAM) y 'disk' (Suspend-to-Disk).
- */sys/power/disk* para el control del modo en que debería llevarse a cabo la suspensión a disco. Sus posibles valores son 'platform', 'shutdown', 'reboot', 'testproc' o 'test' .
- */sys/power/image_size* que controla el tamaño de la imagen creada por el mecanismo de suspensión a disco.
- */sys/power/pm_trace* que permite controlar aspectos relacionados con la depuración de la implementación.

Por otro lado el *kernel* no sólo expone una interface de comunicación en */sys/power*. Con la que interrogar e interactuar con el sistema “como un todo” en cuanto a gestión de energía se refiere. También hace uso de la capa de entrada, o *input layer*, y *netlink* para comunicar eventos relacionados con el subsistema de energía. Esto lo realiza escuchando en las rutas */dev/input/eventX*

Aparte de la interacción esperada del usuario con el subsistema de energía, esto es, la lectura y escritura de valores en la interface */sys/power*; es habitual que un sistema *GNU/Linux* disponga de

infraestructura software en modo usuario con la intención de reaccionar y abstraer la notificación de eventos.

El demonio de eventos de ACPI o *Advanced Configuration and Power Interface* event daemon (*acpid*) es el responsable de la notificación de eventos ACPI al resto de programas en espacio de usuario. Esta pequeña indirección garantiza un mayor grado de desacoplamiento mientras se mantiene una cohesión aceptable a este nivel.

acpid se ejecuta a lo largo del *booting* del sistema como un proceso en *background* por defecto. Este proceso ejecutándose como *daemon* abre ficheros de eventos empleando la input layer y netlink. También intenta abrir ficheros de eventos en */proc/acpi* con la intención de garantizar la mayor compatibilidad hacia atrás posible.

Cuando *acpid* recibe un evento procedente de alguna de las fuentes anteriores examina una lista de reglas ejecutando aquellas que tienen una coincidencia con el evento.

Las reglas de *acpid* son definidas en simples ficheros de configuración. El fichero de configuración empleado por defecto en la mayoría de sistemas *GNU/Linux* es */etc/acpi/events*. Mencionar que cada fichero de reglas debe definir al menos 2 entradas: un evento y una acción.

El valor del evento es una expresión regular contra la cual el evento es comparado mientras el valor de la acción es una línea de comandos que será invocada a través de la *shell /bin/sh*

Además de los ficheros de reglas anteriores *acpid* también acepta conexiones en un socket UNIX, */var/run/acpid.socket* por defecto. Cualquier aplicación puede conectarse a este socket y recibir el texto de todos los eventos ACPI dejándole la responsabilidad de filtrar aquellos que sean de su interés.

Aparte de la gestión administrativa de los estados del sistema y su información así como el encaminamiento y respuesta a eventos del mismo cabe mencionar que un usuario suele acceder a información de su interés empleando aplicaciones cliente. Una de estas aplicaciones cliente en línea de comando es *acpi*. Los casos de uso ofrecidos por esta aplicación están relacionados con el uso y consumo de batería, zonas termales, adaptadores, etc. Es relevante mencionar que estas aplicaciones cliente acceden a esta información, altamente dependiente del hardware disponible en el sistema, explotando las facilidades del modelo de clases del *kernel*; esto es, en la ruta */sys/class*. En estas rutas aparecen enlaces directos a los dispositivos, estado y sus propiedades energéticas que son debidamene publicadas por el *kernel*.

En general, la experiencia de usuario en un sistema *GNU/Linux* desde el nivel del *kernel* hasta el nivel de aplicación está bien resuelta. La actual infraestructura es madura, eficiente y flexible.

3. Preservación de energía y API en Linux *drivers*

3.1 Drivers, módulos y hacking

El proceso habitual para añadir nuevas funcionalidades en el código del *kernel Linux* conlleva tres acciones específicas: añadir, modificar y/o eliminar fragmentos de código fuente. Este proceso, se lleva a cabo a través de iteraciones hasta alcanzar una nueva versión del *kernel* que se considera estable. De forma coloquial, y entre la comunidad de *Linux*, este proceso iterativo de contribución es conocido como *hacking*.

En el proceso de *hacking*, las acciones anteriores no tienen porque ser ejecutadas en un proceso secuencial. Su objetivo principal es introducir mejoras y correciones incrementales, conocidas como parches. Así, un buen parche presenta una serie de características que ayudan a mantener la calidad y gestión del código fuente del *kernel* en un entorno de desarrollo distribuído y global.

Entre estas características deseables en un parche se puede incluir el contener un número reducido de líneas de código, que incluya una única funcionalidad o que se ajuste a un código de estilo[3] reconocido en comunidad. Estas características son deseables ya que facilitan procesos de mantenimiento, calidad y gestión del código fuente en el proyecto.

La contribución de *drivers* para el *kernel Linux* también sigue este mismo proceso de *hacking*. Es decir, el desarrollo de uno o varios parches que, una vez aplicados, añadan el comportamiento deseado al código fuente del proyecto.

Mencionar también, que el proceso anterior de contribución o mejora continua no es algo aislado y es la forma natural y esperada de realizar contribuciones de código fuente a diversos proyectos en comunidades FLOSS.

Al mismo tiempo, otro aspecto relevante en el *kernel Linux* es la naturaleza de parte de su código fuente una vez compilado; que puede ser añadido de forma dinámica en caso de desearlo, es decir, mientras una imagen del *kernel* está en ejecución. Así, este bloque o fragmento de código que puede ser añadido dinámicamente recibe el nombre de módulo cargable del *kernel* o *loadable kernel module* (LKM).

Mencionar que estos módulos cargables del *kernel* fueron concebidos para facilitar tareas específicas de bajo nivel y suelen ser identificados como *drivers* de dispositivos, *drivers* de sistemas de ficheros o interfaces para realizar llamadas a sistema; pero es necesario conocer que no siempre los conceptos de *driver* o *LKM* son intercambiables. Por ejemplo, un *driver* puede encontrarse compilado de forma estática en la imagen del *kernel* y no ser cargado bajo demanda en tiempo de ejecución. Al mismo tiempo, un módulo cargable puede no ser un *driver*.

Un buen recurso puede ser encontrado en [Henderson-2006]. En el mismo se contempla información técnica en relación a la administración y programación básica de módulos cargables. Dos valiosas referencias, utilizadas en el desarrollo del presente proyecto, para la programación de *drivers* también pueden ser encontradas en los libros [Corbet-2005] y [Venkateswaran-2009]

3 http://www.kernel.org/doc/Documentation/CodingStyle

3.2 Modelos, interface y lógica

En el *kernel Linux* la mayor parte del código pertenece a *drivers* de dispositivo y, es por eso, que la mayor parte del subsistema de gestión de energía es específico para *drivers*.

En *Linux* la gestión de energía se abstrae en base a dos modelos: *System Sleep Model* (SSM) y *Runtime Power Management Model* (RPMM)

En SSM los *drivers* entran en estados de bajo consumo como parte de una transición global o "suspensión" de todo el sistema. Esta transición se conoce como *suspend* o S*uspend-to-RAM* existiendo otro tipo de suspensión, para sistemas con memoria no volatil, denominada *hibernation* o *suspend-to-disk*.

En este primer tipo de suspensión, SSM, diferentes abstracciones (*bus, types y classes*) del modelo de dispositivos de *Linux* cooperan y colaboran implementando punteros a funciones que realizaran acciones específicas y concretas a lo largo de la transición global del sistema a un estado de menor consumo de energía.

La suspensión del sistema con este modelo es total, es decir, el conjunto de dispositivos críticos y no críticos entran de forma ordenada en estado de suspensión.

El segundo modelo, RPMM, permite que los dispositivos del sistema entren en estados de bajo consumo sin necesidad de que todo el sistema sea suspendido. Este segundo modelo plantea mayores desafíos en la implementación que el primero ya que las dependencias que existen entre dispositivos deben ser satisfechas. Por ejemplo, no se puede suspender un bus y mantener los dispositivos asociados al mismo en un estado de no suspensión.

En general, se entiende que un dispositivo está suspendido cuando se encuentra inactivo (total o parcialmente). Esta suspensión puede darse en diferentes grados o estados que dependerán del tipo de dispositivo y su implementación física. Un dispositivo suspendido habitualmente muestra inactividad física respecto a E/S, interrupciones, etc.

La implementación de ambos modelos en *Linux* ha presentado algunas eventualidades a lo largo de los años. El modelo más estable y maduro es SSM. El segundo modelo, RPMM, es de reciente inclusión en el *kernel*; con *bugs* en su implementación propios de un código todavía no maduro.

En relación al código que implementan ambos modelos es importante mencionar que la inclusión del segundo modelo, RPMM, ha provocado una reescritura del subsistema que implementaba el primer modelo SSM. Esta reescritura no conlleva una sustitución del primer código base por el segundo sino una extensión del mismo con la intención de hacerlos cooperar a lo largo de un periodo de transición al segundo código base que se espera sea más robusto y fiable. Al mismo tiempo existen numerosos *drivers* que asumen la implementación del primer código base y será por tanto necesario migrarlos a esta segunda implementación. Mientras tanto, dichos *drivers*, operan en modo compatible al ser tenidos en cuenta por este segundo código base que se introdujo en 2010-2011.

En este trabajo asumimos, por tanto, el modelo de preservación de energía de suspensión completa del sistema (SSM) al mismo tiempo que mostramos colateralmente la interface expuesta en la

implementación del segundo código base. La intención, por tanto, es trabajar con el modelo de suspensión más estable y la implementación de futuro que se está introduciendo en el *kernel* actualmente.

En relación a la interface para la transición entre estados de consumo, todas las abstracciones que participan y colaboran en el proceso de suspensión (buses, tipos de dispositivo, clases de dispositivos) exponen una interface de programación que les permite influir en la gestión de consumo de los dispositivos con los que presentan dependencias.

Estas interfaces, en el segundo código base, cubren ambos modelos de preservación de energía mostrados en el punto anterior si bien, aunque pueden aparecer elementos pertenecientes al segundo modelo (señalados en el código con la palabra *runtime* en la implementación) nos centraremos en el primer modelo.

Así, las operaciones de dispositivo, implementadas como punteros a función, para la gestión de energía en los diferentes niveles de subsistema y *drivers* tienen lugar a través de la definición e inclusión de objetos de tipo *struct dev_pm_ops* en dichas abstracciones. Esta estructura, definida en *include/linux/pm.h* presenta el siguiente contenido:

```
struct dev_pm_ops {
        int (*prepare)(struct device *dev);
        void (*complete)(struct device *dev);
        int (*suspend)(struct device *dev);
        int (*resume)(struct device *dev);
        int (*freeze)(struct device *dev);
        int (*thaw)(struct device *dev);
        int (*poweroff)(struct device *dev);
        int (*restore)(struct device *dev);
        int (*suspend_noirq)(struct device *dev);
        int (*resume_noirq)(struct device *dev);
        int (*freeze_noirq)(struct device *dev);
        int (*thaw_noirq)(struct device *dev);
        int (*poweroff_noirq)(struct device *dev);
        int (*restore_noirq)(struct device *dev);
        int (*runtime_suspend)(struct device *dev);
        int (*runtime_resume)(struct device *dev);
        int (*runtime_idle)(struct device *dev);
};
```

Cuando el sistema entra en estado de suspensión (*suspend*) a cada *driver* de dispositivo se le pide que suspenda el dispositivo situándolo en un estado compatible con el estado global objetivo. De igual forma, cuando el sistema deja un estado de bajo consumo, se le pide al *driver* que devuelva (*resume*) el dispositivo a un estado de máxima energía.

Las operaciones de *suspend* y *resume* siempre suceden una a continuación de otra y son consideradas operaciones multifase.

Al mismo tiempo, y con la intención de ofrecer una garantía lógica en la secuencia de llamadas, para una correcta implementación del subsistema de gestión de energía es necesario que las dependencias entre dispositivos sean representadas en una estructura jerárquica. La implementación

actual emplea un árbol como estructura de datos con la intención de recorrer todos los dispositivos en la dirección hojas-raíz en la función *suspend* y en la dirección raíz-hojas en la función *resume*.

El orden del árbol de dispositivos es definido por el orden en el cual los dispositivos logran registrarse. Así, un hijo nunca puede ser registrado, probado o reanudado antes que su padre; y no puede ser eliminado o suspendido después de su padre. La política es, por tanto, que el árbol de dispositivos deberá coincidir con la topología de bus hardware.

Los estados *suspend* y *resume* de sistema se lleva a cabo en varias fases. Diferentes fases son empleadas para los estados *standby* o *memory sleep* ("S*uspend-to-RAM*") y el estado *hibernation* ("*suspend-to-disk*"). Cada fase involucra *callbacks* de ejecución para cada dispositivo antes de que la próxima fase comience. No todos los buses o clases soportan todas las *callbacks* y no todos los *drivers* usan todas las *callbacks*.

Todas las fases se ejecutan después de que todas las tareas del sistema o tasks son congeladas y antes de que vuelvan a descongelarse. En todas ellas se emplean *callbacks* de bus, type o class, es decir, métodos definidos en *dev->bus->pm*, *dev->type->pm* o *dev->class->pm*.

Estos *callbacks* son mutuamente exclusivos. Así, el orden será *dev->type->pm*, *dev->class->pm* o *dev->bus->pm* siempre que los objetos apuntados estén definidos y no sean nulos (NULL)

Mencionar que los *callbacks* anteriores pueden finalizar invocando métodos específicos de dispositivo o *driver* localizados en *dev->driver->pm*, pero esto no tiene por que ocurrir en todos los casos.

Una descripción técnica del orden en que se realizan los *callbacks* en la entrada y salida de estados de suspensión e hibernación puede encontrarse en la documentación del *kernel* (Documentation/power/devices.txt)

En definitiva, el soporte de preservación de energía para un dispositivo en un sistema operativo *GNU/Linux* consiste en desarrollar un *driver* que implemente la lógica específica en las transiciones entre estados.

El nivel de desarrollo de software que será necesario incluir en el *driver* viene dado en función del tipo y clase de dispositivo que se trata de soportar. Así, a modo de ejemplo, para un dispositivo compatible con bus PCI se partirá de un esqueleto que permita registrar el dispositivo como un elemento compatible con este bus. En este mismo *driver* se implementarán los comandos e instrucciones hardware específicos del mismo; dejando la activación, desactivación y cambios de estado comunes a dispositivos PCI al propio bus que implementará por defecto los métodos adecuados para todos sus dispositivos.

Este soporte, a modo de orientación a objetos implementada con estructuras de datos y punteros a función en el lenguaje C, facilita y acelera el desarrollo de *drivers* permitiendo que el desarrollador implemente aquello que no forma parte del *standard* o es una excepción en la gestión de energía del dispositivo para el que desea desarrollar un *driver*.

3.3 Análisis de un driver tipo: 3Com EtherLink PCI III/XL ethernet

En este apartado, y a modo de ilustración, resaltamos los fragmentos de código fuente que son relevantes en la implementación de un *driver* para tarjetas de red ethernet 3Com. Dicho *driver* soporta dispositivos PCI y EISA. El código del *driver* para este dispositivo se encuentra localizado en la ruta *drivers/net/3c59x.c* en las fuentes del *kernel Linux*.

Las partes de código relevantes para dar soporte a los estados *suspend* y *resume* son las siguientes:

- Estructura de registro del *driver* (*struct pci_driver vortex_driver*)
- Estructura de operaciones para la gestión de energía (*struct dev_pm_ops vortex_pm_ops*)
- Código de las operaciones de gestión de energía *suspend* y *resume* (*vortex_suspend* y *vortex_resume*)
- Directiva condicional del preprocesador para compilación condicional de código

En relación a la estructura de registro del *driver* (*struct pci_driver vortex_driver*) cabe mencionar que esta estructura es empleada en el registro del *driver* en el sistema. Para *drivers* con soporte de gestión de energía se espera que la variable *drivers.pm* de dicha estructura apunte a una estructura con las operaciones de gestión de energía que soporta. En el código del *driver* dicha asignación sucede en la línea 3267[4]:

```
static struct pci_driver vortex_driver = {
        .name           = "3c59x",
        .probe          = vortex_init_one,
        .remove         = __devexit_p(vortex_remove_one),
        .id_table       = vortex_pci_tbl,
        .driver.pm      = VORTEX_PM_OPS,
};
```

La estructura de operaciones para la gestión de energía (*struct dev_pm_ops vortex_pm_ops*) aparece “oculta” debajo de una definición de preprocesador dinámica denominada VORTEX_PM_OPS. Así, dicha definición aparece en la línea 895 del código:

```
#define VORTEX_PM_OPS (&vortex_pm_ops)

#else /* !CONFIG_PM */

#define VORTEX_PM_OPS NULL
```

En ella, VORTEX_PM_OPS toma el valor de la dirección en memoria de la estructura *vortex_pm_ops* o el valor nulo. Decidiendo esto último en base a si el *kernel* está siendo compilado con soporte para gestión de energía o no.

La estructura de operaciones para la gestión de energía *vortex_pm_ops* es inicializada en la línea de código 886:

```
static const struct dev_pm_ops vortex_pm_ops = {
```

4 La versión del kernel Linux empleada en este proyecto, donde no mencionada, es la 2.6.38

```
        .suspend = vortex_suspend,
        .resume = vortex_resume,
        .freeze = vortex_suspend,
        .thaw = vortex_resume,
        .poweroff = vortex_suspend,
        .restore = vortex_resume,
};
```

En ella se observa que sólo dos funciones, *vortex_suspend* y *vortex_resume*, tienen la responsabilidad de responder a la notificación de cambios de estado en el sistema. El uso de operaciones comunes es algo habitual en los *drivers* debido, principalmente, a que la lógica de suspensión y reanudación es reutilizable para otros estados de más bajo consumo que comparten lógica de activación/desactivación del dispositivo.

El resto de operaciones que no son inicializadas explícitamente son inicializadas implícitamente a un valor nulo (NULL) con la intención de que se lleven a cabo comportamientos por defecto.

El código para las operaciones *vortex_suspend* y *vortex_resume* aparece listado a continuación:

```
static int vortex_suspend(struct device *dev)
{
        struct pci_dev *pdev = to_pci_dev(dev);
        struct net_device *ndev = pci_get_drvdata(pdev);

        if (!ndev || !netif_running(ndev))
                return 0;

        netif_device_detach(ndev);
        vortex_down(ndev, 1);

        return 0;
}

static int vortex_resume(struct device *dev)
{
        struct pci_dev *pdev = to_pci_dev(dev);
        struct net_device *ndev = pci_get_drvdata(pdev);
        int err;

        if (!ndev || !netif_running(ndev))
                return 0;

        err = vortex_up(ndev);
        if (err)
                return err;

        netif_device_attach(ndev);

        return 0;
}
```

El método *vortex_suspend*, tal y como es empleado en el *driver* es llamado a modo de *callback*, en las trasnsiciones de sistema S*uspend-to-RAM* e *hibernation*. En ambos casos, el *driver* se comporta igual marcando el dispositivo como no disponible al sistema (*netif_device_detach*) para posteriormente llamar a código de bajo nivel que permita al dispositivo entrar en un estado de bajo consumo (*vortex_down*).

El código de ambos métodos, *netif_device_detach* y *vortex_down*, no es relevante en tanto es código específico para un determinado tipo de dispositivo (tarjeta de red) y hardware de un determinado fabricante (3Com). En otros dispositivos, la lógica incluída realizará una función similar interactuando con otros subsistemas (memoria, ttys, etc) con dependencias con el dispositivo.

El código *vortex_resume* es directo y complementario al código *vortex_suspend* anotado en líneas anteriores.

Por último, mencionar que el código fuente relacionado con gestión de energía es incluído de forma dinámica empleando una estrategia de preprocesamiento dinámico a través de las directivas condicionales *#ifdef* e *#ifndef* del preprocesador. Así, la presencia o ausencia de CONFIG_PM permite incluir o deshechar código de forma dinámica. Esto último se está utilizando en el *kernel* para dos objetivos principalmente: no incluir código del que no se haga uso e inicializar variables de forma condicional en base a la inclusión o no inclusión de código fuente en el proceso de compilación.

3.4 Contribución de parches para la nueva gestión de energía en el driver sm7xx de Linux

3.4.1 Soporte actual para el driver gráfico sm7xx en el kernel Linux

El *driver* gráfico sm7xx en el *kernel Linux* (*drivers/staging/sm7xx*) proporciona soporte a periféricos de la compañía *Silicon Motion Technology Corporation* (*NasdaqGS: SIMO*); en concreto, a la familia de controladores gráficos basados en la línea comercial de semiconductores denominada *Lynx*.

Este *driver* gráfico se introduce en el *kernel*, a través del subsistema de *staging*, a finales de Noviembre de 2009. Su inclusión, por parte de la empresa *Lemote, Inc.*, intenta que el código base originalmente desarrollado por *Silicon Motion Corportation* logre *upstream* en el *kernel* obteniendo un ciclo virtuoso de desarrollo entre la comunidad del *kernel* y la comunidad más modesta que *Lemote, Inc.* mantiene como parte de su estrategia para comercializar el netbook *Yeeloong* a nivel global.

En el momento de la inclusión del *driver* en el *kernel* éste es funcional. Si bien, gran número de funcionalidades descritas en la especificación técnica del fabricante no se encuentran implementadas, otras funcionalidades básicas de visualización, *timing* o aceleración 2D se encuentran presentes y permiten que el *driver* constituya el soporte oficial más estable en la comunidad *Linux* para los controladores gráficos sm7xx.

En Noviembre de 2009 también se realiza un análisis técnico superficial del estado actual del *driver* (*drivers/staging/sm7xx/TODO*) señalando el estado del mismo. El análisis señala algunas de las funcionalidades ausentes del *driver*, como el soporte gráfico dual, y hace evidente la necesidad de refinar el código así como adaptarlo a los estándares de calidad del *kernel Linux*. Una de las funcionalidades requeridas, no presentes en el *driver*, era también el soporte de configuración dinámica de modo gráfico para el *framebuffer*.

En Enero de 2010 el código base del *driver* sufre una regresión y pierde el soporte de aceleración 2D.

3.4.2 Desarrollo de mejoras y actualización del código base del driver

Como parte del actual proyecto, y como resultado del análisis previo y documentación en puntos anteriores, a lo largo de los meses de Marzo y Abril se desarrollan varias mejoras que actualizan el código base del *driver* sm7xx. Dichas mejoras son implementadas como parches, o diferencias de código fuente entre versiones software, para su posterior contribución al *kernel Linux*.

En concreto, cuatro parches de entre los desarrollados son relevantes en relación al trabajo que nos ocupa. Estos son identificados y enumerados a continuación:

Título

staging: sm7xx: fixed defines

Descripcion

Deleted redundant __KERNEL__ define

PM methods (suspend and resume) enabled under CONFIG_PM only

Signed-off-by: Javier M. Mellid <jmunhoz@igalia.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

Referencias

https://patchwork.kernel.org/patch/674271/

Título

staging: sm7xx: smtcfb.c: Use the new PCI PM

Descripcion

The sm7xx driver uses the legacy PCI power management (suspend and
resume) callbacks.

This patch adds the new PCI PM and let the PCI core code handles the
PCI-specific details of power transitions.

Tested in 2.6.38, including standby and hibernation support.

Tested-by: Wu Zhangjin <wuzhangjin@gmail.com>
Signed-off-by: Javier M. Mellid <jmunhoz@igalia.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

Referencias

https://patchwork.kernel.org/patch/743882/

Título

staging: sm7xx: minor cleanup

Descripcion

Sync code comments with TODO, fix some style and format issues

Signed-off-by: Javier M. Mellid <jmunhoz@igalia.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

Referencias

https://patchwork.kernel.org/patch/743872/

Título

staging: sm7xx: Use kernel framebuffer mode setting

Descripcion

This patch implements dynamic framebuffer mode setting.

Previous code works with mode setting in a hard code way. Previous hard
code configuration is used as default configuration if dynamic mode
setting or boot mode setting (via sm712vga_setup) is not used.

Tested with SM712 supporting 1024x600x16 as default hardware resolution.

Changes:

- Implement fb_check_var and fb_set_par callbacks
- Remove __maybe_unused decorator in function being used (sm712vga_setup)
- Minor cleanup on initialization structs related with mode settings
- Updated author copyright
- Updated TODO file

Signed-off-by: Javier M. Mellid <jmunhoz@igalia.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

Referencias

https://patchwork.kernel.org/patch/763682/

3.4.3 Contribución e integración de mejoras en el kernel Linux

La contribución e integración de los parches anteriores se hizo públicamente enviando los mismos a las listas de desarrollo del *kernel Linux*. En el momento de la contribución de los parches el *kernel* estable actual era la versión 2.6.38 con *release candidates* siendo emitidas para la versión 2.6.39 Esto tiene como consecuencias que los parches considerados correctivos entrasen dentro de la versión del *kernel* 2.6.39 mientras aquellos parches considerados mejoras entrasen dentro de la versión del *kernel* 3.0.0

En esta fase de integración del *kernel* los parches fueron aceptados en la rama *master* del repositorio y forman parte de las siguientes versiones del *kernel*:

Parche	Integrado en kernel	Id. commit
staging: sm7xx: fixed defines	2.6.39	392a002a0066812480e1b55639bbced5936d26aa
staging: sm7xx: smtcfb.c: Use the new PCI PM	3.0.0	59815677555746f8263672f25cebcf4c27fc5d31
staging: sm7xx: minor cleanup	3.0.0	3b70a26bcbe05db12965de702368ca0b9ec945f1
staging: sm7xx: Use kernel framebuffer mode setting	3.0.0	dc762c4f8514f23094927e0a62ef305d90651535

En resumen, la contribución de estos parches al *kernel* supone la mejora del soporte para el *driver* sm7xx en cuanto a migración de código para el uso del nuevo framework de gestión de energía desarrollado y contribuído por Rafael Wysocki, solución de diferentes problemas de integración y calidad con el código actual y aporte de una nueva funcionalidad para el cambio dinámico de modo empleando el *framebuffer*.

4. Gestión de energía y aplicaciones web

4.1 Granularidad y gestión de energía en servidores web

Dada la naturaleza de la web, un servidor web se encuentra típicamente diseñado para atender un pico de carga mucho más elevado que su carga media. Como consecuencia, el sistema tiene intervalos de tiempo significativos de baja utilización. Durante estos intervalos de baja utilización el sistema puede operar a frecuencias de CPU inferiores a las que habitualmente necesitaría para atender un pico de carga, ofreciendo así un consumo menor. Un comportamiento dinámico en el escalado de la frecuencia de la CPU, por tanto, permite reducir el consumo energético en su operación diaria.

Evidencias y estudios de lo anterior pueden ser encontrados en [Bohrer-2002], pág 4. donde podemos observar soluciones web para eventos significativos que operaron al 11% y al 25% de la capacidad de pico observada. En [Bohrer-2002] también se presenta un estudio sobre gestión de energía en servidores web donde se identificó el procesador como el componente de mayor impacto energético.

Una de las aproximaciones para reducir el consumo por parte de la CPU es reducir su voltaje operacional. Una reducción de voltaje generalmente también requiere una reducción proporcional en frecuencia. Este enfoque de variar el voltaje del procesador proporcionalmente a su frecuencia es conocido como *voltage scaling* o escalado de voltaje.

Voltage scaling presenta ventajas notables a causa de que la energía consumida por un procesador es directamente proporcional a V^2, donde V es el voltaje operacional. Actuando sobre el voltaje y la frecuencia de un procesador es posible, por tanto, obtener una reducción cuadrática en el consumo.

En [Bohrer-2002] se evalua la eficacia de escalar la frecuencia de la CPU de forma dinámica en el dominio de servidores web, validando los resultados obtenidos en base a hardware instrumentalizado y modelos de simulación que corroboran lo mencionado anteriormente.

Por tanto, podemos afirmar que es deseable que la computación de un algoritmo se realice a la menor frecuencia posible de CPU en un compromiso de rendimiento/consumo siempre que el principal objetivo sea obtener un consumo reducido y ajustado.

Al mismo tiempo, *GNU/Linux* proporciona una interface de gestión de escalado de las frecuencias de la CPU. Esta interface está modelada a través de abstracciones basadas en una serie de gobernadores de bajo nivel que actúan bajo una política o diseño predeterminado.

Estos gobernadores propios del sistema operativo tienen en cuenta los parámetros globales del sistema que actúan sobre la CPU (carga, tiempos de ejecución, latencias de E/S, etc) y actúan en base al trabajo que llevan a cabo sin discriminar su naturaleza, procedencia o incluso el interés funcional del mismo.

Lo anterior parece un enfoque adecuado para estaciones de trabajo donde la actividad del usuario es heterogénea e impredecible. En un servidor web, en cambio, el objetivo último de cada computadora que forma parte de la solución se encuentra predeterminado. Parece razonable que el

gobierno de un servidor web esté basado y dirigido por la funcionalidad específica que da sentido a su existencia, y que el consumo de energía se reduzca y ajuste a las necesidades reales de la misma. Es importante mencionar que soluciones de servicio web masivas conectadas permanentemente tiene un gran impacto energético.

El caso de estudio que se realiza en este punto, por tanto, trata de profundizar y conectar dos áreas de desarrollo actuales como son los servidores web y la Ingeniería de Software en aplicaciones web. Todo ello dentro del marco de gestión de energía que ofrecen los sistemas operativos basados en *GNU/Linux*.

En concreto, en este punto se profundizará sobre las abstracciones software necesarias que dan soporte a la toma de decisiones en cuanto a escalar la frecuencia de la CPU de forma dinámica respecto a la carga que sufre una aplicación web construída con principios de Ingeniería actuales y considerados como mejores prácticas dentro de la industria.

4.2 Arquitectura web modificada para gestión de energía

4.2.1 El patrón de diseño Modelo-Vista-Controlador

El patrón de diseño Modelo-Vista-Controlador[5] (MVC) es uno de los patrones más empleados en diseño y desarrollo de soluciones de interacción. En la actualidad es ampliamente usado en aplicaciones web gracias a la adopción y difusión del mismo realizado por la plataforma Java[6].

Las implementaciones de este patrón son variadas y habitualmente cuestionadas por expertos en el campo. En general, es comúnmente aceptado que el paradigma MVC es un marco flexible que permite dividir las distintas responsabilidades de una aplicación, o incluso una parte de la interface de la aplicación, en tres elementos bien definidos y delimitados: el modelo, la vista y el controlador.

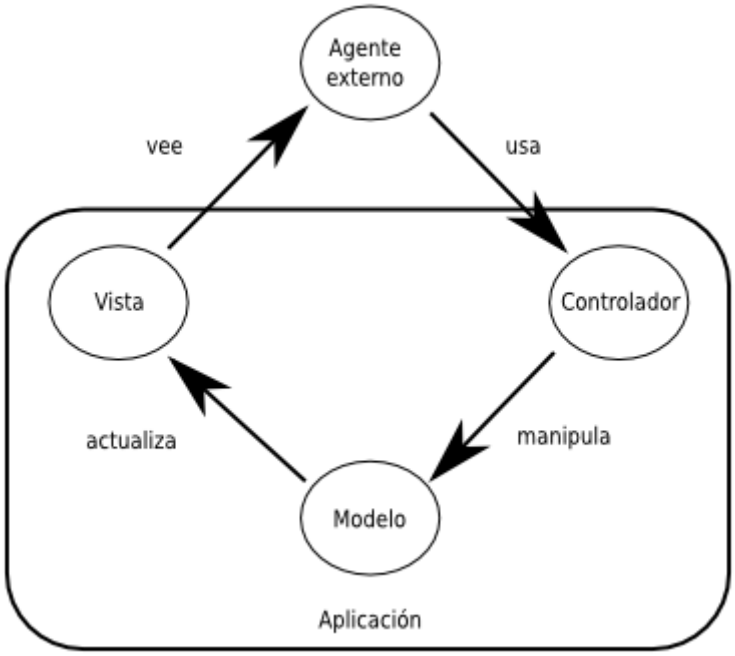


Figura 4. Patrón de diseño Modelo-Vista-Controlador

MVC fué originalmente desarrollado para asociar los tradicionales roles de entrada, procesado y salida en una interface gráfica de usuario. Para nuestro trabajo entendemos cada uno de los tres elementos identificados en el siguiente marco descriptivo:

Modelo

- El modelo es un objeto representando datos o incluso actividad
- El modelo gestiona el comportamiento y datos del dominio de la aplicación, responde a peticiones de información en relación a su estado al igual que a cambios de estado
- En el dominio de negocio, como marco de trabajo genérico y abstracto, el modelo representa los datos de la empresa y sus reglas de negocio. A menudo el modelo se emplea como una

5 http://en.wikipedia.org/wiki/Model-view-controller
6 http://en.wikipedia.org/wiki/Java_(software_platform)

abstracción software que permite capturar procesos del mundo real

- El modelo no tiene conocimiento específico de los elementos controlador o vista
- El modelo puede no utilizar necesariamente un almacen de persistencia como, por ejemplo, una base de datos.

Vista

- Una vista es un tipo de visualización del estado total o parcial del modelo
- Una vista gestiona cualquier información de salida, gráfica o textual, que necesita ser visualizada o representada

Controlador

- Un controlador ofrece facilidades para cambiar el estado de un modelo. El controlador interpreta una entrada y toma la responsabilidad del encaminamiento adecuado hacia el modelo y/o vista según proceda
- Un controlador es el medio por el cual un agente externo interactúa con la aplicación. Así, acepta una entrada e instruye al modelo y vista para llevar a cabo las acciones esperadas por esa entrada.
- Un controlador traduce interacciones con la vista en acciones para ser realizadas por el modelo. En una aplicación web estas acciones son peticiones HTTP, por ejemplo peticiones GET o POST.

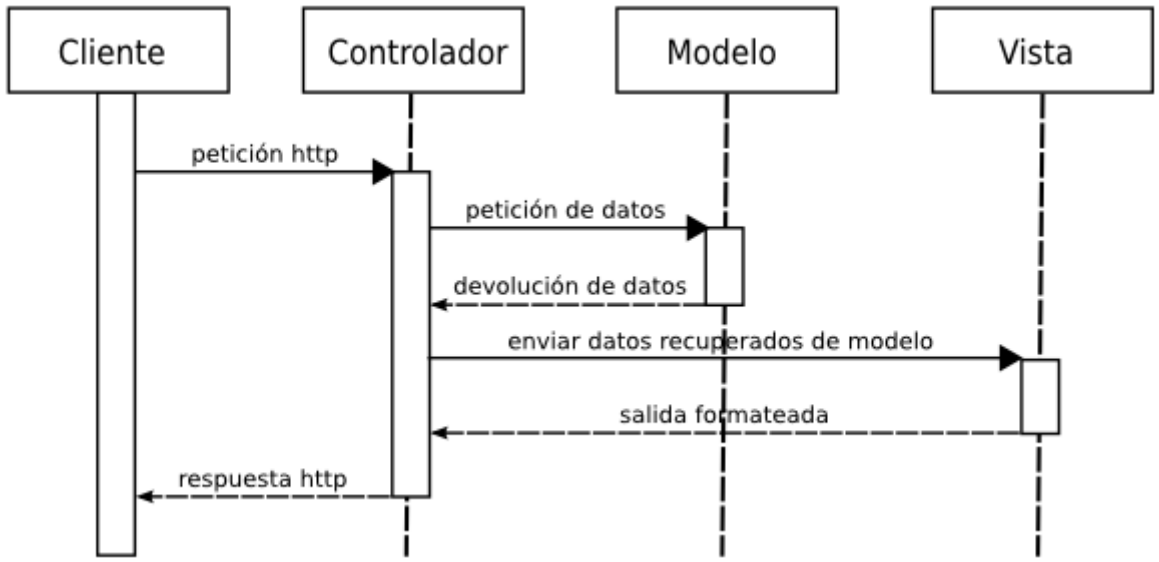


Figura 5. Diagrama de secuencia del patrón Modelo-Vista-Controlador

En el diagrama de la figura anterior podemos ver la secuencia habitual de interacción entre un cliente, o agente externo, y una aplicación web. En este diagrama se aprecia como el modelo es cohesivo y desacoplado en cuanto a que agrupa y aglutina la lógica de la aplicación frente a otros elementos más cercanos a la infraestructura como pueden ser el controlador o la vista.

Con este enfoque identificamos y aislamos en una aplicación web la lógica de negocio, el modelo,

de la infraestructura software que debe soportar la interacción. Es coherente decir por tanto, que el modelo presenta el conocimiento y la responsabilidad de aquello que es necesario realizar para satisfacer los requisitos funcionales de la aplicación. Supervisar e instruir al modelo es, por tanto, necesario para realizar una correcta gestión de energía sobre la aplicación web puesto que será en el modelo donde se encuentren las variables que afectaran al consumo real: carga espacial y temporal.

4.3.2 Arquitectura propuesta para la gestión de energía en aplicaciones web

La arquitectura propuesta para gestión y gobierno de aplicaciones web basadas en el patrón de diseño Modelo-Vista-Controlador se basa fundamentalmente en los siguientes dos puntos:

- Inclusión de soporte para la gestión de energía en la propia aplicación web con la intención de instrumentalizar el modelo en cuanto a su supervisión y control
- Empleo de un agente externo automático y síncrono que guía el gobierno de consumo de la aplicación web actuando sobre las facilidades que ofrece el Sistema Operativo.

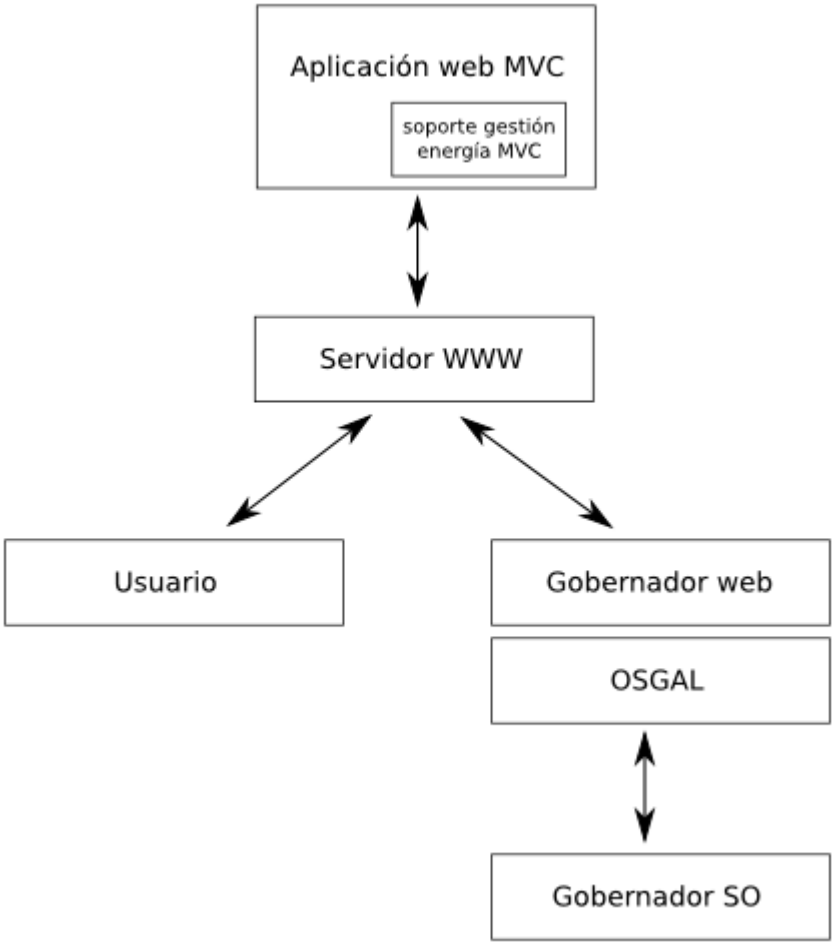


Figura 6. Arquitectura propuesta para gestión de energía en aplicaciones web MVC

En la figura anterior se representa la arquitectura extendida, que incluye las modificaciones y elementos propuestos con la intención de responder al comportamiento y naturaleza del modelo de la aplicación.

El prototipo desarrollado en el presente trabajo se ajustará a esta arquitectura en la medida de lo posible. Una descripción de sus elementos y responsabilidades dentro de esta arquitectura se realiza a continuación:

- Usuario. Un usuario representa el cliente funcional del sistema, habitualmente es un ser humano que interactúa con el sistema a través de un navegador web empleando el protocolo estándar HTTP o HTTPS.
- Servidor WWW. Un servidor web representa el programa que se ejecuta en el servidor y recibe las peticiones web del usuario. Su misión es interactuar con el navegador del cliente y mediar entre el mismo y la aplicación web que alberga.
- Aplicación web MVC. Representa el código interpretado o compilado que reside en el servidor web y que éste invocará como resultado de las diferentes peticiones y responsabilidades que debe satisfacer en relación a los usuarios. La aplicación es estructurada siguiendo el patrón MVC visto en el punto 4.2.1
- Soporte gestión energía MVC. Código autocontenido en la aplicación web MVC. Su responsabilidad es instrumentalizar el modelo, extendiéndolo con nuevos métodos y supervisando el comportamiento de los métodos y variables originales del mismo. En general, su integración con la aplicación debe ser mínimamente intrusiva y no tener ningún impacto en la lógica de la aplicación ni en el diseño de sus elementos.
- Gobernador web. En este elemento reside la responsabilidad del gobierno y gestión de energía de la solución propuesta. Para ello, el gobernador actúa síncrona y periódicamente a intervalos regulares con la intención de obtener información de consumo del modelo a través de los métodos extendidos por el código de soporte de gestión de energía MVC. Una vez obtenida esta información, y previa carga e interpretación de una política de gobierno, actuará sobre la interface de gobierno de frecuencia proporcionada por el sistema operativo. Esto último lo lleva a cabo empleando una capa intermedia de abstracción (OSGAL) que desacopla la interface del Sistema Operativo y facilita posibles implementaciones distribuídas. El gobernador web también puede presentar una interface para consulta del registro de estadísticas.
- Gobernador SO. Constituye la interface de gestión de frecuencias de la CPU. Es proporcionada por el sistema operativo y debe ser configurada para ser usada por el Gobernador web.

4.3 Desarrollo de un gobernador para aplicaciones web

4.3.1 Instrumentalización dinámica del patrón Modelo-Vista-Controlador en PHP5

En este punto se describe la aplicación web con soporte para gestión de energía. La aplicación web, implementada a modo de prototipo y como base de conocimiento sobre la que tomar decisiones de consumo, sigue el patrón Modelo-Vista-Controlador mencionado anteriormente.

Funcionalmente, la aplicación realiza cálculos de factorial y permite su instrumentalización; es decir, conocer el estado en el que se encuentra el cálculo de un determinado factorial en un instante determinado y el tiempo invertido en el cálculo del mismo una vez finalizado.

Este cálculo de factoriales, sin pérdida de generalidad en cuanto a la funcionalidad de cualquier Modelo, es interesante ya que permite que la CPU disponga de tiempos de espera mínimos en las computaciones. Esto último es algo notable para facilitar la evaluación del prototipo.

El Modelo, como mencionábamos anteriormente, encapsula la lógica de cálculo para la obtención del factorial de un número. Por motivos prácticos e ilustrativos, en la implementación es preferible implementar tres métodos de cálculo con un cuerpo de código similar que iteran 100 000 veces para tres diferentes valores (10, 50 y 100).

Lo anterior se lleva a cabo por razones prácticas. Así, la implementación de la infraestructura de encaminamiento permanece sencilla, no se sobrecarga la implementación con el paso y comprobación de parámetros, se obtienen salidas válidas sin generar desbordamientos aritméticos y se dota al modelo de tres métodos de negocio sobre los que aplicar técnicas dinámicas de instrumentalización.

La implementación del Modelo es, por tanto, una implementación práctica y sencilla que simula y ofrece elementos de interés sobre los que trabajar las ideas del presente proyecto.

En base a lo anterior, sobre esta aplicación web se tomó la decisión de implementar un bloque de código que permitiese instrumentalizar el Modelo. Básicamente, las responsabilidades de este código son las siguientes:

- Modificar el elemento Controlador para encaminar acciones propias de gestión de energía. En concreto, las acciones *show_status* y *reset_status* que actúan sobre el estado de consumo de energía del Modelo.
- Modificar el elemento Controlador para, a través de la implementación de una clase implementando el patrón adaptador[7] y *proxy*[8] instrumentalizar la clase Modelo original sin que ésta tenga conocimiento de esto último.
- Añadir una nueva clase ModeloPM que proporciona contabilidad a las llamadas de los métodos originales del Modelo de forma dinámica, al mismo tiempo que implementa los métodos extendidos de gestión de energía y gestiona su concurrencia.
- Añadir una nueva vista compartida para el soporte de visualización en la información de

7 http://en.wikipedia.org/wiki/Adapter_pattern
8 http://en.wikipedia.org/wiki/Proxy_pattern

salida de las nuevas acciones *show_status* y *reset_status*

En resumen, la aplicación web con soporte de gestión de energía permite el cálculo de tres factoriales (10, 50 y 100) calculados 100 000 iteraciones antes de devolver el resultado. Cada petición web que finaliza en la llamada a uno de los métodos de cálculo contenidos en el Modelo es instrumentalizada para conocer si la petición se ha realizado, si ha logrado computarse y el tiempo que ha empleado en este último paso. Toda esta información de instrumentalización forma parte del estado de gestión de energía de la aplicación web que es consultado y, en caso de ser necesario, reinicializado a través de los métodos *show_status* y *reset_status*.

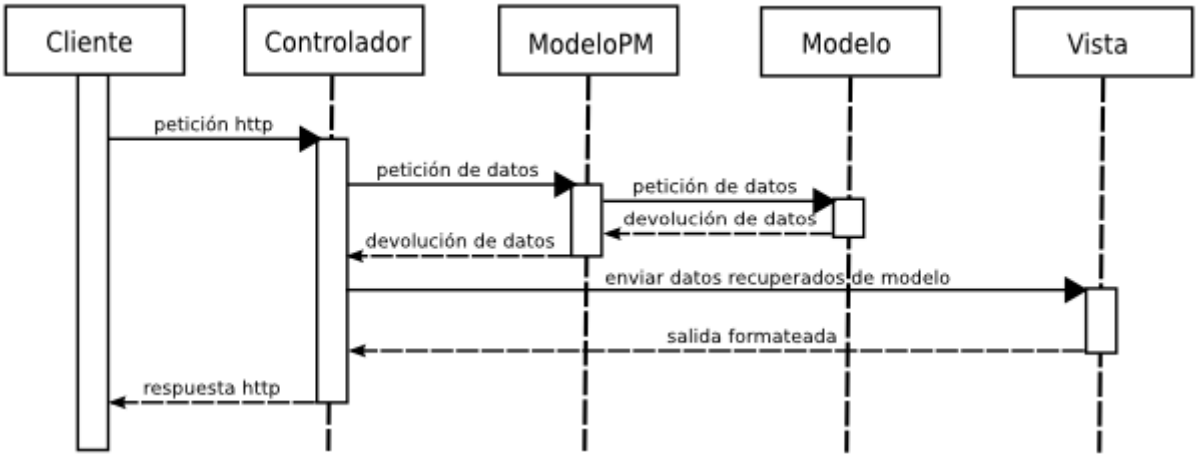


Figura 7. Diagrama de secuencia del patrón Modelo-Vista-Controlador modificado

En la Figura 7 se observa el rol mediador que desarrolla el elemento ModeloPM supervisando, de forma dinámica, las llamadas a los métodos del elemento Modelo y realizando tareas de contabilidad de consumo para su posterior consulta o modificación empleando los mismos canales de comunicación e interface que el Modelo original.

La inclusión del elemento ModeloPM, con este enfoque, presenta enormes ventajas de cara al diseño y desarrollo de la solución web. No siendo invasiva en cuanto a que hace uso de la infraestructura de comunicación e interfaces disponibles, al mismo tiempo que permanece independiente a las modificaciones de diseño e implementación que pueden suceder en un futuro dentro del Modelo. Es decir, el Modelo puede sufrir cualquier tipo de modificación en relación a sus métodos (nuevos métodos, cambio en sus firmas, sobrecarga orientada a objetos, eliminación de métodos, etc) sin que el ModeloPM sea necesario modificarlo. El desarrollador de la aplicación web por tanto, es ajeno a la existencia del soporte de gestión de energía en cuanto a la extensión y modificación del Modelo se refiere.

En relación a la interface que presenta la aplicación web modificada. Un usuario puede solicitar el cálculo del factorial para el valor 10, 50 y 100 sin ser consciente de que sus peticiones están siendo instrumentalizadas y monitorizadas.

El acceso a la información de gestión de energía se lleva a cabo mediante dos consultas directas al servidor web. Estas consultas son las siguientes:

http://localhost/index.php?pm=show_status

http://localhost/index.php?pm=reset_status

La primera consulta, *show_status*, devuelve en la pantalla del navegador una cadena similar a la siguiente:

{ result : OK, getFact_10 : {2,2,0.4375}, getFact_50 : {3,3,1.34375}, getFact_100 : {12,4,2.98046875} }

La cadena está codificada. En el ejemplo, la cadena nos informa que la petición de estado de gestión de energía ha sido correcta. Nos hace saber también que actualmente el Modelo dispone de 3 métodos (getFact_10, getFact_50 y getFact_100) junto a información del número de veces que se ha solicitado el método, el número de veces que se ha computado con éxito y la tendencia temporal de ejecución del mismo en segundos.

A modo de ejemplo, podemos observar que el método getFact_100 se ha solicitado 12 veces, de las cuales 4 veces ha finalizado y su tendencia temporal de ejecución está próxima a los 3 segundos.

La segunda consulta, *reset_status*, nunca falla. El objetivo de la misma es establecer al valor 0 todos los contadores de estado con los que se supervisa el modelo. A modo de ejemplo, el valor de la cadena devuelta es el siguiente:

{ result : OK }

En cuanto al despliegue, el prototipo asume que su localización será la raíz del servidor web. Copiando el código a la raíz del servidor e invocando la consulta http://localhost/index.php es el único paso necesario para un correcto despliegue de la aplicación web.

4.3.2 Implementación y despliegue del gobernador web en Python

Para la implementación del gobernador web se decidió utilizar el lenguaje Python[9], dada su flexibilidad y velocidad de desarrollo en la implementación de prototipos. En el diagrama de responsabilidades mostrado en la Figura 8 se muestra el conjunto de módulos implementados, así como sus dependencias.

La implementación está constituída por 10 módulos. En el diagrama, una línea horizontal indica el grado de reusabilidad que muestra cada componente. Así, los módulos situados arriba de la misma son potencialmente reutilizables en otros sistemas operativos, mientras que los módulos situados por debajo presentan dependencias duras con las interfaces del sistema operativo y necesitarían reescribirse o adaptarse para su uso en otros sistemas operativos.

El módulo *gobweb.py* es el módulo principal. Este módulo dirige la lógica del goberandor. Sus principales funciones son muestrear los datos necesarios de las distintas fuentes de información (CPU, ACPI y MVC) para, a continuación, salvar la muestra a base de datos. Posterior a dicha recopilación de datos el algoritmo delega en *policy.py* la selección y comportamiento de la política de gobierno para, una vez decidida, ejecutarla.

El módulo *policy.py* es el módulo que implementa la política de gestión de energía. Es el

9 http://www.python.org/

responsable de interpretar la carga de trabajo del Modelo, así como seleccionar la próxima acción de gobierno. Su implementación es genérica. Entre *gobweb.py* y *policy.py* existe un contrato ligero en el que *policy.py* toma decisiones genéricas y *gobweb.py* las lleva a cabo.

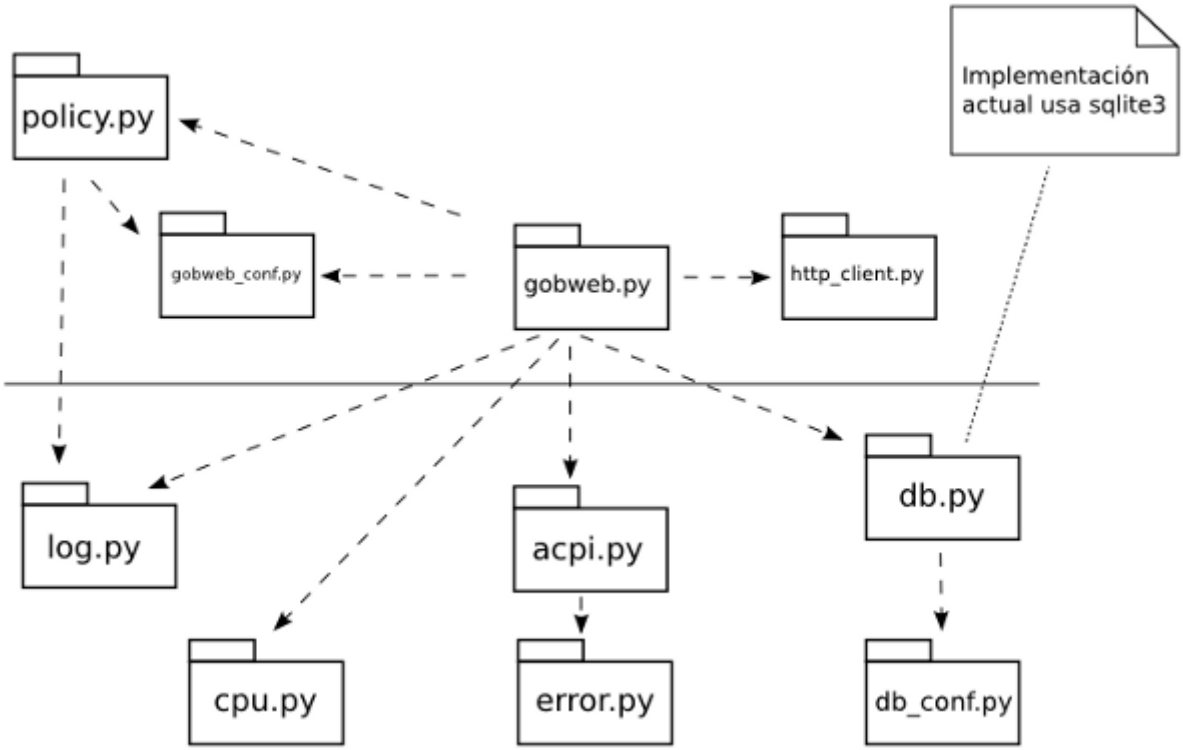


Figura 8. Implementación gobernador web

gobweb_conf.py y *db_conf.py* son módulos de configuración. El primero hace alusión a la configuración general del gobernador mientras el segundo contiene información de configuración propia de la base de datos.

El acceso y abstracción a la base de datos es realizada por el módulo *db.py*. La base de datos usada en la implementación actual es sqlite3[10].

cpu.py, *acpi.py* y *http_client.py* son módulos consumidores de datos. Abstraen el acceso a las fuentes de información de CPU, batería y Modelo principalmente.

log.py y *error.py* son módulos de registro y comunicación. El primero permite visualizar contenido mientras el segundo informa de errores y aborta la ejecución.

Funcionalmente, el gobernador web tiene las siguientes tres responsabilidades:

- acceder de forma periódica y registrar información útil en memoria no volátil
- delegar en la política vigente la próxima acción de gobierno a tomar en base a la información actual
- llevar a cabo la acción de gobierno seleccionada, traduciéndola a una posible selección de nueva frecuencia operativa para la CPU

10 http://www.sqlite.org/

Lo anterior puede traducirse en que el gobernador permite realizar una selección dinámica de las frecuencias operativas mientras se está ejecutando en base a la información que muestrea regularmente. Posteriormente, y en caso de interés, una base de datos contiene el registro de toda la información empleada en el intervalo de gobierno. La base da datos sqlite3 permite una inspección conforme al estándar SQL facilitando así cualquier futuro análisis o consulta de los datos empleados.

En relación al despliegue del gobernador web éste no presenta mayores dependencias que aquellas producidas por el lenguaje Python 2.6.6 y sqlite3. La implementación se desarrolló para que el software fuese configurable a través de los módulos que contienen el sufijo _conf.

La ejecución del gobernador se lleva a cabo invocando la línea de comando *python gobweb.py* en el shell.

4.3.3 Infraestructura hardware

La infraestructura hardware empleada para el desarrollo y evaluación del prototipo de gobernador web está formada por dos portátiles conectados directamente por Gigabit Ethernet.

Figura 8. Arquitectura Cliente-Servidor a través de conexión GigaE directa

El portátil actuando como servidor es un Thinkpad x201 basado en Intel(R) Core(TM) i7 CPU M620 @ 2.67GHz. El sistema expone 4 *cores*. La memoria RAM instalada es de 8G con un controlador Ethernet basado en Intel 82577LM.

El portátil actuando como cliente es un Thinkpad x61s basado en Intel(R) Core(TM)2 Duo CPU L7500 @ 1.60GHz. El sistema expone 2 *cores*. La memoria RAM instalada es de 4G con un controlador Ethernet basado en Intel 82566MM.

Se considera que los requisitos de disco son despreciables en cuanto a la necesidad de almacenamiento o latencia de acceso. La evaluación del prototipo se centra en la importancia de la variable frecuencia de la CPU frente a consumo. El parámetro fundamental a evaluar son las diferentes frecuencias utilizadas así como el tiempo consumido en una computación. Se tratará por tanto de minimizar operaciones de E/S e interrupciones, salvo las mínimas necesarias debidas a gestión de concurrencia, en un escenario donde se intentará que la CPU ejecute continuamente trabajo útil sin tiempos de espera.

En relación al uso de portátiles frente a servidores o *desktops* más potentes mencionar que la elección de los primeros se debe a la necesidad de optar por una solución de captura de consumo y energía flexible y económica sin la necesidad de instrumentalizar a nivel eléctrico los sistemas. Así, ejecutando la evaluación y siendo la fuente de energía la batería para el portátil denominado

servidor, podemos obtener una medida de la caída porcentual de potencia realizando un muestreo a intervalos conocidos a través de ACPI. Una lectura de ejemplo es mostrada a continuación:

~servidor$ acpi -b
Battery 0: Discharging, 99%, 08:06:27 remaining

4.3.4 Software instalado y entorno de desarrollo

Para el desarrollo del prototipo software se decidió emplear la distribución *Debian GNU/Linux 6.0 Squeeze*. Dicha distribución se desplegó por defecto en ambos equipos para posteriormente pasar a instalar y configurar software de forma personalizada.

En el equipo servidor se instaló o actualizó software, a través del gestor de paquetes APT, a las siguientes versiones:

Paquete	Versión
apache2-mpm-prefork	*2.2.16-6*
apache2-utils	*2.2.16-6*
apache2.2-common	*2.2.16-6*
libapache2-mod-php5	*5.3.3-7*
libonig2	*2 5.9.1-1*
libqdbm14	*1.8.77-4*
php5	*5.3.3-7*
php5-cli	*5.3.3-7*
php5-common	*5.3.3-7*
php5-suhosin	*0.9.32.1-1*
sqlite3	*3.7.3-1*
python-sqlite	*1.0.1-7+b1*
libsqlite0	*2.8.17-6*

La instalación de los paquetes anteriores a través del gestor de paquetes despliegan el servidor web Apache, el lenguaje interpretado PHP5 y el módulo de integración entre el servidor y el intérprete sin necesidad de interacción por parte del usuario. Dicha instalación contiene la configuración en la ruta */etc/apache2*, y la raíz del servidor web desde donde se servirá el contenido en */var/www*

La instalación y configuración del software se validó incluyendo un fichero denominado *info.php* en la ruta */var/www* que incluía las siguientes líneas:

```
<html>
        <head>
                <title> PHP Test Script </title>
        </head>
        <body>
                <?php phpinfo(); ?>
        </body>
</html>
```

La salida del fichero, consumida a través de red vía interface HTTP, permite comprobar que la función *phpinfo()* es evaluada y traducida correctamente mostrando diferente información de

configuración y estado en cuanto al servidor web, el lenguaje PHP y los módulos de soporte e integración del mismo.

A continuación se descargó la versión estable *Linux* 2.6.38.5. Utilizando el fichero de configuración del *kernel* que viene con la distribución Debian *GNU/Linux* como base, se personalizó un nuevo *kernel* con todas aquellas opciones necesarias. En concreto, en "Power management and ACPI options" se activaron las siguientes opciones:

- Power Management support (CONFIG_PM)
- Suspend to RAM and standby (CONFIG_SUSPEND)
- Hibernation (aka 'suspend to disk') (CONFIG_HIBERNATION)
- Run-time PM core functionality (CONFIG_PM_RUNTIME)
- ACPI (Advanced Configuration and Power Interface) Support (CONFIG_ACPI) [*]
- CPU Frequency scaling (CONFIG_CPU_FREQ) [*]
- CPU idle PM support (CONFIG_CPU_IDLE)
- Cpuidle Driver for Intel for Intel Processors (CONFIG_INTEL_IDLE)

En las opciones marcadas con '*' los valores por defecto que presentan los submenus son adecuados y no es necesario modificar ninguna opción o configuración.

Una vez configurado el *kernel* pasamos a compilarlo e instalarlo. Bien empleando el sistema tradicional[11] o las facilidades[12] de la propia distribución.

Tras la instalación y configuración del software anterior obtenemos un sistema *GNU/Linux* con un servidor web interpretando el lenguaje PHP5 y soporte en el *kernel* para gestión de energía. En concreto, debajo de la ruta */sys/devices/system/cpu* podemos encontrar los 4 *cores* que presenta este modelo de CPU, así como los módulos de gestión de energía relacionados con la CPU (cpufreq y cpuidle).

Para cada *core* existirá una ruta con ficheros virtuales para solicitar o establecer información de frecuencia para dicho *core*. Así para el *core* 0 y gobernador *ondemand*, tenemos el siguiente listado:

servidor~$ ls /sys/devices/system/cpu/cpu0/cpufreq
affected_cpus
cpuinfo_cur_freq
cpuinfo_max_freq
cpuinfo_min_freq
cpuinfo_transition_latency
ondemand
related_cpus
scaling_available_frequencies
scaling_available_governors
scaling_cur_freq
scaling_driver
scaling_governor
scaling_max_freq
scaling_min_freq

11 http://www.sysdesign.ca/guides/linux_kernel.html
12 http://www.debian.org/releases/stable/i386/ch08s06.html.en

scaling_setspeed
stats

En el listado, salvo los directorios virtuales *ondemand* y *stats*, el resto son ficheros virtuales cuyo nombre es autoexplicativo. Así, *scaling_setspeed* permite establecer una determinada frecuencia para este *core* mientras que *scaling_cur_freq* informa de la frecuencia actual a la que opera dicho *core*.

Para nuestro hardware las frecuencias de operación permitidas son 13:

servidor~$ cat scaling_available_frequencies
2667000 2666000 2533000 2399000 2266000 2133000
1999000 1866000 1733000 1599000 1466000 1333000
1199000

Los gobernadores de frecuencia que hemos decidido soportar en el *kernel* son 5:

$ cat scaling_available_governors
userspace powersave conservative ondemand performance

Si bien, el que muestra más interés a la hora de permitir cambiar la frecuencia operativa del *core* por una aplicación es *userspace*.

Así, en el transcurso del desarrollo del prototipo el gobernador web podrá seleccionar entre 13 frecuencias operativas para cada uno de los 4 *cores* presentes en la CPU. El cambio de frecuencia, por tanto, se hará por *core* a través del fichero *scaling_setspeed* habiendo previamente establecido el gobernador a *userspace* en el fichero virtual *scaling_governor*.

Mencionar que los últimos paquetes hacen referencia a las base de datos ligera sqlite en su versión 3 así como las librerías de uso por parte de Python. Estos paquetes son empleados en el código del gobernador web.

4.4 Evaluación de un gobernador para aplicaciones web

4.4.1 Estrategia de evaluación

La estrategia de evaluación del gobernador web que se presenta en este trabajo persigue como resultado final verificar que el gobierno de las frecuencias operativas de la CPU es el adecuado, para así lograr un consumo de energía en el modelo lo más reducido posible. Como hemos comentado en el punto 4.1, frecuencias operativas de CPU bajas implican un menor consumo que frecuencias operativas más elevadas; y es por esto último, que la frecuencia de trabajo del modelo debería ser la mínima posible en un compromiso entre rendimiento y consumo.

Cuando hablamos de gobierno estamos hablando de la toma y ejecución de acciones que suponen un cambio en la frecuencia operativa de la CPU. Dichas acciones están asociadas a la información que obtenemos a través de un muestreo continuado del modelo de la aplicación web.

En dicho muestreo, y para nuestra implementación particular, la información recuperada consta de tres variables asociadas a cada uno de los métodos disponibles en el modelo. Las tres variables, que son objeto de análisis en este punto, constituyen la información disponible que el gobernador puede utilizar para ayudarle a tomar una decisión respecto a la frecuencia operativa más apropiada en cada instante temporal. Comprender su significado y evolución, se hace necesario; para así poder anticipar y reaccionar ante estados de carga no deseados mediante la modificación de las frecuencias operativas de la CPU.

Estas variables, son las siguientes:

- Número de veces que se ha iniciado la ejecución de un método del modelo. Contiene el número de veces que el método ha tomado el control. Este valor es diferente que el número de peticiones web recibidas por el método. Por ejemplo, en un escenario de *stress* el servidor web puede encolar más peticiones de las que podría procesar en un futuro. En este caso la petición puede ser anulada por el propio servidor web antes de ejecutar el método del modelo.

- Número de veces que ha finalizado la ejecución de un método del modelo. Habitualmente una página web recibe una entrada, la procesa y ofrece un resultado muy rápidamente. En situaciones de *stress* el servidor web puede decidir abortar el proceso de una página si ésta presenta un tiempo elevado. El valor, por tanto, registra el final de la computación de un método si éste logra terminar.

- Tiempo promedio que tarda en ejecutarse un método. Cada vez que un método es invocado, y éste termina, el tiempo de ejecución es calculado. Este valor representa una tendencia temporal ponderada con la intención de que ejecuciones anómalas del método no influyan en la historia temporal del mismo. Este valor puede interpretarse como el tiempo empleado en las ejecuciones pasadas, incluída la actual, de un método determinado.

Teniendo en cuenta lo anterior, la evaluación del gobernador web comienza con el estudio detenido de estas tres variables. De tal manera que la salida de este estudio, recogido en los puntos 4.4.2 y 4.4.3, es la base que inspira la actual implementación de la política del gobernador web que

acompaña este trabajo.

Una vez obtenida esta propuesta de implementación, la explicación y evaluación de la misma se hace en base a la simulación de un escenario de carga representativo. Es importante mencionar que en el punto 4.4.4 se recoge una explicación y evaluación detallada de este segundo y último paso en la evaluación del gobernador web.

Se busca por tanto, con esta evaluación, identificar variables candidatas a trabajar como heurísticas de comportamiento para el modelo; junto a una caracterización y validación del comportamiento del modelo ante situaciones de *stress* computacional.

4.4.2 Evolución temporal en los tiempos de respuesta de un método

Anterior a obtener una evolución temporal en los tiempos de respuesta para un método del modelo medimos el tiempo que se emplea en la ejecución de los distintos métodos del modelo a mínima y máxima frecuencia. La siguiente figura ilustra las medidas.

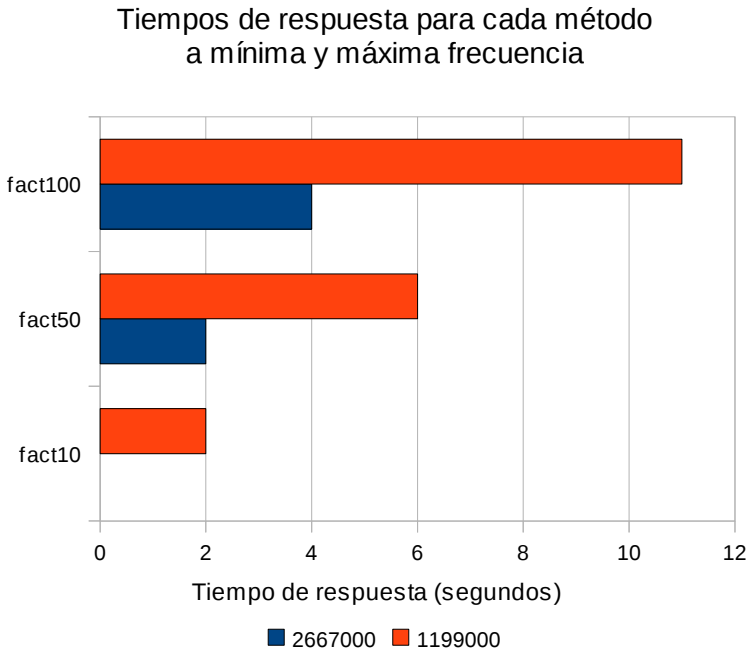


Figura 9. Tiempos de respuesta en segundos para cada método a mínima y máxima frecuencia

Vemos que el orden de magnitud de trabajo es de segundos para la totalidad de los métodos. Este orden es esperado y fué contemplado en el diseño para trabajar cómodamente con diferentes cargas de *stress* y sus tiempos de ejecución asociados.

Podemos validar también que los tiempos de ejecución son coherentes. Tanto a máxima como a mínima frecuencia la carga temporal va aumentando progresivamente, tal y como se esperaba.

La tasa max freq / min freq es aproximadamente 2.22. En un escenario ideal esto permite anticipar que actualizando de mínima a máxima frecuencia la CPU el rendimiento debería doblarse. Vemos que esto es consistente con los datos mostrados en los que los métodos son intensivos en instrucciones aritmético-lógicas manteniendo la CPU la mayor parte del tiempo ocupada, sin

tiempos de espera notables debidos a intervalos de E/S.

Otro aspecto a tener en cuenta que revela la figura es que la frecuencia de operación de la CPU es aproximada y variable. Así, en la evaluación lo que consideramos máxima y mínima frecuencia como notificadas por el propio hardware son frecuencias de trabajo aproximadas pudiendo operar a mayor y menor frecuencia que la establecida en base a requisitos y necesidades de diseño hardware de la CPU. Es importante mencionar también que en una CPU *multi-core* todos los *cores* necesitan sincronizar y notificar cambios de frecuencia con lo cual la frecuencia de operación no sólo es aproximada a la establecida sino también debe ser acordada en *quorum* por todos los *cores*.

Con esta información podemos asumir, sin pérdida de generalidad, que las conclusiones obtenidas en el estudio temporal de uno de los métodos del modelo serán aplicables al resto al tener todos una naturaleza computacional similar.

La evolución temporal en tiempos de respuesta será obtenida para el método fact50 operando a máxima frecuencia. En esta configuración el tiempo medio de ejecución del método es de 0.5 segundos; es decir, un cliente web tendría que esperar del orden de medio segundo para obtener el resultado del factorial una vez solicitado. La carga de *stress* con la que trataremos de evaluar este método es el escenario en el que 3 usuarios solicitan el cálculo del factorial cada segundo a lo largo de 60 segundos tal y como se muestra en la siguiente figura.

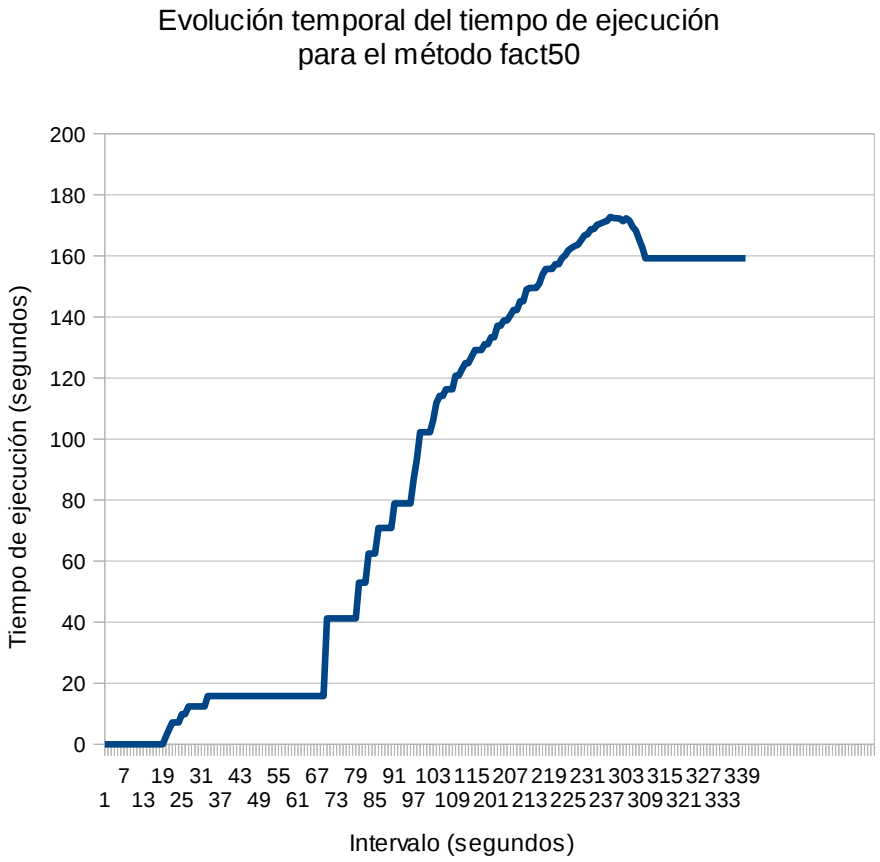


Figura 10. Evolución temporal del tiempo de respuesta en segundos (fact50)

En la figura podemos comprobar que la tasa de llegada de peticiones provoca congestión en la ejecución del método. Comprobamos también que los tiempos de ejecución del método van aumentando. Estos nuevos tiempos más elevados son debidos principalmente al tiempo de espera por cambio de contexto en la CPU. Es decir, cada método debe esperar y compartir la CPU con el resto de llamadas que se están produciendo y esto implica aumentar sus tiempos de respuesta significativamente.

Observamos también que durante el primer minuto la congestión es máxima y, al no finalizar ningún método las lecturas de tiempo se estabilizan en el orden de los 18 segundos. Esto es señal de máxima congestión.

Posteriormente, aparece una vertical pronunciada de 18 a 42 segundos en tiempo de respuesta motivada por la finalización de nuevos métodos con tiempos medios muy superiores a los 18 segundos.

A partir del minuto la tasa de llegada de peticiones cae a 0 y no se realizan, por tanto, nuevas peticiones; esto provoca que la aplicación web se recupera progresivamente si bien la ejecución completa de todas las llamadas realizadas a métodos se extienden a lo largo de un total de 5 minutos.

Podemos ver a continuación la evolución en el tiempo de ejecución del método, en azul, frente al diferencial de los tiempos de ejecución a lo largo de su evolución, en naranja.

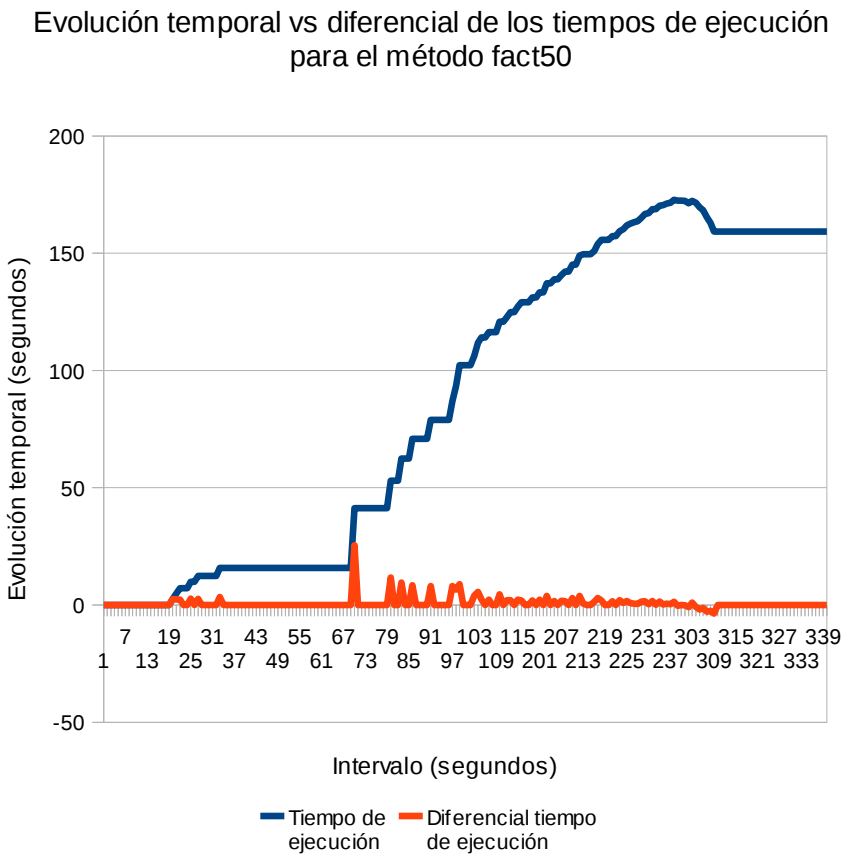


Figura 11. Evolución temporal vs diferencial de los tiempos de ejecución (fact50)

Concluímos, por tanto, que el diferencial en la evolución de tiempos que tiene lugar en la ejecución de un método puede ser empleado como heurística para detectar congestión e inducir cambios en la frecuencia operativa de la CPU.

Se proponen por tanto las siguientes acciones en la implementación de un gobernador web para tratar de detectar y actuar estados no deseables:

- Un diferencial temporal nulo continuado, en presencia de nuevas llamadas al método, es señal de gran congestión. Seleccionamos la frecuencia de operación máxima para la CPU.
- Un diferencial temporal nulo continuado, en ausencia de nuevas llamadas al método, es señal de reposo. Seleccionamos la frecuencia de operación mínima para la CPU.
- Un diferencial temporal positivo continuado es señal de aumento o presencia de trabajo. Seleccionamos una frecuencia de operación inmediatamente superior a la actual para la CPU.
- Un diferencial temporal negativo continuado es señal de disminución o ausencia de trabajo. Seleccionamos una frecuencia de operación inmediatamente inferior a la actual para la CPU.

Por último es necesario establecer un umbral temporal que matiza el concepto de diferencial continuado. Este umbral, es crítico, ya que debe superar el rizado que se produce en el diferencial debido a la ponderación de tiempos de ejecución para métodos que han finalmente sido ejecutados en tiempos notablemente diferentes. Un valor elevado de este umbral disminuye la sensibilidad del gobernador para establecer una nueva frecuencia mientras que un valor mínimo provoca que el gobernador sea muy sensible en el seguimiento de la carga consumiendo energía extra asociada al propio cambio de frecuencia en la CPU. Este umbral temporal es configurable por el administrador.

4.4.3 Evolución temporal en las llamadas a un método

Como se ha mencionado anteriormente el gobernador dispone de dos variables que le permiten conocer el número de invocaciones a método que se han iniciado junto al número de invocaciones a método que han terminado. Así, si una computación es abortada bien por un exceso de tiempo o por cualquier otra situación en la que el servidor web tome esa decisión ambos valores diferiran indicando que un método no terminó de ejecutarse.

En la siguiente figura, en azul, vemos la progresión temporal del número de métodos iniciados por segundo. Observamos que durante el primer minuto el patrón de crecimiento es el esperado; 3 usuarios cada segundo durante un periodo de 60 segundos.

En naranja se aprecia la progresión temporal del número de métodos que concluyen por segundo. Observamos que durante la congestión inicial la progresión es reducida y sólo el crecimiento es destacable a partir del minuto 3.

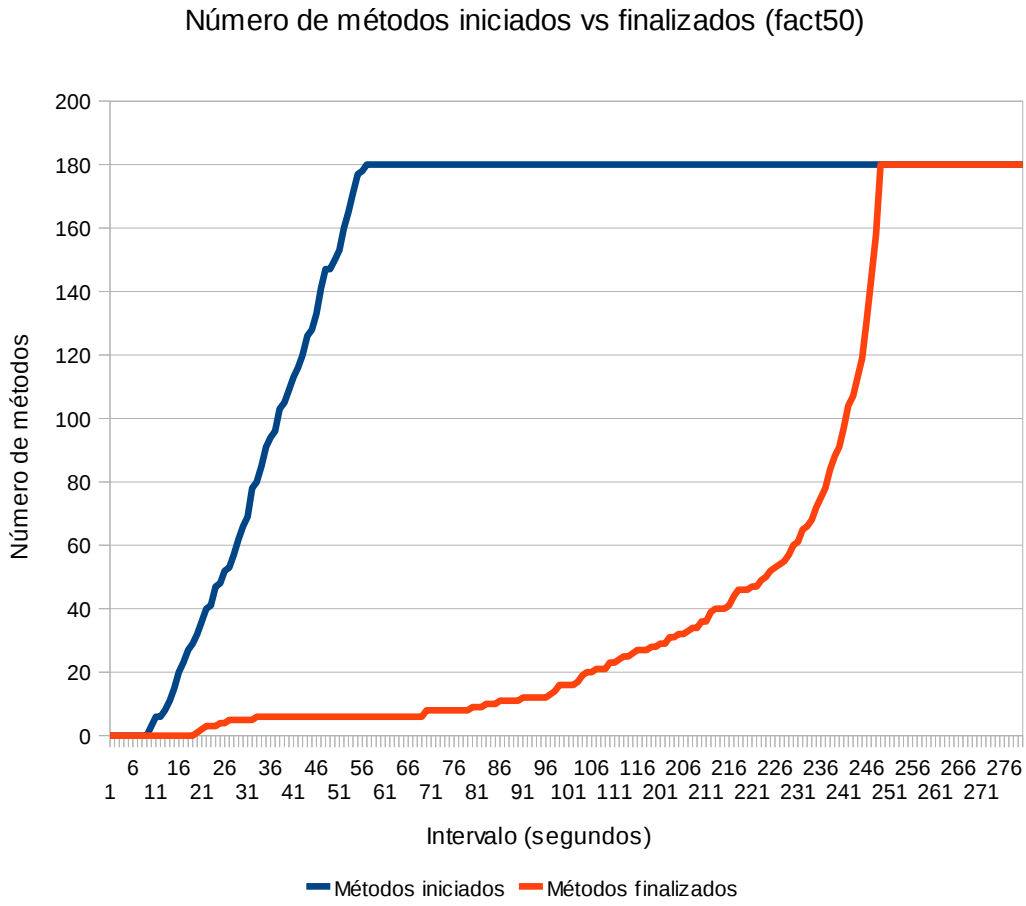


Figura 12. Número de métodos iniciados vs finalizados por segundo (fact50)

La misma información puede visualizarse como el diferencial instantáneo para cada de las dos variables. Esta información nos permite observar más en detalle la variabilidad de las mismas. En la siguiente figura se muestra esta información.

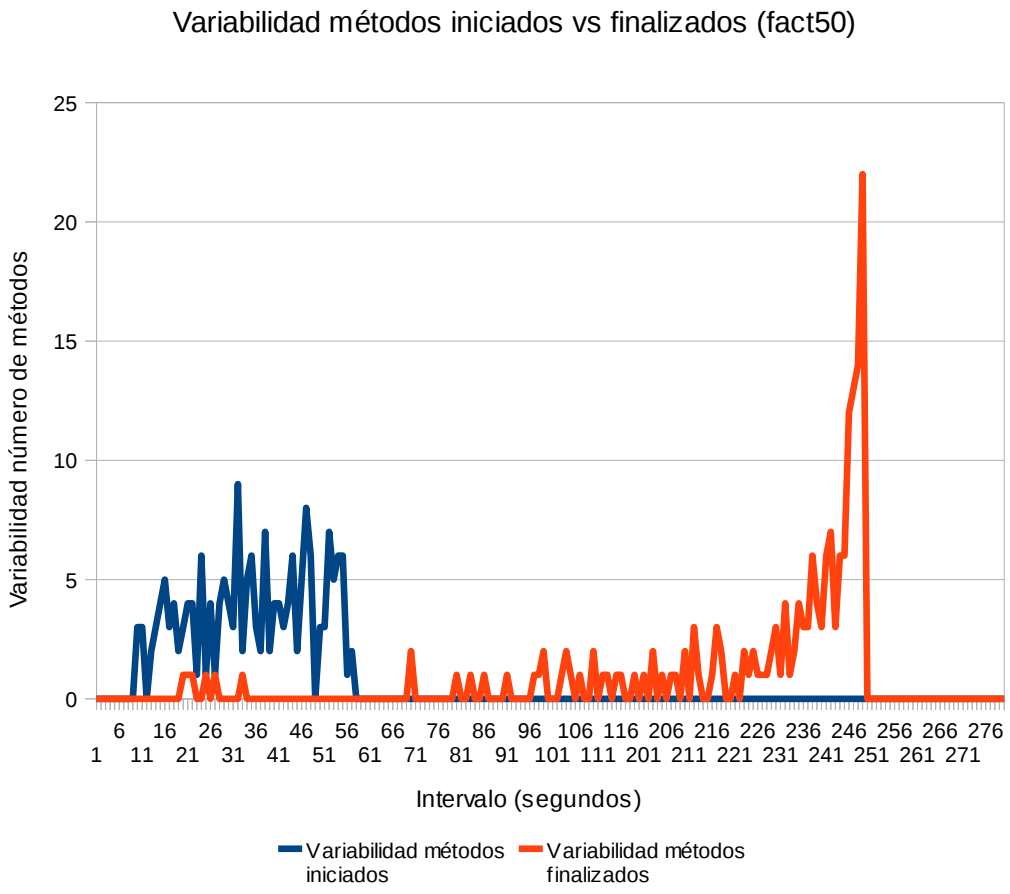


Figura 13. Variabilidad en el número de métodos iniciados vs finalizados por segundo (fact50)

Observamos que el patrón de variabilidad se corresponde con el escenario de *stress* que buscábamos. Así se observa gran congestión en el primer minuto, en azul, y como la aplicación se comporta en esta situación. Al mismo tiempo observamos la variabilidad de la aplicación mientras ésta se recupera y la congestión se reduce.

Con esta información se presenta también, en la Figura 14, el diferencial de métodos iniciados frente a finalizados por segundo (color amarillo). En esta gráfica podemos apreciar que en situación de reposo o carga mínima el valor del diferencial es nulo, mientras que ante una congestión elevada el diferencial es máximo.

Observamos, por tanto, que este diferencial entre el número de metodos iniciados y finalizados por segundo; puede ser empleado como heurística de gobierno. Así, las siguientes reglas de gobierno pueden ser consideradas en la implementación de un gobernador web:

- Ante un valor nulo continuado del diferencial se opera en la frecuencia operativa mínima de la CPU

- Ante un valor máximo continuado del diferencial se opera en la frecuencia operativa máxima de la CPU
- Ante un aumento del diferencial se opera en la frecuencia operativa de la CPU inmediatamente superior a la actual
- Ante un descenso del diferencial se opera en la frecuencia operativa de la CPU inmediatamente inferior a la actual

Mencionar que las acciones anteriores tienen un coste de energía debido a la modificación en el cambio de la frecuencia operativa de la CPU. Para minimizar esto último, empleamos un umbral temporal configurable por el administrador, el cual nos permite conocer si nos encontramos en una situación en la que las acciones anteriores podrían ser significativas en cuanto a reducir el consumo que se está produciendo y, por tanto, asumir el coste energético de un cambio de frecuencia en la CPU.

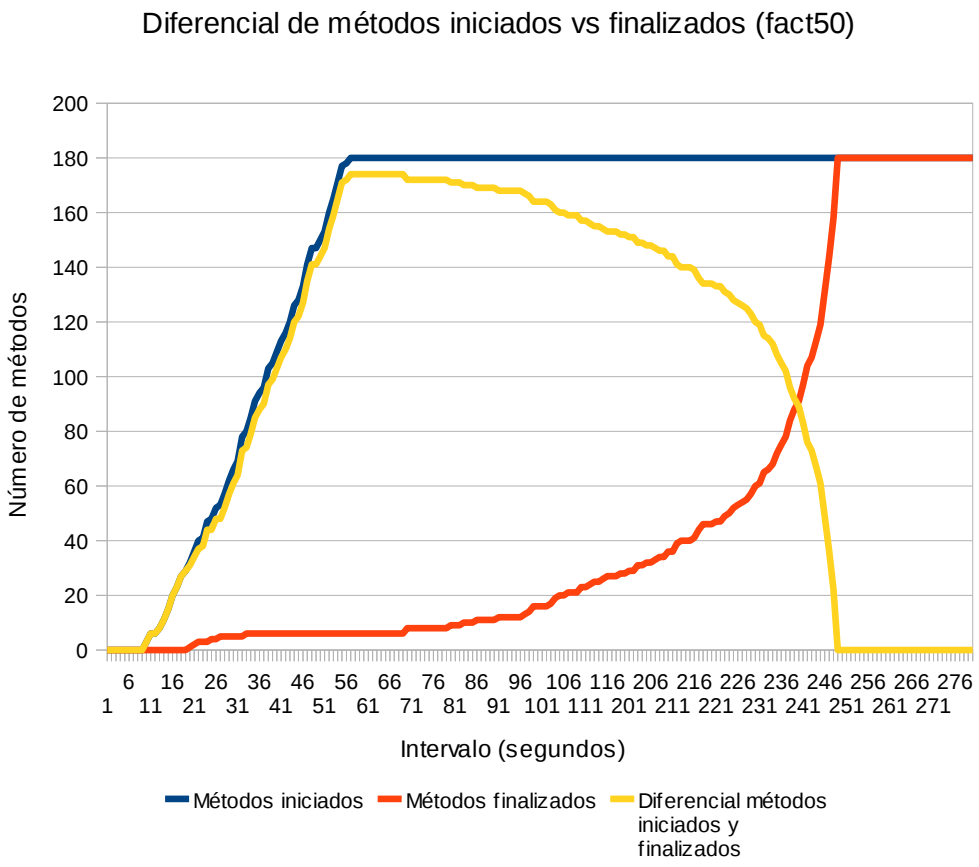


Figura 14. Diferencial de métodos iniciados vs finalizados por segundo (fact50)

4.4.4 Propuesta y evaluación del algoritmo de gobierno

La propuesta de algoritmo de gobierno se encuentra implementada en el prototipo denominado *gobweb*. En concreto, la parte más relevante del mismo en cuánto a toma de decisiones y políticas de acción aparece recogida en el módulo *policy.py*. En este módulo están contenidas dos funciones denominadas *get_next_action_by_policy()* y *get_next_policy_action()*

La primera de ellas es responsable de proponer una nueva frecuencia operativa en caso de considerarlo oportuno. Para ello, se asegura que se dan las condiciones para un posible cambio de frecuencia operativa en la CPU. Estas condiciones son principalmente que la función no haya sido llamada recientemente y sea posible identificar al menos uno de los métodos del Modelo en un estado de carga. Ante estas condiciones, la función obtiene las tendencias de petición y tiempo para delegar en *get_next_policy_action()* la selección de la siguiente acción a tomar.

Así, esta última función recibe información sobre diferenciales actuales de petición y tiempo al igual que sus correspondientes tendencias. Un listado de la función se muestra a continuación.

```
def get_next_policy_action(current_delta_pending_requests, requests_tendency,
                           current_delta_time, time_tendency):

    next_policy_action = 'none'

    if current_delta_pending_requests > 0:
        if requests_tendency > 0:
            if requests_tendency > (current_delta_pending_requests *
                                    gw_conf.CHANGE_THRESHOLD / 100):
                next_policy_action = 'max_freq'
            else:
                next_policy_action = 'inc_freq'
        else:
            if time_tendency > 0:
                if time_tendency > (math.fabs(current_delta_time) *
                                    gw_conf.CHANGE_THRESHOLD / 100):
                    next_policy_action = 'inc_freq'
                else:
                    next_policy_action = 'dec_freq'
            else:
                if (math.fabs(time_tendency) > (math.fabs(current_delta_time)
                                    * gw_conf.CHANGE_THRESHOLD / 100)):
                    next_policy_action = 'min_freq'
                else:
                    next_policy_action = 'dec_freq'
    else:
        next_policy_action = 'min_freq'

    return next_policy_action
```

La lógica detrás de esta segunda función es sencilla. En líneas generales, esta segunda función selecciona una acción de entre cuatro posibles: *max_freq*, *min_freq*, *inc_freq* y *dec_freq*

Cada una de estas acciones provocará, en último término y fuera del módulo *policy.py*, que se incremente (*inc_freq*), decremente (*dec_freq*) o se establezca la frecuencia operativa de la CPU a un máximo (*max_freq*) o a un mínimo (*min_freq*).

La toma de la decisión esta básada en un análisis de casos dirigido por variables que consideran el estado del número de invocaciones a método efectuadas, así como los tiempos invertidos en cada uno de los estados de ejecución de un método del Modelo.

El algoritmo comienza inspeccionando si el Modelo tiene pendiente trabajo para realizar. Si no existen peticiones que atender; la salida, por lo tanto, será *min_freq*

En caso de que existan peticiones pendientes comprueba si la tendencia de trabajo realizado es positiva o negativa.

Si es positiva, la decisión se basa en el crecimiento de esta tendencia con la intención de comprobar si ésta supera un umbral de crecimiento preestablecido. Esto último permite decidir como de brusco debe ser el aumento de frecuencia: incremental (*inc_freq*) o máximo (*max_freq*)

Si la tendencia de peticiones es negativa, entonces la aplicación es capaz de finalizar más trabajo del que se le hace llegar. En este caso la decisión se toma en base a la tendencia temporal en la ejecución del método. Si esta tendencia temporal es positiva se decide incrementar la frecuencia en caso de superar un determinado umbral. En caso de que no lo supere, se propone decrementar la frecuencia.

Al mismo tiempo, si la tendencia temporal es negativa el Modelo presenta un rendimiento mejor al que mostró en un pasado reciente y, por lo tanto, es necesario decidir como de brusco se desea reducir la frecuencia en base a un umbral entre las salidas *dec_freq* o *min_freq*.

En resumen, la política inspecciona la tendencia en la cantidad de trabajo realizado por el Modelo y el tiempo invertido en llevarlo a cabo. Esto permite proponer un cambio de frecuencia operativa en la CPU alineada con el *stress* que sufre el Modelo. Las funciones de este módulo tratan de identificar y clasificar el estado de carga en el que se encuentra el Modelo para sugerir la siguiente acción a tomar.

Para evaluar y validar lo anterior se propone el siguiente escenario donde, con una tasa de llegada de 3 usuarios por segundo durante un intervalo de 30 segundos, se observa el comportamiento del gobernador web, CPU y Modelo de la aplicación web.

Este escenario es completo en el sentido de que se presenta una evolución en el *stress* de carga del Modelo. Así, esta tasa de llegada de usuarios permite ver como evoluciona el Modelo en los diferentes tramos de carga. En los primeros 30 segundos las llamadas a método se suceden. Esto permite congestionar el Modelo y ver como reacciona ante la llegada de nuevas invocaciones a método que no puede atender. Tras este periodo inicial se dejan de invocar nuevos métodos del Modelo con la intención de ver como el Modelo se recupera y empieza a finalizar invocaciones pendientes de método hasta alcanzar de nuevo un estado de reposo.

Este escenario permite no sólo evaluar un ciclo de carga completo respecto a la selección y cambio

de frecuencia operativa que ocurre en la CPU; sino validar otros comportamientos del gobernador como pueden ser detectar cambios de carga en el Modelo o latencias de comunicación en la fase de muestreo de información.

En la Figura 15, se muestran las diferentes gráficas de salida para este escenario de evaluación y validación del gobernador web. Estas cinco gráficas, en el orden presentado, han demostrado ser una herramienta útil para la comprensión y rápida inspección visual del comportamiento del Modelo.

La gráfica a) es la gráfica de evolución de número de métodos iniciados frente a número de métodos finalizados. En concreto, permite identificar para este escenario el periodo de evaluación total derivado del periodo de *stress* continuado de 30 segundos. Confirmamos que para este escenario nos movemos en el orden de los 120 segundos. La gráfica también permite ver el tiempo de recuperación entre la finalización del stress y el comienzo de recuperación del Modelo a máxima carga computacional, unos 20 segundos.

La gráfica b) es una gráfica de variabilidad. Comparándola con la gráfica a) podemos conocer el instante temporal donde existe una variabilidad máxima en el número de métodos iniciados y finalizados por segundo. La distancia entre picos nos indica el intervalo temporal asociado a la computación total de la carga simulada para este escenario.

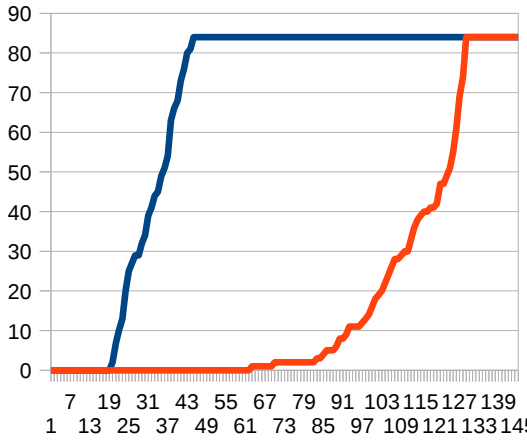
La gráfica d) representa la evolución del tiempo de ejecución de un método del Modelo y la variabilidad temporal del mismo. Al encontrarse situada visualmente debajo de la gráfica b) facilita validar que la evolución del tiempo de ejecución y su variabilidad coinciden con el rango temporal de la señal de variabilidad de métodos finalizados por segundo donde esta última no es nula. Esto es coherente con el hecho de que la información de tiempos de ejecución esté disponible sólo cuando el Modelo comienza a finalizar invocaciones de métodos y, por tanto, se dispone de esta información.

La gráfica c) es la gráfica más relevante en cuanto a que permite confirmar que el gobernador interpreta la carga del Modelo correctamente y selecciona nuevas frecuencias operativas para la CPU en base a la misma, de acuerdo con la política mostrada en líneas anteriores. Su posición, situada debajo de la gráfica a), nos permite evaluar que los cambios de frecuencia son adecuados así como inspeccionar visualmente la latencia de los mismos.

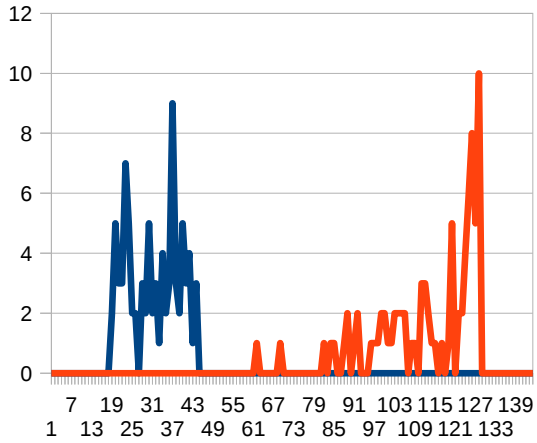
Así, en la gráfica c) se observa que el Modelo opera a máxima frecuencia operativa mientras la tasa de llegada de trabajo es máxima. Sólo empieza a reducir la frecuencia progresivamente cuando el trabajo empieza a finalizarse y, siempre que el mismo se lleve a cabo en un intervalo temporal aceptable. Por último, una vez detectada una situación de reposo o ausencia de trabajo la frecuencia pasa a ser mínima.

La gráfica d) es una gráfica porcentual de carga. Es relevante confirmar que la CPU mantiene un 100% de carga a lo largo del escenario de evaluación como estaba previsto dada la naturaleza computacional del Modelo bajo evaluación. Analizando esta gráfica con la gráfica c) se observa como un gobernador de grano grueso no hubiera podido tomar una decisión óptima a lo largo de todo el escenario debido a que habitualmente sus decisiones son tomadas en relación a tendencias de carga en la CPU y tiempos de E/S

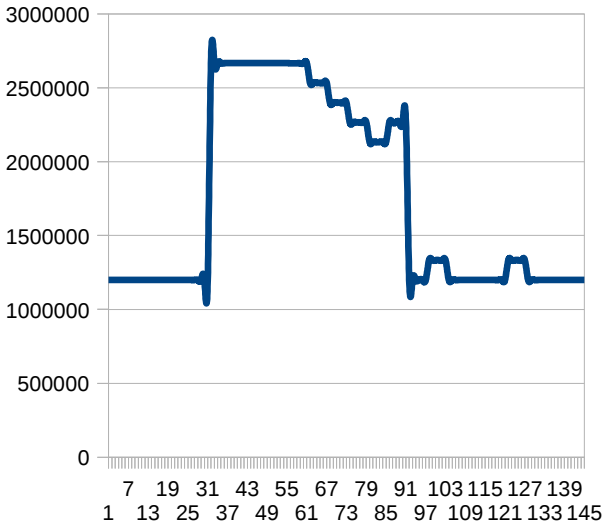
Se valida por tanto que el gobierno de la CPU se realiza en base a la política de consumo implementada y los resultados obtenidos son coherentes y esperados.



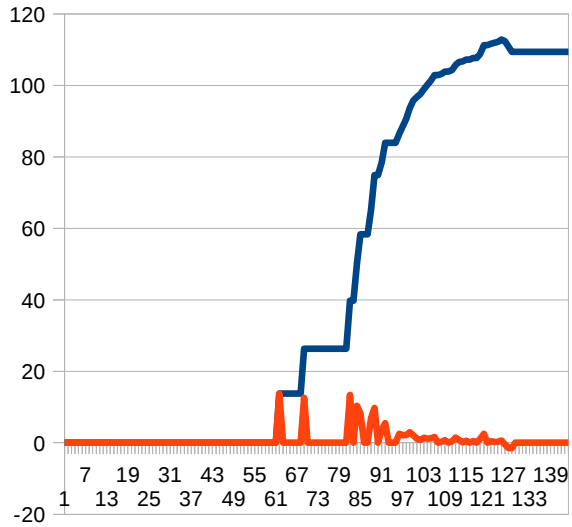
a) Número de métodos iniciados (azul) frente a número de métodos finalizados (naranja) por segundo



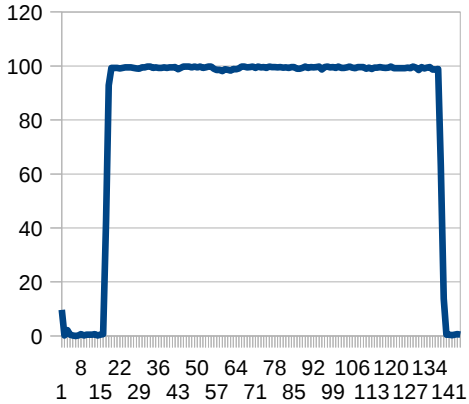
b) Variabilidad en el número de métodos inicializados (azul) frente a finalizados (naranja) por segundo



c) Evolución temporal (segundos) de la frecuencia operativa (MHz) en la CPU



d) Evolución tiempo ejecución (azul) frente a variabilidad del mismo (naranja) en segundos



e) Carga porcentual CPU

Figura 15. Gráficas de evaluación gobernador web (3 usuarios/segundo, 30 segundos, fact50)

4.4.5 Herramientas de evaluación empleadas

Para la prueba de evaluación se decidió utilizar principalmente la herramienta de prueba *Tsung*. *Tsung* es un framework de prueba que permite simular cargas distribuídas multiprotocolo. El objetivo de *Tsung* es simular usuarios con la intención de probar la escalabilidad y rendimiento de aplicaciones basadas en la arquitectura cliente/servidor.

En nuestro caso *Tsung* fué empleado para simular una carga determinista sobre la aplicación web, con la intención de observar el comportamiento del gobernador web en un tiempo determinado de *stress*.

Tsung fué instalado y configurado en la máquina cliente. Su instalación y configuración se realizó desde código fuente tal y como indica su documentación[13]. La versión utilizada fué *Tsung 1.3.3*

La obtención de la información de instrumentalización fué realizada a través del código desarrollado en este trabajo; bien bajo demanda, invocando los métodos de instrumentalización del Modelo vía interface web o bien mediante el análisis de la información recopilada por el gobernador web y almacenada en base de datos periódicamente.

13 http://tsung.erlang-projects.org/user_manual.html

5. Conclusiones y trabajo futuro

En la realización de este proyecto se profundizó en el soporte de gestión de energía de un sistema operativo *GNU/Linux*. En los apartados dos y tres se documentó gran parte del conocimiento obtenido para la administración, gestión y programación de soporte de energía en software a nivel de *kernel* y a nivel de aplicación.

En concreto, en el apartado dos se presento la interface ACPI, su implementación en *Linux* y la experiencia de usuario final en gestión de energía. Mientras la experiencia de usuario es realmente abierta y su comprensión no presenta ningún problema; la interface ACPI y la implementación de la misma en *Linux*, presenta zonas oscuras confirmadas en las cuales la información y código fuente no se encuentra disponible.

En general, aunque ACPI presenta una aproximación interesante y soluciona un gran número de problemas en la integración hardware-software dentro de la Industria es probable que algunos fabricantes y usuarios decidan no adoptarla en su estado actual. Los primeros por añadir un coste considerable a la manufactura de hardware en volúmenes de producción pequeños mientras que los segundos por considerar ACPI un sistema no totalmente abierto, empleado como barrera de entrada por grandes fabricantes que convierten el código embebido en un secreto de negocio, como pueden ser los fragmentos de código AML.

En el apartado tres, se profundizó en la preservación de energía a bajo nivel, deteniéndonos en donde suceden 2/3 de la gestión de energía en el *kernel Linux*, los *drivers* de dispositivo. Tras una breve introducción teórica de los modelos e interfaces vigentes se realizó un análisis de un *driver* tipo. El apartado permite obtener una visión de la gestión de energía realizada en el *kernel* con foco en el bus PCI.

Mencionar que a lo largo del trabajo realizado en el apartado tres se desarrollaron también una serie de parches para el *driver* gráfico sm7xx. Estos parches están incluídos en las versiones 2.6.39 y 3.0.0 del *kernel Linux*. La inclusión de estos parches en el *kernel* sucedió a lo largo de unas pocas semanas en las que el código fué potencialmente revisado por millones de personas en general y dos responsables de subsistemas en particular, con anterioridad a que Linus[14] los incluyese *upstream* en el *kernel*. Este breve lapso de tiempo tan reducido en el proceso de inclusión de nuevo código permite capturar la agilidad, apertura y dinamismo que tiene un proyecto de las dimensiones y el alcance de un *kernel* como es *Linux*.

La contribución de este proyecto a nivel de aplicación tiene lugar principalmente en el apartado cuatro, en el que se cuestiona la necesidad de proponer una implementación para un gobernador web basada en la arquitectura software de una aplicación web y no en la carga computacional genérica del hardware que la alberga. La correcta instrumentalización de una aplicación web que implementa el patrón Modelo-Vista-Controlador permite que dicha aplicación sea gobernada a través del diseño e implementación propuesto con una mayor granularidad de gestión.

La evaluación y validación de la propuesta de gobernador web presentada en el proyecto tiene lugar en este apartado cuatro, arrojando resultados positivos y prometedores de cara a realizar una investigación más profunda.

14 http://es.wikipedia.org/wiki/Linus_Torvalds

Las líneas anteriores permiten, por tanto, verificar que los objetivos principales propuestos inicialmente están satisfechos en su totalidad. Al igual, que la experiencia obtenida en la consecución de los mismos plantea nuevos retos de futuro con el ánimo de progresar en las líneas abiertas por este proyecto.

En concreto, surgen tres vías de progreso directo a raíz de los resultados obtenidos en este proyecto:

- Patrón de migración de *drivers* PCI al nuevo marco de gestión de energía del *kernel* en el cual confirmaríamos que uno de los parches desarrollados presenta una naturaleza de patrón, es decir, aplicando su misma lógica sobre otros *drivers* PCI, y con cambios muy reducidos en el nuevo *driver*, se lograría que este nuevo *driver* PCI hiciese uso del nuevo marco de gestión de energía siendo compatible con el antiguo e invirtiendo un tiempo de desarrollo reducido.
- Nuevo diseño distribuído del gobernador web con la intención de que ningún bloque de su arquitectura comparta CPU con la aplicación gobernada. El objetivo de este nuevo diseño sería un gobernador web tolerante a *stress* ante altas cargas computacionales y distribuído, redundando en mejores latencias y tiempos de respuesta entre los diferentes elementos.
- Extensión de la interface de instrumentalización de la aplicación web con la intención de soportar filtros de carga, es decir, que invocaciones a métodos en estados de alta carga puedan ser interceptados y gestionados de forma transparente al modelo.

Como reflexión final, me gustaría compartir el hecho de que en la actualidad el desarrollo de software no suele contemplar requisitos energéticos en su proceso de captura. Esto último, es algo a destacar ya que como se ha demostrado la gestión energética puede ser incluída como parte de la arquitectura de la solución sin afectar a la lógica de negocio de la misma en alto grado.

6. Referencias

[ACPI-2010] ACPI. (2010). *Advanced Configuration & Power Interface*. [en línea]. http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf [fecha de consulta: 1 de junio de 2011]

[Bohrer-2002] Bohrer, P.; Elnozahy, E.; Keller, T.; Kistler, M.; Lefurgy, C.; McDowell, C.; Rajamony, R. (2002). *The case for power management in web servers*. Kluwer Academic Publishers Norwell, MA, USA. Power aware computing. IBM Research, Austin TX. ISBN-10: 0306467860

[Brown-2005] Brown, L.; Keshavamurthy, A.; Shaohua Li, D.; Moore, R.; Pallipadi, V.; Yu, L. (2005). *ACPI in Linux*. Intel Open Source Technology Center. Proceedings of the Linux Symposium. Volune One.

[Corbet-2005] Corbet, J., Rubini, A., Kroah-Hartman, G. (2005). Linux Device Drivers, 3rd Edition. O'Reilly Media; 3 edition. ISBN-13: 978-0596005900

[Frenkil-1997] Frenkil, J. (1997). *Tools and Methodologies for Low Power Design*. [en línea] http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.7225 [fecha de consulta: 1 de junio de 2011]

[Henderson-2006] Henderson, Bryan (2006). *Linux Loadable Kernel Module HOWTO*. [en línea]. http://tldp.org/HOWTO/Module-HOWTO/ [fecha de consulta: 1 de junio de 2011]

[Intel-2011] Intel. (2011). *Advanced Configuration & Power Interface (ACPI)*. [en línea]. http://www.intel.com/technology/iapc/acpi/ [fecha de consulta: 1 de junio de 2011]

[Microsoft-2011] Microsoft; Intel. (2011). *Advanced Power Management*. [en línea]. http://es.wikipedia.org/wiki/Advanced_Power_Management [fecha de consulta: 1 de junio de 2011]

[Miñana-2009] Miñana, G. (2009). *Reducción del Consumo de Potencia en Unidades Funcionales Mediante Cotejo de Códigos de Operación*. Madrid: Universidad Complutense de Madrid, Departamento de Arquitectura de Computadores y Automática. ISBN-13: 978-8469284223

[Pallipadi-2007] Pallipadi, V.; Li, S.; Belay, A. (2007). *cpuidle - Do nothing, efficiently ...* Intel Open Source Technology Center; Novell, Inc. Proceedings of the Linux Symposium. Volume Two.

[Shearer-2007] Shearer, F. (2007). Power Management in Mobile Devices. Newnes. ISBN-13: 978-0750679589

[Venkateswaran-2009] Venkateswaran, S. (2009). Essential Linux Device Drivers. Prentice Hall. ISBN-13: 978-0132396554

Anexo A. Acrónimos

AC. *Alternating Current*
ACPI. *Advanced Configuration and Power Interface*
ACPICA. *ACPI Component Architecture*
AML. *ACPI Machine Language*
APM. *Advanced Power Management*
APT. *Advanced Packaking Tool*
BIOS. *Basic Input/Output System*
CPU. *Central Processing Unit*
DC. *Direct Current*
FLOSS. *Free/Libre/Open Source Software*
GNU. *GNU is Not Unix*
HTTP. *Hypertext Transfer Protocol*
HTTPS. *Hypertext Transfer Protocol Secure*
ISA. *Instruction Set Architecture*
LKM. *Loadable Kernel Module*
MVC. *Model-View-Controller*
OS. *Operating System*
OSPM. *Operating System-directed and Power Management*
PCI. *Peripheral Component Interconnect*
PHP. *PHP: Hypertext Preprocessor*
PM. *Power Management*
PNP. *Plug and play*
RAM. *Random Access Memory*
RPMM. *Runtime Power Management Model*
SCI. *System Control Interrupts*
SQL. *Structured Query Language*
SSM. *System Sleep Model*
WWW. *World Wide Web*

Anexo B. Software adjunto

El presente proyecto adjunta un fichero comprimido en formato ZIP[15] que contiene la implementación software del trabajo realizado. El fichero, denominado PFC-46902656K.zip, una vez descomprimido presenta la siguiente estructura:

- *INDEX*. Un fichero describiendo el contenido de cada directorio.
- *md5sum.txt*. Un fichero de control que permite validar contenido mediante *hashes* MD5[16].
- *sm7xx*. Código fuente de los parches para el *driver sm7xx* enviados a las listas públicas del *kernel*.
- *mvc_pm*. Código fuente de la aplicación web MVC con soporte para gestión de energía.
- *gobweb*. Código fuente del gobernador web.

15 http://en.wikipedia.org/wiki/ZIP_(file_format)
16 http://en.wikipedia.org/wiki/MD5

Anexo C. Código fuente

Código fuente parches driver sm7xx

```
commit 392a002a0066812480e1b55639bbced5936d26aa
Author: Javier M. Mellid <jmunhoz@igalia.com>
Date:   Wed Mar 30 16:24:10 2011 +0200

    staging: sm7xx: fixed defines

    Deleted redundant __KERNEL__ define

    PM methods (suspend and resume) enabled under CONFIG_PM only

    Signed-off-by: Javier M. Mellid <jmunhoz@igalia.com>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

diff --git a/drivers/staging/sm7xx/smtcfb.c b/drivers/staging/sm7xx/smtcfb.c
index d007e4a..0d65971 100644
--- a/drivers/staging/sm7xx/smtcfb.c
+++ b/drivers/staging/sm7xx/smtcfb.c
@@ -26,10 +26,6 @@
  *       Boyod.yang <boyod.yang@siliconmotion.com.cn>
  */

-#ifndef __KERNEL__
-#define __KERNEL__
-#endif
-
 #include <linux/io.h>
 #include <linux/fb.h>
 #include <linux/pci.h>
@@ -1019,6 +1015,7 @@ static void __devexit smtcfb_pci_remove(struct pci_dev *pdev)
         smtc_free_fb_info(sfb);
 }

+#ifdef CONFIG_PM
 /* Jason (08/14/2009)
  * suspend function, called when the suspend event is triggered
  */
@@ -1111,6 +1108,7 @@ static int __maybe_unused smtcfb_resume(struct pci_dev *pdev)

         return 0;
 }
+#endif

 /* Jason (08/13/2009)
```

* pci_driver struct used to wrap the original driver

```
commit 3b70a26bcbe05db12965de702368ca0b9ec945f1
Author: Javier M. Mellid <jmunhoz@igalia.com>
Date:   Sat Apr 30 17:44:26 2011 +0200

    staging: sm7xx: minor cleanup

    Sync code comments with TODO, fix some style and format issues

    Signed-off-by: Javier M. Mellid <jmunhoz@igalia.com>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

diff --git a/drivers/staging/sm7xx/TODO b/drivers/staging/sm7xx/TODO
index a66d9e4..b0f5318 100644
--- a/drivers/staging/sm7xx/TODO
+++ b/drivers/staging/sm7xx/TODO
@@ -2,7 +2,6 @@ TODO:
 - Dual head support
 - 2D acceleration support
 - use kernel coding style
-- checkpatch.pl clean
 - refine the code and remove unused code
 - use kernel framebuffer mode setting instead of hard code
 - move it to drivers/video/sm7xx/ or make it be drivers/video/sm7xxfb.c
diff --git a/drivers/staging/sm7xx/smtcfb.c b/drivers/staging/sm7xx/smtcfb.c
index 976a080..c0d943f 100644
--- a/drivers/staging/sm7xx/smtcfb.c
+++ b/drivers/staging/sm7xx/smtcfb.c
@@ -14,15 +14,15 @@
  *
  * Version 0.10.26192.21.01
  *      - Add PowerPC/Big endian support
- *      - Add 2D support for Lynx
- *      - Verified on2.6.19.2  Boyod.yang <boyod.yang@siliconmotion.com.cn>
+ *      - Verified on 2.6.19.2
+ *      Boyod.yang <boyod.yang@siliconmotion.com.cn>
  *
  * Version 0.09.2621.00.01
- *      - Only support Linux Kernel's version 2.6.21.
- *      Boyod.yang  <boyod.yang@siliconmotion.com.cn>
+ *      - Only support Linux Kernel's version 2.6.21
+ *      Boyod.yang <boyod.yang@siliconmotion.com.cn>
  *
  * Version 0.09
-*      - Only support Linux Kernel's version 2.6.12.
+ *      - Only support Linux Kernel's version 2.6.12
  *      Boyod.yang <boyod.yang@siliconmotion.com.cn>
  */

@@ -99,17 +99,17 @@ struct vesa_mode_table     {
 static struct vesa_mode_table vesa_mode[] = {
```

```
        {"0x301", 640,  480,  8},
        {"0x303", 800,  600,  8},
-       {"0x305", 1024, 768, 8},
+       {"0x305", 1024, 768,  8},
        {"0x307", 1280, 1024, 8},

        {"0x311", 640,  480,  16},
        {"0x314", 800,  600,  16},
-       {"0x317", 1024, 768, 16},
+       {"0x317", 1024, 768,  16},
        {"0x31A", 1280, 1024, 16},

        {"0x312", 640,  480,  24},
        {"0x315", 800,  600,  24},
-       {"0x318", 1024, 768, 24},
+       {"0x318", 1024, 768,  24},
        {"0x31B", 1280, 1024, 24},
 };
```

```
commit 59815677555746f8263672f25cebcf4c27fc5d31
Author: Javier M. Mellid <jmunhoz@igalia.com>
Date:   Sat Apr 30 17:44:25 2011 +0200

    staging: sm7xx: smtcfb.c: Use the new PCI PM

    The sm7xx driver uses the legacy PCI power management (suspend and
    resume) callbacks.

    This patch adds the new PCI PM and let the PCI core code handles the
    PCI-specific details of power transitions.

    Tested in 2.6.38, including standby and hibernation support.

    Tested-by: Wu Zhangjin <wuzhangjin@gmail.com>
    Signed-off-by: Javier M. Mellid <jmunhoz@igalia.com>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

diff --git a/drivers/staging/sm7xx/smtcfb.c b/drivers/staging/sm7xx/smtcfb.c
index 3e2230f..976a080 100644
--- a/drivers/staging/sm7xx/smtcfb.c
+++ b/drivers/staging/sm7xx/smtcfb.c
@@ -1016,13 +1016,10 @@ static void __devexit smtcfb_pci_remove(struct pci_dev *pdev)
 }

 #ifdef CONFIG_PM
-/* Jason (08/14/2009)
- * suspend function, called when the suspend event is triggered
- */
-static int __maybe_unused smtcfb_suspend(struct pci_dev *pdev, pm_message_t msg)
+static int smtcfb_pci_suspend(struct device *device)
 {
+       struct pci_dev *pdev = to_pci_dev(device);
        struct smtcfb_info *sfb;
-       int retv;

        sfb = pci_get_drvdata(pdev);

@@ -1032,25 +1029,9 @@ static int __maybe_unused smtcfb_suspend(struct pci_dev *pdev,
pm_message_t msg)
        smtc_seqw(0x20, (smtc_seqr(0x20) | 0xc0));
        smtc_seqw(0x69, (smtc_seqr(0x69) & 0xf7));

-       switch (msg.event) {
-       case PM_EVENT_FREEZE:
-       case PM_EVENT_PRETHAW:
-               pdev->dev.power.power_state = msg;
-               return 0;
-       }
-
```

```
-	/* when doing suspend, call fb apis and pci apis */
-	if (msg.event == PM_EVENT_SUSPEND) {
-		console_lock();
-		fb_set_suspend(&sfb->fb, 1);
-		console_unlock();
-		retv = pci_save_state(pdev);
-		pci_disable_device(pdev);
-		retv = pci_choose_state(pdev, msg);
-		retv = pci_set_power_state(pdev, retv);
-	}
-
-	pdev->dev.power.power_state = msg;
+	console_lock();
+	fb_set_suspend(&sfb->fb, 1);
+	console_unlock();

	/* additionally turn off all function blocks including internal PLLs */
	smtc_seqw(0x21, 0xff);
@@ -1058,22 +1039,13 @@ static int __maybe_unused smtcfb_suspend(struct pci_dev *pdev,
pm_message_t msg)
	return 0;
 }

-static int __maybe_unused smtcfb_resume(struct pci_dev *pdev)
+static int smtcfb_pci_resume(struct device *device)
 {
+	struct pci_dev *pdev = to_pci_dev(device);
	struct smtcfb_info *sfb;
-	int retv;

	sfb = pci_get_drvdata(pdev);

-	/* when resuming, restore pci data and fb cursor */
-	if (pdev->dev.power.power_state.event != PM_EVENT_FREEZE) {
-		retv = pci_set_power_state(pdev, PCI_D0);
-		pci_restore_state(pdev);
-		if (pci_enable_device(pdev))
-			return -1;
-		pci_set_master(pdev);
-	}
-
	/* reinit hardware */
	sm7xx_init_hw();
	switch (hw.chipID) {
@@ -1108,22 +1080,30 @@ static int __maybe_unused smtcfb_resume(struct pci_dev *pdev)

	return 0;
 }
-#endif
```

```
-/* Jason (08/13/2009)
- * pci_driver struct used to wrap the original driver
- * so that it can be registered into the kernel and
- * the proper method would be called when suspending and resuming
- */
+static const struct dev_pm_ops sm7xx_pm_ops = {
+	.suspend = smtcfb_pci_suspend,
+	.resume = smtcfb_pci_resume,
+	.freeze = smtcfb_pci_suspend,
+	.thaw = smtcfb_pci_resume,
+	.poweroff = smtcfb_pci_suspend,
+	.restore = smtcfb_pci_resume,
+};
+
+#define SM7XX_PM_OPS (&sm7xx_pm_ops)
+
+#else  /* !CONFIG_PM */
+
+#define SM7XX_PM_OPS NULL
+
+#endif /* !CONFIG_PM */
+
 static struct pci_driver smtcfb_driver = {
	.name = "smtcfb",
	.id_table = smtcfb_pci_table,
	.probe = smtcfb_pci_probe,
	.remove = __devexit_p(smtcfb_pci_remove),
-#ifdef CONFIG_PM
-	.suspend = smtcfb_suspend,
-	.resume = smtcfb_resume,
-#endif
+	.driver.pm  = SM7XX_PM_OPS,
 };

 static int __init smtcfb_init(void)
```

```
commit dc762c4f8514f23094927e0a62ef305d90651535
Author: Javier M. Mellid <jmunhoz@igalia.com>
Date:   Sat May 7 03:11:58 2011 +0200

    staging: sm7xx: Use kernel framebuffer mode setting

    This patch implements dynamic framebuffer mode setting.

    Previous code works with mode setting in a hard code way. Previous hard
    code configuration is used as default configuration if dynamic mode
    setting or boot mode setting (via sm712vga_setup) is not used.

    Tested with SM712 supporting 1024x600x16 as default hardware resolution.

    Changes:

    - Implement fb_check_var and fb_set_par callbacks
    - Remove __maybe_unused decorator in function being used (sm712vga_setup)
    - Minor cleanup on initialization structs related with mode settings
    - Updated author copyright
    - Updated TODO file

    Signed-off-by: Javier M. Mellid <jmunhoz@igalia.com>
    Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

diff --git a/drivers/staging/sm7xx/TODO b/drivers/staging/sm7xx/TODO
index b0f5318..7304021 100644
--- a/drivers/staging/sm7xx/TODO
+++ b/drivers/staging/sm7xx/TODO
@@ -3,7 +3,6 @@ TODO:
 - 2D acceleration support
 - use kernel coding style
 - refine the code and remove unused code
-- use kernel framebuffer mode setting instead of hard code
 - move it to drivers/video/sm7xx/ or make it be drivers/video/sm7xxfb.c

 Please send any patches to Greg Kroah-Hartman <greg@kroah.com> and
diff --git a/drivers/staging/sm7xx/smtcfb.c b/drivers/staging/sm7xx/smtcfb.c
index c0d943f..94cb4e8 100644
--- a/drivers/staging/sm7xx/smtcfb.c
+++ b/drivers/staging/sm7xx/smtcfb.c
@@ -8,6 +8,9 @@
  * Copyright (C) 2009 Lemote, Inc.
  * Author: Wu Zhangjin, wuzhangjin@gmail.com
  *
+ * Copyright (C) 2011 Igalia, S.L.
+ * Author: Javier M. Mellid <jmunhoz@igalia.com>
+ *
  *  This file is subject to the terms and conditions of the GNU General Public
  *  License. See the file COPYING in the main directory of this archive for
```

```
  *  more details.
@@ -39,16 +42,16 @@
 #include <linux/pm.h>
 #endif

-struct screen_info smtc_screen_info;
-
 #include "smtcfb.h"

 #ifdef DEBUG
-#define smdbg(format, arg...)          printk(KERN_DEBUG format , ## arg)
+#define smdbg(format, arg...) printk(KERN_DEBUG format , ## arg)
 #else
 #define smdbg(format, arg...)
 #endif

+struct screen_info smtc_screen_info;
+
 /*
  * Private structure
  */
@@ -127,6 +130,29 @@ u16 smtc_ChipIDs[] = {

 #define numSMTCchipIDs (sizeof(smtc_ChipIDs) / sizeof(u16))

+static struct fb_var_screeninfo smtcfb_var = {
+       .xres           = 1024,
+       .yres           = 600,
+       .xres_virtual   = 1024,
+       .yres_virtual   = 600,
+       .bits_per_pixel = 16,
+       .red            = {16, 8, 0},
+       .green          = {8, 8, 0},
+       .blue           = {0, 8, 0},
+       .activate       = FB_ACTIVATE_NOW,
+       .height         = -1,
+       .width          = -1,
+       .vmode          = FB_VMODE_NONINTERLACED,
+};
+
+static struct fb_fix_screeninfo smtcfb_fix = {
+       .id             = "sm712fb",
+       .type           = FB_TYPE_PACKED_PIXELS,
+       .visual         = FB_VISUAL_TRUECOLOR,
+       .line_length    = 800 * 3,
+       .accel          = FB_ACCEL_SMI_LYNX,
+};
+
 static void sm712_set_timing(struct smtcfb_info *sfb,
                                struct par_info *ppar_info)
```

```
 {
@@ -268,29 +294,6 @@ static void smtc_set_timing(struct smtcfb_info *sfb, struct par_info
         }
 }

-static struct fb_var_screeninfo smtcfb_var = {
-        .xres = 1024,
-        .yres = 600,
-        .xres_virtual = 1024,
-        .yres_virtual = 600,
-        .bits_per_pixel = 16,
-        .red = {16, 8, 0},
-        .green = {8, 8, 0},
-        .blue = {0, 8, 0},
-        .activate = FB_ACTIVATE_NOW,
-        .height = -1,
-        .width = -1,
-        .vmode = FB_VMODE_NONINTERLACED,
-};
-
-static struct fb_fix_screeninfo smtcfb_fix = {
-        .id = "sm712fb",
-        .type = FB_TYPE_PACKED_PIXELS,
-        .visual = FB_VISUAL_TRUECOLOR,
-        .line_length = 800 * 3,
-        .accel = FB_ACCEL_SMI_LYNX,
-};
-
 /* chan_to_field
  *
  * convert a colour value into a field position
@@ -604,20 +607,6 @@ smtcfb_write(struct fb_info *info, const char __user *buf, size_t count,
 }
 #endif /* ! __BIG_ENDIAN */

-static struct fb_ops smtcfb_ops = {
-        .owner = THIS_MODULE,
-        .fb_setcolreg = smtc_setcolreg,
-        .fb_blank = cfb_blank,
-        .fb_fillrect = cfb_fillrect,
-        .fb_imageblit = cfb_imageblit,
-        .fb_copyarea = cfb_copyarea,
-#ifdef __BIG_ENDIAN
-        .fb_read = smtcfb_read,
-        .fb_write = smtcfb_write,
-#endif
-
-};
-
 void smtcfb_setmode(struct smtcfb_info *sfb)
```

```
 {
        switch (sfb->fb.var.bits_per_pixel) {
@@ -676,6 +665,47 @@ void smtcfb_setmode(struct smtcfb_info *sfb)
        smtc_set_timing(sfb, &hw);
 }

+static int smtc_check_var(struct fb_var_screeninfo *var, struct fb_info *info)
+{
+       /* sanity checks */
+       if (var->xres_virtual < var->xres)
+               var->xres_virtual = var->xres;
+
+       if (var->yres_virtual < var->yres)
+               var->yres_virtual = var->yres;
+
+       /* set valid default bpp */
+       if ((var->bits_per_pixel != 8)  && (var->bits_per_pixel != 16) &&
+           (var->bits_per_pixel != 24) && (var->bits_per_pixel != 32))
+               var->bits_per_pixel = 16;
+
+       return 0;
+}
+
+static int smtc_set_par(struct fb_info *info)
+{
+       struct smtcfb_info *sfb = (struct smtcfb_info *)info;
+
+       smtcfb_setmode(sfb);
+
+       return 0;
+}
+
+static struct fb_ops smtcfb_ops = {
+       .owner        = THIS_MODULE,
+       .fb_check_var = smtc_check_var,
+       .fb_set_par   = smtc_set_par,
+       .fb_setcolreg = smtc_setcolreg,
+       .fb_blank     = cfb_blank,
+       .fb_fillrect  = cfb_fillrect,
+       .fb_imageblit = cfb_imageblit,
+       .fb_copyarea  = cfb_copyarea,
+#ifdef __BIG_ENDIAN
+       .fb_read      = smtcfb_read,
+       .fb_write     = smtcfb_write,
+#endif
+};
+
 /*
  * Alloc struct smtcfb_info and assign the default value
  */
```

```
@@ -796,7 +826,7 @@ static void smtc_free_fb_info(struct smtcfb_info *sfb)
  *       Returns zero.
  *
  */
-static int __init __maybe_unused sm712vga_setup(char *options)
+static int __init sm712vga_setup(char *options)
 {
        int index;
```

Código fuente aplicación web MVC con soporte de gestión de energía

```
<?php

/*

Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA  02110-1301, USA.

index.php

*/

include_once("controller/Controller.php");

$controller = new Controller();

$controller->invoke();

?>
```

```
<html>

<?php

/*

Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA  02110-1301, USA.

view/viewfactorial.php

*/

?>

<head>
	<title>View factorial!</title>
</head>

<body>

	<?php echo 'Resultado: ' . $fact->factorial; ?>

</body>
</html>
```

```
<?php

/*

Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA  02110-1301, USA.

view/viewfacts.php

*/

?>

<html>

<head>
	<title>View available factorials!</title>
</head>

<body>

Factoriales disponibles:

<ul>
	<li> <a href="index.php?factorial=10">Calcula 100000 veces el factorial de 10</a> </li>
	<li> <a href="index.php?factorial=50">Calcula 100000 veces el factorial de 50</a> </li>
	<li> <a href="index.php?factorial=100">Calcula 100000 veces el factorial de 100</a> </li>
</ul>

</body>

</html>
```

```
<?php

/*

Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA  02110-1301, USA.

view/viewpmstatus.php

*/

echo $pmreply;
?>
```

```
<?php

/*

Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA  02110-1301, USA.

model/Factorial.php

*/

class Factorial {

	public $factorial;

	public function nfact($n)
	{
		if ($n == 0)
		{
			return 1;
		}
		else
		{
			return $n * $this->nfact($n - 1);
		}
	}

	public function __construct($n)
	{
		$this->factorial = $this->nfact($n);
	}

}

?>
```

```
<?php

/*

Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA  02110-1301, USA.

model/Model.php

*/

include_once("model/Factorial.php");

class Model {

	// we use three methods to calculate same thing in order to get
	// several available methods in model quickly. Moreover, it runs
	// several times the same thing to simulate load avoiding
	// arithmetic overflows. Last values are goods!

	public function getFact_10()
	{
		for($i=0; $i<100000; $i++)
		{
			$fact = new Factorial(10);
		}
		return $fact;
	}

	public function getFact_50()
	{
		for($i=0; $i<100000; $i++)
		{
			$fact = new Factorial(50);
		}
		return $fact;
	}
```

```
	public function getFact_100()
	{
		for($i=0; $i<100000; $i++)
		{
			$fact = new Factorial(100);
		}
		return $fact;
	}

}

?>
```

```
<?php

/*

Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA  02110-1301, USA.

model/ModelPM.php

*/

include_once("model/Model.php");

class ModelPM {

	private $model;

	private $filename = '/tmp/mutex.lock';

	public function __construct()
	{
		$this->model = new Model();

		if( !file_exists($this->filename) )
			$this->resetPmStatus();
	}

	private function is_method_available($method_name)
	{
		$found = false;

		$class_methods = get_class_methods($this->model);

		foreach ( $class_methods as $method )
		{
			if ( strpos($method, $method_name) === 0 )
			{
				$found = true;
```

```
                        break;
                }
        }

        return $found;
}

private function update_method_stats_on_disk($method, $delta, $method_executed)
{
        $fp = fopen( $this->filename, "r+" );

        while( !flock($fp, LOCK_EX) );

        $class_methods_stats = (array) unserialize(file_get_contents($this->filename));

        $pair = (array) $class_methods_stats[$method];

        if ( !$method_executed )
        {
                $pair['requests']++;
        }
        else
        {
                $pair['time'] = 0.75 * $pair['time'] + 0.25 * $delta;
                $pair['requests_ok']++;
        }

        $class_methods_stats[$method] = $pair;

        file_put_contents($this->filename, serialize($class_methods_stats));

        flock($fp, LOCK_UN);

        fclose($fp);
}

function __call($method, $args)
{
        if ( $this->is_method_available($method) )
        {

                $this->update_method_stats_on_disk($method, 0, FALSE);

                $s_time = time();

                $result = call_user_func_array(array($this->model, $method), $args);

                $delta = time() - $s_time;

                $this->update_method_stats_on_disk($method, $delta, TRUE);
```

```
                return $result;
        }
        else
        {
                return false;
        }
    }

    public function getPmStatus()
    {
        $result = '{ result : OK,';

        $class_methods_stats = (array) unserialize(file_get_contents($this->filename));

        $num_elements = count($class_methods_stats);

        foreach ( $class_methods_stats as $method => $value )
        {
                $entry = (array) $value;

                $result = $result . ' ' . $method . ' : {' . $entry['requests'] . ',' .
$entry['requests_ok'] . ',' . $entry['time'] . '}';

                $num_elements--;

                if ( $num_elements != 0 )
                {
                        $result = $result . ',';
                }
        }

        $result = $result . ' }';

        return $result;
    }

    public function resetPmStatus()
    {
        // create/reset method statistics

        $class_methods = get_class_methods($this->model);

        foreach ( $class_methods as $method )
        {
                $class_methods_stats[$method]['requests'] = 0;
                $class_methods_stats[$method]['requests_ok'] = 0;
                $class_methods_stats[$method]['time'] = 0;
        }
```

```
                while ( !file_put_contents($this->filename, serialize($class_methods_stats),
LOCK_EX) );

                return '{ result : OK }';
        }

}

?>
```

```
<?php

/*

Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
Street, Fifth Floor, Boston, MA  02110-1301, USA.

controller/Controller.php

*/

include_once("model/ModelPM.php");

class Controller {

	public $model;

	public function __construct()
	{
		$this->model = new ModelPM();
	}

	public function invoke()
	{
		if (isset($_GET['pm']) )
		{
			if ( $_GET['pm'] == 'show_status' )
				$pmreply = $this->model->getPmStatus();

			if ( $_GET['pm'] == 'reset_status' )
				$pmreply = $this->model->resetPmStatus();

			include 'view/viewpmstatus.php';
		}
		else
		{
		if ( !isset($_GET['factorial']) )
		{
```

```
                        // no specific factorial request so we offer possibilities
                        include 'view/viewfacts.php';
                }
                else
                {
                        // show the requested fact
                        $method = "getFact_" . $_GET['factorial'];
                        $fact = $this->model->$method();
                        include 'view/viewfactorial.php';
                }
                }
        }
}

?>
```

Código fuente del gobernador web

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# acpi.py
#

import commands

import error

def get_battery_status():
        data = commands.getstatusoutput('acpi -b')[1].split()
        if len(data) < 4:
                error.log_error_and_die ('commands: battery output format not recognised')
        return { 'status' : data[2],
            'charge' : data[3] }
        return l

if __name__ == '__main__':
        # testing
        print(get_battery_status())
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# cpu.py
#

import os
import os.path

PATH_INFO_CPU = '/sys/devices/system/cpu'

def __read_simple_var(path_directory, varname):
        f = open (path_directory + varname, 'r')
        value = f.readline()[:-1]
        f.close()
        return value

def __read_multiple_var(path_directory, varname):
        f = open (path_directory + varname, 'r')
        value = f.readline().split(' ')[:-1]
        f.close()
        return value

def get_cpu_info():

        num_cpu = 0

        status_cpu = []

        # walk on cpus if exist :)

        while (True):

                path_current_cpu = PATH_INFO_CPU + '/cpu' + str(num_cpu)

                if (os.path.isdir(path_current_cpu)):
```

```
                path_directory = path_current_cpu + '/cpufreq/'

                cpuinfo_min_freq = __read_simple_var(path_directory, 'cpuinfo_min_freq')
                cpuinfo_max_freq = __read_simple_var(path_directory, 'cpuinfo_max_freq')
                cpuinfo_cur_freq = __read_simple_var(path_directory, 'cpuinfo_cur_freq')
                scaling_min_freq = __read_simple_var(path_directory, 'scaling_min_freq')
                scaling_max_freq = __read_simple_var(path_directory, 'scaling_max_freq')
                scaling_cur_freq = __read_simple_var(path_directory, 'scaling_cur_freq')
                scaling_governor = __read_simple_var(path_directory, 'scaling_governor')

                scaling_available_frequencies = __read_multiple_var(path_directory,
                                                        'scaling_available_frequencies')
                scaling_available_frequencies = [int(freq) for freq in
                                                        scaling_available_frequencies]

                scaling_available_governors   = __read_multiple_var(path_directory,
                                                         'scaling_available_governors')

                status_cpu.append( { 'cpu' + str(num_cpu) :
                                    { 'cpuinfo_min_freq' : int(cpuinfo_min_freq),
                                      'cpuinfo_max_freq' : int(cpuinfo_max_freq),
                                      'cpuinfo_cur_freq' : int(cpuinfo_cur_freq),
                                      'scaling_min_freq' : int(scaling_min_freq),
                                      'scaling_max_freq' : int(scaling_max_freq),
                                      'scaling_cur_freq' : int(scaling_cur_freq),
                                      'scaling_governor' : scaling_governor,
                                      'scaling_available_frequencies':
                                                        scaling_available_frequencies,
                                      'scaling_available_governors' :
                                                        scaling_available_governors
                                    }
                                   } )

                num_cpu = num_cpu + 1
        else:
                break;

    return status_cpu

def __write_simple_var(path_directory, varname, value):
    f = open (path_directory + varname, 'a')
    f.write(value)
    f.close()

def set_governor(governor):
    num_cpu = 0
    while (True):
        path_current_cpu = PATH_INFO_CPU + '/cpu' + str(num_cpu)
        if (os.path.isdir(path_current_cpu)):
            __write_simple_var (path_current_cpu + '/cpufreq/', 'scaling_governor',
```

```
                governor)
                num_cpu = num_cpu + 1
        else:
                break

def set_scaling_frequency(frequency):
        num_cpu = 0
        while (True):
                path_current_cpu = PATH_INFO_CPU + '/cpu' + str(num_cpu)
                if (os.path.isdir(path_current_cpu)):
                        __write_simple_var (path_current_cpu + '/cpufreq/', 'scaling_setspeed',
                                                                                str(frequency))
                        num_cpu = num_cpu + 1
                else:
                        break


if __name__ == '__main__':
        # dump cpu state (testing)
        print (get_cpu_info())
        set_governor ('userspace')
        print (get_cpu_info())
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# db.py
#

import os
import os.path

import db_conf

from pysqlite2 import dbapi2 as sqlite

DB_NAME = '/tmp/gobweb.db'

def init(reset):
	global con_db
	global cur_db

	if ( reset ):
		try:
			os.remove(DB_NAME)
		except OSError:
			pass

	con_db = sqlite.connect(DB_NAME)
	cur_db = con_db.cursor()

	if ( reset ):
		cur_db.execute(db_conf.SQL_CREATE_STR)
		con_db.commit()

def close():
	cur_db.close()
	con_db.close()

def insert_row(elapsed_time, method_name, requests, requests_ok, time, d_requests,
```

```
d_requests_ok,
                d_time, max_cur_freq, s_max_cur_freq, governor, battery_status, battery_charge):
        cur_db.execute(db_conf.SQL_INSERT_STR, (elapsed_time, method_name, requests,
requests_ok, time, d_requests, d_requests_ok,
                                                d_time, max_cur_freq, s_max_cur_freq, governor,
battery_status, battery_charge))
        con_db.commit()

def save_data(elapsed_time, method_name, requests, requests_ok, time, d_requests, d_requests_ok,
              d_time, max_cur_freq, s_max_cur_freq, governor, battery_status, battery_charge):
        init(False)
        insert_row(elapsed_time, method_name, requests, requests_ok, time, d_requests,
d_requests_ok,
                   d_time, max_cur_freq, s_max_cur_freq, governor, battery_status, battery_charge)
        close()

def reset_storage():
        init(True)
        close()

if __name__ == '__main__':
        # testing
        reset_storage()
        save_data(0, 'test', 1, 2, 3, 4, 5, 6, 7, 8, 9, 'test', 10)
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# db_conf.py
#

SQL_CREATE_STR = 'CREATE TABLE model (     \
                  id INTEGER PRIMARY KEY,  \
                  elapsed_time   INTEGER,  \
                  method_name    TEXT,     \
                  requests       INTEGER,  \
                  requests_ok    INTEGER,  \
                  time           REAL,     \
                  d_requests     INTEGER,  \
                  d_requests_ok  INTEGER,  \
                  d_time         REAL,     \
                  max_cur_freq   INTEGER,  \
                  s_max_cur_freq INTEGER,  \
                  governor       TEXT,     \
                  battery_status TEXT,     \
                  battery_charge INTEGER   );'

SQL_INSERT_STR = "INSERT INTO  model (elapsed_time, method_name, requests, requests_ok,
time, d_requests, d_requests_ok, \
                                      d_time, max_cur_freq, s_max_cur_freq, governor, battery_status,
battery_charge) \
                                      values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# error.py
#

import sys

def log_error_and_die (error):
        print ('ERROR : ' +  error)
        sys.exit(-1)
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# gobweb.py
#

import db
import cpu
import log
import acpi
import policy
import http_client
import time as clock

import gobweb_conf as gw_conf

def get_data():
        return cpu.get_cpu_info(), http_client.get_mvc_app_status(), acpi.get_battery_status()

def get_max_cur_freq_and_governor(cpu_info):

        cpu_winner  = 'null'
        freq_winner = 0

        for item in cpu_info:
                for name in item.keys():
                        freq = item[name]['cpuinfo_cur_freq']
                        if ( freq > freq_winner ):
                                freq_winner = freq
                                cpu_winner  = name

                s_freq_winner   = item[name]['scaling_cur_freq']
                governor_winner = item[name]['scaling_governor']

        return freq_winner, s_freq_winner, governor_winner

def store_data_with_delta(elapsed_time, current_mvc_data, last_mvc_data, cpu_info, battery_info):
```

```
        max_cur_freq, s_max_cur_freq, governor = get_max_cur_freq_and_governor(cpu_info)

        battery_status = battery_info['status']
        battery_charge = battery_info['charge'][:-2]

        last_stored_data = []

        for method_name in current_mvc_data.keys():
                requests      = int(current_mvc_data[method_name]['requests'])
        requests_ok   = int(current_mvc_data[method_name]['requests_ok'])
        time          = float(current_mvc_data[method_name]['time'])
                d_requests    = requests - int(last_mvc_data[method_name]['requests'])
                d_requests_ok = requests_ok - int(last_mvc_data[method_name]['requests_ok'])
                d_time        = time - float(last_mvc_data[method_name]['time'])
        db.save_data(elapsed_time, method_name, requests, requests_ok, time, d_requests,
d_requests_ok,
                    d_time, max_cur_freq, s_max_cur_freq, governor, battery_status, battery_charge
)
                last_stored_data.append( { 'elapsed_time'   : elapsed_time,
                                           'method_name'    : method_name,
                                           'requests'       : requests,
                                           'requests_ok'    : requests_ok,
                                           'time'           : time,
                                           'd_requests'     : d_requests,
                                           'd_requests_ok'  : d_requests_ok,
                                'd_time'         : d_time,
                                           'max_cur_freq'   : max_cur_freq,
                                           's_max_cur_freq' : s_max_cur_freq,
                                           'governor'       : governor,
                                           'battery_status' : battery_status,
                                           'battery_charge' : battery_charge } )

        return last_stored_data

def log_iteration():
        t = clock.strftime('%Y-%m-%d %H:%M:%S')
        log.output_screen(t),
        log.output_screen(gw_conf.ITERATION_MESSAGE)

def get_max_freq(cpu_info):
        freqs = cpu_info[0]['cpu0']['scaling_available_frequencies']
        max_freq = freqs[0]
        for freq in freqs:
                if max_freq < freq:
                        max_freq = freq
        return max_freq

def get_min_freq():
        freqs = cpu_info[0]['cpu0']['scaling_available_frequencies']
```

```
	min_freq = freqs[0]
	for freq in freqs:
		if min_freq > freq:
			min_freq = freq
	return min_freq

def get_next_freq(cur_freq):
	next_freq = cur_freq
	freqs = sorted(cpu_info[0]['cpu0']['scaling_available_frequencies'])
	for freq in freqs:
		if (freq > cur_freq):
			next_freq = freq
			break;
	return next_freq

def get_prev_freq(cur_freq):
	prev_freq = cur_freq
	freqs = reversed(sorted(cpu_info[0]['cpu0']['scaling_available_frequencies']))
	for freq in freqs:
		if (freq < cur_freq):
			prev_freq = freq
			break
	return prev_freq

def set_new_freq(next_freq):
	cpu.set_scaling_frequency(next_freq)

def change_new_freq_by_action(next_policy_action, cpu_info):

	next_freq = 0

	cur_freq, _, _ = get_max_cur_freq_and_governor(cpu_info)

	if   next_policy_action == 'max_freq':
		next_freq = get_max_freq(cpu_info)
	elif next_policy_action == 'min_freq':
		next_freq = get_min_freq()
	elif next_policy_action == 'inc_freq':
		next_freq = get_next_freq(cur_freq)
	elif next_policy_action == 'dec_freq':
		next_freq = get_prev_freq(cur_freq)
	else:
		next_freq = cur_freq

	if cur_freq != next_freq:
		set_new_freq(next_freq)

# main algorithm

db.reset_storage()
```

```
if ( gw_conf.SET_OS_GOVERNOR ):
        cpu.set_governor(gw_conf.OS_GOVERNOR)

cpu_info, last_mvc_data, battery_info = get_data()

elapsed_time = 0

store_data_with_delta(elapsed_time, last_mvc_data, last_mvc_data, cpu_info, battery_info)

private_policy_data = {}

while True:

        clock.sleep(gw_conf.SAMPLING_TIME)

        elapsed_time = elapsed_time + gw_conf.SAMPLING_TIME

        cpu_info, current_mvc_data, battery_info = get_data()

        if gw_conf.LOG_ITERATION:
                log_iteration()

        last_stored_data = store_data_with_delta(elapsed_time, current_mvc_data, last_mvc_data,
cpu_info, battery_info)

        next_policy_action, private_policy_data =
policy.get_next_action_by_policy(last_stored_data, private_policy_data)

        if policy.current_freq_threshold == gw_conf.FREQ_THRESHOLD:
                change_new_freq_by_action(next_policy_action, cpu_info)

        last_mvc_data = current_mvc_data
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# gobweb_conf.py
#

# sampling time
SAMPLING_TIME = 1

# log iteration
LOG_ITERATION = False

# progress message
ITERATION_MESSAGE = 'New iteration running ...'

# governor
OS_GOVERNOR = 'userspace'

# set userspace governor if needed
SET_OS_GOVERNOR = True

# set freq checking threshold
FREQ_THRESHOLD = 5

# set change threshold (%)
CHANGE_THRESHOLD = 5
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# http_client.py
#

import urllib2

SHOW_STATUS_REQUEST  = 'http://localhost/index.php?pm=show_status'
RESET_STATUS_REQUEST = 'http://localhost/index.php?pm=reset_status'

def __read_remote_data(request):
        f = urllib2.urlopen(request)
        reply = f.read()
        return reply

def fetch_mvc_app_status():
        data = __read_remote_data(SHOW_STATUS_REQUEST)
        return data

def get_mvc_app_status():
        data = fetch_mvc_app_status()
        data = data[15:-2]
        data = data.split()
        data = [item for item in data if item != ':']
        ndata = []
        for item in data:
                if item.endswith('},'):
                        ndata.append(item[:-1])
                else:
                        ndata.append(item)
        keys   = ndata[::2]
        values = ndata[1::2]
        dic = {}
        for i in range(0, len(keys)):
                dic[keys[i]] = values[i]
        for key in dic.keys():
```

```
                entry = dic[key][1:-1].split(',')
                dic[key] = { 'requests'    : entry[0],
                             'requests_ok' : entry[1],
                             'time'        : entry[2] }
        return dic

def reset_mvc_app_status():
        data = __read_remote_data(RESET_STATUS_REQUEST)
        return data

if __name__ == '__main__':
        # testing
        print(fetch_mvc_app_status())
        print(get_mvc_app_status())
        print(reset_mvc_app_status())
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# log.py
#

def output_screen_ (message):
        print (message),

def output_screen (message):
        print (message)
```

```
#
# Copyright (C) 2011 by Javier M. Mellid <jmunhoz@igalia.com>
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE. See the GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51 Franklin
# Street, Fifth Floor, Boston, MA  02110-1301, USA.
#
# policy.py
#

import gobweb_conf as gw_conf
import math
import log

current_freq_threshold = gw_conf.FREQ_THRESHOLD

def get_next_policy_action(current_delta_pending_requests, requests_tendency, current_delta_time,
time_tendency):

        next_policy_action = 'none'

        if current_delta_pending_requests > 0:
                if requests_tendency > 0:
                        if requests_tendency > (current_delta_pending_requests *
gw_conf.CHANGE_THRESHOLD / 100):
                                next_policy_action = 'max_freq'
                        else:
                                next_policy_action = 'inc_freq'
                else:
                        if time_tendency > 0:
                                if time_tendency > (math.fabs(current_delta_time) *
gw_conf.CHANGE_THRESHOLD / 100):
                                        next_policy_action = 'inc_freq'
                                else:
                                        next_policy_action = 'dec_freq'
                        else:
                                if (math.fabs(time_tendency) > (math.fabs(current_delta_time) *
gw_conf.CHANGE_THRESHOLD / 100)):
                                        next_policy_action = 'min_freq'
                                else:
                                        next_policy_action = 'dec_freq'
```

```
        else:
                next_policy_action = 'min_freq'

        return next_policy_action

def get_next_action_by_policy(info, data):

        global current_freq_threshold

        policy_action = 'no_new_freq'

        # do we need to run?
        if current_freq_threshold != 0:
                current_freq_threshold = current_freq_threshold - 1
                return policy_action, data

        current_freq_threshold = gw_conf.FREQ_THRESHOLD

        # choose worst method

        worst_entry = { 'method_name' : 'idle', 'requests' : 0, 'requests_ok' : 0 }

        for entry in info:
                current_pending_requests = worst_entry['requests'] - worst_entry['requests_ok']
                entry_pending_requests = entry['requests'] - entry['requests_ok']
                if current_pending_requests < entry_pending_requests:
                        worst_entry = entry

        # retrieve last selected entry
        if data.has_key('last_selected_entry'):
                last_selected_entry = data['last_selected_entry']
        else:
                last_selected_entry = worst_entry

        # save new last selected entry
        data['last_selected_entry'] = worst_entry

        # idle?
        if worst_entry['method_name'] == 'idle':
                log.output_screen('policy: idle')
                next_policy_action = get_next_policy_action(0, 0, 0, 0)
                return next_policy_action, data

        # new worst method detected?
        if worst_entry['method_name'] != last_selected_entry['method_name']:
                log.output_screen('policy: new worst method detected')
                return policy_action, data

        # calculate reqs deltas
        current_delta_pending_requests = worst_entry['requests'] - worst_entry['requests_ok']
```

```
        old_delta_pending_requests = last_selected_entry['requests'] -
last_selected_entry['requests_ok']
        requests_tendency = current_delta_pending_requests - old_delta_pending_requests

        # calculate time deltas
        current_delta_time = worst_entry['d_time']
        old_delta_time = last_selected_entry['d_time']
        time_tendency = current_delta_time - old_delta_time

        # take proper action
        policy_action = get_next_policy_action(current_delta_pending_requests, requests_tendency,
current_delta_time, time_tendency)

        # log action
        log.output_screen_ ('policy: making decisions on ' + worst_entry['method_name'])
        log.output_screen_ (', current_delta_pending_requests : ' +
str(current_delta_pending_requests))
        log.output_screen_ (', requests_tendency : ' + str(requests_tendency))
        log.output_screen_ (', current_delta_time : ' + str(current_delta_time))
        log.output_screen  (', time_tendency : ' + str(time_tendency))
        log.output_screen  ('policy: policy_action : ' + policy_action)

        return policy_action, data
```

Anexo D. Planificación

Fases del proyecto

01. Creación del plan de trabajo

El resultado final de esta actividad generó el plan de trabajo a seguir durante el proyecto.

02. Trabajo de documentación e investigación

Durante esta fase se realizó trabajo de documentación e investigación sobre el dominio en estudio con la intención de estructurar más en detalle y refinar los diferentes aspectos de cada una de las áreas bajo estudio.

03. Estudio interface ACPI en Linux

Estudio del nivel de soporte para la interface Avanzada de Configuración y Energía (ACPI) en el *kernel Linux*. En esta tarea se abordó el estándar ACPI, su nivel de madurez, grado de implementación y particularidades en el diseño del *kernel Linux*.

04. Documentar interface ACPI en Linux

En esta fase se llevó a cabo la documentación y materialización de la fase 03 de cara a su consolidación en la memoria. El material generado fué incompleto y estaba formado por anotaciones, bocetos y gráficos preliminares.

05. Estudio interface gestión PM en GNU/Linux

Estudio de la interface de gestión de energía con sistemas operativos basados en *GNU/Linux*. En esta tarea se abordó la interface de gestión de energía que ofrece el sistema operativo a sus usuarios y subsistemas, comandos, limitaciones, casos de uso habituales y mejores prácticas.

06. Documentar interface gestión PM en GNU/Linux

En esta fase se llevó a cabo la documentación y materialización de la fase 05 de cara a su consolidación en la memoria. El material generado fué incompleto y estaba formado por anotaciones, bocetos y gráficos preliminares.

07. Estudio entorno desarrollo *drivers* PM en Linux

Estudio de las estructuras, interface y lógica de implementación (*frameworks*) disponibles en el *kernel Linux* para su uso en *drivers*. En esta tarea se abordaron detalles de desarrollo a bajo nivel, se estudió la dotación e interacción de código fuente implementado en *drivers* y como el actual soporte de gestión de energía hace uso de él. En concreto, se estudiaron los diferentes estados, modelos y fases que un *driver* debe soportar y gestionar para una correcta gestión energética.

08. Documentar entorno desarrollo *drivers* PM en Linux

En esta fase se llevó a cabo la documentación y materialización de la fase 07 de cara a su consolidación en la memoria. El material generado fué incompleto y estaba formado por anotaciones, bocetos y gráficos preliminares.

09. Análisis y estudio de un *driver* tipo implementando PM

Análisis y estudio de un *driver* tipo implementando gestión de energía. Se analizó y estudió el código de *drivers* utilizando el soporte de gestión energética y el API del *kernel* con la intención de factorizar un esqueleto básico común que permitiese un punto de inicio en el soporte de gestión de energía para *drivers* similares.

10. Documentar análisis y estudio de un *driver* tipo implementando PM

En esta fase se llevó a cabo la documentación y materialización de la fase 09 de cara a su consolidación en la memoria. El material generado fué incompleto y estaba formado por anotaciones, bocetos y gráficos preliminares.

11. Desarrollo mejoras *driver* sm7xx

En esta fase se llevó a cabo el desarrollo de las mejoras para el *driver* sm7xx, diversas pruebas y el trabajo de comunicación para su integración final en el *kernel Linux*.

12. Desarrollo gobernador web

En esta fase se llevó a cabo el desarrollo del gobernador web junto al estudio de heurísticas de gobierno candidatas.

13. Evaluación gobernador web

En esta fase se llevó a cabo la evaluación del gobernador web; la construcción de escenarios, simulaciones de carga y generación de resultados gráficos.

14. Integración y generación de documentación final

En esta fase se integró y generó la documentación final correspondiente a las fases 04, 06, 08, 10, 11 y 13. A través de sucesivas iteraciones sobre las mismas se obtuvo una documentación final homogénea, compacta y cohesiva con la intención de reducir contenidos superfluos y garantizar la calidad de la misma en cuanto a presentación, contenido y estructura.

15. Entrega memoria

Este hito de control marca la entrega de la memoria final revisada.

16. Control PEC1

Hito de control y seguimiento.

17. Control PEC2

Hito de control y seguimiento.

18. Control PEC3

Hito de control y seguimiento.

Seguimiento y desvío sobre la planificación original

La planificación actual con la que se finalizó el presente proyecto difiere de la original. Las principales correcciones son fruto de la naturaleza exploratoria del proyecto y el estado de la actual implementación del sistema operativo *GNU/Linux*. Esto último marcó el nivel de progreso en la segunda mitad del proyecto y ayudó a refinar objetivos técnicos en detalle.

En general, el proyecto avanzó de acuerdo a la planificación original hasta la semana 17, con anterioridad a entrar en los objetivos prácticos perseguidos.

En la semana 18, se evaluó la posibilidad de trabajar en dos líneas en paralelo dentro de los objetivos establecidos: el desarrollo y contribución de mejoras para un *driver* existente en el *kernel Linux* al igual que el gobierno de frecuencias de CPU en base a la carga computacional de una aplicación web.

El punto anterior provocó que se siguiese una nueva aproximación en la planificación. Se decidió que la tarea con mayor riesgo era el desarrollo de las mejoras para el *driver*. Existía incertidumbre en la obtención de conocimiento y desarrollo de las mejoras en cuanto al alcance temporal del proyecto y la obtención de conocimiento que sucedería en paralelo.

Así, se incluyó el estudio y desarrollo de un gobernador web secuencialmente a lo largo de las tareas previamente existentes mientras el desarrollo de las mejoras para el *driver* se realizaría en paralelo a lo largo de cinco semanas.

La tarea de desarrollo de mejoras se definió como genérica y en la misma tuvo lugar trabajo de análisis, desarrollo, pruebas, integración y comunicación con la comunidad *GNU/Linux*.

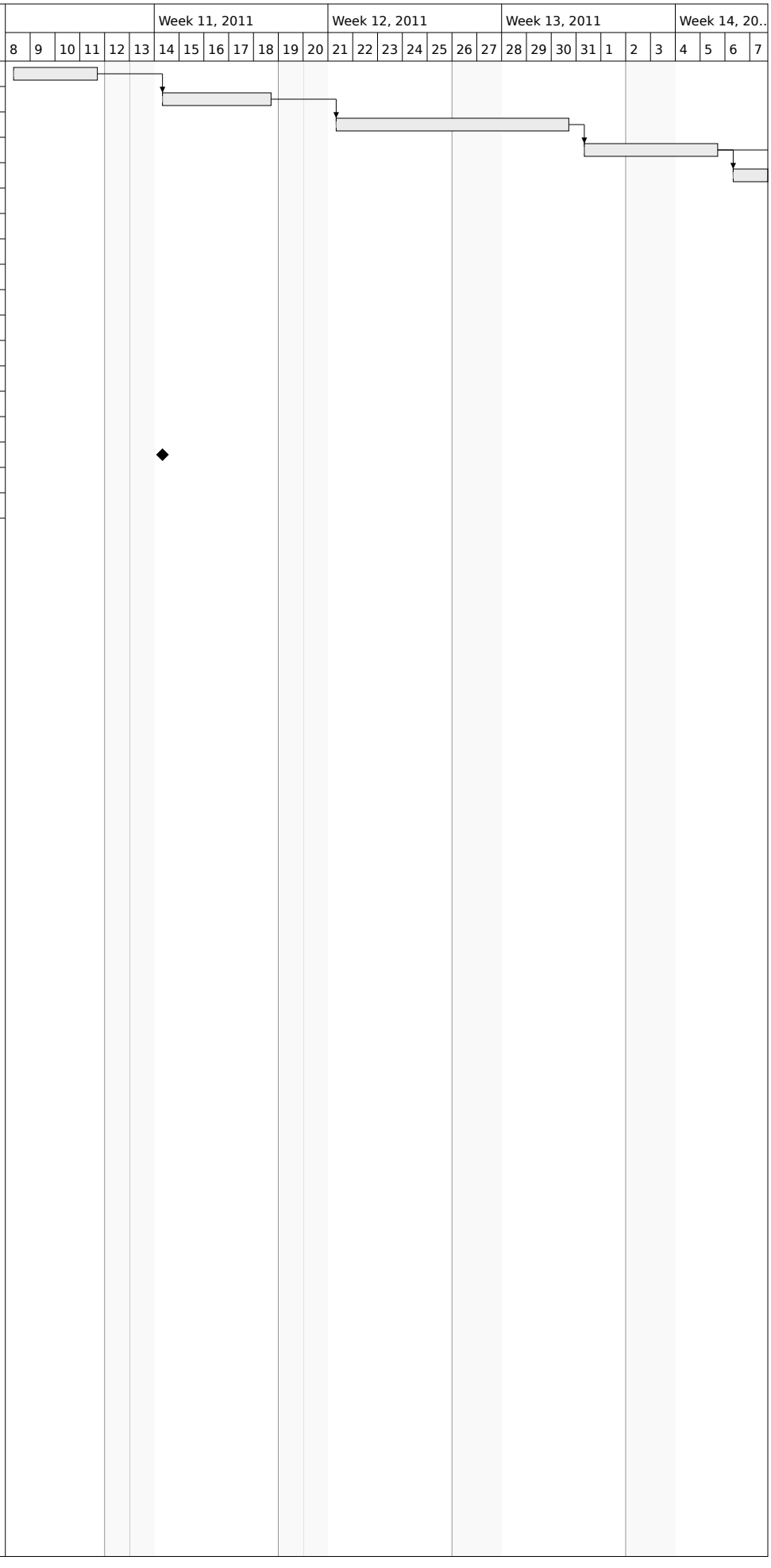
Mencionar también que el desarrollo y evaluación del gobernador web fueron tareas que se realizaron en paralelo. El estudio y exploración de heurísticas provocaron que gran parte del proceso técnico de evaluación se llevase a cabo en los primeros pasos de desarrollo para validar que el progreso era adecuado y coherente. Esto permitió obtener también *feedback* de seguimiento y confirmar que la línea de avance era la adecuada.

No existieron grandes retrasos o imprevistos en el transcurso del proyecto. La comunicación fué ágil y fluída a lo largo del mismo. Todos los hitos fueron alcanzados con holgura.

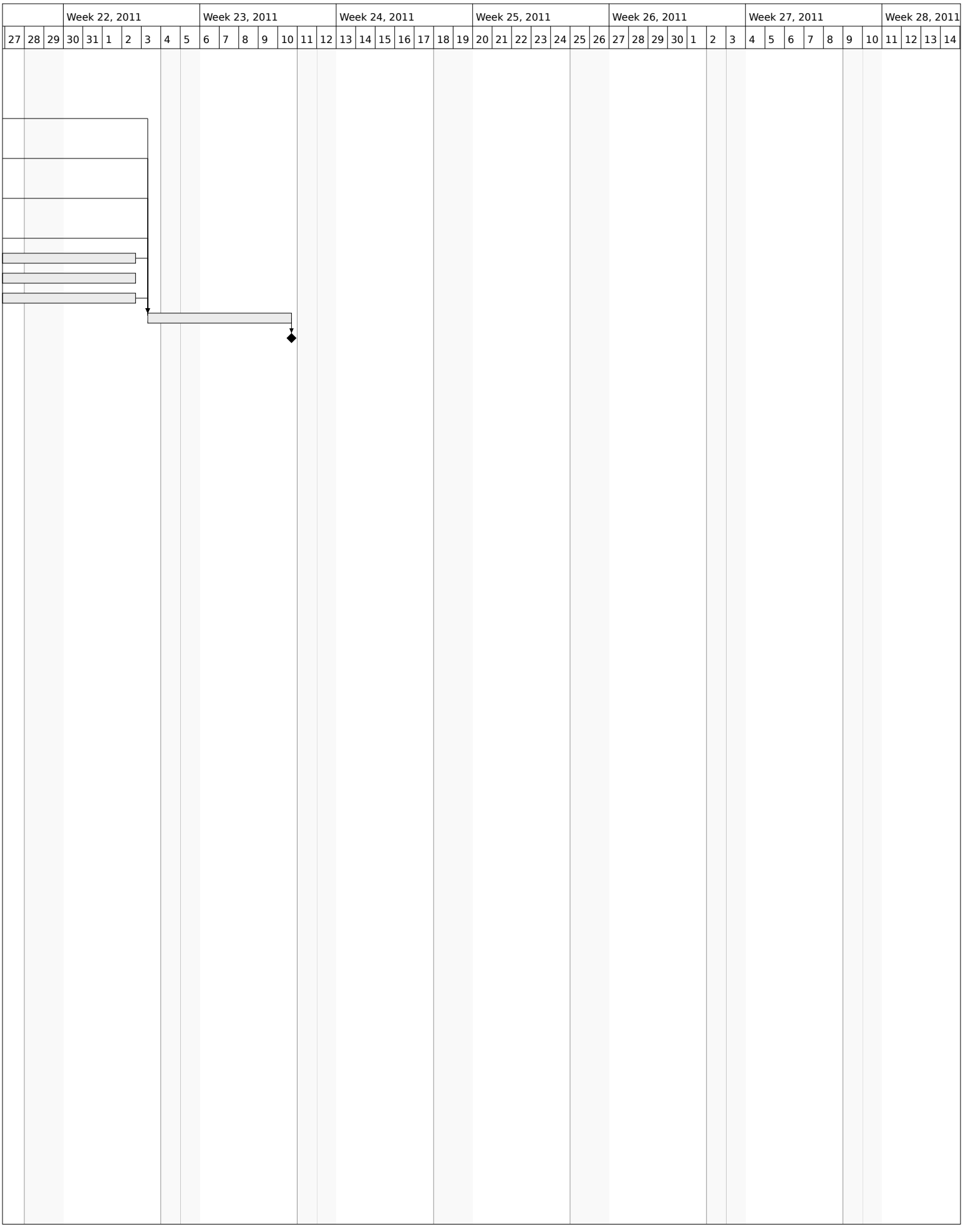
Diagrama de Gantt y Calendario

A continuación se muestra el calendario y diagrama de Gantt empleado para el control y seguimiento del presente proyecto.

Name	Work
Creación del plan de trabajo	4d
Trabajo de documentación e investigación	5d
Estudio interface ACPI en Linux	8d
Documentar interface ACPI en Linux	4d
Estudio interface gestión PM en GNU/Linux	8d
Documentar interface gestión PM en GNU/Linux	4d
Estudio entorno desarrollo drivers PM en Linux	8d
Documentar entorno desarrollo drivers PM en Linux	4d
Análisis y estudio de un driver tipo implementando PM	2d
Documentar análisis y estudio de un driver tipo imp. PM	1d
Desarrollo mejoras driver sm7xx	25d
Desarrollo gobernador web	10d
Evaluación gobernador web	10d
Integración y generación de documentación final	6d
Entrega memoria	
Control PEC1	
Control PEC2	
Control PEC3	




Week 15, 2011
Week 16, 2011
Week 17, 2011
Week 18, 2011
Week 19, 2011
Week 20, 2011
Week 21, 2011
8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26


Week 22, 2011
Week 23, 2011
Week 24, 2011
Week 25, 2011
Week 26, 2011
Week 27, 2011
Week 28, 2011
27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14

WBS	Name	Start	Finish	Work	Duration	Slack	Cost	Assigned to	% Complete
1	Creación del plan de trabajo	Mar 8	Mar 11	4d	4d		0		0
2	Trabajo de documentación e investigación	Mar 14	Mar 18	5d	5d		0		0
3	Estudio interface ACPI en Linux	Mar 21	Mar 30	8d	8d		0		0
4	Documentar interface ACPI en Linux	Mar 31	Apr 5	4d	4d		0		0
5	Estudio interface gestión PM en GNU/Linux	Apr 6	Apr 15	8d	8d		0		0
6	Documentar interface gestión PM en GNU/Linux	Apr 25	Apr 28	4d	4d		0		0
7	Estudio entorno desarrollo drivers PM en Linux	Apr 29	May 10	8d	8d		0		0
8	Documentar entorno desarrollo drivers PM en Linux	May 11	May 16	4d	4d		0		0
9	Análisis y estudio de un driver tipo implementando PM	May 17	May 18	2d	2d		0		0
10	Documentar análisis y estudio de un driver tipo imp. PM	May 19	May 19	1d	1d		0		0
11	Desarrollo mejoras driver sm7xx	Apr 29	Jun 2	25d	25d		0		0
12	Desarrollo gobernador web	May 20	Jun 2	10d	10d		0		0
13	Evaluación gobernador web	May 20	Jun 2	10d	10d		0		0
14	Integración y generación de documentación final	Jun 3	Jun 10	6d	6d		0		0
15	Entrega memoria	Jun 10	Jun 10	N/A	N/A		0		0
16	Control PEC1	Mar 14	Mar 13	N/A	N/A	60d	0		0
17	Control PEC2	Apr 15	Apr 15	N/A	N/A	36d	0		0
18	Control PEC3	May 19	May 19	N/A	N/A	17d	0		0