Universidad de Almería

Titulación de Ingeniero en Informática

Virtualización de servidores de telefonía IP en GNU/Linux

Autor:
Eugenio Eduardo Villar Fernández

Tutores:
Julio Gómez López
Francisco Gil Montoya

Almería, junio 2010

Tanto la memoria de este trabajo como el software desarrollado se distribuyen bajo la licencia GNU GPL v3.

La Licencia Pública General GNU (GNU GPL) es una licencia libre, sin derechos para software y otro tipo de trabajos.

Las licencias para la mayoría del software y otros trabajos prácticos están destinadas a suprimir la libertad de compartir y modificar esos trabajos. Por el contrario, la Licencia

Pública General GNU persigue garantizar su libertad para compartir y modificar todas las versiones de un programa--y asegurar que permanecerá como software libre para todos sus usuarios.

Cuando hablamos de software libre, nos referimos a libertad, no a precio. Las

Licencias Públicas Generales están destinadas a garantizar la libertad de distribuir copias de software libre (y cobrar por ello si quiere), a recibir el código fuente o poder conseguirlo si así lo desea, a modificar el software o usar parte del mismo en nuevos programas libres, y a saber que puede hacer estas cosas.

Para obtener más información sobre las licencias y sus términos puede consultar:

- http://www.gnu.org/licenses/gpl.html (Licencia original en inglés)
- http://www.viti.es/gnu/licenses/gpl.html (Traducción de la licencia al castellano)

Quiero agradecer este trabajo de forma especial a mi abuelo *Eugenio*, por darme su apoyo incondicional en todo momento de mi vida y por enseñarme que *cada escalón debe ser una meta, pero sin que esa meta deje de ser un nuevo escalón.*

A *Carmen* por ser mi compañera de viaje, con el mapa en sus manos, tanto en los buenos momentos como en los no tan buenos, por su ánimo constante con una sonrisa enorme en la cara. A mi familia al completo: mi padre *Jesús* y mi madre *Teodora* por darme esta gran oportunidad, mi hermano *Jesús*, mis hermanas *Antonia*, *Lourdes y Hanna*, y mi abuela *Antonia*. Porque siempre estáis ahí pase lo que pase.

También quiero recordar en estas líneas a mis compañeros durante todos estos años en la Universidad: *Carlinjos, Méndez* y *Popo*. Gracias también a mis amigos de toda la vida y de hoy: *Ginés, Miki, Álvaro, María Ferrer, Lucía, Jose, Juanma, Lorena, Carlos y Pedro*. Sois lo más rock and roll de por aquí. A mis hermanos de la experiencia Erasmus por hacerme sentir que responsabilidad y locura van cogidos de la mano: *Mario "Araña", Brais, Álex, Matteo, Cris, Jordi, Morencos, todas las niñas...* Quiero dar las gracias también a mis compañeros en Enseñanzas Propias y, como no, a mis dos tutores y *gurús* en todo este trabajo, *Julio y Paco*. Sin su experiencia y consejo nunca hubiera sido posible llegar a buen puerto.

Finalmente, a la banda sonora y las imágenes que me han acompañado durante todos estos últimos meses: *El incendio, Aviones, Daiquiri Blues, Las consecuencias, Dexter y Los Soprano.*

A todos ellos, gracias de todo corazón.

INDICE

INTRODUCCION

La virtualización es sin duda alguna la tecnologías con más calado y más demandadas actualmente en la sociedad que está revolucionando el sector de las tecnologías de la información porque está transformando su modelo de trabajo prácticamente a todos los niveles. Virtualizar, en una frase, es lograr la ejecución de máquinas –con sistemas operativos, aplicaciones y servicios propios- dentro de otras máquinas. Es aquí donde yace la diferencia con otras técnicas conocidas también con el sobrenombre de virtualización, aunque bien distintas; como por ejemplo el *Grid Computing o Server Agregation (Agregación de Servidores)*, que a grandes rasgos consiste en lograr la percepción de múltiples servidores como uno sólo.

La tecnología de virtualización, con más de cuarenta años de antigüedad, ha estado presente de forma minoritaria en grandes centros de cálculo. Pero recientemente, el uso de esta tecnología *explotó* debido a la mejora notable del hardware, a su reducción de precio, y a la evolución del software. Partiendo de esta base, se comenzó el desarrollo de grandes proyectos de virtualización tanto privativos como libres, lo que provocó la llegada de la virtualización al escritorio y el consiguiente aumento de popularidad debido a las enormes ventajas que supone su implantación.

La tecnología que ofrece la virtualización permite configurar, instalar, y administrar instancias de múltiples máquinas lógicas (máquinas virtuales o sistemas invitados) con sus respectivos sistemas operativos y aplicaciones, con acceso a los recursos físicos del servidor anfitrión. A diferencia de la emulación por hardware y la simulación, la virtualización convencional necesita de al menos una capa de abstracción intermedia o modificaciones en el sistema operativo anfitrión que permitan el acceso a los recursos de la máquina anfitriona por parte de las máquinas virtuales invitadas. Esta capa acepta diferentes nombres dependiendo de la solución de virtualización en uso, aunque generalmente suele ser llamada *hipervisor* (*Hypervisor* en inglés) o *monitor de máquinas virtuales* (*VMM, Virtual Machine Monitor*).

Partiendo de una arquitectura general de virtualización existen diferenciaciones que provocan la aparición de distintos modelos en práctica. Estas diferencias, sobre todo en el sistema operativo anfitrión y su intervención en el proceso, la existencia o no de hipervisor, la transparencia en la virtualización para las máquinas virtuales o la complejidad de la virtualización y complementación con otras técnicas (por ejemplo, emulación hardware) han provocado que estos modelos se adecúen a distintos casos en los que queramos implantar la virtualización de una manera distinta también, y que ofrezcan rendimiento diferente lo que los

hace recomendables dependiendo de cada situación particular y necesidades. Existen muchas técnicas y modelos de virtualización entre los que debemos elegir el que mejor se adapte a nuestras necesidades como por ejemplo: virtualización completa, paravirtualización, virtualización a nivel de sistema operativo, a nivel del kernel de Linux, etc.

Debido a la multitud de modelos existentes y la accesibilidad de la que dispone, la virtualización está siendo muy popular en los departamentos informáticos de las empresas. Las posibilidades a la hora de implantar un proyecto de virtualización son muy grandes: prácticamente hay un modelo aplicable a la solución que necesitemos; los campos y sectores en los que es aplicable son numerosos.

La aplicación más popular en la actualidad es sin duda la **consolidación de servidores**: *virtualización o encapsulación de servidores en máquinas virtuales*. La causa más importante que origina la necesidad de consolidar servidores es la preocupación por el bajo porcentaje de utilización de las capacidades de los servidores unido al alto consumo de energía que los acompaña –crecen en número, ocupan mayor espacio, pero permanecen infrautilizados- sumado al alto coste del configuración, mantenimiento, administración y soporte que supone un gran número de servidores. Aplicar virtualización resulta entonces necesario y fundamental, ya que permitirá la reducción de costes en todos los ámbitos: espacio utilizado, energía, administración, gestión, recuperación ante desastres… garantizando una mejor eficiencia energética y un mayor uso de los recursos de los servidores al integrar de manera virtual múltiples instancias virtuales de éstos en un sólo servidor anfitrión físico.

El objetivo general del proyecto no es otro que la virtualización de servicios de telefonía IP (consolidación de servidores centralita VoIP). La telefonía IP (*Voice over Internet Protocol*), nacida de la necesidad y la utilidad de hacer uso de una sola red para la transmisión de datos en lo que a telefonía se refiere, es la comunicación que viaja a través de la red permitiendo un ahorro elevado tanto en costes de comunicación como en costes de mantenimiento de las infraestructuras de telefonía, teniendo al mismo tiempo grandes posibilidades de escalabilidad, portabilidad, e integración con las redes locales internas y recursos de las empresas.

De entre las diferentes soluciones software disponibles que implementan centralitas telefónicas VoIP destaca sin duda alguna *Asterisk*, aplicación de software libre bajo la licencia GPL (General Public License). *Asterisk* proporciona características que le brindan un gran potencial; por ejemplo, buzón de voz, salas de conferencias, distribución y gestión automática de llamadas, contestador automático, interacción con la red de telefonía tradicional, reconocimiento de llamadas, posibilidad de creación de números virtuales, entre otras.

La combinación de virtualización y telefonía IP puede ofrecer unos resultados realmente excelentes explotando y uniendo las ventajas de uno y otro lado. Sin embargo, existen desventajas derivadas de esta asociación como el rendimiento final obtenido -muy importante si se quiere dotar al servicio de telefonía IP por ejemplo de alta disponibilidad o alto rendimiento, en ocasiones puede resultar dudoso dependiendo del tipo de virtualización aplicada-, problemas con el uso del hardware específico de telefonía IP por parte de las máquinas virtuales, o el estado del reloj de la centralita virtualizada. Por tanto, las ventajas y desventajas derivadas del uso combinado de virtualización y telefonía IP y su exploración es el objeto principal de estudio en este proyecto.

A continuación podemos ver de forma introductoria la estructura de los diferentes capítulos que componen esta obra:

- **Capítulo 1. Introducción a la virtualización.** En el primer capítulo vamos a presentar las tecnologías de virtualización desde una perspectiva histórica: pasado, presente y futuro. Conocemos los conceptos básicos para comenzar a introducirnos

en el mundo de la virtualización, los modelos actualmente existentes, y son listados los objetivos del proyecto.

- **Capítulo 2. Virtualización de plataforma.** Comprende el estado del arte de las diferentes aproximaciones disponibles en virtualización de plataforma y los sistemas de almacenamiento generalmente usados al consolidar servidores. Se comparan las diferentes posibilidades para determinar una solución de virtualización a implementar para la consolidación de servidores de telefonía IP.

- **Capítulo 3. Xen.** Analizamos en profundidad las características de la más importante de las soluciones de virtualización libres del momento (concretamente siguiendo el modelo de paravirtualización). Vemos cómo realizar su instalación y configuración, la puesta en marcha de dominios *domU* (máquinas virtuales) y cómo es posible monitorizar su estado mediante herramientas gráficas y en modo texto.

- **Capítulo 4. Introducción a la telefonía IP y Asterisk.** En el capítulo cuarto podemos conocer en qué consiste la telefonía IP y *Asterisk*, la solución de centralitas software libre que está revolucionando el sector en los últimos años. Vemos cómo podemos realizar una instalación y configuración básica de *Asterisk*.

- **Capítulo 5. Diseño, implementación y prueba de una infraestructura virtual de telefonía IP.** Comprende las fases de desarrollo práctico del proyecto: análisis y diseño, implementación y prueba de la infraestructura virtual Xen de telefonía IP utilizando *Asterisk*. El objetivo principal es conocer las ventajas y desventajas de la aplicación de la virtualización en la consolidación de este tipo de servidores y obtener conclusiones reales sobre distintas configuraciones gracias a la realización de pruebas de rendimiento.

- **Capítulo 6. Alta disponibilidad en clústeres virtuales.** Finalmente en el sexto capítulo recogemos el desarrollo de prácticas y utilización de herramientas necesarios para dotar a un clúster virtual de alta disponibilidad. Para ello utilizadmos la solución software de alta disponibilidad *Heartbeat*, perfectamente integrada con *Asterisk*, y aplicamos técnicas de migración de dominios virtuales entre diferentes sistemas anfitriones bajo distintas circunstancias.

CAPITULO 1

INTRODUCCION A LA VIRTUALIZACION

En este primer capítulo vamos a ver información general sobre el término *virtualización*, sus diferentes aproximaciones teóricas y sus aplicaciones prácticas hoy en día. Después de hacer una rápida reseña histórica, podemos comprobar como la *virtualización* no es un término nuevo y que además en informática podemos encontrar muchos ámbitos en los que es aplicable la característica *virtual*.

Después, veremos el paso crucial dado con la creación de tecnologías hardware como soporte para la virtualización, *Intel VT* y *AMD*-V, y nos centraremos en qué consiste la *virtualización* en nuestros días bajo el punto de vista de la empresa actual: este es el campo sobre el que desarrollaré el Proyecto Fin de Carrera y no es otro que, simplificándolo en una frase, *la ejecución de servicios en máquinas dentro de otras máquinas*. Aquí introduciremos los conceptos de *máquina virtual* e *hipervisor*, y exploraremos de manera breve la gran variedad de modelos de virtualización existentes como lo son la *virtualización de plataforma, de recursos, aplicaciones y escritorio.* Cada uno de los anteriores puede derivar en varios paradigmas y modos de operar que clasificaremos con detenimiento, haciendo especial hincapié en las categorías englobadas dentro del ámbito de la *virtualización de plataforma*: *virtualización completa, paravirtualización, virtualización a nivel del sistema operativo, a nivel del kernel, emulación y sistemas invitados*.

Virtualizar aporta ventajas y posibilidades únicas en la actualidad. Permite reducir costes en prácticamente todos los campos de actuación de la administración de sistemas; desde la instalación y configuración de equipos hasta los procesos de copias de seguridad, monitorización, gestión y administración de la infraestructura. Disminuye el número de servidores físicos necesarios y el porcentaje de desuso de los recursos de los que disponen, aumentando su eficiencia energética. También nos brinda la posibilidad de centralizar y automatizar procesos cuya administración normalmente consume mucho tiempo, pudiendo aprovisionar y migrar máquinas virtuales de una manera rápida y fiable, manteniendo alta la calidad del servicio y bajo el tiempo de respuesta ante una caída del mismo.

Las técnicas de virtualización unidas a otras herramientas disponibles pueden garantizar requerimientos que de otro modo serían difíciles de cubrir, al menos de manera tan sencilla, como son por ejemplo alta disponibilidad y alto rendimiento. En nuestro caso serán aplicadas a servicios de telefonía IP, por lo que es también de vital importancia revisar el estado actual de este tipo de servicios y más concretamente de *Asterisk*. *Asterisk*, desde su aparición, ha revolucionado por completo el concepto de servicios de telefonía IP debido a sus ventajosas prestaciones proporcionando características telefónicas que lo dotan de un gran potencial, como escalabilidad, portabilidad y flexibilidad, o su integración con Internet y redes locales. Además de todo ello, que no es poco, *Asterisk* supone una gran reducción en costes fundamentalmente debido al uso de Internet como medio de transporte para las comunicaciones, al mismo tiempo que posee un gran abanico de características diferenciadoras como gestión y distribución automática de llamadas, buzones de voz, salas de conferencias, fax virtuales… Tanto la telefonía IP en general como *Asterisk* en particular serán estudiados en el capítulo cuarto *Introducción a la telefonía IP y Asterisk*.

La combinación de virtualización y telefonía IP puede ofrecer unos resultados realmente espectaculares uniendo las ventajas que derivan del uso de uno y otro. Sin embargo, existen desventajas derivadas de esta asociación. Sirva este capítulo como introducción teórica al primer pilar de esta unión, la virtualización, y como análisis previo y necesario para el resto del desarrollo del proyecto.

1 Historia de la Virtualización

En este primer apartado profundizaremos en el concepto de virtualización haciendo un pequeño repaso histórico de la tecnología, no tan reciente como podríamos pensar. Viendo sus orígenes y entendiendo las causas que provocaron que evolucionara como lo ha hecho, llegaremos al punto en el que hoy en día nos encontramos: un amplio abanico de soluciones que encajan perfectamente en las necesidades actuales de las empresas. Esta variedad de soluciones, clasificadas en diferentes modelos, ha provocado que normalmente se confunda el término virtualización con otras actividades con objetivos similares. Haremos una parada en las tecnologías hardware desarrolladas tanto por Intel (*Intel VT*) como por AMD (AMD-V), cuya aparición fue y sigue siendo una referencia muy importante en ciertos casos en los que la solución de virtualización a aplicar requiera de su presencia como soporte físico y asistir en el proceso. Esto ocurre, por ejemplo, cuando queremos aplicar *virtualización completa* con máquinas virtuales que alojen sistemas operativos *invitados sin modificar*. Finalmente, estableceremos las bases de porqué la virtualización juega hoy día un papel tan importante y porqué lo seguirá haciendo en el futuro.

1.1 ANTECEDENTES

Virtualizar ha sido considerado históricamente y de manera general como tomar *algo* en cierto estado y hacer parecer que se encuentra en otro estado diferente. A partir de ello, dos aproximaciones han ido evolucionando: hacer parecer que un computador se trata de múltiples computadores y no solamente de uno –virtualización- o lograr que múltiples computadores sean uno sólo; esto, más que virtualización, comúnmente es llamado *Grid Computing* o *Server Aggregation*.

La virtualización no es un tema novedoso en informática, de hecho se considera que existe, aproximadamente, desde hace cuatro o cinco décadas. Por aquel entonces y hasta hace pocos años era aplicada en ámbitos exclusivos, sólo prácticamente para los grandes centros de cálculo, tanto bancarios como militares y universitarios. Algunos de los usos pioneros de la virtualización incluyen al *IBM 7044* (en el que la máquina física era la *M44*, que albergaba varias máquinas lógicas *44X* para los procesos) el *CTSS (Compatible Time Sharing System)*

desarrollado por el MIT *(Massachusetts Institute of Technology)* en el *IBM 7044*, y el proyecto *Atlas* de la Manchester University (uno de los primeros supercomputadores del mundo, operativo en 1962), pionero en el uso de paginación bajo demanda y llamadas en modo supervisor.

Figura 1.1 Computadores IBM 7040 y 7044 en un centro de computación en 1964.

El **proyecto *Atlas*** tuvo especial importancia ya que Christopher Strachey incluyó en él características novedosas para la época (años sesenta) y que venían a solucionar los graves problemas surgidos del uso común de un único ordenador por parte de muchos trabajadores a través de terminales. Básicamente consistía en un mecanismo para el reparto y uso al mismo tiempo de los recursos del computador (fundamentalmente procesador y disco), y la seguridad y fiabilidad de que el trabajo de un empleado no interfiriera en el de los otros. En la época de los mainframes, estas cuestiones superaban en importancia al rendimiento en la rapidez de los resultados. Así es como **nació la virtualización, con la necesidad de particionar recursos de disco, memoria y capacidad de cómputo**. Estas particiones (máquinas virtuales) podrían acoger una instancia de un sistema operativo, comunicarse a través de red, usar sus recursos o utilizar los del resto en el que caso de que no estén ocupados, se podrían tomar *imágenes* de su estado, o incluso ser migradas entre distintos servidores que las alojaran.

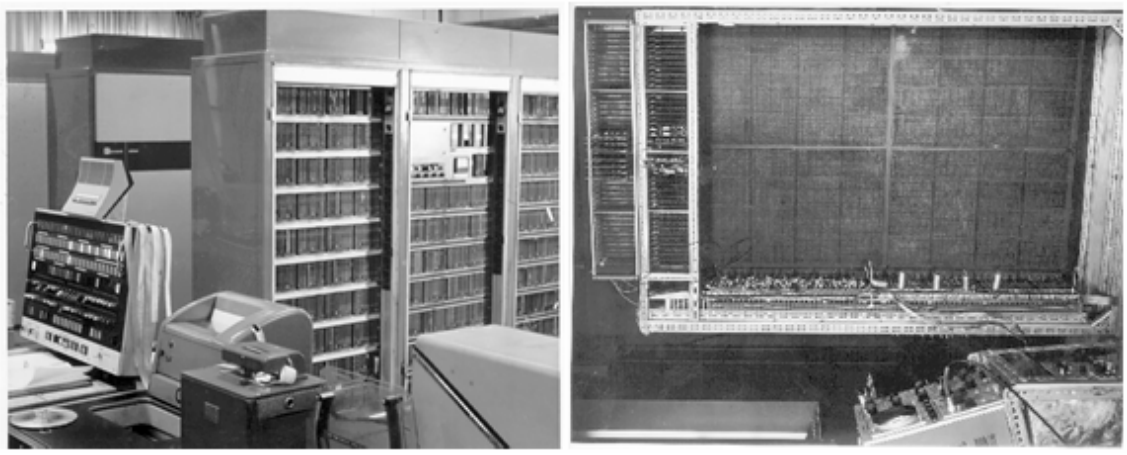


Figura 1.2 (a) Almacenamiento ROM de Muse (prototipo del computador Atlas), 1960. (b) Unidad aritmética del computador Atlas, 1967.

IBM reflejó la importancia de la virtualización en los años sesenta con el desarrollo de varios sucesores para el *IBM 7044*. Uno de ellos, el *Model 67* virtualizó todas las interfaces hardware a través del *VMM (Virtual Machine Monitor)*, un *monitor de máquinas virtuales*, llamado posteriormente en la década de los setenta *hipervisor* debido a la habilidad que poseía de correr sistemas operativos dentro de otros, y que era ejecutado encima del hardware subyacente. En estos primeros días de la virtualización los sistemas operativos que eran ejecutados en máquinas virtuales eran llamados *Conversational Monitor Systems* o *CMS*. Estas primeras *máquinas virtuales* continuaron desarrollándose y avanzando, e incluso en nuestros días se pueden encontrar corriendo en el mainframe *System z9*^{TM}, mostrado en la figura 1.3. Esto muestra un detalle importante en la evolución de la virtualización, que es la compatibilidad hacia atrás.

Figura 1.3 (a) Parte frontal e interior del mainframe IBM 2094 System z9, que corre actualmente desarrollos de máquinas virtuales utilizadas en los primeros mainframes con virtualización de IBM.

Otro de los primeros usos de la virtualización es el uso del procesador simulado, *P-code (Pseudo-code)*. *P-Code* es un lenguaje máquina que es ejecutado en una máquina virtual más que en el hardware real, lo que permitió a los programas codificados en *P-Code* ser altamente

portables y correr en cualquier lugar en el que esté disponible la *máquina virtual P-Code*. Máquinas virtuales de uso extendido en la actualidad siguieron este mismo modelo, como es el caso de la *Java Virtual Machine (JVM)*. El mismo concepto que en el que se fundamentó *P-Code* fue usado en los años sesenta también por el *Basic Combined Programming Language o BCPL*, predecesor de C.

Otro aspecto diferente de la virtualización y más reciente es la llamada ***virtualización del juego de instrucciones***, o traducción binaria. Un juego de instrucciones virtual es traducido al conjunto de instrucciones físico del hardware subyacente, en la mayoría de los casos de manera dinámica. Un ejemplo reciente de este modelo fue usado en el *Crusoe Central Processing Unit (CPU)*, diseñado por *Transmeta*, que implementó traducción binaria bajo el nombre comercializado de *Code Morphing*. Un ejemplo similar es el escaneo de código en tiempo de ejecución (*runtime code scanning*) usado por las soluciones de *virtualización completa* (será vista más adelante) para encontrar y redirigir instrucciones privilegiadas.

Con la llegada de los computadores personales el concepto de acceso al mismo tiempo a los recursos de un único supercomputador fue desapareciendo, y con él se vio eclipsada la virtualización: lo importante era el rendimiento más que la seguridad y fiabilidad. Al ocaso de la virtualización también contribuyó el que no fuera una buena idea la partición de los recursos de los miniordenadores o computadores personales debido a su escasez; los mainframes quedaron reducidos a lugares críticos y puntuales. La evolución con los años siguió la misma línea, hasta llegar a la situación que conocemos en la que prácticamente existe un ordenador por persona. Afortunadamente la virtualización junto a tecnologías como los sistemas operativos multiusuario y multitarea sobrevivieron en las Universidades y en sectores en los que su uso y fiabilidad eran críticos: grandes empresas, bancos, sistemas militares, etc. Estos sistemas fueron evolucionando y ya no eran los mainframes usados antiguamente, sino que eran sistemas que usaban hardware de miniordenador y con arquitectura mainframe, como la familia ***IBM AS/400***, cuyo primer modelo vio la luz en 1988.

Figura 1.4 Serie B de la familia IBM AS/400. Como se puede ver, los tamaños y configuraciones disponibles muestran una gran evolución en los mainframes.

Con el aumento de complejidad y potencia de los ordenadores que ya podían ejecutar sistemas multitarea y multiusuario, se pudieron retomar las características del sistema Unix que fueron eliminadas en sus primeras versiones (reducidas para posibilitar la ejecución en sistemas de baja potencia); entre ellas la virtualización. Surgió de nuevo el término de ***consolidación de almacenamiento***, recorriendo el camino inverso desde *un disco duro por persona* a *un disco duro para* todos.

En el presente, la **virtualización ha llegado al *escritorio***, lo que ha hecho que incremente exponencialmente de nuevo su popularidad y esto provoque que sea una de las tecnologías más innovadoras del momento debido a las notables ventajas que supone su aplicación. Uno de los hechos que justifican esto es que prácticamente todas las grandes empresas dentro del mundo informático han desarrollado productos de virtualización o han adquirido empresas que los ofrecían.

Hoy en día, las empresas disponen de ordenadores con una potencia de cálculo muy superior a la de decenas de servidores de hace varios años. Ahora que el rendimiento no es problema éste consiste en la seguridad, fiabilidad, y separación de privilegios necesaria, es decir, como ocurría hace aproximadamente cuarenta años en bancos, organizaciones militares y universidades. Estos problemas son ahora las únicas razones para seguir manteniendo servicios separados en diferentes servidores en las empresas. A partir de ahí queda explorar las innumerables ventajas que ofrece la virtualización como solución, y que mostraremos a lo largo del desarrollo del presente proyecto: planificación conjunta de servidores, creación automática de máquinas, migración en caliente a través de distintos equipos para llevar a cabo tareas de mantenimiento, creación de entornos de prueba, mayor aprovechamiento de los recursos hardware disponibles,…

1.2 VIRTUALIZACION ASISTIDA POR HARDWARE: INTEL VT Y AMD-V

El origen de las actuales tecnologías de virtualización por hardware está en los problemas creados en la arquitectura x86 por algunas de sus instrucciones cuando técnicas de virtualización quieren ser aplicadas: hay instrucciones pertenecientes al modo privilegiado que no pueden ser capturadas y que incluso pueden devolver diferentes valores dependiendo del nivel de privilegios de quien originó la llamada.

La arquitectura x86 dispone de cuatro anillos de protección, desde el nivel 0 (el de mayor privilegio) donde se ejecuta normalmente el sistema operativo al nivel 3 (menos privilegios) el cual soporta las aplicaciones, pasando por los niveles 1 y 2 en los que corren los servicios del sistema operativo. El problema fue entonces identificado por las empresas fabricantes de hardware –las máquinas virtuales no trabajarían adecuadamente si no eran ejecutadas con suficientes privilegios- y produjeron diseños que soportaran eficientemente y aceleraran la virtualización. La virtualización asistida por hardware, disponible desde décadas atrás en los mainframes IBM y los servidores Sun y otras máquinas, vivía así su gran relanzamiento en 2004 con la presentación de la tecnología *VT* de Intel, seguida después de la correspondiente *AMD-V* de AMD en 2006.

Tanto **Intel** como **AMD** disponen de estándares que definen características implementadas en muchos de sus procesadores más usados en ámbitos empresariales que permiten que tecnologías o soluciones de virtualización que hacen uso de la paravirtualización (como *Xen*, por ejemplo) puedan virtualizar tal y como lo hacen los procesadores instalados en los mainframes, pudiendo realizar **virtualización completa** y usar como sistema operativo invitado en las máquinas virtuales cualquier sistema.

En términos generales, la virtualización asistida por hardware hace uso de circuitería en la CPU y chips controladores que mejoran la ejecución y rendimiento de múltiples sistemas operativos en máquinas virtuales. Las tecnologías que implementan virtualización con soporte hardware específico suelen tratar con funcionalidades y funciones como el almacenamiento y recuperación del estado de la CPU en transiciones entre el sistema operativo invitado (que corre en la máquina virtual) y el *VMM* (*Virtual Machine Monitor*), capa de virtualización que actúa como medio entre éstos y el sistema operativo anfitrión y el hardware real disponible, gestionando los recursos y llamadas.

Así, con **virtualización soportada por hardware**, podemos implementar virtualización pura, sin necesidad de modificar los sistemas operativos invitados como hace *Xen* en la paravirtualización, y sin necesidad de emular las instrucciones cuyo procesamiento es problemático como hace *VMware*. El rendimiento es notablemente mejorado como consecuencia.

Intel

La tecnología diseñada e implementada por Intel, y que incluye en sus procesadores de gamas media y alta es ***Intel VT –Virtualization Technology-***. Intel introduce mejoras en sus procesadores x86 (*VT-x*) e *Itanium* (*VT-i*). *Intel VT* permite al *VMM (Virtual Machne Monitor o Monitor de Máquina Virtual)* correr en modo privilegiado –habiendo otro modo disponible para los sistemas invitados-, optimizando y acelerando las transiciones entre los sistemas operativos invitados de las máquinas virtuales y el *VMM*. Captura las llamadas al hardware desde el sistema operativo invitado, almacena el estado de la CPU y lo restaura después de que el *VMM* maneje el evento. Intel también sigue ampliando la funcionalidad de su tecnología de virtualización con los años, por ejemplo, en 2008 lanzando ***VT-d***, *VT* para E/S Directa (*VT for Directed I/O*), que permite transferencias de acceso directo a memoria (*DMA*) entre dispositivos y la memoria de los sistemas operativos invitados sin el uso del *VMM* como un paso intermedio. Esto es muy importante porque permite a los adaptadores de red y gráficos ser asignados de manera exclusiva a máquinas virtuales específicas para incrementar el rendimiento.

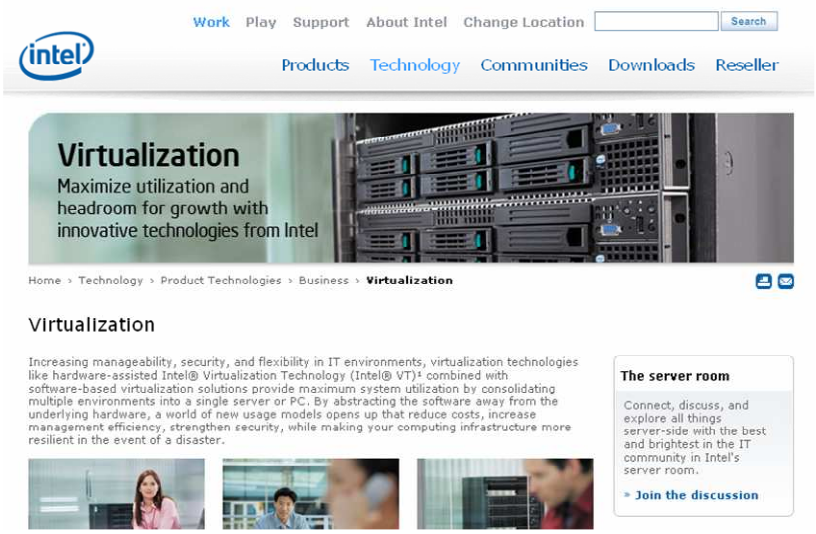


Figura 1.5 Página Web de Intel VT en Intel http://www.intel.com/technology/virtualization.

Intel VT proporciona un complemento ideal y necesario para implementar en nuestra infraestructura virtualización completa asistida por hardware. Maximiza las ventajas del uso de la virtualización, optimizando su rendimiento y reduciendo sobre todo el consumo de potencia. Como complemento, Intel proporciona otras tecnologías como *Intel® vPro™* para equipos de sobremesa y portátiles permitiendo la administración remota de sistemas virtualizados. Todo esto, sumado a la confianza que imprime una empresa en el mundo informático como Intel, hace que *Intel VT* sea ampliamente usada en entornos de virtualización y que sea una opción siempre considerada a la hora de afrontar la implantación de consolidación de servidores.

AMD

Por su parte, AMD dispone de una tecnología análoga a la de Intel denominada ***AMD-V o AMD-SVM*** **–**originalmente bajo el nombre ***Pacífica-*** que incluye también igualmente en sus procesadores tanto de gama media como de gama alta. La tecnología de virtualización de AMD proporciona entornos robustos y escalables de virtualización mientras que mantiene la eficiencia en consumo de potencia. Las capacidades y funcionalidades que proporciona esta tecnología en la virtualización x86 permiten por ejemplo alojar un mayor número de máquinas virtuales, más usuarios y más transacciones por máquina virtual (*Direct Connect Architecture*), acelerar las aplicaciones que se ejecutan en las máquinas virtuales (*RVI* o *Rapid Virtualization Indexing*), mejoras en los cambios de una máquina virtual a otra, o *migración en caliente* de máquinas virtuales.

AMD-V por ejemplo incluye opciones de configuración que permiten al *VMM* adaptar los privilegios de cada una de las máquinas virtuales. La tecnología de virtualización *AMD-V* está íntimamente relacionada con la familia de procesadores ***AMD Opteron™***. Los efectos de la arquitectura *Direct Connect*, que proporciona un manejo rápido y eficiente de la memoria, combinados con el **controlador de memoria integrado** –el cual compensa la pérdida en rendimiento en la traducción de instrucciones-, la tecnología *HyperTransport™*, y el uso de *RVI* ayudan a reducir el consumo de potencia, permiten soportar un mayor número de usuarios, más transacciones, y más aplicaciones que demanden un uso intensivo de recursos, alcanzando altos niveles de eficiencia y utilización en los entornos virtuales.

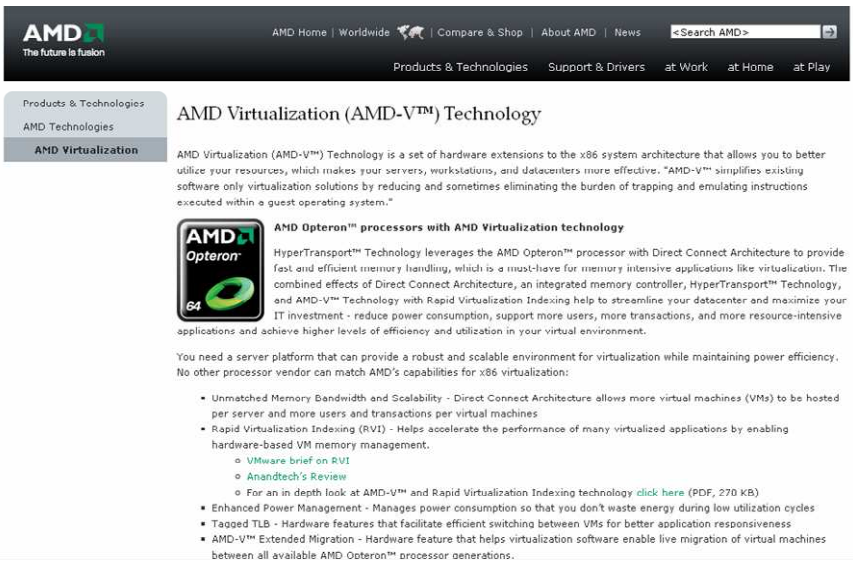


Figura 1.6 Página web de AMD-V en AMD
http://www.amd.com/us/products/technologies/virtualization/Pages/amd-v.aspx.

Ambos estándares son prácticamente idénticos y equivalentes en cuanto a funcionalidad ofrecida a las soluciones software de virtualización que quieran hacer uso de sus características. Así, por ejemplo, *Xen* emplea la tecnología ***HVM*** *(Hardware Virtual Machine)* desarrollada por IBM para la creación y uso de máquinas virtuales con virtualización completa (pudiendo ejecutar sistemas operativos no modificables), que dispone de la posibilidad de acceder y tomar ventaja de las características tanto de *AMD-V* como *Intel VT* haciendo uso de una interfaz común, accediendo a ambas de la misma forma.

Existen grandes diferencias en las implementaciones de ambas tecnologías fundamentalmente debido a razones técnicas, casi siempre relacionadas con la **gestión de la memoria**. La memoria es muy importante, ya que la virtualización necesita enmascarar la organización de la memoria a las máquinas virtuales: los procesadores AMD disponen de la

gestión de la memoria integrada en el chip del procesador, mientras que los procesadores Intel la tienen fuera del chip. Así, AMD lo tuvo más fácil para ofrecer virtualización, mientras que Intel sufre la penalización en la gestión de la memoria cuando la virtualizan. Otras diferencias de índole comercial han provocado por ejemplo que AMD se posicione mejor para ser usada en consolidación de servidores, mientras que Intel ofrece mayor seguridad evitando intrusiones y ataques en ordenadores de sobremesa y portátiles. Todo esto, de todas formas, no implica que los procesadores Intel no puedan ser usados para consolidar servidores ni que los procesadores AMD no sean seguros, simplemente es lo que históricamente los desarrollos realizados por cada empresa y la manera en que han sido gestionados han tenido como consecuencia.

1.3 PRESENTE Y FUTURO

Que hoy día la virtualización es uno de los puntos calientes del sector informático a nivel mundial nadie lo duda. Las cifras, con un creciente, cada vez más, número de empresas que virtualizan a prácticamente todos los niveles posibles la infraestructura de sus servicios y servidores, *data centers*,… y los buenos resultados obtenidos tras su implantación en la mayoría de ellas lo pone de manifiesto. Lo más sorprendente de todo es que se trate de una tecnología disponible desde hace más de cuarenta años, aunque no explotada en todos los estratos –fundamentalmente en grandes centros de cálculo- y mucho menos en el ámbito más interesante en la actualidad: la consolidación de servidores y la virtualización de sistemas operativos.

La virtualización proporciona muchas mejoras en rendimiento, portabilidad y flexibilidad; características insignia también de GNU/Linux, por lo que la elección de soluciones de virtualización en sistemas que hacen uso de GNU/Linux hace que tengamos un abanico enorme de posibilidades para virtualizar según nuestras necesidades con la mayor libertad. Más aún ahora que vivimos un período de recesión económica, las empresas ven la virtualización –sobre todo con soluciones de virtualización software libre y GNU/Linux como telón de fondo- como una solución que les permitirá a medio plazo un gran ahorro de costes en prácticamente todos los ámbitos relacionados con las tecnologías de la información, desde la energía consumida por los servidores de la empresa hasta los costes de mantenimiento, pasando por la administración, soporte, recuperación del servicio… y aumentando la calidad del mismo.

En el futuro aparece la virtualización como una de las claves en la explotación óptima de las actuales tendencias tecnológicas en informática. Tendencias actuales como por ejemplo el direccionamiento de 64 bits, CPUs multicore (más de 16 cores/CPUs por servidor), el tratamiento de manera importante de la refrigeración y ahorro de energía en los servidores, la convergencia de las interfaces de E/S mediante el uso de interfaces de red y almacenamiento de alta velocidad compartidas, o el almacenamiento virtualizado basado en red, y teniendo en cuenta las características que hemos estudiado a lo largo del presente capítulo, hacen ver que la virtualización en un futuro juegue sin duda un papel de suma importancia en el aprovechamiento de todos estos avances tecnológicos. Todo ello hace presagiar que vamos encaminados a la implantación de datacenters complemente virtuales.

Todo esto apoyado por hechos en la industria informática:

- Las nuevas tecnologías del hardware proporcionan soporte de manera explícita a la virtualización, como hemos visto han hecho Intel (con *Intel VT*) y AMD (con *AMD-V*) –mejor captura de instrucciones, ocultamiento de la memoria al hipervisor o *VMM*…- o como se está viendo con el desarrollo de las *MMUs (Memory Management Units)* de E/S y dispositivos de E/S que aceleran la virtualización.

- Los sistemas operativos comienzan a ser conscientes de la virtualización: mejora de la comunicación con el hipervisor o *VMM*, términos de licencia más amables, reducción de las operaciones costosas para la virtualización…

- Evolución de la gestión de E/S hacia el uso directo de los dispositivos por parte de los sistemas operativos de las máquinas virtuales invitadas (*Passthrough I/O Virtualization*).

Con la virtualización por todas partes, se puede empezar a pensar en nuevos modelos en la distribución del software, instancias de máquinas virtuales:

- Todos los beneficios de las tradicionales máquinas computacionales pero sin su coste y complejidad.

- Máquinas con dispositivos virtuales pre configurados, construidas con un propósito específico.

- Sistemas operativos y aplicaciones instalados/as y configurados/as de antemano.

- Personalización y configuración limitada al usuario.

- Instalación y configuración simple y fácil.

- No requieren hardware dedicado para su ejecución.

En definitiva, la virtualización es la clave para la gestión completa de los *data centers* en el futuro y el máximo aprovechamiento de las grandes posibilidades tecnológicas que nos ofrece la industria del hardware: es necesaria para incrementar la eficiencia. El soporte por parte de toda la comunidad a la virtualización es total, por lo que se puede decir que estamos viviendo una revolución en la informática a todos los niveles (además conjuntamente con técnicas como el *Grid Computing*, *Cloud Computing*…) como se puede ver en la transformación de modelos de todo tipo: económicos, de diseño, de los servicios ofrecidos, de gestión de las infraestructuras informáticas, del desarrollo de software,… que ocasiona su aplicación.

2 Terminología sobre Virtualización

Vista la evolución seguida por la virtualización en los últimos cuarenta años, es necesario definir los conceptos que resultan fundamentales en la actualidad y que actúan como base a los distintos *modelos de virtualización* existentes. Es por ello que este segundo apartado comienza con la introducción a los conceptos imprescindibles de *máquina virtual* e *hipervisor*. El primero porque, aunque haya tipos de virtualización en los que no sea necesario su uso es imposible entender el fundamento de las principales técnicas de *virtualización* sin saber en qué consiste una *máquina virtual*, tanto si es *de sistema* como si es *de aplicación*. El segundo, por ser el aspecto más importante y diferenciador de las técnicas de *virtualización completa* y *paravirtualización*, los dos modelos más extendidos actualmente en el uso de infraestructuras virtuales.

Conociendo ambos términos estaremos en disposición de presentar la clasificación de las diferentes técnicas de virtualización en cuatro modelos fundamentales: *virtualización de plataforma, virtualización de recursos, de aplicaciones y de escritorio*. Se apreciará que no es sencillo a priori distinguir unos de otros, ya que en ocasiones las características que los diferencian pueden ser mínimas. Tras haber estudiado los distintos modelos y la forma de operar

de las soluciones que los componen nos será más fácil finalmente ver las ventajas y desventajas derivadas de *virtualizar* en la última sección del presente apartado.

2.1 DOS CONCEPTOS FUNDAMENTALES: MAQUINA VIRTUAL E HIPERVISOR

Veamos con mayor detalle los que han sido considerados los dos conceptos más importantes en la terminología de la virtualización: máquina virtual e hipervisor. Es fundamental comprender con claridad el papel que juegan cada uno de ellos ya que ello nos permitirá acercarnos un poco más al nivel de abstracción impuesto por la virtualización y su funcionamiento interno. Comenzamos hablando de máquinas virtuales.

Es conveniente distinguir entre dos contextos muy importantes en los que en la actualidad se ubica el concepto de máquina virtual. Según las características y funcionalidad de la propia máquina podemos hablar bien de *máquinas virtuales de hardware o de sistema* o bien de *máquinas virtuales de proceso o de aplicación.*

Las ***máquinas virtuales de hardware o de sistema***, que son las que conforman el corazón del modelo de virtualización que será aplicado en el desarrollo del proyecto (*virtualización de plataforma*), son las que corren paralelamente sobre una máquina física anfitrión o host, de manera que tienen acceso y hacen uso de los recursos hardware que son abstraídos de él. Cada máquina virtual es engañada ya que cree que posee de forma exclusiva los recursos hardware de los que dispone cuando en realidad lo hace de manera virtual, ejecuta una instancia de sistema operativo sobre el que corren determinados servicios o aplicaciones tal y como consideremos necesario (véase la figura 1.7 en contraposición a la figura 1.8, que recoge un servidor sin virtualización y por tanto sin máquinas virtuales).

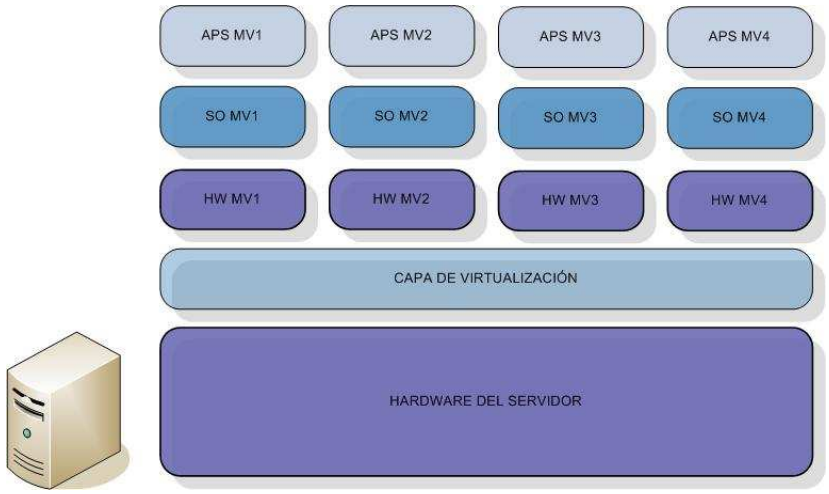


Figura 1.7 Arquitectura general de Virtualización sobre un servidor, en el que se puede apreciar como corren cuatro máquinas virtuales.

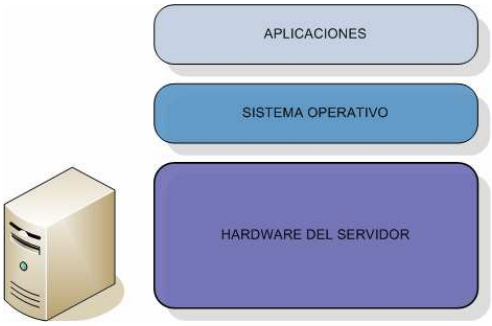


Figura 1.8 Arquitectura general de un servidor sin Virtualización.

La funcionalidad de este tipo de máquinas virtuales es muy amplia, aunque algunas de las características más destacables son la coexistencia de diferentes sistemas operativos, la *consolidación de servidores* (virtualización de servidores), y la prueba o testeo de proyectos tanto software, como hardware ya que pueden proporcionar arquitecturas de instrucciones (*ISA*) diferentes a las que implemente la máquina física anfitriona.

Éstas son las máquinas virtuales que componen la base de la virtualización que estudiaremos, junto a la capa software existente debajo de ellas y que permite esta distribución y administración de los recursos, el *hipervisor* (*hypervisor* en inglés) –también llamado habitualmente *monitor*, ya que una de sus funciones fundamentales es la monitorización de las máquinas virtuales-. Esta capa, que puede correr directamente sobre el hardware de la máquina física anfitriona o sobre un sistema operativo anfitrión –también como otra máquina virtual- será presentada en este mismo apartado a continuación y estudiada en mayor profundidad en posteriores capítulos. De esta acepción de máquina virtual brotan todos los proyectos desarrollados que se presentarán bajo el modelo de virtualización de plataforma, como *Xen*, *VMware*, *Hiper-V*, *Linux V-Server*, *User-mode Linux*, *KVM* u *OpenVZ*.

Al otro lado se encuentran las ***máquinas virtuales de proceso o de aplicación***. La primera diferencia es que éstas no representan una máquina completa al uso. Son ejecutadas como un único proceso sobre el sistema operativo y como lo hacen habitualmente los procesos, y además soportan la ejecución de tan sólo un proceso sobre ellas. Su objetivo fundamental es proporcionar un entorno de ejecución independiente del hardware y del propio sistema operativo para las aplicaciones que ejecutarán; éstas arrancan la máquina a su inicio y de igual manera la apagan cuando finalizan. Las dos máquinas virtuales de proceso o de aplicación de mayor importancia en la actualidad son *JVM* (*Java Virtual Machine*, entorno de ejecución para lenguaje Java de Sun Microsystems) y *CLR* (*Common Language Runtime*, entorno de ejecución para la plataforma *.NET* de Microsoft, ya citadas anteriormente). Tras haber realizado esta pequeña distinción inicial entre ambos tipos de máquinas virtuales, más adelante ambas serán presentadas con mayor detalle y apreciaremos mejor sus características estudiando las soluciones más relevantes en cada categoría. Es turno de introducir el concepto *hipervisor*.

El ***hipervisor* o *hypervisor*** es un pequeño monitor de bajo nivel de máquinas virtuales que se inicia durante el arranque, antes que las máquinas virtuales, y que normalmente corre justo sobre el hardware (denominado en alguna bibliografía como ***native o baremetal***) como podemos observar en la figura 1.9, aunque también lo puede hacer sobre un sistema operativo (llamado en este caso hipervisor ***hosted***):

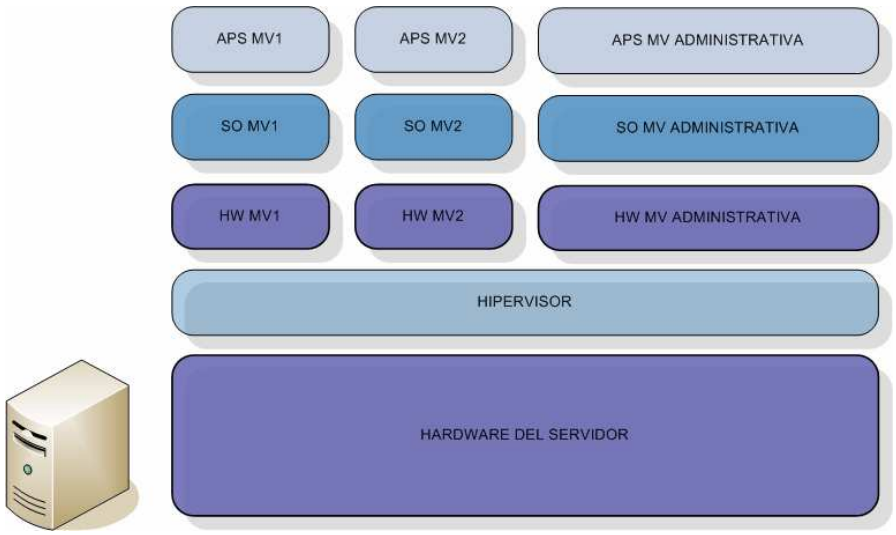


Figura 1.9 El hipervisor corre justo sobre el hardware del servidor. Sobre él, a su vez, encontramos un sistema operativo/máquina virtual monitor/a.

Como podemos imaginar esta capa adicional de virtualización no es usada en todos los modelos y soluciones existentes: es incluida en técnicas de virtualización completa y paravirtualización –ambas pudiendo además apoyarse en hardware con soporte de virtualización-, aunque tradicionalmente ha sido identificado más con soluciones de paravirtualización. Proporciona dos funcionalidades básicas:

- Identifica, capta, maneja y responde a operaciones de CPU e instrucciones privilegiadas o protegidas emitidas por las máquinas virtuales.

- Maneja el encolado, envío y devolución de resultados de peticiones de acceso a los recursos hardware instalados en el host anfitrión por parte de las máquinas virtuales.

Como se muestra en la figura 1.9, el sistema operativo corre sobre el hipervisor tal y como lo hacen las máquinas virtuales, el cual puede comunicarse con el hipervisor y cuyo objetivo fundamental es la gestión y administración de las instancias de las máquinas virtuales, por lo que lo habitual es que incluya diversas herramientas de gestión y monitorización, funcionalidad que puede ser también extendida con otras que deseemos instalar por cuenta propia.

El hipervisor es utilizado como capa de virtualización en los modelos virtualización completa y paravirtualización independientemente de la existencia y uso de hardware con soporte de virtualización específico. La paravirtualización es el modelo basado en hipervisor más popular debido a que introduce cambios en los sistemas operativos invitados permitiéndoles la comunicación directa con el hipervisor, mejorando así el rendimiento ofrecido y no introduciendo penalizaciones adicionales a la emulación usada en otros modelos como la virtualización completa. Cualquiera de los modelos basado en hipervisor sólo podrá gestionar máquinas virtuales con sistema operativo, librerías y utilidades compiladas para el mismo hardware y juego de instrucciones que el de la máquina física. El que el sistema operativo de la máquina virtual deba ser modificado o no ya dependerá de si hablamos de paravirtualización o virtualización completa.

2.2 MODELOS DE VIRTUALIZACION

En nuestros días muchos conceptos y tecnologías son englobados bajo el paradigma de la virtualización, en ocasiones de manera errónea y en otras acertada. Y es que comparando

todos en ocasiones o son prácticamente iguales o no tienen ninguna similitud, lo que puede provocar en el usuario confusión y que no llegue a comprender fielmente qué es lo que puede ofrecer cada solución. No es correcto mezclar por ejemplo los conceptos emulación, simulación, virtualización o paravirtualización en el mismo paquete. A continuación se intentará arrojar un poco de luz en este aspecto y plantear todo el entramado con mayor claridad.

De manera general se puede decir que **virtualización es el efecto de abstraer los recursos de un computador, proporcionar acceso lógico a recursos físicos**. La virtualización separa de manera lógica la petición de algún servicio y los recursos físicos que realmente proporcionan el servicio. Dependiendo del recurso que se abstraiga, que puede ser un recurso individual –almacenamiento, red- o bien una plataforma - un servidor, máquina- completa, y de por quién sea usado ese recurso, atenderemos entonces a distintos **modelos de virtualización**.

Por ejemplo, en el caso de que mediante algún mecanismo un sistema hardware completo sea abstraído de forma que pueda ser usado por diferentes instancias de sistemas operativos (y sus respectivas aplicaciones) de forma que éstas tengan la ilusión de que poseen los recursos de manera exclusiva y no compartida, estaremos hablando de un tipo de virtualización concreto, *virtualización de plataforma*, en el que el recurso que se abstrae es un servidor completo hardware y estamos virtualizando (disponen de algún tipo de recurso de forma virtual, aunque no sean conscientes) diferentes instancias de diferentes sistemas operativos.

Por lo tanto, es importante distinguir para entender con mayor claridad la virtualización entre dos conceptos como son el **recurso virtual que se abstrae** y el **ente (aplicación, sistema operativo, máquina…) que, virtualizado, dispone de ese recurso.** Dependiendo de ambos términos, al unirse, hablaremos de un modelo de virtualización distinto.

Teniendo en mente todo esto, podemos distinguir cuatro **modelos principales de virtualización**:

- ***Virtualización de plataforma.*** El recurso abstraído es un sistema completo, por ejemplo un sistema o servidor. En términos generales consiste en la abstracción de todo el hardware subyacente de una plataforma de manera que múltiples instancias de sistemas operativos puedan ejecutarse de manera independiente, con la ilusión de que los recursos abstraídos les pertenecen en exclusiva. Esto es muy importante, ya que cada *máquina virtual* no ve a otra *máquina virtual* como tal, sino como otra máquina independiente de la que desconoce que comparte con ella ciertos recursos.

  Este es un modelo especialmente a tener en cuenta, ya que es el aplicado para lo que se llama ***consolidación de servidores.*** La ***virtualización* o *consolidación de servidores*** puede verse como un particionado de un servidor físico de manera que pueda albergar distintos *servidores dedicados (o privados) virtuales* que ejecutan de manera independiente su propio sistema operativo y dentro de él los servicios que quieran ofrecer, haciendo un uso común de manera compartida y aislada sin ser conscientes del hardware subyacente. La *consolidación de servidores* será tratada en mayor profundidad en el capítulo segundo, donde podremos apreciar y diferenciar los distintos tipos y paradigmas de virtualización de plataforma existentes, que son los siguientes:

  - ***Sistemas operativos invitados.*** Sobre una aplicación para virtualización –no hace uso de hipervisor u otra capa de virtualización- que corre sobre la instancia de un sistema operativo –sistema operativo host- se permite la ejecución de servidores virtuales con sistemas operativos independientes. Si la aplicación de virtualización implementa traducción del juego de instrucciones o emulación podrán ser ejecutadas máquinas virtuales cuyo sistema operativo, utilidades y

aplicaciones hayan sido compiladas para hardware y juego de instrucciones diferentes al de la máquina física anfitriona, en caso contrario no. Algunos ejemplos de soluciones de este tipo son **VMware Workstation**, Parallels Desktop, **Sun xVM VirtualBox**, VMware Player, y **Microsoft Virtual PC**.

- ***Emulación.*** Un emulador que replica una arquitectura hardware al completo –procesador, juego de instrucciones, periféricos hardware- permite que se ejecuten sobre él máquinas virtuales. Por lo tanto se permite la ejecución de sistemas operativos y aplicaciones distintos al instalado físicamente en la máquina que ejecuta el emulador. Los emuladores más importantes actualmente son **Boch**s, MAME, DOSBox, Hercules, MESS, VirtualPC, y **Qemu**.

- ***Virtualización completa.*** También llamada nativa. La capa de virtualización, un hipervisor, media entre los sistemas invitados y el anfitrión, la cual incluye código que emula el hardware subyacente –si es necesario- para las máquinas virtuales, por lo que es posible ejecutar cualquier sistema operativo sin modificar, siempre que soporte el hardware subyacente. El código de emulación puede provocar pérdida en el rendimiento.

  Puede hacer uso de soporte hardware específico para virtualización y así mejorar su rendimiento. Sin duda dentro de esta categoría podemos encontrar algunas de las soluciones más importantes sobre virtualización –junto a las correspondientes a *pararvirtualización*- como **VMware Server**, **XenServer**, **z/VM**, Oracle VM, Sun xVM Server, Virtual Server, **VMware ESX Server**, VMware Fusion, **Xen**, **Hyper-V** (en algunos casos solo es posible si existe hardware con soporte de virtualización).

- ***Paravirtualización.*** Similar a la virtualización completa porque introduce hipervisor como capa de virtualización, pero además de no incluir emulación del hardware, introduce modificaciones en los sistemas operativos invitados que por consiguiente están al tanto del proceso (deben poder ser modificables). Éstos cooperan así en la virtualización eliminando la necesidad de captura de instrucciones privilegiadas o conflictivas por parte del hipervisor, mejorando el rendimiento hasta obtenerlo casi similar a un sistema no virtualizado (supone más una ventaja que una desventaja la modificación de los sistemas operativos invitados). Las librerías y utilidades ejecutadas por las máquinas virtuales deben estar compiladas para el mismo hardware y juego de instrucciones que el de la máquina física anfitriona.

  Puede hacer uso de soporte hardware específico para virtualización y así mejorar su rendimiento, además de para la ejecución de sistemas operativos no modificados ya que este soporte hardware puede manejar operaciones privilegiadas y protegidas y peticiones de acceso al hardware, además de comunicarse con y gestionar las máquinas virtuales. Las soluciones más extendidas e importantes dentro del paradigma de la *paravirtualización* son **Xen**, Logical Domains, Oracle VM, y Sun xVM Server.

- ***Virtualización a nivel del sistema operativo.*** Virtualiza los servidores sobre el propio sistema operativo, sin introducir una capa intermedia de virtualización. Por lo tanto, simplemente aísla los servidores independientes, que comparten el mismo sistema operativo. Aunque requiere cambios en el núcleo del sistema operativo, ofrece rendimientos próximos al sistema sin virtualizar. Compartiendo el mismo núcleo, entonces las máquinas no pueden correr

sistemas operativos diferentes (sí distintas distribuciones Linux o versiones del sistema operativo dependiendo de la solución utilizada), y además las librerías y utilidades ejecutadas deben estar compiladas para el mismo hardware y juego de instrucciones que el de la máquina física Como ejemplos representativos de este modelo podemos citar **OpenVZ**, **Linux V-Server**, Virtuozzo, FreeBSD´s chroot jails, Free VPS, Solaris Containers y Solaris Zones.

- ***Virtualización a nivel del kernel.*** Convierte el núcleo Linux en hipervisor utilizando un módulo, el cual permite ejecutar máquinas virtuales y otras instancias de sistemas operativos en el espacio de usuario del núcleo Linux anfitrión. Las librerías, aplicaciones y sistemas operativos de las máquinas virtuales deben ser soportados por el hardware subyacente del anfitrión. Dos soluciones destacan en esta categoría: **KVM** y **User-mode Linux**.

- ***Virtualización de recursos***. En este segundo caso el recurso que se abstrae es un recurso *individual* de un computador, como puede ser la conexión a red, el almacenamiento principal y secundario, o la entrada y salida. Existe un gran número de ejemplos dentro de la virtualización de recursos, como por ejemplo el uso de ***memoria virtual***, los sistemas ***RAID*** (*Redundant Array of Independent Disks*), ***LVM*** *(Logical Volume Manager),* ***NAS*** *(Network-Attached Storage)* o la ***virtualización de red***. Veamos con mayor detenimiento los distintos modelos de *virtualización de* recursos, los recursos que abstraen y las tecnologías y aplicaciones más notables a clasificar dentro de cada uno:

  - ***Encapsulación.*** Se trata de la ocultación de la complejidad y características del recurso creando una interfaz simplificada. Es el caso más simple de *virtualización de recursos*, como se puede ver.

  - ***Memoria virtual.*** Permite hacer creer al sistema que dispone de mayor cantidad de memoria principal y que se compone de segmentos contiguos. Como sabemos, es usada en todos los sistemas operativos modernos. Por lo tanto, en este caso el recurso individual que es abstraído es la memoria y disco. Ejemplos conocidos por todos son el espacio Swap utilizados por los sistemas operativos Unix, o las técnicas de paginado de memoria usadas en sistemas operativos Microsoft.

  - ***Virtualización de almacenamiento.*** Abstracción completa del almacenamiento lógico sobre el físico (disco y almacenamiento son el recurso abstraído). Es completamente independiente de los dispositivos hardware. Como ejemplos de *virtualización de almacenamiento* tenemos soluciones tan extendidas como RAID (*Redundant Array of Independent Disks*), LVM *(Logical Volume Manager)*, SAN (*Storage Area Network*), NAS *(Network-Attached Storage)*, NFS (*Network File Systems*), AFS, GFS, iSCSI (*Internet SCSI*), AoE (*ATA over Ethernet*).

  - ***Virtualización de red.*** La *virtualización de red* consiste en la creación de un espacio de direcciones de red virtualizado dentro de otro o entre subredes. Es fácil ver que el recurso abstraído es la propia red. Ejemplos bien conocidos de *virtualización de red* son OpenVPN y OpenSwarm, que permiten crear VPNs.

  - ***Unión de interfaces de red (Ethernet Bonding).*** Combinación de varios enlaces de red para ser usados como un único enlace de mayor ancho de banda. El recurso abstraído son por tanto los enlaces de red. Soluciones ejemplo de *Ethernet Bonding* son vHBA (*Virtual Host Bus Adapter*), y vNIC (*Virtual Network Interfaces Card*).

- ***Virtualización de Entrada/Salida.*** Abstracción de los protocolos de capas superiores de las conexiones físicas o del transporte físico. En este caso, los recursos que se abstraen son las conexiones de entrada/aalida y transporte. Ejemplo(s): Xsigo Systems, 3Leaf Systems, y en el futuro lo será: Cisco Systems, Brocade.

- ***Virtualización de memoria.*** Virtualizaremos bajo este modelo cuando unamos los recursos de memoria RAM de sistemas en red en una memoria virtualizada común.

- ***Virtualización de aplicaciones.*** Las aplicaciones son ejecutadas encapsuladas sobre el sistema operativo -recurso usado en este tipo de virtualización- de manera que aunque creen que interactúan con él –y con el hardware- de la manera habitual, en realidad no lo hacen, sino que lo hacen bien con una *máquina virtual de aplicación* o con algún *software de virtualización*. Este tipo de virtualización es usada para permitir a las aplicaciones de características como portabilidad o compatibilidad, por ejemplo para ser ejecutadas en sistemas operativos para los cuales no fueron implementadas. Debe quedar claro que la virtualización es solamente de las aplicaciones, lo que no incluye al sistema operativo anfitrión.

  Un ejemplo bien conocido es ***Wine***, que permite la ejecución de aplicaciones de Microsoft Windows virtualizadas correr sobre GNU/Linux, dentro de lo que son llamadas técnicas de *simulación*. Otros ejemplos muy importantes son *JVM* (*Java Virtual Machine*, entorno de ejecución para lenguaje Java de *Sun Microsystems*) y *CLR* (*Common Language Runtime*, entorno de ejecución para la plataforma *.NET* de Microsoft). Podemos diferenciar además entre los dos siguientes tipos de *virtualización de aplicaciones*:

  - ***Virtualización de aplicaciones limitada. Aplicaciones Portables.*** Aplicaciones que pueden correr desde dispositivos de almacenamiento extraíbles. También se incluyen dentro de esta categoría las aplicaciones heredades que son ejecutadas como si lo hicieran en sus entornos originales. Lo normal es que en este caso, en *virtualización de aplicaciones limitada*, no medie ninguna *capa de virtualización* o *software* con las mismas prestaciones y que la portabilidad se encuentre limitada al sistema operativo sobre el que correrá la aplicación. El recurso abstraído es el sistema operativo sobre el que son ejecutadas las *aplicaciones virtualizadas.*

  - ***Virtualización de aplicaciones completa.*** En este segundo tipo de *virtualización de aplicaciones*, una *capa intermedia o software de virtualización* es introducido para mediar entre la aplicación *virtualizada* y el sistema operativo y hardware subyacentes.

    - ***Portabilidad Multiplataforma (Cross-platform).*** Permite a aplicaciones compiladas para una CPU y sistema operativo específicos ser ejecutadas en diferentes CPUs y sistemas operativos sin ser modificadas, usando una traducción binaria dinámica y mapeado de llamadas del sistema operativo. No requiere recompilación o porting al correr en un entorno virtualizado, normalmente una máquina virtual de proceso o aplicación. Por tanto, el recurso abstraído en este caso es la CPU y el sistema operativo. Ejemplos utilizados en la mayoría de los sistemas son Java Virtual Machine, Common Language Runtime, Mono, LLVM, Portable .NET, Perl Virtual Machine, Citrix XenApp,

Novell ZENworks Application Virtualizacion, VMware ThinApp, Microsoft Application Virtualization

- ***Simulación.*** Reproducción del comportamiento de una aplicación concreta o una funcionalidad específica de una aplicación. Ahora, el recurso que se abstrae es la API (*Application Program Interfaces*) del sistema operativo, o cualquier interfaz. Antes ya se comentó Wine como ejemplo de este modelo de *virtualización de aplicaciones*, además disponemos de Crossover office, coLinux, Zebra, o Quagga.

- ***Virtualización de escritorio***. Consiste en la manipulación de forma remota del escritorio de usuario (aplicaciones, archivos, datos), que se encuentra separado de la máquina física, almacenado en un servidor central remoto en lugar de en el disco duro del computador local. El escritorio del usuario es encapsulado y entregado creando *máquinas virtuales*. De esta forma, es posible permitir al usuario el acceso de forma remota a su escritorio desde múltiples dispositivos, como pueden ser computadores, dispositivos móviles, etc. Por lo tanto, en este caso el recurso que se abstrae es el almacenamiento físico del entorno de escritorio del usuario –como usuarios, no somos conscientes del lugar físico en el que se encuentra nuestro escritorio, simplemente tenemos acceso a él-. Ejemplos muy importantes de soluciones que trabajan con *virtualización de escritorio* son Wyse Technology, **VMware View**, Sun VDI, vDesk de Ring Cube, **XenDesktop de Citrix**, vWorkspace de Quest Software, o ThinLinc de Cendio.

Como resumen de lo anteriormente expuesto se presenta a continuación la tabla 1.1 que recoge los distintos *modelos de virtualización* comentados, el recurso o recursos que abstrae y los ejemplos comentados.

Tabla 1-1. Modelos de virtualización en función del recurso que se abstrae

Modelo	Submodelo	Recurso abstraído	Ejemplo(s)
Virtualización de Plataforma	Sistemas operativos invitados	Plataforma hardware completa	VMware Workstation, Parallels Desktop, Sun xVM VirtualBox, VMware Player, Microsoft Virtual PC
	Emulación	Plataforma hardware completa	Bochs, MAME, DOSBox, Hercules, MESS, VirtualPC, Qemu
	Virtualización Completa	Plataforma hardware completa	VMware Server, XenServer, z/VM, Oracle VM, Sun xVM Server, Virtual Server, VMware ESX Server, VMware Server, VMware Fusion, Xen, Hyper-V (en algunos casos solo es posible si existe hardware con soporte de virtualización)
	Paravirtualización	Plataforma hardware completa	Xen, Logical Domains, Oracle VM, Sun xVM Server
	Virtualización a nivel del Sistema Operativo	Plataforma hardware completa	OpenVZ, Linux V-Server, Virtuozzo, FreeBSD´s chroot jails, Free VPS, Solaris Zones y Solaris Containers
	Virtualización a nivel del kernel	Plataforma hardware completa	KVM, User-mode Linux
Virtualización de Recursos	Encapsulación	Recurso individual	
	Memoria virtual	Memoria y disco	Espacio Swap, técnicas de

				paginado de memoria
	Virtualización de almacenamiento		Disco, almacenamiento	RAID, LVM, SAN, NAS, NFS, AFS, GFS, iSCSI, AoE
	Virtualización de red		Red	OpenVPN, OpenSwarm, que permiten crear VPNs
	Unión de interfaces de red (Ethernet Bonding)		Enlaces de red	vHBA (Virtual Host Bus Adapter), vNIC (Virtual Network Interfaces Card)
	Virtualización de E/S		Conexiones de entrada/salida y transporte	Xsigo Systems, 3Leaf Systems, en el futuro: Cisco Systems, Brocade
	Virtualización de memoria		Memoria RAM	
Virtualización de aplicaciones	Virtualización de aplicaciones limitada	Aplicaciones Portables	Sistema operativo	
	Virtualización de aplicaciones completa	Portabilidad Multiplataforma (Cross-platform)	CPU y sistema operativo	Java Virtual Machine, Common Language Runtime, Mono, LLVM, Portable .NET, Perl Virtual Machine, Citrix XenApp, Novell ZENworks Application Virtualizacion, VMware ThinApp, Microsoft Application Virtualization
		Simulación	API del Sistema Operativo, Interfaz	Wine, Crossover office, coLinux, Zebra, Quagga
Virtualización de escritorio			Sistema completo - localización física del escritorio, que se encuentra en un servidor remoto-	Wyse Technology, VMware View, Sun VDI, vDesk de Ring Cube, XenDesktop de Citrix, vWorkspace de Quest Software, o ThinLinc de Cendio

Viendo el amplio abanico de soluciones de virtualización disponibles una de las empresas líderes en el sector como es *VMware* propuso en 2006 el desarrollo de una *Interfaz de Máquina Virtual o VMI* (del inglés *Virtual Machine Interface*) genérica que permitiera el acceso múltiple de tecnologías de virtualización basadas en hipervisor para usar una interfaz común a nivel del kernel. Esto inicialmente no le pareció buena idea a *Xen*, aunque finalmente en la reunión USENIX de 2006 tanto *Xen* como *VMware* decidieron trabajar junto a otras tantas empresas en el desarrollo de una interfaz genérica conocida como *paravirt_ops*¸ que está siendo desarrollada por *IBM*, *VMware*, *Red Hat*, *XenSource* y coordinada por Rusty Russel. Este proyecto permitirá a las tecnologías de virtualización basadas en hipervisor competir más en lo que respecta a sus méritos técnicos y administrativos.

2.3 VENTAJAS Y DESVENTAJAS DE LA VIRTUALIZACION

Ya hemos visto recorriendo un poco la historia de la virtualización como en los años sesenta se hizo necesaria su implantación, cómo después sufrió una época de ocaso en la que permaneció olvidada, para finalmente en nuestros días regresar con fuerzas debido a la eclosión del hardware, su llegada a escritorio y en general las enormes posibilidades que puede proporcionar. Busquemos y revisemos el porqué hoy en día las tecnologías de virtualización han cobrado tanta importancia en el sector empresarial hasta llegar a ser uno de los sectores punteros en cuanto a uso y proyección dentro de la ingeniería informática.

Vamos a ver algunas de las razones y causas que han motivado que la virtualización resulte fundamental en nuestros tiempos, cómo después de más de cuarenta años ha surgido la necesidad de emplear la virtualización en los *CPDs* (*Centros de Procesado de Datos*):

- **Hardware de los Servidores Infrautilizado**

Hoy es habitual que los servidores que se ubican en los data centers de las empresas utilicen apenas un 15% o 20% de su capacidad de computación. Esto nos conduce lógicamente a uno 80% o 85% de capacidad que no es utilizada y por lo tanto desaprovechada. Aún así, con este uso tan bajo, el espacio que ocupan los servidores es el mismo y el consumo eléctrico que conllevan es el mismo que si se encontraran con usos cercanos al 100%. Como se puede concluir fácilmente, esto es un desperdicio de recursos computacionales.

Las características del hardware en cuanto a rendimiento y capacidad se duplican prácticamente cada año, lo que lleva a buscar soluciones que nos permitan aprovechar de mejor forma estos avances hardware con una carga de trabajo mayor. Es aquí donde surge el rol de la virtualización permitiendo que en un solo equipo o servidor almacenemos múltiples sistemas. Por lo tanto, usando virtualización en sus servidores las empresas pueden elevar las tasas de utilización de los mismos haciendo un uso más eficiente de los recursos y capital de la empresa. Ante este crecimiento sinfín de la potencia computacional proporcionada por la industria del chip, no hay más remedio que usar virtualización.

- **Se agota el Espacio en los Data Centers**

Como todos sabemos durante las últimas décadas el imponente crecimiento de las tecnologías de la información ha llevado a casi todas las empresas a reconducir sus actividades para adaptarse a los nuevos modelos de negocio, basado en software y automatizado, pasando del almacenamiento físico en papel al almacenamiento masivo de la información de forma electrónica. Toda esta transformación, si cabe, ha experimentado incluso un incremento y aceleración mucho mayor en los últimos años. Como es lógico, para soportar todos estos cambios las empresas han ido aumentando también el número de servidores de los que disponen en sus data centers, llegando a la situación en la que se les agota el espacio disponible para los mismos.

Así, esta situación requiere nuevos métodos de almacenamiento, como son los ofrecidos por la ***virtualización de almacenamiento***, que permite el manejo del almacenamiento independientemente de cualquier dispositivo particular hardware, logrando una abstracción completa del almacenamiento lógico sobre el físico. Con el uso de la virtualización, alojando múltiples sistemas invitados en un único servidor físico, se permite a las empresas recoger el espacio en el que se ubica su data center y así evitar los costes de la ampliación de su espacio. Este es un beneficio muy importante que aporta la virtualización, ya que la construcción de un data center puede llegar a costar del orden de unos 7 millones de euros.

- **Demanda de una mejor Eficiencia Energética**

Hace años parecía que cualquier coste energético en actividades empresariales era totalmente asumible, barato y que los recursos estarían disponibles sin dificultad. Desde hace un tiempo, aunque en la mayoría de los casos instanciados por iniciativas de grupos ecológicos y no por propia iniciativa, las empresas empezaron a considerar y darse cuenta que la energía es finita y que quizás habría que buscar nuevas estrategias en su forma de operar para llegar a situaciones en las que dependieran mucho menos de los recursos energéticos y potencia, y en los que su consumo fuera muchísimo menor. Lógicamente, el primer lugar en el que se fijaron para reducir todo este consumo fueron los data centers.

Para poder apreciar mejor el impacto del consumo que existe hoy día en los data centers veremos un ejemplo extraído de [4]:

"Un estudio comisionado por AMD y ejecutado por un científico del Lawrence Berkeley National Laboratory mostró que la cantidad consumida de energía por los data centers en Estados Unidos se duplicó entre los años 2000 y 2005. Además, éste consumo de energía se espera que aumente otro 40% para el final de la década. El consumo actual de energía de los servidores de los data centers y los costes asociados de refrigeración representa el 1.2% de la energía total consumida en Estados Unidos."

"Basado, en parte, en los resultados de este estudio, el United States Environmental Protection Agency (EPA) formó un grupo de trabajo para establecer estándares para el consumo de energía de los servidores y planear establecer una nueva clasificación "Energy Star" para los servidores eficientes."

Viendo este ejemplo es fácil reflexionar y llegar a la conclusión que en el momento histórico en el que nos encontramos es de vital importancia gestionar de manera eficiente el consumo de energía de los servidores que instalamos en los data centers, reduciendo al máximo su consumo en pro del ahorro energético tan necesario. El coste de los servidores que están dispuestos en los data centers sumado al bajo porcentaje de utilización de los mismos hacen muy necesario el usar técnicas de virtualización para reducir el número de servidores físicos funcionando, y de ésta manera el consumo energético que conllevan.

- **Costes de la Administración de Sistemas**

Como sabemos, las tareas de administración de sistemas pueden llegar a ser muy intensas y laboriosas, además en la mayoría de los casos los administradores de sistemas deben estar ubicados juntos a los servidores en los data centers porque necesitan tener acceso al hardware físico para realizar muchas de sus actividades. Entre las actividades que suelen realizar podemos destacar como principales la monitorización de los servidores, tanto de los servicios como del hardware –reemplazando hardware defectuoso cuando sea necesario- y sus recursos de CPU y memoria así como uso de disco y tráfico de red, instalación de sistemas operativos y aplicaciones, la realización de copias de seguridad periódicas de los datos que almacenan los servidores para recuperación en posibles pérdidas, seguridad y redundancia.

El uso de la virtualización ofrece una gran reducción en costes de administración en prácticamente todas las actividades que la componen. Por ejemplo proporcionando una monitorización simplificada y centralizada, provisión de máquinas de forma automatizada, simplificación en el proceso de copia de seguridad y restauración, dando más seguridad a nivel de máquina al aislarlas, redundancia y replicación de servidores para lograr alta disponibilidad y alto rendimiento… Aunque algunas de las tareas pueden permanecer prácticamente iguales, es decir, permanecen inalteradas en los entornos virtualizados, otras desparecen ya que los servidores que antes eran físicos pasan a ser instancias de máquinas virtuales.

- **Necesidad de alto rendimiento y alta disponibilidad**

Cada día más el modelo de negocio actual provoca que una mejor calidad de servicio y prestaciones sean requeridas a las empresas, a las que en la mayoría de los casos se les exige que sus servicios se encuentren disponibles las 24 horas al día los 365 días del año, y al mismo tiempo que su fiabilidad y rendimiento se mantengan elevados e inalterables. En un escenario así, sumado al hecho de que dispongamos servidores cuyos recursos se encuentran infrautilizados, se hace aún más patente la necesidad de aplicar alguna *técnica de virtualización*. Por ejemplo, en los casos más habituales en los que características de alto rendimiento y/o alta disponibilidad son implementadas son necesarias máquinas o servidores adicionales, bien para situar servicios a la espera la caída de otros o bien para la distribución de carga, por ejemplo.

Como se puede intuir, tanto las máquinas primarias que sirven los servicios como estas máquinas adicionales pueden ser integradas en una *infraestructura virtual* de forma que no necesitemos adquirir nuevos sistemas ni hardware adicional, al mismo tiempo que consolidamos los servidores en un único servidor físico anfitrión cuyo porcentaje de utilización aumentará.

Resumiendo, usando *consolidación de servidores* nos otorga la posibilidad de cambiar el modelo de gestión de nuestros data centers reduciendo costes en todos los sentidos, número de servidores físicos, la infrautilización de su capacidad y recursos, energía, espacio, y administración asociada. Además, y como vamos a ver a continuación, la virtualización además añade una serie de funcionalidades en la administración de sistemas cuyas ventajas son innumerables, como por ejemplo la **migración en caliente de máquinas virtuales**, la cual bien por motivos de mantenimiento, o balanceo de carga, alta disponibilidad y alto rendimiento, nos permite migrar una máquina virtual –su memoria, sistema operativo, y aplicaciones- de un servidor físico de un clúster a otro servidor físico dentro del mismo clúster o incluso a otros data centers –incluso, situados en continentes distintos-. Como veremos en el desarrollo de este proyecto se le ha otorgado gran importancia a la migración de las máquinas virtuales, por lo que se ha realizado un estudio más profundo de esta funcionalidad y diversas pruebas de rendimiento que podremos ver en los próximos capítulos del presente documento.

Además la virtualización también es **muy importante para los desarrolladores** ya que permite crear entornos de prueba virtualizados exactamente iguales a los reales, que pueden ser usados para desarrollo y depuración con todas las ventajas que ello supone. Por ejemplo, un núcleo Linux ocupa un solo espacio de direcciones, lo que quiere decir que si hubiera algún fallo en el núcleo o en los drivers provocaría la caída del sistema operativo al completo. Al usar virtualización, estamos ejecutando varias instancias de sistemas operativos, luego si falla uno de ellos, tanto el hipervisor como el resto de los sistemas operativos que tengamos en funcionamiento lo seguirán haciendo. Así, esto supone que depurar el núcleo de Linux sea lo más parecido a depurar cualquier otra aplicación en el espacio de usuario. Si fuera poco, existen numerosas soluciones de virtualización con distintos paradigmas de trabajo, ofreciendo distintas soluciones de rendimiento, portabilidad y flexibilidad que los administradores de sistemas pueden explotar para su beneficio según sus necesidades.

A continuación se exponen las **ventajas derivadas del uso de la virtualización**, tanto las comentadas anteriormente como otras. Como se puede apreciar, la lista es considerablemente extensa:

- ***Consolidación de servidores.*** Quizás una de las características más notables del uso de la virtualización y el hecho por el cual se encuentra en continua expansión en el mundo empresarial informático. En sí, consolidar servidores consiste en **reducir el número de los mismos al mismo tiempo que aumenta el porcentaje de su utilización.** Al consolidar servidores, se permitirá usar despliegues más modulares y escalables y centralizar su administración, notablemente simplificada. Como veremos, muchas de las ventajas restantes de la virtualización derivan del hecho de consolidar servidores.

- ***Administración de sistemas simplificada.*** Puede simplificar prácticamente todas las actividades relacionadas con la administración, sobre todo las que suelen ejecutarse de manera estandarizada (como las copias de seguridad), aunque por otro lado introduzca nuevas como el establecimiento de políticas de recuperación mediante migración o clonación de máquinas virtuales, mantenimiento de repositorios….

  Por ejemplo, la instalación y despliegue de nuevos sistemas es una tarea que se ve enormemente simplificada al introducir virtualización gracias a técnicas como la clonación de máquinas o las *virtual appliances* (instancias de máquinas virtuales ya preconfiguradas). Otro ejemplo: la *virtualización de escritorio* puede simplificar enormemente el despliegue de nuevos sistemas reduciendo la cantidad de software que

se requiere sea instalado localmente; sin duda incrementando la centralización de recursos compartidos y estandarizando el proceso de despliegue de sistemas puede proporcionar grandes ventajas a los administradores.

Siempre hay que recordar que el número de máquinas de las que es responsable el administrador siempre será el mismo; sean físicas o sean virtuales. Para aprovechar al máximo las ventajas derivadas del uso de la virtualización es fundamental mantener la infraestructura lo más independiente posible de los sistemas físicos; el propósito de las máquinas físicas que alojan máquinas virtuales no deber ser otro que exclusivamente ese, y no deben proporcionar servicios externos por ellas mismas.

- ***Alta disponibilidad y recuperación ante desastres.*** Al tener reducción en los tiempos de parada de los servicios y datos críticos del negocio. Podemos disponer de varias instancias de un servidor en espera de posibles fallos del que está en funcionamiento (simplemente son ficheros de configuración). Sin virtualización, se requieren múltiples sistemas físicos en espera sin ser utilizados para implementar esto mismo. Es posible la recuperación efectiva ante desastres y el mantenimiento de niveles de disponibilidad del servicio acordados gracias a mecanismos como la *migración de máquinas*. Si un sistema físico falla, los sistemas lógicos contenidos en él pueden ser migrados o distribuidos en caliente o dinámicamente a otros sistemas.

  La migración puede ser usada también para aplicar actualizaciones en la máquina, o en sistemas que alojan las máquinas, que vuelven a ser migradas a su localización original tras la culminación de las operaciones planificadas. Por lo tanto, si adoptamos una estrategia para detección automática de problemas y migración de máquinas virtuales ello nos llevará a reducir los costes asociados con recuperación de fallos y aumentar la disponibilidad de los servicios.

- ***Alto rendimiento y redundancia.*** Es muy fácil mantener una serie de servidores virtuales redundantes esparcidos en varios servidores físicos. Crear, instalar, configurar y mantener estas réplicas también es extremadamente sencillo, sin costes adicionales. A ello ayuda mucho el hecho de la posibilidad de aprovisionamiento de instancias preconfiguradas de máquinas virtuales. Operando de esta forma resulta sencillo disponer de mecanismos para balancear la carga de trabajo entre los servidores virtuales disponibles.

- ***Reducción de costes.*** La aplicación de técnicas de virtualización supone el ahorro de costes en prácticamente todos los ámbitos, pudiendo destinar esfuerzos y recursos a otros aspectos como la innovación. Se ahorrará en costes de instalación, configuración, monitorización, administración y soporte del servicio, asociados a licencias, del software -usando soluciones software libre como *Xen* con unos grandes beneficios en rendimiento y un bajo coste-, copias de seguridad, recuperación, consumo energético, seguridad… tanto a corto como largo plazo, al disponer de escalabilidad y agilidad sostenible.

  También nos permite ahorrar costes en la adquisición de nuevo hardware combinando la consolidación de servidores con una planificación adecuada de las capacidades para hacer un mejor uso del hardware existente. La virtualización también puede ayudar en la reducción de los costes en nuestra infraestructura informática en cuanto a potencia y requerimientos de refrigeración; añadir máquinas virtuales a un anfitrión existente no aumentará su consumo. Otros aspectos en los que es posible ahorrar son: costes de acceso remoto y fiabilidad (menos equipos con teclado, video y ratón necesarios), menos conexiones a aparatos suministradores de potencia ininterrumpible –los cuales se encontrarán más liberados y disponibles en tiempos de

fallo en el suministro de potencia-, y relacionados con la infraestructura de red: si dependiendo de cómo sea configurado el acceso a la red por parte de las máquinas virtuales, es posible simplificar el cableado de la red y reducir el número de hubs y switches necesarios.

- ***Mejora de las políticas de puesta en marcha, copias de seguridad y recuperación.*** Por ejemplo mediante el establecimiento de puntos de control en las máquinas virtuales y el uso de almacenamiento centralizado como *SAN*, *iSCSI*, *AoE*, *NAS*, *NFSs*…. Nuestros servidores pasarán a ser simplemente directorios y archivos de configuración, fácilmente replicables en copias de seguridad. En muchos casos la recuperación puede ser reducida a *copiar y pegar* estos directorios y archivos de configuración de una copia de seguridad o desde una máquina virtual preinstalada. En la mayoría de las soluciones de virtualización que existen en la actualidad es posible la toma de imágenes del estado de la máquina virtual, posibilitando también su posterior recuperación… teniendo a nuestra mano otro sencillo mecanismo adicional relacionado con este tema.

- ***Optimización del uso y control de los recursos.*** Derivada de la consolidación de servidores virtuales en servidores físicos infrautilizados. De esta manera, recursos como memoria, capacidad de procesamiento, red, disco… presentan porcentajes de utilización mayores a los habituales ofrecidos por servidores físicos dedicados, por lo que los servidores físicos de los que dispongamos son utilizados de manera óptima. Por ejemplo, en sistemas multiprocesador o multi-core las máquinas virtuales pueden correr usando diferentes procesadores o cores, haciendo un uso mejor de los recursos de computación totales disponibles.

- ***Gran escalabilidad.*** Crecimiento ágil soportado y con gran contención de costes. Una infraestructura virtual proporciona características de escalabilidad muy superiores a una física tradicional, al tratarse de máquinas virtuales lógicas. Un servidor físico, podrá gestionar más número de máquinas virtuales a medida que disponga de mayores recursos. Así, por ejemplo, si en nuestra infraestructura de máquinas virtuales quisiéramos integrar un nuevo servidor web, sólo tendríamos que crear y configurar la máquina virtual correspondiente (si dispusiéramos de una preconfigurada, bastaría sólo con copiarla) en un servidor físico que ya estuviera funcionando –salvo que no tuviéramos recursos suficientes en alguno- ahorrando tiempo, espacio, costes de administración, licencias, instalación, configuración… En cambio, si actuáramos como siempre lo han hecho las empresas, habría que adquirir un nuevo servidor o, en el mejor de los casos, integrar el servicio con otros diferentes en uno mismo.

- ***Aprovisionamiento de máquinas virtuales.*** El uso de máquinas virtuales preconfiguradas o virtual appliances es una solución rápida, flexible y asumible para desarrollar nuevos sistemas. Listas para cargar y funcionar, ahorrando tiempo de administración, instalación, configuración. Incluso preconfiguradas encapsulando máquinas virtuales determinadas aplicaciones o servicios (por ejemplo, centralitas *VoIP*, un servidor web, un balanceador de carga…), que luego pueden ser reutilizadas en la empresa según las necesidades. Tendremos disponibles la provisión de aplicaciones y sistemas dinámicamente, rápidamente, y justo a tiempo, moviendo máquinas virtuales de un servidor a otro según la necesidad, en lugar de malgastar tiempo configurando e iniciando nuevos entornos físicos en servidores. Este aprovisionamiento de máquinas virtuales puede planificarse, e incluso automatizarse, según la política que establezca el departamento TI y el uso de herramientas destinadas para ello.

- ***Compatibilidad hacia atrás.*** La virtualización posibilita el uso y mantenimiento de sistemas y aplicaciones heredados que no fueron adaptados a versiones actuales, y por lo tanto sin compatibilidad garantizada con los sistemas en uso hoy día. Usando

virtualización no es necesario crear y establecer sistemas para compatibilidad: con crear una máquina virtual con el entorno clásico de operación del sistema o la aplicación que queremos usar será necesario, eliminando cualquier riesgo de convivencia con las nuevas versiones del entorno. La virtualización es sin duda una solución excelente y simple para estos casos en los que queremos continuar ejecutando software heredado del que las empresas mantienen una fuerte dependencia. Sólo podremos ejecutar software heredado que sea soportado por el hardware sobre el que corre la solución de virtualización; en el caso en el que queramos ejecutar uno que necesite otra arquitectura hardware diferentes, haremos uso de una solución de virtualización que integre *emulación*, como *Qemu*.

- ***Disminución del número de servidores físicos.*** Derivada de la consolidación de servidores, al integrar múltiples instancias de servidores lógicos dentro de los servidores físicos, conseguiremos disminuir el número de estos últimos a utilizar en el CPD (Centro de Proceso de Datos) o Data Center. Esta es una ventaja muy importante ya que repercute en muchos aspectos; los más importantes referidos a la administración de nuestra infraestructura informática. Lógicamente, al disminuir el número de servidores físicos se simplificará y reducirá ésta a la vez que disminuirá también el espacio físico requerido en nuestro CPD o data center para ellos, cuestiones que pueden llegar a resultar de gran importancia.

- ***Mejora de la eficiencia energética.*** Al existir un menor número de servidores físicos el consumo de potencia de los mismos consecuentemente será menor. Además, este consumo será más eficiente: ahora los servidores no se encontrarán infrautilizados como antes, que consumían la misma potencia con un menor porcentaje de utilización.

- ***Prueba y depuración.*** Fácil establecimiento de entornos virtuales iguales a los reales en los que realizar prueba y depuración de sistemas operativos, aplicaciones,… sin las consecuencias que lógicamente eso tendría en un entorno físico real. También, por ejemplo, una aplicación muy importante es la prueba y depuración de software que se desarrolla para correr sobre sistemas y arquitecturas hardware aún no desarrolladas ni fabricadas, y que sí estarán disponibles en el futuro, no teniendo que esperar a que ello ocurra para su prueba. Hay que considerar antes qué solución de virtualización es más apropiada para el conjunto de pruebas y tests que vayamos a desarrollar, por ejemplo soluciones basadas en hipervisor no suelen ser muy recomendadas para la depuración de drivers de hardware debido al propio hecho de introducir el nivel adicional de operación y acceso al hardware.

- ***Seguridad y aislamiento.*** La virtualización puede proporcionarnos mayores niveles de seguridad y aislamiento, y a un coste menor. Tenemos la posibilidad de proteger aplicaciones y sistemas operativos aislándolos en máquinas virtuales que son totalmente independientes entre sí y con el hipervisor o sistema anfitrión. Cada una de las máquinas virtuales tiene un acceso en modo supervisor único, por lo que un ataque de seguridad que logre acceder a una aplicación o sistema operativo de una de las máquinas afectará sola y exclusivamente a la máquina en la que ocurrió el fallo de seguridad, y no en el resto de máquinas ni en el anfitrión por lo que no los comprometerá. Esto es beneficioso tanto para empresas como a nivel de usuario particular.

- ***Tipificación y estandarización de plataformas y configuraciones.*** Es sencillo al aplicar virtualización establecer estándares para la configuración tanto de los servidores anfitriones como de las máquinas virtuales que se alojan en ellos. Así, asistiremos a escenarios en los que habrá homogeneidad, ahorrando por tanto en administración y soporte. En cuanto a las máquinas virtuales, y aunque difieran algunas de otras bastante

en cuanto a su funcionalidad y configuración, todas serán tratadas también de manera uniforme por parte del virtualizador que esté en uso. Toda esta transformación puede influir de forma muy positiva en una mejor integración y simplificación del negocio en la infraestructura informática; por ejemplo cambios en el negocio pueden ser direccionados en cambios en las máquinas virtuales los cuales pueden ser llevados a cabos con rapidez.

- ***Mejora de la calidad de servicio y fiabilidad.*** Mediante la prueba y depuración de aplicaciones, sistemas operativos o nuevas versiones de sistemas operativos y software asociado es posible asegurar niveles de calidad y fiabilidad en ellos (*Quality assurance o QA*), al poder implementar un testeo fácil de aplicaciones en múltiples sistemas operativos, o de los propios sistemas operativos sin tener que disponer y configurar múltiples entornos hardware dedicados. Esta es, junto a la consolidación de servidores y la reducción de costes, una de las grandes ventajas al aplicar virtualización, debido a la efectividad mostrada en términos de tiempo y coste, reduciendo la cantidad de hardware requerido, reduciendo o incluso eliminado la mayoría del tiempo requerido para la instalación de sistemas, reinstalación –en muchas ocasiones es útil retomar el estado guardado previamente de una máquina virtual-, configuración….

- ***Clonación de máquinas virtuales.*** Asociada a términos como redundancia, aprovisionamiento de máquinas virtuales o recuperación ante desastres. Es una de las características de la virtualización que hacen que sea una fantástica solución en la instalación, desarrollo y despliegue de nuevos sistemas. También tiene una gran utilidad cuando por ejemplo, queremos aplicar un parche a un sistema operativo de una máquina virtual, clonándola previamente para guardar su estado. En muchas de las soluciones de virtualización existentes es posible la toma y recuperación de imágenes del estado de las máquinas virtuales, algo que puede resultar de gran utilidad.

- ***Personalización.*** En el caso de las soluciones de virtualización que son software libre, como *Xen, KVM, Qemu,…* éstas pueden ser personalizadas y extendidas para alcanzar los requisitos necesarios específicos.

- ***Uso en ámbitos docentes.*** Uso como ayuda didáctica: se pueden configurar e iniciar entornos para que estudiantes puedan aprender e interactuar con sistemas operativos, aplicaciones o drivers de dispositivos. Por ejemplo, es posible el aprendizaje de conocimientos informáticos estudiando en máquinas virtuales configuradas para emular hardware o sistemas operativos particulares. Unido a las capacidades que proporciona la virtualización acerca de prueba y depuración, ésta ventaja resulta muy interesante en estos ámbitos.

- ***Clústeres de servidores virtuales.*** Creación de clústeres de servidores virtuales para unificar múltiples servidores en un solo sistema. Proporciona las ventajas de la creación de clústeres trasladadas al ámbito de una infraestructura virtual, pudiendo crear un clúster dentro de un solo equipo.

- ***Flexibilidad.*** Las características hardware y software de las máquinas virtuales son totalmente configurables a nuestro gusto. Así, podemos crear servidores virtuales con RAM, CPU, disco, y red que estrictamente necesitemos. Gran flexibilidad en el reparto de recursos del sistema anfitrión entre las máquinas que aloja. Esta flexibilidad se ve aumentada por el hecho de la posibilidad de asignación de determinados dispositivos de manera exclusiva a ciertas máquinas virtuales según nuestros intereses. Por ejemplo, si dispondremos de un servidor que deberá soportar una gran cantidad de tráfico de red podremos asignar a su máquina virtual de manera exclusiva una de las tarjetas de red disponibles físicamente en el servidor anfitrión.

- ***Gran agilidad.*** La creación de máquinas virtuales es un proceso que se puede llevar a cabo con gran rapidez e incluso se puede automatizar. Disponiendo incluso de máquinas virtuales pre configuradas, podemos poner en funcionamiento servidores con un simple clic o ejecutando tan sólo un comando. Esto nos posibilita también ganar en rapidez ante cambios bajo demanda, para realizar mejoras, impuestos por el modelo de negocio…

- ***Portabilidad, migración.*** Incrementando el aislamiento de las máquinas virtuales de hardware físico específico aumenta la disponibilidad de los sistemas aumentando la portabilidad de las máquinas virtuales, permitiéndoles ser migradas. La migración de máquinas virtuales es un proceso transparente al usuario al mismo tiempo que transparente a cualquiera de los procesos que se encuentren corriendo en las máquinas virtuales. Portar máquinas virtuales entre distintos servidores físicos es muy sencillo. Mecanismos de migración en parada o en caliente aparte, podemos portar máquinas virtuales simplemente copiando los archivos de configuración que definen las mismas o los ficheros/imágenes que constituyen sus discos, de un tamaño que permite usar dispositivos de almacenamiento USB, discos duros externos, ….

- ***Automatización.*** Existen herramientas de automatización que permiten fijar métricas comunes (por ejemplo sobre rendimiento) de las máquinas virtuales y que pueden llegar a reconfigurarlas si es necesario para cumplir con las condiciones expuestas. Por ejemplo, se puede asignar dinámicamente mayor CPU a una máquina que lo precise para cumplir unos determinados tiempos de respuesta, tomando CPU de otra máquina que tenga menos carga o migrándola a un servidor con mayor CPU disponible. Así, la infraestructura virtual está altamente ligada con el negocio.

- ***Cola unificada.*** Trato de los servidores como una cola unificada de recursos. Tratando de esta forma los servidores de que dispongamos ganaremos en agilidad, consistencia y efectividad en el proceso de administración de sistemas.

No sería recomendable entrar a valorar solamente las ventajas que puede aportar la adopción de infraestructuras virtuales en lugar de las usadas tradicionalmente. Antes de aplicar técnicas de virtualización es completamente necesario y debe ser considerado de obligatoriedad tener en cuenta las posibles desventajas que pueden surgir de su implantación; algunas de ellas podrían condicionar el plantear si finalmente llevar a cabo nuestro proyecto de virtualización o no.

A pesar de ello, no debemos temer el encontrarnos con grandes dificultades y barreras que nos impidan virtualizar; debemos poner en una balanza las ventajas y desventajas que pueden aparecer en nuestro caso particular, teniendo en cuenta la variedad de posibilidades y modelos de virtualización disponibles. Presentamos a continuación las **desventajas más habituales que suelen aparecer cuando virtualizamos**:

- ***Pérdida de rendimiento.*** Como es normal, la ejecución de un sistema operativo y de aplicaciones en una máquina virtual nunca ofrecerá un rendimiento igual y mucho menos superior al obtenido con la ejecución directamente sobre el servidor físico. Como sabemos, algunas soluciones introducen capas intermedias como son los hipervisores, que capturan las llamadas de las máquinas virtuales, gestionan su acceso concurrente a los recursos físicos y las monitorizan. Por lo general, una aplicación que corre en una máquina virtual lo hace de maneras más lenta a como lo haría en una máquina física directamente, aunque recientemente se estén obteniendo performances cercanas al rendimiento nativo de los sistemas anfitriones (más con paravirtualización que con virtualización completa). La pérdida en rendimiento depende por lo general de tres

factores: la aplicación en sí, la tecnología de virtualización utilizada, y la configuración del hipervisor. Aplicaciones con un gran volumen de operaciones de entrada y salida experimentan peor rendimiento.

- ***Compartición del servidor.*** Una de las principales ventajas del uso de la virtualización puede llegar a ser una desventaja importante si no es gestionada y administrada correctamente. Las capacidades y recursos de los servidores físicos anfitriones deben ser monitorizadas y controladas en todo momento. El uso de capacidad, memoria, procesador,… puede variar considerablemente y de manera dinámica al existir nuevos procesos y procedimientos como por ejemplo *migraciones de máquinas virtuales*, que al tener lugar permite que *máquinas virtuales* puedan ubicarse en diferentes servidores en distintos momentos. Lógicamente, visto de esta manera, la compartición del servidor puede acarrear una mayor complejidad en la administración, que no llevada a cabo correctamente puede conducir a importantes e impredecibles problemas.

- ***Soporte del hardware.*** Por lo general, no es posible utilizar hardware que no esté soportado por el hipervisor. El software de virtualización suele imponer una serie de dispositivos hardware (como tarjetas de vídeo y de red) que son las disponibles para las máquinas virtuales. Otras soluciones de virtualización pueden emular el hardware necesario, aunque por lo tanto ofreciendo peor rendimiento.

- ***Hardware virtual obsoleto.*** Puede darse la posibilidad de que el hardware virtual esté obsoleto. Aunque se suele ir actualizando con nuevas versiones del hipervisor, lo normal es que dispongamos de dispositivos anteriores para virtualización como por ejemplo USB 1.0, Firewire 400 o Ethernet 100. Es por ello que es altamente recomendable disponer de un hipervisor actualizado.

- ***Aceleración de video por hardware.*** No está disponible la aceleración de vídeo por hardware, por lo que aplicaciones que disponen de efectos 3D no funcionarán en condiciones normales sobre una máquina virtual. Hay alguna excepción (como puede ocurrir con VMware Fussion o Parallels, que soportan algunas versiones de OpenGL o DirectX), pero conviene comprobar en primer lugar su rendimiento.

- ***Riesgo en la organización al implantar una infraestructura virtual.*** La proliferación de máquinas virtuales puede llegar a ser un inconveniente ya que, no organizada de una manera satisfactoria, puede conllevar un crecimiento en la complejidad de la administración, gestión de licencias de los servidores, aumento del riesgo en cuestiones de seguridad,… todo lo contrario a lo que se pretender llegar cuando implantamos un proyecto de virtualización.

- ***Número inadecuado de máquinas virtuales.*** La creación de máquinas virtuales que no son necesarias lleva un consumo mayor y elevado de recursos computacionales, RAM, disco, CPU… por lo que puede llegar a desaprovecharse recursos. No es conveniente la creación sin control de máquinas virtuales para disponer de servidores con mayor porcentaje de utilización; el uso de los recursos debe ser el conveniente para la actividad que estemos desarrollando.

- ***Anfitrión como único punto de fallo.*** Con una única avería en el servidor anfitrión pueden caer múltiples servidores virtuales con sus respectivos servicios teniendo ello un gran impacto en la organización. La solución de este gran problema de la virtualización es sin duda una planificación detallada que cubra disponibilidad y recuperación ante desastres siempre que sea posible.

Algunas aproximaciones para conseguirlo son las siguientes: redundancia de hardware en el sistema host haciendo que el fallo de un componente sea transparente a las máquinas virtuales, la compra y mantenimiento de hardware que replique los sistemas físicos que alojan máquinas virtuales de gran importancia (en ocasiones este coste es asumible en comparación con el daño que causaría la caída del sistema en funcionamiento), el uso de clustering, redundancia y replicación de las máquinas virtuales que ofrecen servicios críticos en varios servidores físicos y así evitar la caída del servicio, y como no la ejecución de manera centralizada de software de monitorización de sistemas que nos alerte de problemas hardware y software emergentes antes de que lleguen a ser críticos. Además hay que implementar funcionalidades de migración de máquinas virtuales –*Xen*, por ejemplo puede llegar a eliminar estos puntos de fallo únicos haciendo los servidores virtuales totalmente portables de un host físico a otro en el momento de detección de los problemas-. La importancia del hardware del servidor físico anfitrión en la virtualización es crítica como podemos apreciar.

- ***Portabilidad condicionada.*** La portabilidad de las máquinas entre distintas plataformas está condicionada por el software de virtualización que elijamos. Dependiendo si elegimos soluciones sobre GNU/Linux, MacOS, Windows,… para nuestro sistema anfitrión tendremos unas posibilidades u otras para esta portabilidad. Puede que en un futuro sea un requisito indispensable esta portabilidad y migración de las máquinas virtuales, por lo que debe ser planificado y estudiado con antelación.

- ***Disminución de las ventas del hardware.*** Como efecto colateral del uso de la virtualización, disminuyen en gran porcentaje las ventas de hardware. El número de máquinas vendidas será inferior al actual, a pesar de que el hardware vendido será de una potencia notablemente superior. A pesar de esta "desventaja", las empresas del hardware –como Intel y AMD- han apostado fuerte desde el principio por la virtualización lanzando tecnologías dotadas de soporte para ello.

- ***Dependencia del sistema operativo anfitrión.*** La elección del sistema operativo de la máquina anfitriona es crítica. Todos los sistemas operativos de las máquinas virtuales dependerán de la estabilidad y seguridad que ofrezca el anfitrión. Una vez más en este punto podremos establecer debates sobre qué solución de virtualización y qué sistema operativo deben residir en el servidor físico anfitrión.

- ***Dependencia de la solución de virtualización elegida.*** Es fundamental elegir la tecnología y solución de virtualización adecuada en función del servicio que ofrezcamos. Hay algunas que ofrecen mejor rendimiento en servidores con servicios críticos o de negocio, y otras que son mejores en servicios no críticos. Por ejemplo, *VMware* no es la solución óptima para virtualizar las aplicaciones críticas. El hardware sobre el que se ejecutan las máquinas virtuales *VMware* (arquitectura x86) no es capaz de direccionar tantos datos de una sola vez como otras arquitecturas (porque no es una arquitectura nativa 64 bits), tiene características de fiabilidad y disponibilidad medias. Las soluciones óptimas para virtualizar los servidores críticos son las que se ejecutan en servidores de alta gama, por ejemplo *HP Integrity Virtual Machines*, o *Xen.*

- ***Disponibilidad de recursos suficientes.*** La disponibilidad de los recursos es una cuestión de gran importancia y es un problema potencial debido al hecho de que las máquinas virtuales comparten los recursos disponibles físicamente en el servidor anfitrión, muchas veces compitiendo por ellos, otras veces no. Hay que disponer de recursos suficientes para que las máquinas virtuales estén funcionando de manera simultánea, no solamente de forma independiente. Siempre debemos situarnos en el

peor de los casos a la hora de planificar el uso futuro de los recursos por parte de las máquinas virtuales.

Debemos pensar *dinámicamente*, ya que el uso de los recursos será cambiante: por temas de carga de trabajo, al disponer de un mayor o menor número de máquinas, si las máquinas son portadas entre distintos servidores al implementar migraciones,… hacen que sea extremadamente importante la monitorización de los recursos de procesamiento, memoria, red, capacidad….

- ***Congestión de red por servidor.*** Si el servidor físico que aloja a diversas máquinas virtuales dispone de una o pocas interfaces de red y las máquinas virtuales ejecutan operaciones con una carga intensiva en la red ello puede provocar que la demanda del hardware de red sea excesiva y exista congestión, dando lugar a problemas de rendimiento para el host anfitrión o para las máquinas virtuales que comparten la (s) interfaz (interfaces) de red. Una solución trivial a este problema es la instalación de un mayor número de tarjetas de red en el anfitrión así como la asignación de manera exclusiva de algunas de ellas a las máquinas virtuales que demande un mayor tráfico. Sin embargo, estas asignaciones de recursos a las máquinas pueden provocar que aumenten la complejidad de procesos como la migración de máquinas virtuales.

- ***Incremento de la complejidad y tiempo de depuración de las actividades de Networking.*** La administración de red de máquinas virtuales, cada una con sus respectivas interfaces de red (ya sean físicas o virtuales) y configuraciones asociadas (*MAC*, dirección IP, encaminamiento) puede llegar a ser más compleja que la administración de red de máquinas físicas con las mismas características, debido no sólo a la capa software de red virtual introducida en las máquinas sino también a la configuración que debe ser establecida en cortafuegos, filtrados, y otros mecanismos de control. Una práctica extendida en este aspecto es la creación de subredes de máquinas virtuales, cada una de las cuales con su propio servidor DHCP para así controlar el rango de direcciones de red que se asignan a las máquinas virtuales, simplificando de manera notable las actividades de filtrado, encaminamiento y cortafuegos al trabajar con bloques de direcciones de red. Además hay que tener en cuenta las configuraciones de las interfaces de red cuando clonamos y caracterizamos nuevas máquinas virtuales, ya que algunos de los parámetros pueden dar problemas (como por ejemplo la dirección *MAC* de la tarjeta de red) al existir duplicados en la red, dificultando las actividades de networking.

- ***Posible aumento de complejidad en la administración.*** A pesar de que en la mayoría de los casos las actividades de administración se verán simplificadas, hay configuraciones en las que es posible que el efecto al presentar virtualización sea el contrario del esperado. Por ejemplo, esto ocurrirá si usamos utilidades de administración y/o monitorización de sistemas de forma distribuida que no pueden trabajar con máquinas virtuales –no pueden comunicarse con ellas-, o puede que ocurra también si usamos al mismo tiempo diferentes soluciones de virtualización. Aunque no sea un problema de gran importancia, siempre es bueno tenerlo en mente.

- ***Nuevas problemáticas.*** Se introducen nuevas problemáticas que no existían en los entornos físicos al virtualizar: por ejemplo, debemos conocer en cada momento qué máquinas se encuentran arrancadas en cada servidor físico de las que aloja, o si hay máquinas que pueden ejecutarse en varios servidores físicos en cual se encuentran en funcionamiento en cada momento, si es necesario migrar máquinas para balanceo de carga,… Otra problemática es la convivencia en la monitorización tanto de servidores físicos habituales como de servidores virtuales. Lo mismo ocurre si disponemos de

diversas tecnologías de virtualización: deben ser gestionadas centralizadamente y de una manera transparente, lo más homogénea posible.

- ***Licencias del software.*** Siempre que apliquemos virtualización debemos considerar temas relacionados con licencias del software. Cuando almacenaremos múltiples cuentas de usuarios en un único servidor, cuando repliquemos máquinas virtuales… En el caso de que no pudiésemos permitirnos nuevas licencias al aplicar virtualización, siempre podemos buscar esquemas de licencias flexibles, lo que en ocasiones es crítico para servidores que alojan un gran número de usuarios. Otro posible problema relacionado con licencias es el causado por vendedores de software los cuales no apoyan el uso de sus productos en entornos virtualizados –hay que comprobar la licencia del producto para estar seguros de ello.

En la tabla 1.2 podemos ver de manera resumida las ventajas y desventajas comentadas en las páginas anteriores. Aunque se trate de una muestra de las principales ventajas y desventajas, no por ello se debe pasar por alto el detalle de que el número de desventajas es menor al mismo tiempo que generalmente tienen lugar en casos específicos. La importancia y facilidad de obtención de las ventajas que han sido expuestas hace que día a día las técnicas de virtualización cobren mayor fuerza, y sin duda compensa aplicarlas a pesar de las posibles y menores desventajas que puedan surgir.

Tabla 1-2. Ventajas y desventajas derivadas del uso de técnicas de virtualización

Ventajas	Desventajas
Consolidación de servidores	Pérdida de rendimiento
Administración de Sistemas simplificada	Compartición del servidor
Alta disponibilidad y recuperación ante desastres	Soporte del hardware
Alto rendimiento y redundancia	Hardware virtual obsoleto
Reducción de costes	Aceleración de vídeo por hardware
Mejora de las políticas de puesta en marcha, copias de seguridad y de recuperación	Riesgo en la organización al implantar una infraestructura virtual
Optimización del uso y control de los recursos	Número inadecuado de máquinas virtuales
Gran escalabilidad	Anfitrión como único punto de fallo
Aprovisionamiento de máquinas virtuales	Portabilidad condicionada
Compatibilidad hacia atrás	Disminución de las ventas del hardware
Disminución del número de servidores físicos	Dependencia del sistema operativo anfitrión
Mejora de la eficiencia energética	Dependencia de la solución de virtualización elegida
Prueba y depuración	Disponibilidad de recursos suficientes
Seguridad y aislamiento	Congestión de red por servidor
Tipificación y estandarización de plataformas y configuraciones	Incremento de la complejidad y tiempo de depuración de las actividades de Networking
Mejora de la calidad y fiabilidad	Posible aumento de complejidad en la administración
Clonación de máquinas virtuales	Nuevas problemáticas
Personalización	Licencias del software
Uso en ámbitos docentes	
Clústeres de servidores virtuales	
Flexibilidad	
Gran agilidad	
Portabilidad, Migración	
Automatización	
Cola unificada	

Como podemos ver, casi todas las ventajas que nos ofrece la virtualización son aplicables en el campo de la **consolidación de servidores en las empresas**, quienes están

obteniendo el mayor beneficio de ésta tecnología y que a su vez impulsan. Casi observando todo lo descrito anteriormente podemos llegar a responder la siguiente pregunta: **¿por qué consolidar, y no seguir con el modelo de servidores independientes?**

Hasta hace unos años, la instalación de servidores en las empresas se regía por las necesidades que tuviera, y dependiendo del servicio que se quisiera configurar se adquiría un servidor optimizado en hardware, sistema operativo, y software para desempeñar esa función. No importaba el hecho de que existieran otros servidores previamente instalados en la empresa, ni el hardware que utilizaban, sistema operativo, y software. Los servidores eran comprados, configurados e instalados porque eran los idóneos para soportar el servicio que quisiéramos implantar: una base de datos, *CRMs*, alojamiento web… Desarrollando de esta manera los *CPDs* en las empresas se llegó a una situación inmanejable llena de una mapa de servidores heterogéneos completamente en sus características: creciente número de servidores en la empresa, dificultad de administración, dificultad en el manejo de cambios, dificultad en la monitorización de todos los servidores… sumado a los costes de espacio y energéticos que implica tal número de máquinas. Y además… ¡¡infrautilizados!! Las estadísticas muestran que los servidores suelen estar usados en torno al 30% como máximo, mientras las empresas desaprovechan el otro 70% por el que pagaron, a lo que hay que sumar licencias, soporte, mantenimiento…

Otros estudios muestran además que el 75% del presupuesto de los departamentos de Tecnologías de la Información es invertido en operaciones de mantenimiento, mientras que tan sólo el 25% del mismo es usado para innovación. Como hemos visto, el uso de la virtualización repercute notablemente en el ahorro de costes en administración, lo que además permitiría a las empresas dedicar un porcentaje mayor de sus presupuestos a **innovación y gestión del servicio**. **Virtualizar es una respuesta natural y actual a las necesidades que están surgiendo en las empresas y *CPDs*. La virtualización permite a las empresas evolucionar de un *CPD* basado en hardware a otro basado en software, en el cual los recursos compartidos se asignan dinámicamente a las aplicaciones que los precisen.** Si no se virtualiza, además de no frenar todos estos problemas, lo grave es que continúan agravándose, afectando a la competitividad de la empresa. La mayoría de las empresas adoptan soluciones intermedias, en las que se virtualiza sólo una parte de los servicios y aplicaciones, típicamente los no productivos. De esta forma, incluso, las ventajas y ahorros en costes y agilidad apreciables.

Como curiosidad, y ya que hablamos de la gestión de los *CPDs* actualmente en las empresas, otra técnica empleada que es considerada opuesta conceptualmente a la virtualización es el ***grid computing* o *agregación de servidores***, en la que varios servidores son observados como uno sólo. Como podemos ver, no se trata más que de otra forma de virtualización.

La virtualización es el siguiente paso en el modelo de gestión de los servicios, equiparable en forma y contenido al que surgió al pasar de sistemas operativos de una única tarea a sistemas operativos multitarea, en el sentido de aprovechamiento de recursos y aumento de la eficiencia.

3 Conclusiones

Como hemos podido ver en los apartados anteriores el mayor campo de aplicación de la virtualización es el de la **consolidación de los servidores** que forman la infraestructura de los data centers en cualquier empresa, universidad, institución… En estos equipos corren servicios fundamentales para los productos que ofrezcan, como bases de datos, páginas web, comercio electrónico, hosting, correo electrónico,… con presencia cada vez más importante y creciente.

El objetivo principal del presente proyecto es el diseño, implementación y prueba de una pequeña infraestructura virtual, consolidando servidores con servicios de centralita *VoIP*.

Dentro de un solo servidor físico, o si se prefiere un clúster de servidores, puede ser desarrollada una completa infraestructura de servidores o centralitas *VoIP* virtuales (infraestructura virtualizada), manteniendo el servicio ofrecido a prácticamente el mismo nivel de rendimiento y con las considerables ventajas de aplicar virtualización. En las *máquinas virtuales* correrán los servicios *VoIP* y sus instancias de sistemas operativos independientes, de manera que el hardware del servidor físico/clúster de servidores físicos es abstraído a las máquinas, creyendo éstas que son servidores de telefonía IP dedicados.

Las centralitas *VoIP (Voice over IP)* se encuentran en auge actualmente y están disfrutando de una gran acogida por parte del sector empresarial debido a sus inmejorables prestaciones; la gestión de llamadas por software ofrece unas posibilidades con las que no puede competir la telefonía tradicional.

La combinación de virtualización y telefonía IP puede ofrecer unos resultados realmente excelentes explotando y uniendo las ventajas de uno y otro lado. Sin embargo, existen desventajas derivadas de esta asociación como el rendimiento final obtenido -muy importante si se quiere dotar al servicio de telefonía IP por ejemplo de alta disponibilidad o alto rendimiento, en ocasiones puede resultar dudoso dependiendo del tipo de virtualización aplicada-, problemas con el uso del hardware específico de telefonía IP por parte de las máquinas virtuales, o el estado del reloj de la centralita virtualizada. Por tanto, las ventajas y desventajas derivadas del uso combinado de virtualización y telefonía IP y su exploración serán objeto de estudio..

Así, es la *VoIP* el servicio elegido para medir sus prestaciones y rendimiento en entornos virtuales, en escenarios simples y de alta disponibilidad, y realizar una comparativa con entornos en los que no hay implantada virtualización. De esta forma experimentaremos las ventajas y desventajas de la virtualización y telefonía IP, siendo testigos de la sencillez de todo el proceso a la par de poder observar los grandes beneficios que reporta al consolidar este tipo de servidores. Finalmente, se realizarán pruebas de migración en caliente de servidores *VoIP* entre distintos servidores físicos de un clúster, midiendo tiempos de caída de servicio durante el proceso.

El Proyecto Fin de Carrera además debemos recordar es abordado desde la perspectiva del software libre y GNU/Linux, partiendo de la implementación libre de centralitas *VoIP* como es *Asterisk*, y desarrollando la virtualización de servidores de este tipo con soluciones libres de gran potencia sobre GNU/Linux, como desglosaremos en los capítulos posteriores. Concluyendo, se trata de unir sobre GNU/Linux la funcionalidad y potencia de la herramienta libre de *telefonía IP Asterisk* y de una solución de virtualización libre a elegir, que será discutida próximamente.

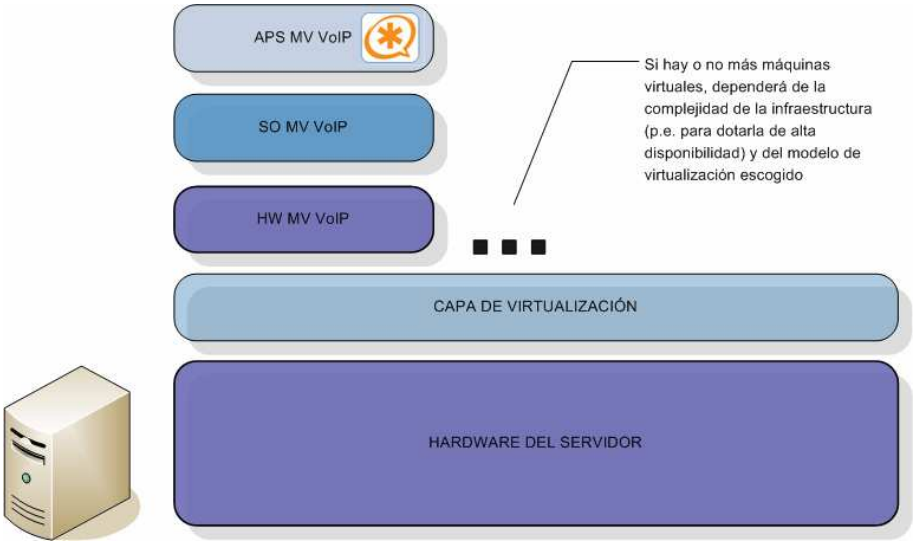


Figura 1.10 Consolidando servidores de telefonía IP Asterisk.

CAPITULO 2

VIRTUALIZACION DE PLATAFORMA

En el capítulo anterior se introdujeron y clasificaron los distintos modelos de virtualización existentes en la actualidad: ***virtualización de plataforma, virtualización de recursos individuales, virtualización de aplicaciones y virtualización de escritorio***. Profundizaremos ahora en el que es considerado el más importante de ellos, la **virtualización de plataforma, en el que es implementada la abstracción de sistemas completos para lograr *consolidación de servidores***. Dentro de esta categoría analizaremos los distintos tipos existentes –***sistemas operativos invitados, emulación, virtualización completa, paravirtualización, virtualización a nivel del sistema operativo y virtualización a nivel del kernel***- para lo que estudiaremos la arquitectura presentada así como las soluciones más relevantes.

Para la implementación de una infraestructura virtual haciendo uso de virtualización de plataforma es fundamental la **elección de un sistema de almacenamiento adecuado para los *servidores virtuales* que tengamos pensado desplegar**. Lo habitual es disponer de un espacio de almacenamiento *compartido* entre los diferentes servidores físicos que actuarán como anfitriones; de esta forma estaremos **dando soporte también a funcionalidades *extra* como es el caso de la *migración de máquinas virtuales***. Debido a la importancia de esta cuestión es necesario incluir también en este capítulo el estado del arte de los distintos tipos de almacenamiento comúnmente usados al consolidar servidores: ***sistemas de ficheros distribuidos o en red, sistemas de ficheros replicados (DRBD), y redes de almacenamiento (NAS y SAN)***.

Al terminar el capítulo realizaremos una **comparativa teórica** tanto entre las características y funcionalidad de los diferentes modelos de virtualización de plataforma como entre los diferentes sistemas de almacenamiento vistos. Así, ello establecerá la base para tomar la decisión de **elegir qué modelo/tipo de solución será implementada en el proyecto para la consolidación de servidores de *telefonía IP* con *Asterisk* y en qué tipo de almacenamiento para las máquinas virtuales usará como apoyo**.

1 Virtualización de plataforma

Hablaremos en este primer apartado del capítulo del que es sin duda el modelo de virtualización más popular y en auge en la actualidad: la virtualización de plataforma; además, éste es el modelo sobre el cual se trabajará en el desarrollo del proyecto fin de carrera debido a sus características.

Ya vimos en el capítulo anterior una introducción a este tipo de virtualización, que recordamos consiste fundamentalmente en la **abstracción de un sistema hardware completo permitiendo que diversas instancias de sistemas operativos corran sobre él**. Las *máquinas virtuales* creen que los recursos de los que disponen les pertenecen, y ven a otras *máquinas virtuales* como sistemas totalmente independientes. De esta forma, los servidores que conocíamos hasta hoy pasan a ser alojados en entes lógicos (*máquinas virtuales*) destinados a convivir con otros y a compartir los recursos físicos de los que disponen; **cuando hablamos de consolidación de servidores hablamos de virtualización de plataforma**. Dependiendo de cómo sea gestionada esta compartición y la convivencia de las máquinas virtuales, y cómo éstas sean organizadas e integradas dentro de una determinada infraestructura virtual estaremos hablando de un tipo u otro de virtualización de plataforma.

Sirva el presente apartado para diferenciar de manera clara los distintos tipos de virtualización de plataforma existentes así como para presentar el estado del arte de las más notables soluciones que pertenecen a él, presentando de manera general su forma de operar: ***sistemas operativos invitados, emulación, virtualización completa, paravirtualización, virtualización a nivel del sistema operativo y virtualización a nivel del kernel.***

1.1 ARQUITECTURA GENERAL

Aunque ya fue presentada de alguna u otra forma a lo largo del presente documento es conveniente recordar la **estructura de la arquitectura general de las soluciones de virtualización de plataforma**, justo antes de ver cuál es la arquitectura particular de cada una de ellas; en algunos casos, como se podrá apreciar, las diferencias serán bastantes. En la figura 2.1 podemos ver los elementos que la componen:

- **Hardware del servidor**. Lógicamente debemos disponer siempre de un servidor físico sobre el que aplicaremos nuestras técnicas de virtualización. **Sobre él será instalada y configurada una herramienta para virtualización, y aportará todos los recursos hardware de los que harán uso las máquinas virtuales**.

- **Capa de virtualización**. **Es el elemento virtualizador y dependiendo de la solución de virtualización escogida la capa estará ubicada de forma diferente**. Así, como veremos en los siguientes apartados, es posible que esta capa se encuentre integrada de forma conjunta con el sistema operativo en el servidor anfitrión, que sea una aplicación en el área de usuario que es ejecutada sobre un sistema operativo como cualquier otra aplicación, o bien una capa que corre directamente sobre el hardware anfitrión, como es el caso de los ***hipervisores***, introducidos en el capítulo anterior.

- **Máquinas virtuales**. Creadas, configuradas, iniciadas, monitorizadas… por la capa de virtualización, nos permiten tener de forma virtual varios equipos en uno solo. Como ya sabemos, **disponen de su propio hardware de forma virtual** (ya sea real o emulado), **su propia instancia de un sistema operativo y como es lógico corren sus propias aplicaciones** como si de un computador real se tratara. Podemos tener tantas máquinas virtuales como los recursos del servidor anfitrión nos lo permitan.

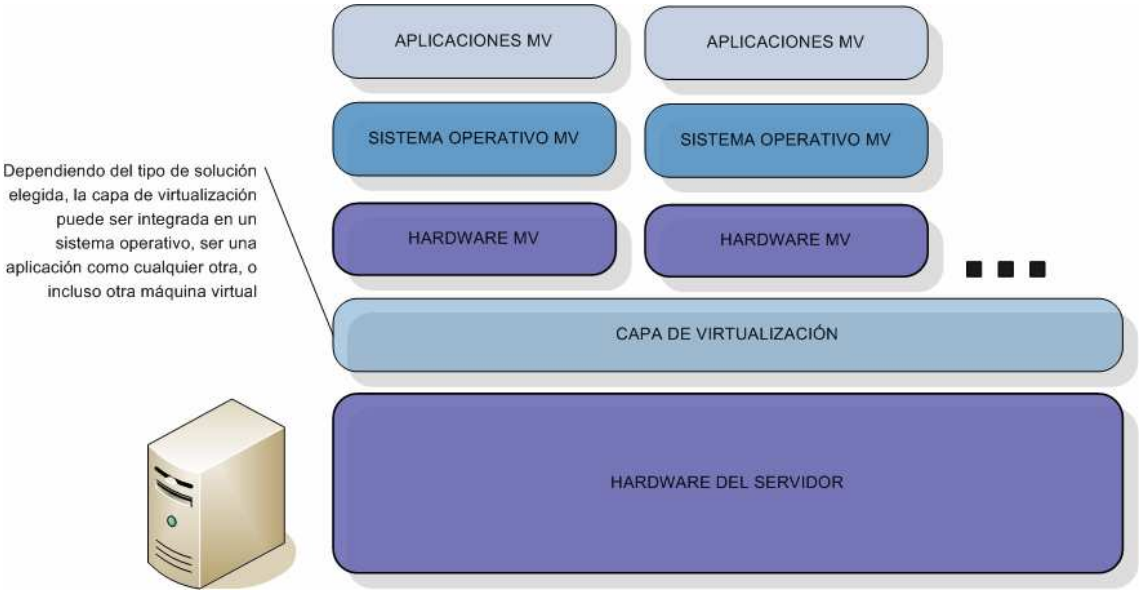


Figura 2.1 Virtualización de plataforma: arquitectura general.

A partir de aquí es posible acceder a configuraciones más complejas, ya no sólo a nivel de virtualización sino también integrando otras tecnologías. A continuación vamos a analizar la arquitectura de **dos de las aplicaciones fundamentales de la virtualización como son la creación de *clústeres de máquinas virtuales* y la *migración de máquinas virtuales*.** Por separado ya se trata de aplicaciones muy potentes, combinándolas aún más.

1.1.1 Clústeres de máquinas virtuales

Un *clúster* es una agrupación de dos o más computadores que se encuentran interconectados y que normalmente trabajan de forma conjunta con algún propósito determinado. Desde fuera, en la mayoría de los casos, el *clúster* es visto como un único equipo que ofrece unos determinados servicios.

Lo habitual es que **los componentes que forman el *clúster*, llamados *nodos***, se encuentren **interconectados mediante conexiones LAN** (*Local Area Network*) **de alta velocidad**. El objetivo principal por el que suelen ser implantados es ampliar la funcionalidad de un servidor o bien mejorar su rendimiento y/o disponibilidad. **En función de las características que presente el *clúster* y sus objetivos, podemos clasificarlos en diferentes tipos**:

- ***Clústeres* de alta disponibilidad.** El rasgo característico de este tipo de *clústeres* es la **compartición de un determinado servicio entre los diferentes nodos** que lo componen (o un grupo de ellos), los cuales **se monitorizan constantemente entre sí**, **de forma que se garantiza el funcionamiento ininterrumpido del servicio**. Tanto si se produce un fallo hardware como si ocurre alguno en las aplicaciones o servicios que corren en el nodo que las ejecuta del *clúster*, las aplicaciones o servicios son migrados por el software de alta disponibilidad de forma automática a cualquiera de los nodos restantes del *clúster*. Una vez que el problema es subsanado, los servicios migrados pueden retornar a su ubicación original en el nodo primario (llamado así porque es el que originalmente los ejecuta). **Existen diferentes configuraciones, como Activo/Pasivo** (donde sólo un nodo ejecuta servicios) **o Activo/Activo** (todos los nodos ejecutan servicios, normalmente

diferentes). **La alta disponibilidad puede ser garantizada a nivel de servicio y también a nivel de datos**, existiendo para cada uno numerosas herramientas en GNU/Linux tanto software como hardware.

- ***Clústeres* de alto rendimiento.** En este tipo de *clústeres* **el objetivo principal es proporcionar altas prestaciones de capacidad de cómputo** superior a las que pudiera ofrecer un único servidor. **El ejemplo más conocido son los llamados *clústeres* Beowulf**, en los que los nodos se encuentran localizados conjuntamente, son homogéneos y comparten una red dedicada. Normalmente son formados cuando el tamaño del problema a resolver es inmanejable por un único servidor o cuando la máquina que podría hacerlo tiene un coste muy alto. El **alto rendimiento** puede ser aplicado en distintos niveles también, siendo especialmente interesante en nuestro caso **enfocado a servicios** –en especial en aquellos que han de soportar una mayor congestión o tráfico-. Como ejemplos de software para la implementación de *clústeres* de alto rendimiento podemos citar **OSCAR** (Open Source Cluster Application Resources), distribuido bajo la licencia GPL, o **Windows HPC Server 2008** para sistemas operativos Microsoft Windows. Algunas herramientas GNU/Linux enfocadas al alto rendimiento concretamente en servicios son **KeepAlived, UltraMonkey, o LifeKeeper**.

- ***Clústeres* de balanceo de carga.** Según como se mire el balanceo de carga también podría ser interpretado como una solución de alto rendimiento. Algunos de los nodos del *clúster* actúan como *frontend* del mismo y se encargan de repartir las peticiones que reciben para un servicio determinado entre otro grupo de nodos que conforman el *backend* del *clúster*. Existen soluciones también enfocadas para servicios específicos, como Web, FTP, DNS,… es decir, **servicios que esperan recibir proporciones de tráfico elevadas**. Por ejemplo, para telefonía IP existen varias herramientas cuya funcionalidad es la de implementar un proxy SIP (protocolo de sesión más usado en la actualidad) de alto rendimiento y muy configurable; **KAMAILIO y OpenSIPS** son dos de las más importantes, derivadas de la desaparecida OpenSER.

Escalabilidad y robustez son dos características que siempre deben estar presentes en un *clúster*. Escalabilidad tanto administrativa como en carga soportada, y robustez garantizando siempre la disponibilidad de nodos que estén activos en el *clúster*.

Ahora que nos hemos introducido en los conceptos fundamentales relacionados con *clustering* podemos comenzar a ver cuál es la **relación existente entre virtualización y *clustering***. A grandes rasgos podemos decir que **consiste en el establecimiento de *clústeres virtuales* cuyos nodos son *máquinas virtuales* ubicadas en el mismo equipo.**

Teniendo en mente lo visto hasta el momento acerca de virtualización no es difícil pensar que las técnicas *clustering* pueden ser aplicadas en entornos virtualizados para aprovechar todas las ventajas particulares de estos entornos, uniéndolas a las que proporciona en los casos que son aplicables el establecimiento de *clústeres*. Un *clúster* es un caso especial en el que **las importantes ventajas que aporta la virtualización pueden llegar a apreciarse más intensamente**: ahorro en espacio y consumo energético, mayor porcentaje de utilización del hardware de los servidores, menor coste en la administración del *clúster*... pero sobre todo, que **los *nodos* del clúster pasen a ser entes lógicos** con todo lo que ello conlleva: rápida recuperación ante desastres, mejora de las políticas y rapidez de procesos como puesta en marcha, recuperación y copias de seguridad, gran escalabilidad, aumento de la seguridad, flexibilidad, etc.

De esta forma es posible implementar *clústeres* con tan sólo un único servidor físico (o varios si se deseara mayor redundancia y seguridad ante fallos en él): los nodos físicos de un clúster real pasan a ser máquinas virtuales (***nodos virtuales***) alojadas en el servidor. **Las máquinas virtuales pueden operar como *nodos* de un clúster de igual forma a como lo harían los servidores físicos**, ya que es posible la instalación y configuración en ellas de software de alta disponibilidad, alto rendimiento, balanceo de carga… Como veremos posteriormente en el proyecto, se realizará la prueba de este tipo de configuraciones, más en concreto de una **solución de alta disponibilidad para el servicio de telefonía IP *Asterisk* mediante el software *Heartbeat***, en entornos virtualizados; **el problema principal viene de establecer y configurar las máquinas virtuales que formarán el *clúster virtual*, y de ver y valorar si los recursos disponibles físicamente en el servidor que las aloja serán suficientes** o por el contrario no permitirán el funcionamiento del *clúster* de forma óptima. Prácticamente todas las soluciones de virtualización que conocemos y que ofrecen las funcionalidades necesarias para su aplicación en entornos empresariales con grandes exigencias, como *Xen* o *VMware* como ejemplos más claros, permiten crear *clústeres* de una forma sencilla ya que proveen mecanismos de configuración (especialmente de la red que intercomunica los *nodos virtuales*), monitorización, recuperación y migración de máquinas virtuales, etc.

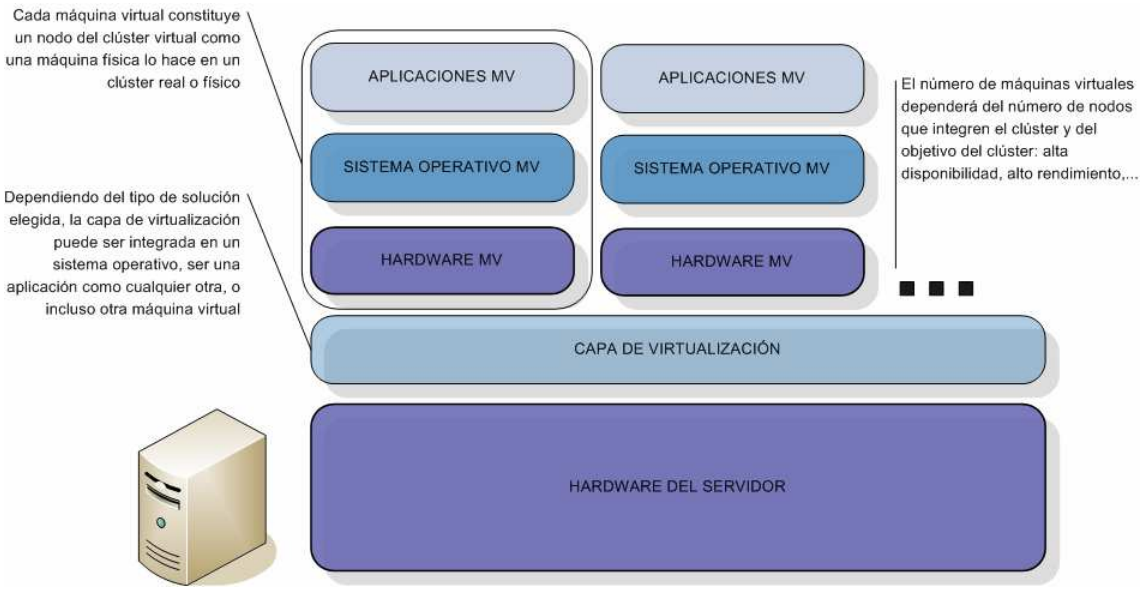


Figura 2.2 Clúster de máquinas virtuales.

Es especialmente interesante debido a la gran rapidez que puede experimentar su implantación el caso de crear un *clúster virtual* tipo *Beowulf*. En este tipo de *clústeres* los *nodos* que lo conforman son idénticos, teniendo un *clúster* completamente homogéneo; además, dispone de una red dedicada para el mismo y los *nodos* se encuentran ubicados conjuntamente. Crear un *clúster virtual Beowulf* sería tan sencillo como crear una *máquina virtual – nodo tipo* y replicarlo tantas veces como nodos deseemos; para ello sabemos ya que existen mecanismos de aprovisionamiento de *máquinas virtuales* que pueden agilizar el proceso muchísimo.

Como vimos anteriormente la **escalabilidad y robustez** son dos características que siempre deben estar presentes en un *clúster*. **En el caso particular de *clústeres virtuales* estas dos características están aún más aseguradas** ya que las máquinas virtuales son mucho más escalables que equipos físicos siempre que dispongamos de los recursos suficientes, además es mucho más económico, proporcionando mayores niveles de seguridad y aislamiento a los servicios que ofrecen y rápida recuperación ante los fallos que pudieran surgir en ellas, sin influir en el resto. Si existe un único servidor físico, la escalabilidad es limitada por los recursos

disponibles en él, y la robustez a los fallos que pueda presentar –si cae el servidor físico por un fallo hardware, las máquinas virtuales que aloja caerían también. Por este motivo **es totalmente recomendable que un *clúster virtual* no sólo disponga de un servidor físico como anfitrión**.

En las figuras 2.3 y 2.4 podemos ver de forma rápida la **arquitectura de un *clúster de alta disponibilidad para el servicio de telefonía IP Asterisk con el software Heartbeat* y su versión *virtual***, la cual será implementada en este proyecto. En este caso, además, dispondremos de un servidor físico de archivos compartidos adicional para el almacenamiento de todos los datos de los clientes, los privilegios a la hora de llamar, mensajes de voz, preferencias, etc, conectado a cada uno de los *nodos* del clúster a través de la red.

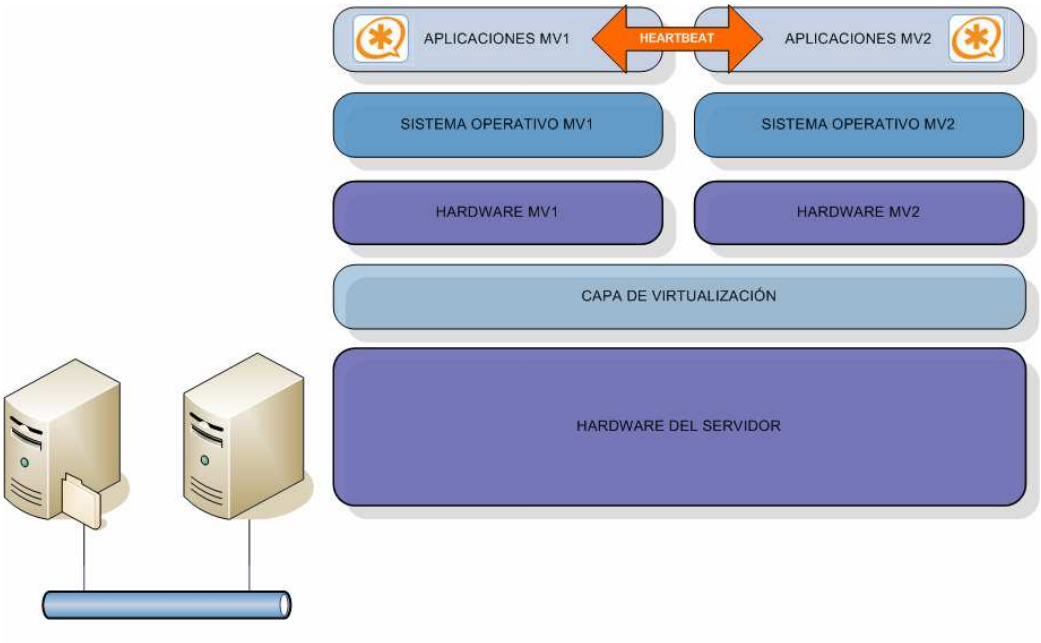


Figura 2.3 Clúster virtual de alta disponibilidad.

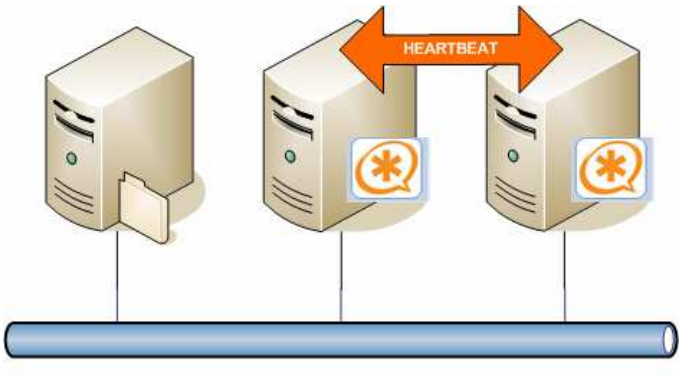


Figura 2.4 Clúster de alta disponibilidad.

1.1.2 Migración de máquinas virtuales

La migración de máquinas virtuales resulta en la actualidad un **requisito indispensable a la hora de decidirnos entre una u otra solución de virtualización**. **Su aplicación permite dotar a nuestra infraestructura virtual de mayor consistencia** al apoyar algunos de los procesos más importantes: aprovisionamiento de máquinas virtuales, balanceo de carga en los servidores físicos, recuperación efectiva ante desastres… Provoca cambios importantes en la arquitectura de virtualización en la que tenemos que empezar a pensar de forma dinámica en lugar de estática, por lo que lo más probable es que aparezcan nuevas cuestiones y problemáticas, como la monitorización del estado de las máquinas virtuales y su consumo de recursos, o la localización física en la que se encuentren actualmente en ejecución.

Sirva este pequeño apartado como una introducción teórica al proceso de migración de máquinas virtuales, que será ampliado y puesto en práctica en posteriores capítulos al implementarlo y probarlo con la solución de virtualización que haya sido elegida. La migración de máquinas virtuales, entendida como el **proceso por el cual una máquina virtual y su estado pueden ser realojadas en un servidor físico diferente al que originalmente disponía de ellos**, **puede ser realizada mediante dos técnicas diferentes**:

- **Salvando y restaurando**. Fue el primer tipo de migración que ofrecieron los productos de virtualización. Consiste a grandes rasgos en salvar y restaurar una máquina virtual: **el estado actual de una máquina virtual en ejecución es guardado en un fichero en disco, creando una *imagen* o *snapshot* del mismo, el cual puede ser usado a posteriori en la misma o en cualquier otra localización** (servidor físico anfitrión) **para iniciar de nuevo el estado de ejecución de la máquina virtual**. Lo normal es que el tamaño del estado de la máquina virtual salvado sea equivalente a la memoria que estaba siendo usada en ese momento. De esta forma es posible transportar el estado de la máquina virtual de un servidor físico a otro.

  Cuando se salva el estado de una máquina virtual se recomienda asegurar el directorio en el que se encuentre, o bien encriptar su contenido para evitar cualquier tipo de ataque. Cuando usamos este tipo de migración y restauramos una máquina virtual hay que tener muy presente el estado previo de la máquina virtual y las aplicaciones que se estaban ejecutando y si crea conflictos con otras máquinas virtuales en el servidor destino, si hay recursos disponibles para su ejecución, su configuración de red, etc.

- ***En caliente o en vivo*.** Este tipo de migración es **mucho más compleja** que la vista en el punto anterior y **es la que debe ser aplicada en los casos en los que dispongamos de servicios críticos** corriendo en las máquinas virtuales que tenemos que migrar por el motivo que sea –optimización del uso de los recursos, mantenimiento, fallos hardware o software, etc. Así, si tenemos *Service Level Agreements* (*SLA*, acuerdos de nivel del servicio) y necesitamos **alta disponibilidad e interrupción mínima de servicios**, éste es el tipo de migración que debemos aplicar.

  Con este tipo de migración *en caliente* es posible el **movimiento de una máquina virtual desde un servidor anfitrión a otro diferente mientras permanece continuamente en ejecución**. Cada solución de virtualización puede que disponga de una implementación diferente del proceso, aunque el objetivo es siempre el mismo: que el usuario final no aprecie ningún efecto en el servicio que esté usando durante la migración y permitir a administradores llevar a cabo actividades de mantenimiento o actualización offline de servidores físicos sin que

ello implique no disponer del servicio. **Por lo general suele constar de una serie de pasos comunes: pre-migración, reserva, pre-copia iterativa, parada y copia, finalización y activación**. Los efectos y ventajas de poder aplicar migración en vivo son notables: mantenimiento de servidores físicos proactivo, implementación de soluciones de alta disponibilidad, garantía de *SLAs*, balanceo de carga a través de múltiples servidores para optimizar el uso de recursos, mejora y reemplazo de hardware…

Para migrar máquinas virtuales hay una serie de requisitos de configuración, almacenamiento y red que es necesario cumplir para que sea realizada correcta y eficientemente. **Algunos de estos requisitos son**:

- Por lo general ambos **servidores físicos origen y destino deben estar ejecutando la misma solución de virtualización**.

- **El servidor destino de la máquina virtual debe disponer de suficientes recursos** de almacenamiento, memoria y procesamiento para su ejecución.

- **Los servidores físicos origen y destino deben tener la misma arquitectura y extensiones de virtualización**. Así se evitarán problemas de compatibilidad entre los juegos de instrucciones usados por librerías y sistemas operativos.

- En algunas configuraciones **es posible la necesidad de utilizar algún tipo de almacenamiento distribuido para las máquinas virtuales y su configuración**, por ejemplo algún *NFS*, sistema *NAS*, *SAN*, o el uso de herramientas y utilidades como *DRBD*.

- Lo más probable es que haya que **habilitar explícitamente el soporte de este tipo de operaciones en la solución de virtualización que estemos usando**. También, por razones de seguridad, habrá que configurar y habilitar su uso solamente en los servidores que sea necesario.

- **Puede que sea necesario establecer reglas específicas para las conexiones de la migración en los cortafuegos**, si se dispone de ellos.

En la figura 2.5 podemos ver gráficamente un sencillo ejemplo de migración de máquinas virtuales. La máquina virtual MV2 que se encuentra ejecutándose en un servidor físico origen es migrada y puesta en marcha en un servidor físico destino que ejecuta la misma solución de virtualización, a través de una red de interconexión local que une los dos servidores y haciendo uso de un *sistema de ficheros distribuido* desde el cual ambos servidores pueden tener acceso a los archivos de configuración y estado de la máquina virtual. Gracias a la funcionalidad proporcionada por la solución de virtualización que escojamos la máquina virtual podrá ser migrada *en caliente*; copiada iterativamente de origen a destino, detenida en origen y arrancada en destino con tiempo de caída inapreciable por parte de los usuarios, dotando a nuestro *clúster* de alta disponibilidad. Si la migración no fuera *en caliente*, simplemente habría que salvar el estado en origen y restaurarlo en destino: como hace uso de un sistema de ficheros distribuido, la localización de la *imagen o snapshot* de la máquina virtual será siempre la misma, cambiando solamente de localización el software que la lanza.

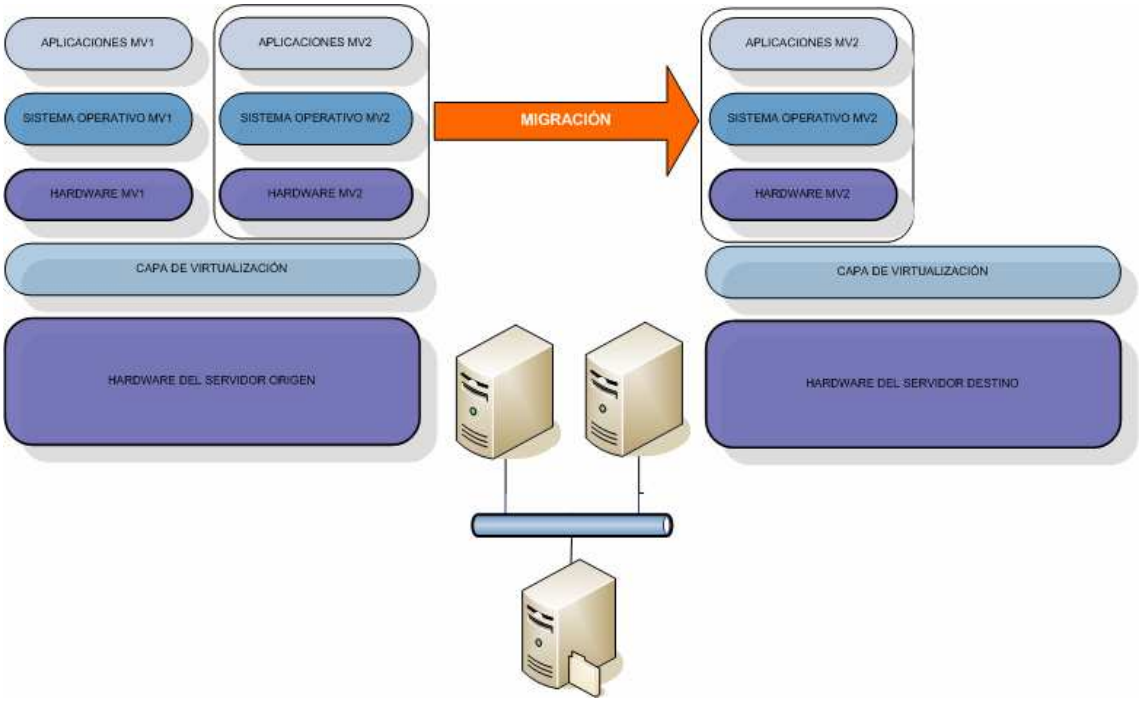


Figura 2.5 Migración de máquinas virtuales.

1.2 SISTEMAS OPERATIVOS INVITADOS

Como se introdujo brevemente en el capítulo primero la aproximación *sistemas operativos* invitados **permite crear, configurar y mantener máquinas virtuales que son ejecutadas dentro de una aplicación, llamada *aplicación de* virtualización**, que corre sobre un sistema operativo de la misma forma a como lo hace cualquier otra aplicación. **Esta aplicación gestiona y administra las máquinas virtuales, controla el acceso a los recursos** hardware disponibles en el equipo físico **e intercepta y trata cualquier instrucción privilegiada** emitida por las máquinas virtuales.

Algunos ejemplos de soluciones de este tipo son **VMware Workstation**, **Parallels Desktop, Sun xVM VirtualBox, VMware Player, y Microsoft Virtual PC** (puede ser clasificado también dentro de esta categoría, al igual que dentro de los emuladores). VirtualBox, la que es a mi juicio la más relevante debido a su fuerte "pegada" en la actualidad y el enorme abanico de funcionalidades de calidad que aporta, será tratado más extensamente en un apartado posterior.

Figura 2.6 Productos de virtualización con sistemas operativos invitados.

1.2.1 Arquitectura

En la figura 2.7 podemos contemplar la arquitectura general de las soluciones que implementan virtualización haciendo uso del modelo de sistemas operativos invitados. Sobre un equipo físico anfitrión corre un sistema operativo que actúa como anfitrión y que proporciona el entorno de ejecución para la ***aplicación de virtualización***, esto es el *elemento virtualizador*, que permite la construcción de ***máquinas virtuales invitadas*** que corren en alto porcentaje en área de usuario, las cuales disponen de sus respectivos *sistemas operativos (invitados)* y *aplicaciones virtualizadas.*

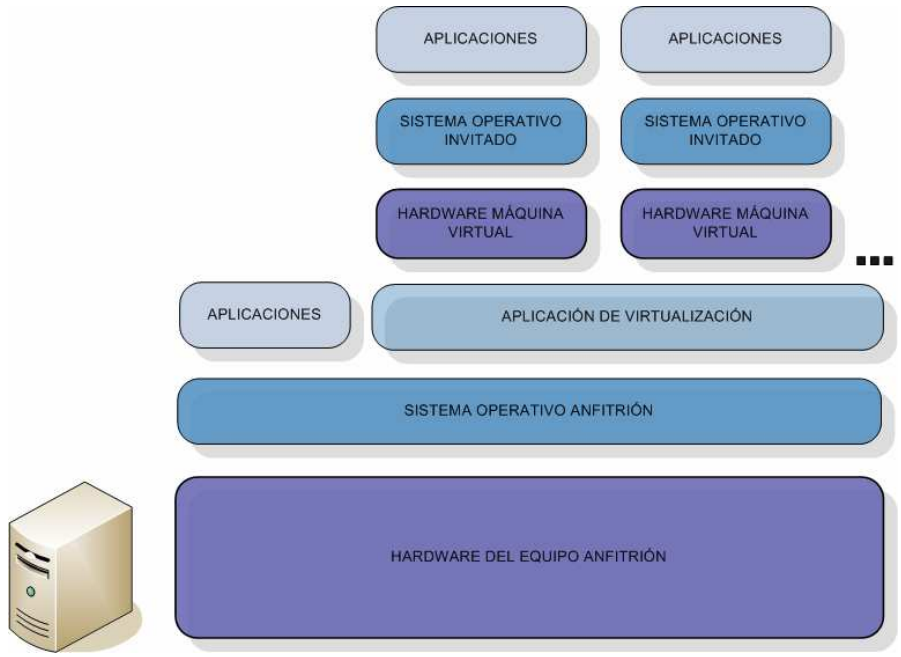


Figura 2.7 Sistemas operativos invitados: arquiectura general.

Se puede notar la ausencia por completo de hipervisor, ya que la aplicación de virtualización es ejecutada al mismo nivel que cualquier otra aplicación sobre el sistema operativo anfitrión. **Los servidores o máquinas virtuales actúan como equipos totalmente independientes** y en el caso de que la aplicación de virtualización implemente traducción de instrucciones o emulación podrán ser ejecutados sistemas operativos y aplicaciones en las máquinas virtuales compilados para un procesador y arquitectura diferentes a las del equipo host. A continuación veremos **la herramienta más importante en lo que a sistemas operativos invitados se refiere, VirtualBox**.

1.2.2 VirtualBox

VirtualBox (http://www.virtualbox.org/) es una solución de código abierto (en su versión *Open Source Edition*, ya que también dispone de una edición comercial) de virtualización con sistemas operativos invitados desarrollada por Innotek, Sun Microsystems y la comunidad Linux que **puede correr máquinas virtuales de 32 y 64 bits Linux, Microsoft Windows, Solarios, BSD, o IBM OS/2 en hosts Microsoft Windows, Mac OS, Linux y OpenSolaris**.

Se trata de una solución altamente profesional y muy bien cualificada para virtualizar a nivel de usuario y en ocasiones a nivel empresarial. Se trata de un proyecto desarrollado y mantenido por una comunidad muy activa, proporcionando con gran frecuencia nuevas versiones que amplían sus características poco a poco; la versión actual es la 3.1.2. **Depende**, como otras tantas soluciones libres Linux de virtualización, **de la emulación realizada por Qemu**. De hecho, VirtualBox trabaja de forma análoga a como lo hace *Qemu* con el módulo *Kqemu* en el *kernel*: **opera en área de usuario, aunque algunas de las operaciones las realiza en el anillo 0; ejecuta todo lo que puede en modo nativo, y simula código en modo real o instrucciones delicadas**. VirtualBox, sin embargo, **el código de área de kernel que no necesita interpretar lo ejecuta en el anillo 1, con la considerable ventaja en el rendimiento**; antes de ejecutar el anillo 1 localiza instrucciones que puedan resultar problemáticas y las sustituye por código nativo. VirtualBox **soporta tanto la tecnología Intel VT como AMD-V: en este caso actúa como un virtualizador puro**.

VirtualBox proporciona **interfaces gráfica y basada en línea de comandos** para la manipulación de las máquinas virtuales: creación, administración, configuración,… La interfaz gráfica, como se puede observar en la figura 2.8, es bastante similar a la de herramientas como Parallels Workstation, VMware Workstation y Server, o incluso Microsoft Virtual PC.

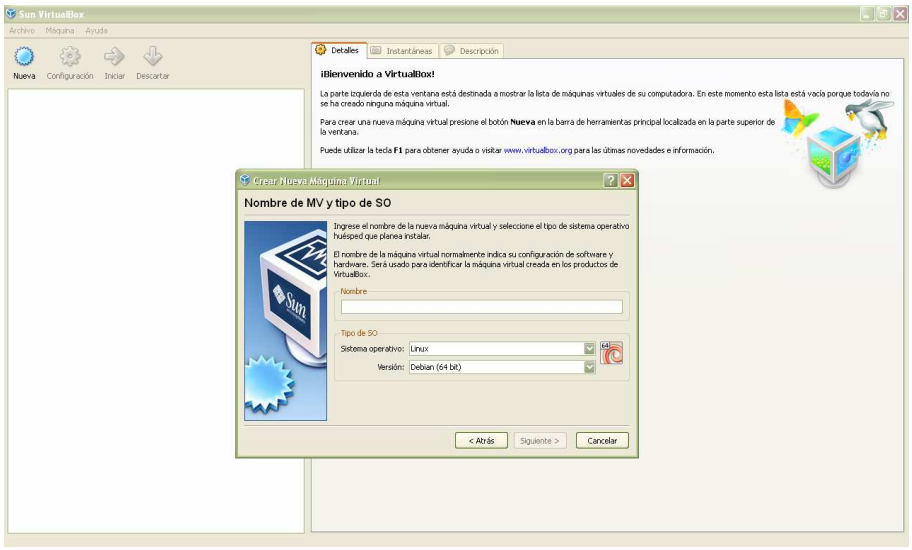


Figura 2.8 Interfaz gráfica de VirtualBox: creando una nueva máquina virtual.

VirtualBox dispone de **características que lo hacen especialmente atractivo**:

- **Es altamente modular**. Dispone de un diseño que permite el desarrollo de nuevas interfaces y uso de las existentes de una forma sencilla y eficiente: es posible por ejemplo iniciar una máquina virtual y manipularla bien desde la interfaz gráfica o línea de comandos con una gran consistencia.

- **Descripciones XML de las máquinas virtuales**. Ello permite que sean altamente portables.

- **Existe software adicional para los sistemas operativos invitados** que permite integrar su manipulación de mejor forma; por ejemplo para cambiar de una máquina virtual a otra, para el manejo del puntero del ratón, etc.

- **Es posible el uso de carpetas compartidas** entre las diferentes máquinas virtuales y los hosts.

- **Permite la toma de instantáneas del estado de las máquinas virtuales** con gran facilidad, permitiendo la recuperación de las mismas o vuelta a un estado anterior si se desea.

Otras características extra se pueden encontrar en la versión comercial de VirtualBox, tales como controladores USB virtuales que permiten una gestión más eficiente de los dispositivos USB locales conectados o la implementación del RDP (Remote Desktop Protocol) para el acceso remoto anfitrión virtualizador (a los sistemas operativos invitados se puede tanto con la versión libre como con la comercial, ya que se puede instalar el servidor RDP), y a dispositivos USB conectados localmente desde el sistema invitado remoto. Además, ésta versión de pago permite emular discos duros virtuales no solo como IDE, sino también como SATA; esto significa que es posible tener todos los discos duros que queramos en nuestro sistema invitado, mientras que en la versión gratuita tenemos un límite de 4 (contando el CD o DVD).

La versión comercial está sólo disponible gratuitamente para uso personal o propósitos de evaluación bajo la licencia VirtualBox Personal Use and Evaluation License (PUEL). Si quisiéramos usar esta versión de forma permanente en empresas deberíamos pagar por ello, sin tener acceso al código fuente. En cambio, como sabemos la edición libre permite obtener las fuentes y binarios completamente de forma gratuita bajo los términos GPL2 (VirtualBox OSE – *Open Source Edition*) para cualquiera que sea nuestro objetivo.

VirtualBox **permite configurar de diversas formas las conexiones de red entre el host y las máquinas virtuales**. Por ejemplo, **es posible para cada máquina integrar hasta 4 adaptadores de red con configuraciones NAT** (el host actuará como router de las máquinas virtuales y hará NAT con sus direcciones de red y conexiones), **bridge o puente** (proporcionando conexión directa al medio físico para la máquina virtual), **red interna** (entre las máquinas virtuales, actuando el host como router) **o host-only** (red privada compartida con el host), **seleccionando el tipo de adaptador entre una lista y la dirección MAC del mismo**. Otra opción interesante es a la hora de crear los medios físicos de almacenamiento, que podamos hacerlo bien reservando el espacio total en disco desde el principio o dejando que ese espacio vaya ocupándose de forma dinámica. Podemos incluso habilitar aceleración 3D para la pantalla o configurar la cantidad de memoria de video, puertos serie y USB con facilidad. Como vemos con estos ejemplos, **las opciones que presenta VirtualBox para las máquinas virtuales son altamente configurables**.

Concluyendo, **VirtualBox es extremadamente fácil de configurar y usar** –no hace falta ser un experto en virtualización para comenzar a poner en funcionamiento sistemas invitados en nuestro host- obteniendo al mismo tiempo unos beneficios y ventajas relativas a virtualización muy grandes; claro está, todo ello a costa de suprimir funcionalidades que serían fundamentales para su aplicación en entornos empresariales donde los requisitos sean exigentes. Es por todo ello que considero VirtualBox como la **herramienta de virtualización ideal para su uso en entornos de escritorio**, en los cuales queremos obtener un gran rendimiento con rapidez, cuando queremos crear máquinas virtuales para unos propósitos muy específicos y en la mayoría de los casos de prueba de aplicaciones.

1.3 EMULACION

La finalidad de la *emulación* no es otra que la de **imitar o suplantar una arquitectura al completo** (procesador, memoria, conjunto de instrucciones, comunicaciones…). De esta

forma, **el software llamado *emulador* puede hacer creer a los programas y sistemas operativos diseñados para una arquitectura concreta que son ejecutados sobre ella realmente; ésta es la *arquitectura emulada***. En la figura 2.9 Podemos apreciar la arquitectura general con la que trabajamos al usar emulación para virtualizar una plataforma.

La ventaja obvia primera es que permite la **emulación de cualquier máquina** incluso siendo diferente de la que disponemos: desde máquinas del pasado y que actualmente no son fabricadas hasta probables futuras arquitecturas y que necesitamos poner a prueba antes de su proceso de fabricación. Nos permite además diseñar, implementar y probar software sobre éstas arquitecturas sin la necesidad de disponer de ellas realmente. En cambio, y como contraprestación, la emulación **suele ofrecer un rendimiento bastante bajo**; lógicamente al emular arquitecturas más sencillas el rendimiento será mejor, pero lo habitual es tener que convivir con tiempos de ejecución realmente lentos.

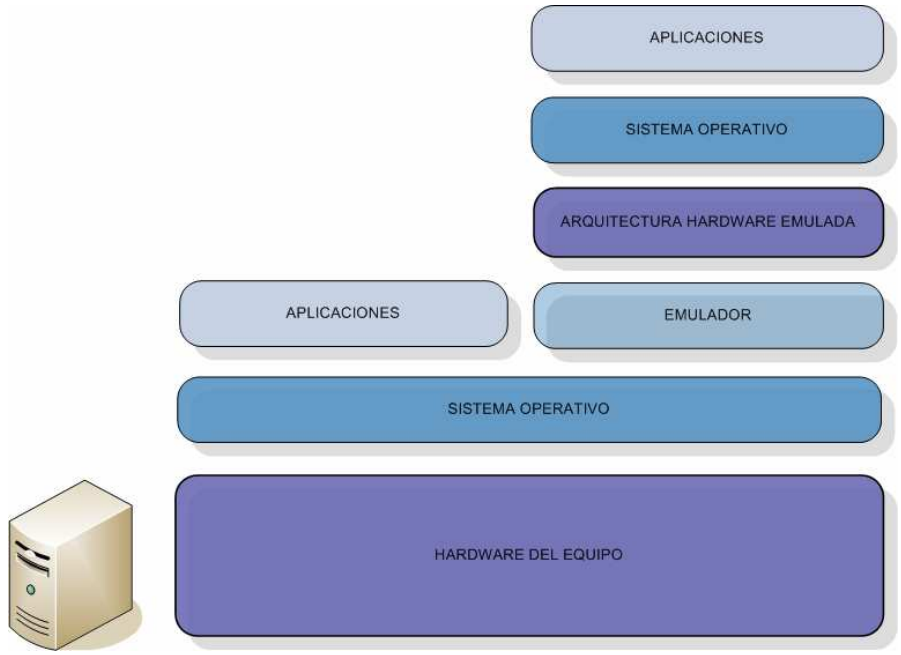


Figura 2.9 Emulación: arquitectural general.

Aunque no es recomendable el uso de emuladores en producción, puede llegar a ser realmente útil en numerosos casos incluso a pesar de la penalización en la ejecución. Por ejemplo, el desarrollo de sistemas empotrados, la programación y depuración en ciertas áreas del kernel Linux, o la prueba de distribuciones son actividades que se ven claramente beneficiadas con el uso de emuladores.

1.3.1 Bochs, MAME y VirtualPC

Uno de los emuladores libres con mayor difusión y utilizado en la actualidad, con permiso de Qemu que será analizado posteriormente, es **Bochs**.

Figura 2.10 Logotipo del emulador Bochs.

Bochs, tal y como es descrito en su web oficial en sourceforge (http://bochs.sourceforge.net/), es un **emulador altamente portable de *IA-32 (x86)* libre**. Codificado en C++, logra la emulación de la *CPU Intel x86*, dispositivos de entrada y salida comunes, y dispone de una BIOS configurable. El uso típico de Bochs es para proporcionar emulación completa de un PC x86, y puede ser compilado para **emular 386, 486, Pentium/PentiumII/PentiumIII/Pentium4 o CPU x86-64** incluyendo opcionalmente instrucciones MMX, SSEx y 3DNow! En cuanto a sistemas operativos que soporta, Bochs **permite la ejecución de Linux, DOS, Windows 95/98 y Windows NT/2000/XP o Vista**… De esta forma Bochs consigue ejecutar sistemas operativos y aplicaciones dentro del emulador en nuestro sistema, más que el concepto de *ejecutar máquinas dentro de máquinas.*

La emulación además del sector informático tiene un amplio uso en sectores lúdicos como es el caso de emuladores de consolas o videojuegos. Uno de los ejemplos más conocidos es el emulador libre ***MAME (Multiple Arcade Machine Emulator)***, **emulador de las famosas máquinas recreativas de los años ochenta y noventa**, y que nos permite volver a disfrutar de aquellos juegos ahora en sistemas operativos como Windows, Linux o MacOS X sin la necesidad de disponer de la máquina original. El número de sistemas emulados por MAME es realmente impresionante, lo que nos permite acceder a un sinfín de juegos emulados. Recientemente nació el **proyecto *MESS*** para el desarrollo de un multiemulador capaz de emular hasta 427 sistemas distintos –incluyendo algunos como *MSX, CPC, Commodore 64, o Spectrum-*.

Aparte de los usos lúdicos, **la emulación también es usada en distintos ámbitos de forma didáctica**: algunos emuladores (*MESS*) permiten a estudiantes por ejemplo instalar sistemas operativos de mainframes de IBM y aprender los detalles de su estructura y funcionamiento. Como se puede demostrar, los emuladores resultan realmente muy útiles en multitud de ocasiones.

Por lo general la emulación de videojuegos resulta incluso en la actualidad más cómoda y fácil de usar que la ejecución de los mismos en sus dispositivos originales. Sin embargo, y aunque ocurre en muchas ocasiones, esto no siempre es cierto porque en ocasiones puede ser necesario lo que es llamado ***emulación de alto nivel*** (*high-level emulation*), **en la que los desarrolladores de los emuladores deben disponer de una licencia** para ser habilitados y escribir software que imite la funcionalidad de BIOS y ROM de la arquitectura hardware original. **La emulación en definitiva no es un proceso sencillo**, incluso en muchos casos requiere del uso de ingeniería inversa para deducir cómo implementar la funcionalidad de la arquitectura a emular debido a la falta de documentación o información.

Para finalizar podemos destacar también otros emuladores como ***Microsoft VirtualPC***. Aunque la versión 2007 esté disponible gratuitamente, el resto de versiones disponen de licencia propietaria. Dependiendo del sistema operativo sobre el que sea ejecutado VirtualPC la emulación actuará de manera diferente; por ejemplo, en el caso que emulemos sobre Windows el procesador no será emulado sino que éste podrá intervenir directamente en la ejecución de instrucciones, mientras que si emulamos sobre Mac OS el procesador también deberá ser

emulado –*Intel Pentium III, 32 bits*- junto al resto del hardware. En ambos sistemas operativos **la emulación es de *recompilación dinámica***; en Windows para traducir los modos kernel y real x86 de las *máquinas virtuales* a código de usuario x86, y en Mac OS para traducir código x86 a código para el procesador PowerPC (no hay versión de VirtualPC para ordenador Macintosh con procesadores Intel). **Uno de los principales inconvenientes de VirtualPC es el soporte de aplicaciones**, ya que no soporta *cualquier programa*.

La emulación ha sido utilizada tradicionalmente por productos como por ejemplo *VMware* para proporcionar una especie de virtualización cuando realmente no se dispone de ella en procesadores que no la soportan de manera nativa.

1.3.2 Qemu

Qemu (http://www.qemu.org/) es sin duda una de las soluciones de virtualización más interesantes en la actualidad. Y es que **aunque haya sido incluido en el presente apartado como un *emulador***, cuyos motivos serán explicados a continuación, **permite también *virtualización completa* de equipos**. No en el sentido de *virtualizar* una infraestructura completa de servidores, ya que **no proporciona funcionalidades complejas como puede tratarse de la *migración de máquinas virtuales***, y sí en el de satisfacer las condiciones necesarias para la ejecución de máquinas virtuales diversas en arquitectura hardware, sistemas operativos, aplicaciones… por lo que en la mayoría de los **casos en los que queremos *virtualizar* de una manera más modesta Qemu será la solución más adecuada**, ya que no será necesario llevar a cabo complejas configuraciones –como por ejemplo recompilar el *kernel* de Linux.

Qemu es por lo general **más sencillo de instalar, configurar, y utilizar que el resto de soluciones de *virtualización de plataforma*** –las cuales veremos en siguientes secciones-, lo que lo hace ideal para situaciones en las que queremos *virtualizar en el escritorio*; esto es, ejecutar de manera rápida y sencilla máquinas virtuales en el escritorio de usuario. Eso, sumado a que se encuentra disponible también para los sistemas operativos Windows y Mac OS lo hace más **portable**.

Figura 2.11 Logotipo de Qemu.

La diferencia principal entre Qemu y otras soluciones es que **funciona como un *emulador de procesadores*, por lo que no ejecuta el código generado por las máquinas virtuales de forma nativa, sino que lo *interpreta***. La importancia de **la emulación de procesadores que lleva a cabo Qemu, incluso usada por otras soluciones de *virtualización de plataforma* como parte de su *proceso de virtualización*** (como es el caso de KVM o *Kernel-based Virtual Machine*), es el motivo principal el cual ha sido incluido como un emulador, aunque volvemos a recalcar que permite implementar también *virtualización completa*. Esta decisión tan importante en el diseño y funcionamiento interno conlleva la aparición de diferentes **ventajas e inconvenientes** directamente derivados de ello. Como se puede observar, la tabla 2.1 refleja las mismas.

Tabla 2-1. Ventajas y desventajas al virtualizar con Qemu

Ventajas	Desventajas
Portabilidad	Penalización en el rendimiento al emular y no ejecutar de forma nativa el código del anillo 0
Instancias de sistemas operativos privativos en las máquinas virtuales al trabajar como emulador	Carece de características propias de soluciones de virtualización para entornos más complejos; su ámbito puede quedar reducido al entorno de usuario
Instalación y configuración sencillas Puede ejecutar de forma nativa el código de generado por las máquinas virtuales haciendo uso del módulo complementario Kqemu en el kernel	

Es fácil intuir que **cuando Qemu es usado en entornos de usuario proporciona muchas más ventajas que en cualquier otro tipo de entorno**; de hecho, para otros entornos no estaría preparado ya que no dispone de funcionalidades que son necesarias, como la *migración* de máquinas virtuales. Qemu es altamente portable como fue comentado anteriormente, aspecto que se ha visto beneficiado sobre todo por la facilidad en su instalación y configuración.

El que trabaje con emulación tiene su lado bueno y su lado *malo*. Al emular el código del anillo 0 en lugar de ejecutarlo de forma nativa, aplicaciones como las bases de datos ven mermado bastante su rendimiento, que es una de las principales desventajas del uso de Qemu. Sin embargo, la emulación de procesadores permite la creación de máquinas virtuales que corren instancias de sistemas operativos privativos, todo ello sin tener la obligación de disponer de virtualización asistida por hardware.

Qemu no sólo permite *interpretar* el código generado por las máquinas virtuales, sino que haciendo uso de un módulo complementario para el *kernel* Linux llamada *Kqemu* es posible ejecutarlo de forma nativa. Usando *Kqemu*, Qemu puede llegar a ofrecer una **velocidad en la ejecución superior**, similar a la que ofrece una de las soluciones privativas más potentes, VMware. Sin el uso de este módulo, el rendimiento en velocidad será menor, cercano aunque todavía superior al que obtiene el emulador visto en el apartado anterior, Bochs.

Qemu puede trabajar de tres formas diferentes:

- **Ejecutando procesos que han sido compilados para un procesador diferente** al que tenemos instalados en nuestro sistema,

- **Emulando un procesador y una arquitectura concretos**. Por ejemplo puede emular x86 de 32 y 64 bits, PowerPC de 32 y 64 bits, Motorola, Sun, MIPS, ARM, ColdFire V2, PXAA270, Texas Instruments OMAP, entre otras. A diferencia de otras soluciones como puede ser Xen, emula los periféricos a los que tienen acceso las máquinas virtuales: éstas no manipulan los periféricos reales, sino otros que son mapeados en los reales. Entre todos los dispositivos emulados se encuentran los más usados controladores de disco, tarjetas gráficas, de sonido, o ratón, teclado, puertos series, tarjetas PCI/ISA de red, puertos USB, etc. La emulación que realiza Qemu es tan excepcional que implementaciones de virtualización como Xen o KVM usan esta emulación para dar soporte a conexiones series, hardware gráfico, imágenes de discos y otros.

- **Y finalmente permite emular solamente el *kernel* y ejecutar el código de área de usuario**. Cuando Qemu trabaja en este modo es **equivalente a la solución privativa VMware** en cuanto a funcionalidad ofrecida, rendimiento –su punto débil- o ventajas y desventajas derivadas de su implantación.

Qemu puede proporcionar toda la funcionalidad necesaria para virtualizar *a nivel de escritorio*. De hecho, las características de las que carece y que posee por ejemplo Xen (*migración* de máquinas virtuales) son completamente prescindibles en los niveles en los que Qemu es usado.

1.4 VIRTUALIZACION COMPLETA

La virtualización completa, **también llamada *nativa***, es un modelo de virtualización muy parecido a la paravirtualización. En ambos modelos a la hora de virtualizar hay un hecho común: el **uso de *hipervisor***. Lo que los diferencia en cambio es la **inclusión de código dentro del hipervisor para emular el hardware subyacente cuando sea necesario** en la virtualización completa, **permitiendo así la ejecución de sistemas operativos invitados *no modificados* y en general cualquier tipo de software ejecutable en el hardware disponible lo podrá hacer también en las máquinas virtuales**.

Al introducir código en el hipervisor, por lo general en la mayoría de los casos las virtualización completa **obtendrá un rendimiento relativamente menor que la paravirtualización**. **Para optimizar el rendimiento es posible la utilización de soporte hardware** específico para virtualización. En la mayoría de los casos, además, el uso de este soporte será de carácter obligatorio.

El reto principal de la virtualización completa es la **interceptación y simulación de operaciones que son privilegiadas**, como por ejemplo las instrucciones de entrada/salida. Por otro lado, las instrucciones generadas por máquinas virtuales que no afectan ni acceden a otras máquinas virtuales ni a la máquina anfitriona suelen ser ejecutadas directamente por el hardware, sin simulación alguna.

Algunas de las soluciones más importantes de virtualización completa lo son también en general del mercado de la virtualización –junto a las correspondientes a *pararvirtualización*- como por ejemplo **VMware** (en la mayoría de sus variantes), **XenServer**, **z/VM**, **Oracle VM**, **Sun xVM Server**, **Virtual Server**, o **Hyper-V**. Otras soluciones, por ejemplo implementan lo que se llama *traducción binaria* de las instrucciones que genera la máquina virtual; esto no es virtualización completa, sólo aparenta serlo.

Hay **escenarios en los cuales a lo largo de los años la virtualización completa ha sido puesta en práctica con bastante éxito**:

- Compartición de equipos entre múltiples usuarios.
- Aislamiento de usuarios.
- Emulación de nuevo hardware para obtener mejores registros de fiabilidad, seguridad y productividad.

1.4.1 Arquitectura

En la figura 2.12 podemos ver la **arquitectura usada en el modelo de virtualización completa. Consta de los siguientes elementos**:

- Hardware la máquina anfitriona que incluya un procesador con soporte a tecnología de virtualización, como **Intel VT o AMD-V.**

- **Un hipervisor normalmente *native*** o ***bare metal***, esto es, que es ejecutado directamente sobre el hardware disponible, **y en el que se han realizado diversas modificaciones (emulación hardware) para el manejo y administración de la infraestructura virtual**. Sin embargo, **también es posible encontrar arquitecturas de virtualización completa en las que el hipervisor, llamado en este caso *hosted*, está integrado con un sistema operativo determinado**; este tipo es usado en escenarios en los que los requisitos son de menor importancia o el entorno es menos exigente.

- La presencia de una **máquina virtual con carácter administrativo**, **aunque también puede que sea una consola que disponga de este control administrativo**.

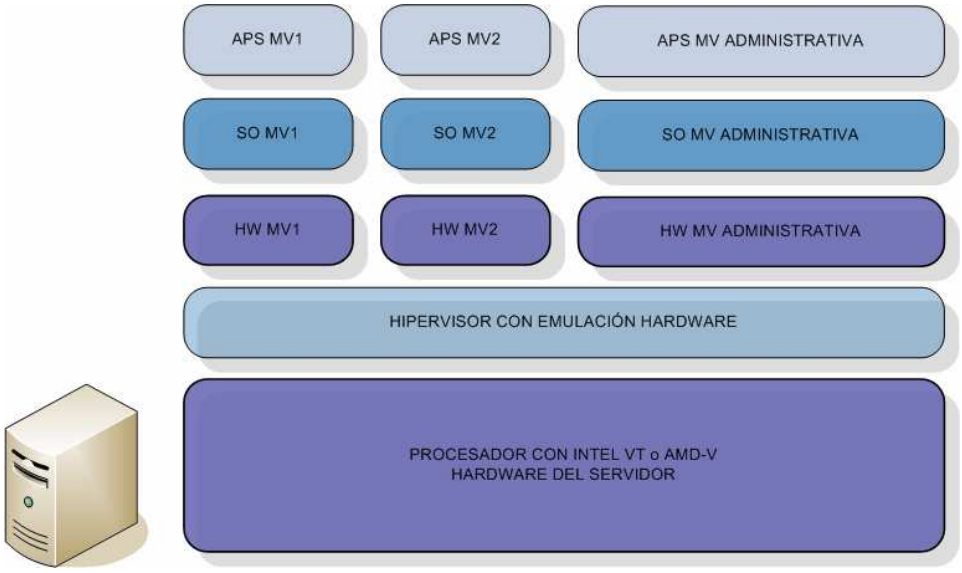


Figura 2.12 Arquitectura general de virtualización completa.

La emulación de ciertas instrucciones privilegiadas sumado a las modificaciones que hay que llevar a cabo en el hipervisor que opere en la infraestructura virtual hace que **por lo general las soluciones de virtualización completa sean consideradas de un rendimiento menor que otras como por ejemplo las de paravirtualización**.

1.4.2 VMware

VMware (http://www.vmware.com/) es sin duda alguna **uno de los gigantes de la virtualización**. **Fue pionero a principios de los años noventa** a la hora de ofrecer excelentes y completas soluciones para sistemas operativos tanto Microsoft Windows como Linux, en equipos cuyos recursos hardware en teoría hasta entonces no eran suficientes para la

implementación de este tipo de actividades. Aunque lo haya clasificado bajo el paradigma de virtualización de plataforma y más concretamente en virtualización completa, **la empresa ofrece soluciones de virtualización prácticamente a todos los niveles y para todas las necesidades**. No contento con eso, desde hace poco VMware está tratando de hacerse camino en las novedosas técnicas del ***cloud computing***.

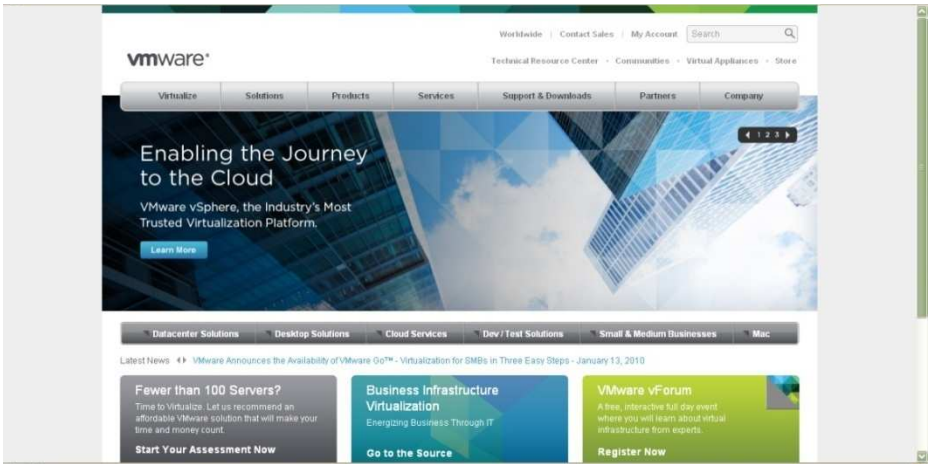


Figura 2.13 Portal web de VMWare: http://www.VWware.com/.

Un momento importante en la evolución de VMware y la virtualización fue la puesta en marcha como software gratuito (que no libre, que hubiera sido aún mejor) de VMware Player y VMware Server, hace ya años, demostrando que ello puede llegar a ser una aproximación muy beneficiosa para cualquier empresa. Es difícil resumir en un par de páginas toda la línea de trabajo que ha seguido durante los últimos años y que continúa ofreciendo VMware. Es por ello que, **agrupadas conceptualmente como las agrupa la propia VMware, clasificamos y presentamos muy brevemente las soluciones desarrolladas y que ofrecen en la actualidad** en la tabla 2.2.

Tabla 2-2. Productos VMware

Categoría	Breve descripción	Productos
Data Centers	Soluciones para entornos empresariales e infraestructuras de servidores virtuales con requerimientos exigentes.	VMware vSphere 4, VMware Server, VMWare ESXi
Escritorio	Productos de virtualización a nivel de usuario particular, enfocado sobre todo a desarrollo y prueba de software. También hay herramientas para implementar virtualización de escritorio.	VMware View 4, WMware ThinApp, VMware ACE, VMware Workstation, VMWare Player
Administración	Herramientas de administración sofisticadas y eficientes para mejorar el rendimiento de la infraestructura de servidores virtuales. Aumentan el potencial de la virtualización.	VMware vCenter Product Family, VMware vCenter Server, Server Heartbeat, Orchestrator, Site Recovery Manager, Lab Manager, Lifecycle Manager, Converter, Chargeback, ConfigControl,

		CapacityIQ, AppSpeed
Mac	Software de virtualización para el escritorio en equipos Macintosh.	VMware Fusion
Servicios Cloud	Soluciones de administración segura de aplicaciones tanto para nubes privadas como para su comunicación con públicas.	VMware vCloud, VMware vCloud Express

Como podemos ver, **la cantidad y la variedad de las soluciones que aporta VMware en el mundo de la virtualización son impresionantes**. Se hace necesario hablar un poco más de algunas de ellas por su relevancia en la actualidad, sobre todo las encuadradas en la categoría *data centers*:

- **VMware vSphere 4**. Es una solución que se encuentra diseñada para ofrecer **en servicios y aplicaciones altos niveles de respuesta y disponibilidad**. Está considerada por VMware como su producto más fiable a nivel de virtualización en *data centers*. **Permite unir todo el potencial de la virtualización y *cloud computing***.

- **VMware Serve**r. Logra que se alcancen los beneficios de la virtualización de una manera rápida a la vez que eficiente. Se considera como un paso previo antes de usar vSphere. Su aplicación para **consolidación de servidores** es inmediata. VMware Server se integra con un sistema operativo Linux o Windows, por lo que no requiere de un servidor dedicado. Su descarga es gratuita.

- **VMware ESXi**. Se trata de un producto mucho más completo que VMware Server; incluye administración centralizada y su rendimiento es también mejor, siendo muy fácil su escalabilidad hacia vSphere. En cambio, al tratarse de una solución **basada en *hipervisor bare metal*** (que es ejecutado directamente sobre el hardware) sí precisa el uso de un servidor dedicado de 64 bits para la virtualización. Su uso típico también es la consolidación de servidores, además del desarrollo y prueba de software. Incluye importantísimas características de última generación en virtualización referentes a **administración avanzada de los recursos, escalabilidad, alta disponibilidad o seguridad**. Al igual que VMware Server, su descarga es gratuita, teniendo disponible la compra de soporte y material para de aprendizaje.

- **VMware Workstation**. Es una solución muy popular debido a su **facilidad de uso**. Permite a la mayoría de los usuarios adentrarse en el mundo de la virtualización, **ofreciendo una gran flexibilidad en la creación y utilización de las máquinas virtuales, con un interfaz totalmente intuitivo a la vez de potente**, por lo que es muy utilizada en entornos educativos con fines de prueba y depuración de sistemas operativos y aplicaciones. Es un producto que ha acumulado muchos años de experiencia y eso se nota en su rendimiento. Incluye una importante aplicación para la administración de las redes virtuales a crear (NAT, Bridge, red privada, host-only), además de importantes opciones para distribuir máquinas virtuales a otros usuarios y establecer sus políticas de uso en las últimas ediciones (ACE).

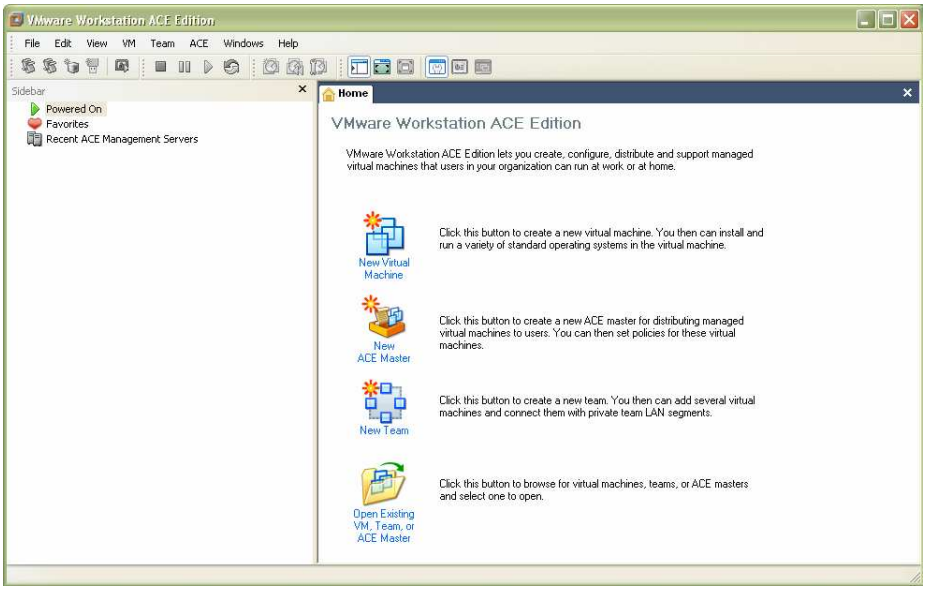


Figura 2.14 Interfaz administrativa de VMWare Workstation ACE Edition.

- **VMware ACE y Player**. Son otras de las soluciones cuyo uso se encuentra más extendido. **VMware Player permite simplemente la ejecución de máquinas virtuales** creadas previamente y localizadas en local, o bien descargándolas de repositorios creados a tal efecto. **VMware ACE** en cambio tiene una funcionalidad más compleja, y se presenta en la mayoría de los casos como una extensión a Workstation; **es usado para crear instancias de máquinas virtuales a distribuir para distintos usuarios**, junto con las políticas de uso asociadas y términos de seguridad. Después, estas copias pueden ser administradas de manera centralizada desde el *ACE Management Server*, como podemos ver en la figura 2.15.

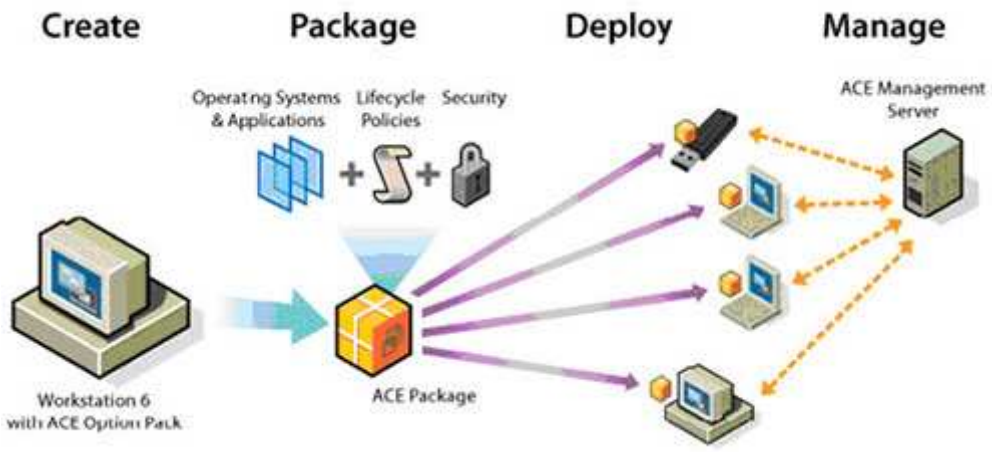


Figura 2.15 Arquitectura de uso de VMWare ACE.

Las soluciones a nivel *data center* permiten aprovechar al máximo las ventajas derivadas del uso de la virtualización, permitiendo a los administradores centrarse en actividades creativas y de negocio en lugar de rutinas para el mantenimiento software y hardware. Independientemente del producto que estemos hablando en VMware, siempre soportará la instalación y ejecución de sistemas operativos x86 no modificados. Incluso algunas

de las soluciones que pueden aparecer al usuario como más limitadas disponen de características excelentes de virtualización; por ejemplo, la ejecución de múltiples máquinas virtuales simultáneamente y una completa configuración y personalización de las mismas.

1.4.3 Citrix XenServer

En el año 2007 **Citrix compró una empresa llamada XenSource que ofrecía diferentes servicios de virtualización, originalmente formada por ex miembros del proyecto original del *hipervisor* Xen en la Universidad de Cambridge**, dedicada por completo al desarrollo y soporte en el uso de Xen. Desde antes de este momento, XenSource ya era uno de los principales *sponsors* de Xen; ahora con Citrix sucede lo mismo, algo que es muy positivo para un proyecto de software libre como Xen. De esta forma Citrix, históricamente inmersa en soluciones de acceso remoto a equipo Windows, se adentraba en el mercado de la virtualización.

Aunque queramos considerar XenServer de Citrix y Xen como dos soluciones de virtualización diferentes, en realidad no lo son, ya que **la base de ambas es exactamente la misma**, independientemente de características *extra* que sean añadidas: **el *hipervisor* Xen**, que corre directamente sobre el hardware del servidor anfitrión.

Tomando el *hipervisor* Xen como punto de partida, el equipo primero de XenSource y en la actualidad de Citrix desarrolló sobre él una completa plataforma de virtualización de servidores, proporcionando **consolidación de servidores efectiva**. Una de las ventajas de XenServer es que ha sido **utilizado mucho en entornos empresariales, y más recientemente, con infraestructuras *cloud***.

Las principales diferencias entre Citrix XenServer y Xen son dos: **XenServer precisa de una máquina *host* de 64 bits y un equipo con el sistema operativo Windows instalado para poder gestionar tanto la propia máquina *host* como la infraestructura virtual que deseemos crear**; Xen *open source*, por el contrario, puede ser ejecutado en una máquina con hardware de 32 bits sin la necesidad de disponer de un cliente Windows para administrarlo. El soporte hardware para virtualización en los procesadores es necesario cuando queremos utilizar sistemas operativos no modificables (por ejemplo alguno de la familia Microsoft Windows).

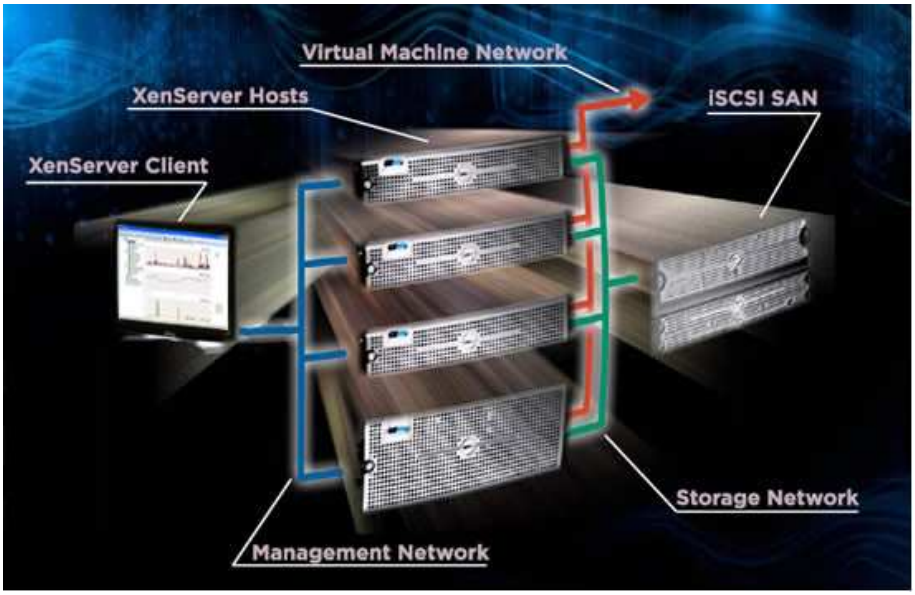


Figura 2.16 XenServer necesita de un cliente para la administración de la infraestructura virtual.

XenServer, en principio gratuito pero no libre, **añade ciertos elementos que pueden hacerlo más atractivo que Xen** (completamente gratuito y *open source*), aunque haya que pagar por su uso y no sea ni mucho menos *open source*. Además, en cierto modo es más sencillo de usar ya que **desde la propia instalación de la solución ésta muestra que está muy enfocada a facilitar su uso entre los usuarios**. **Nuestra elección entre ambas soluciones depende entonces de las necesidades que tengamos de disponer de ciertas características *extra*, que XenServer denomina *"Essentials"*.**

Los *essentials* de Citrix incluyen por ejemplo herramientas como XenMotion para la gestión de migraciones de máquinas virtuales o XenCenter para administración avanzada de forma centralizada. Además **dos versiones se encuentran disponibles, *Enterprise* y *Platinum***, la segunda más completa aún que la primera. Algunos de los aspectos que pueden ser mejorados gracias a las características que ofrecen son:

- Integración y administración del almacenamiento disponible.
- Balanceo de carga de trabajo de forma dinámica. Migración en caliente de máquinas virtuales.
- Administración avanzada de procesos en la infraestructura virtual y su estado.
- Aprovisionamiento de máquinas virtuales dinámico.
- Alta disponibilidad.

Para más información podemos visitar la web oficial de XenServer en Citrix: http://www.citrix.com/English/ps2/products/feature.asp?contentID=1686939

1.4.4 z/VM

z/VM (http://www.vm.ibm.com/) **es la solución ofrecida por IBM en el sector de la virtualización**. Puesto en el mercado a finales del año 2000, supuso un paso muy importante en una empresa que desde la década de los sesenta ha creado los *mainframes* más eficientes y ha trabajado con técnicas relacionadas con *virtualización.* Así, representó y sigue representando la **evolución de las soluciones basadas en los conceptos y tecnologías desarrollados bajo la experiencia de IBM**, con origen en el CP/CMS en los System/360-67 de IBM.

z/VM es ejecutado en los zSeries de IBM, los computadores System z9 y System z10. La versión 6.1 requiere el uso de la z/Architecture 2 (ARCHLVL 3), que se encuentra implementada en los modelos System z10 de IBM. **El *hipervisor* usado por z/VM es de tipo *native***, es decir, se encuentra corriendo directamente sobre el hardware disponible en el equipo anfitrión. z/VM **soporta como sistemas operativos en las máquinas virtuales Linux, z/OS, z/OS.e, TPF** *(Transaction Processing Facility)* **y z/VSE. También es posible instalar el propio z/VM, lo que nos permite anidar máquinas virtuales**.

En la figura 2.17 podemos ver **un ejemplo de utilización de z/VM.** En ella podemos ver un servidor con dos particiones lógicas independientes pero que comparten los recursos físicos, cada una de las cuales ejecuta una instancia de z/VM. A su vez, cada una de las instancias de z/VM se encuentra ejecutando tres máquinas virtuales, con su propio entorno y aplicaciones, independientes entre sí, aunque compartiendo los mismos recursos físicos también.

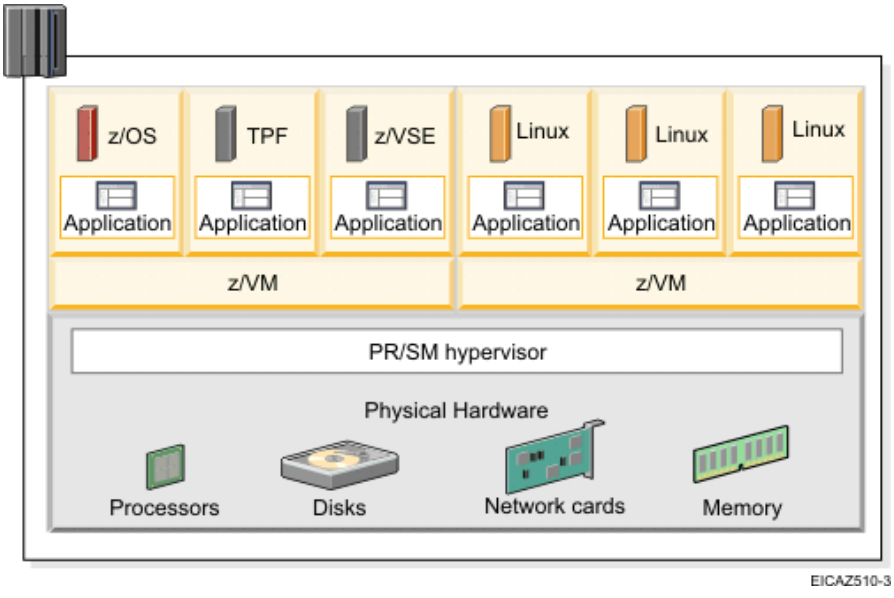


Figura 2.17 Ejemplo de uso de z/VM en dos particiones independientes.

Típicamente z/VM permite la creación de millares de máquinas virtuales *Linux* en un único sistema o partición lógica, siempre con la limitación impuesta de la cantidad de recursos disponibles. z/VM emula los distintos recursos físicos disponibles a las máquinas virtuales creando colas de recursos compartidas. De esta forma, z/VM tiene una gran capacidad para administrar cómo, y hasta qué punto, cada recurso físico es usado por cada máquina virtual.

1.5 PARAVIRTUALIZACION

Como he dicho anteriormente, se trata del **modelo de virtualización quizás más importante junto al de virtualización completa**. Comentado con brevedad en el primer capítulo, es momento de entrar en detalle en cuanto a su modo de operar, arquitectura, qué lo hace diferente del resto y lo más importante: la solución de virtualización **Xen**, sino el que más, uno de los más extendidos productos de virtualización en el sector empresarial hoy en día.

La similitud que guarda con la virtualización completa es evidente: **hace uso de un *hipervisor*** (bien *native/bare metal* o bien *hosted* en un sistema operativo) **como capa de virtualización y así compartir y administrar el acceso al hardware subyacente**. Sin embargo, las siguientes diferencias son determinantes: **no incluye emulación del hardware en el *hipervisor***, como hace la virtualización completa, **y sí modificaciones en los sistemas operativos invitados**, **haciendo que éstos estén al tanto del proceso de virtualización, cooperando en él al comunicarse directamente con el hipervisor**.

La introducción de estas modificaciones en los sistemas operativos de las máquinas virtuales tiene dos implicaciones de importancia:

- **En principio, los sistemas operativos invitados deben ser modificables** lógicamente. Esto excluye a todos los sistemas operativos privativos, como por ejemplo Microsoft Windows. **En principio porque en caso de poder hacer uso de soporte hardware específico para virtualización, además de mejorar el rendimiento final de todo el proceso, podremos ejecutar sistemas operativos no modificados** al manejar por hardware todas las operaciones privilegias/protegidas, peticiones de acceso al hardware y la comunicación con las máquinas virtuales.

- **Los sistemas operativos invitados son conscientes y cooperan en el proceso, eliminando la necesidad de captura de instrucciones privilegiadas/conflictivas en el *hipervisor*, lo que mejora notablemente el rendimiento final, que es cercano al que obtendríamos en un sistema no virtualizado**. Es decir, se lograr eliminar la sobrecarga asociada a la emulación requerida para las otras tecnologías basadas en hipervisor, como es el caso de la virtualización completa.

Las modificaciones en los sistemas operativos de las máquinas virtuales siempre han sido objeto de debate; bajo mi punto de vista, considero que es más una ventaja que una desventaja, sobre todo cuando no hay necesidad alguna de usar sistemas operativos *no modificables* en las máquinas virtuales, obteniendo finalmente un mejor rendimiento. Como suele ser habitual, tanto las librerías como las aplicaciones que son ejecutadas por las máquinas virtuales deben ser compatibles con la arquitectura física del anfitrión.

Aparte de Xen, que veremos a continuación con el detalle que se merece debido a su gran importancia, hay otras soluciones encuadradas en este modelo que debemos al menos citar: **Logical Domains, Oracle VM y Sun Xvm Server, etc**.

1.5.1 Arquitectura

En la figura 2.18 se muestra de forma gráfica la **arquitectura utilizada en el modelo de *paravirtualización*** y que ya ha sido descrita en el apartado anterior. Las máquinas virtuales, con modificaciones en sus sistemas operativos, cooperan en el proceso de virtualización comunicándose de forma directa con el hipervisor, que de esta forma queda *liberado* de realizar la captura de las instrucciones privilegiadas que creen las máquinas virtuales. **Una máquina virtual actúa como entorno administrativo de toda la infraestructura virtual**.

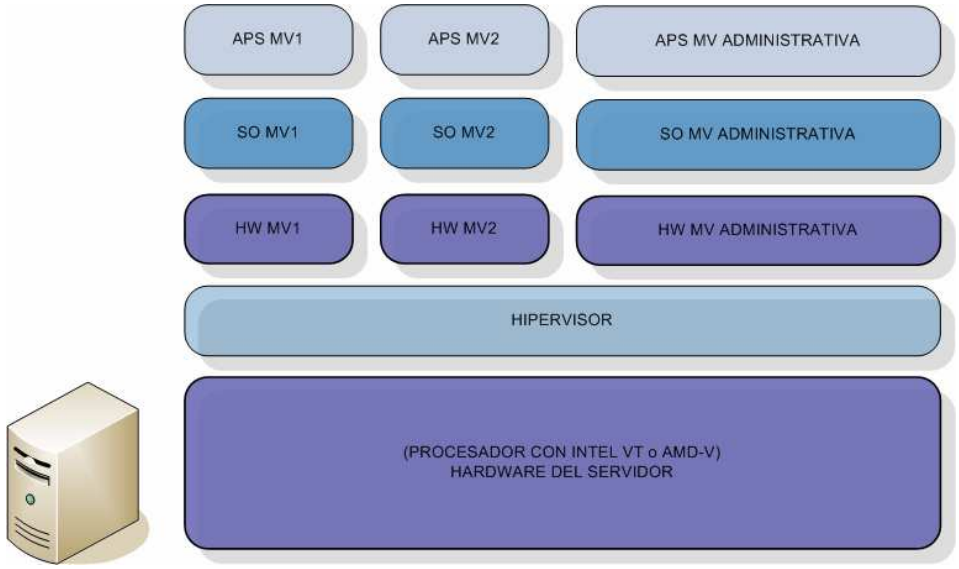


Figura 2.18 Arquitectura general de paravirtualización.

En el hardware de la máquina anfitriona no es requisito disponer de un procesador con soporte para virtualización, Intel VT o AMD-V, sólo en el caso de que quisiéramos desplegar máquinas virtuales con instancias de sistemas operativos no modificables.

1.5.2 Xen

La solución más relevante de las que ofrece *paravirtualización* y en general virtualización es Xen. Más aún, casi todo el mundo cuando habla de *paravirtualización* piensa en Xen, **pionero en la implementación del modelo** y su principal valedor a lo largo de los años. Además muchas empresas líderes en informática han apostado por Xen como solución de virtualización, como IBM, Oracle, SuSe o Sun, y hasta hace poco también Red Hat –recientemente ha sustituido a Xen por KVM como software de virtualización en sus sistemas operativos *Enterprise*-.

Xen (http://www.xen.org/) **es una solución completamente *open source* surgida en la Universidad de Cambridge** en 2003, creada por un equipo liderado por Ian Pratt y que es desarrollada y **mantenida activamente por una comunidad que también ha incluido a los mejores ingenieros en soluciones para *data centers*** en empresas incluyendo AMD, Cisco, Dell, Fujistu, HP, IBM, Intel, Mellanox, Network Appliance, Novell, Red Hat, Samsung, SGI, Sun, Unisys, Veritas, Voltaire, yCitrix. Como sabemos, aparte de esta versión libre de Xen existen otras comerciales basadas en ella con características empresariales adicionales, desarrolladas y soportadas por XenSource –adquirida, como sabemos, por Citrix Systems-.

Figura 2.19 Logotipo de Xen

Hoy en día la comunidad Xen no sólo se dedica al desarrollo del *hipervisor Xen*, que mantiene una dura competencia con *KVM* por alcanzar un mejor rendimiento y del cual hablaremos más adelante, sino que **dispone de varios proyectos que amplían el campo de actuación de la virtualización e incluso comienzan a trabajar con tecnologías *cloud***:

- **XCI**. Se trata de una versión del *hipervisor Xen* para dispositivos cliente. Es un *hipervisor* embebido que usa pequeñas librerías, por lo que requiere menos memoria y disco que la versión estándar del *hipervisor*. Está **enfocado a su uso en ordenadores portátiles u otros dispositivos móviles con arquitectura Intel o AMD**.

- **Xen Cloud Platform**. **Solución *cloud* que ofrece una pila para integrarla con virtualización**. Proporciona una completa infraestructura *cloud* con una ponderosa pila administrative basada en APIs abiertas y estandarizadas, soporte para multi-tenencia, garantías SLA (*Service Level Agreement*) y detalladas métricas para cargos basados en consumo.

- Otros proyectos, como **HXEN** (para ejecutar Xen como un *hipervisor hosted* en algún sistema operativo), **Xen ARM** (para portar Xen a procesadores ARM) o el **Proyecto Satori**. Este último es un poco particular, ya que su objetivo es dar soporte para máquinas virtuales Xen Linux paravirtualizadas en Hiper-V, el *hipervisor* desarrollado por Microsoft, y nació como consecuencia del acuerdo colaborativo entre **XenSource** y Microsoft, después de que XenSource fuera comprada por Citrix Systems.

Xen proporciona una plataforma de virtualización en la que es posible ejecutar múltiples máquinas virtuales, llamados dentro de la *terminología Xen* ***dominios***, en paralelo en una sola máquina física anfitriona que proporciona los recursos a los *dominios*. Xen permite todo esto haciendo uso de un ***hipervisor* que es ejecutado directamente sobre el hardware**, esto es, es de tipo *bare metal* o *native*.

Al tratarse de *paravirtualización* se requieren cambios en los sistemas operativos de los dominios para su cooperación en el proceso, con todo lo que ello implica y que ya hemos analizado en el apartado anterior. Así, si no disponemos de procesadores con tecnología de virtualización sólo podremos instanciar máquinas virtuales con sistemas operativos modificables (Linux, Minix, Plan 9, NetBSD, FreeBSD, OpenSolaris), algo que desde el punto de vista Linux es un compromiso totalmente razonable debido a las mejoras en el rendimiento que se obtienen respecto a otros modelos. En cambio si se requiere un soporte más amplio, ello puede presentarse como un inconveniente. Si disponemos de la nueva generación de chipsets AMD-V o Intel VT, podremos entonces ejecutar lo que Xen denomina ***Hardware Virtual Machine*, o *virtualización acelerada*, que nos permitirá la creación y ejecución de dominios con instancias de sistemas operativos no modificados**. La *HVM* media así entre los dominios y el hardware administrando sus llamadas.

Cuando Xen dispone de soporte hardware para la virtualización es considerada como la tecnología más rápida y segura al mismo tiempo como solución de virtualización, ya que **se encuentra altamente optimizado para tomar todas las ventajas de las capacidades que aportan las instrucciones Intel VT y AMD-V**. Además, **el *hipervisor Xen* es excepcionalmente ligero** (aproximadamente 50.000 líneas de código), **lo que provoca que experimente baja sobrecarga en las operaciones y así un rendimiento cercano al nativo en los dominios.**

Otro aspecto muy interesante es la **muy buena base de herramientas administrativas existentes para Xen**, fruto de la activa comunidad que lo desarrollo y soporta. Algunas de las **funcionalidades que han hecho ser a Xen una solución fundamental son las siguientes**:

- Soporte para las arquitecturas x86, x86_64 e ia64.
- Rendimiento cercano al nativo en los dominios invitados incluso para aplicaciones con grandes cargas de CPU y entrada/salida.
- Fuerte aislamiento entre los dominios invitados. Ello permite un particionamiento completo entre ellos lo que implica mejorar la seguridad de la virtualización
- Posibilidad de salvar y posteriormente restaurar dominios.
- *Migración en caliente* desde un servidor físico a otro manteniendo la disponibilidad total del dominio.
- Soporte *Hardware Virtual Machine* para ejecutar instancias de sistemas operativos no modificados en hardware Intel VT y AMD-V.
- Escalable para soportar un gran número de dominios invitados.
- Soporte de procesadores SMP de 32 bits.
- *Intel PAE* (*Physical Addressing Extensions*) para el soporte de servidores de 32 bits con más de 4Gb de memoria física.
- Herramientas de control con grandes prestaciones.
- Gráficos *AGP/DRM (Accelerated Graphics Port/Direct Rendering Manager).*
- Soporte *ACPI (Advanced Configuration and Power Interface,* Interfaz Avanzada de Configuración y Energía*)* mejorado.

Xen llama a cada máquina virtual que es ejecutada en el sistema *dominio.* Cuando se inicia Xen, en primer lugar arranca el *hipervisor*, responsable de iniciar a su vez un dominio llamado *Domain0* (*dom0*), en el cual corre el sistema operativo anfitrión. *dom0* es un dominio privilegiado que tiene acceso a los recursos hardware del servidor y que también contiene herramientas administrativas para la gestión del resto de dominios (creación, monitorización, configuración,…). A los dominios no privilegiados se les llama *domUs*.

Cada dominio interactúa con el *hipervisor* (encargado de comprobar la tabla de páginas y de asignar los recursos a los dominios) a través de una llamada denominada *hypercall* (*hiperllamada*), tal y como las aplicaciones se comunicación con un *kernel*¸ por ejemplo para solicitar operaciones privilegiadas como la actualización de la tabla de páginas; el *hipervisor* responderá mediante canales de eventos. Todas las peticiones de acceso al hardware de los *domUs* son realizadas al *dom0* que es quien tiene acceso directo; esto es posible gracias a las modificaciones realizadas en los sistemas operativos de los *domUs*.

Para finalizar presentamos de forma gráfica la arquitectura de *paravirtualización* de Xen, haciendo uso de la terminología propia. Se puede apreciar como las máquinas virtuales se denominan *dominios*: *dom0* para el dominio privilegiado de carácter administrativo y *domUs* para el resto de los dominios. El *hipervisor* o *VMM* (*Virtual Machine Monitor*) corre

directamente sobre el hardware del servidor físico, que proporciona los recursos para el entorno virtual y que se llama *host* (anfitrión); así, las instancias de máquinas virtuales se llaman también *guests* (invitados).

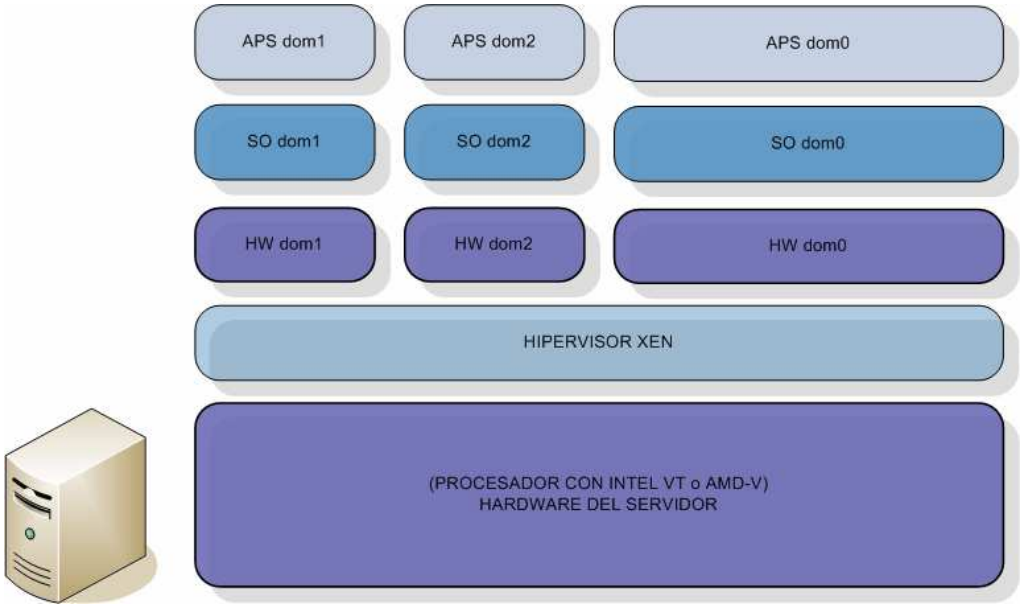


Figura 2.20 Arquitectura de paravirtualización con Xen.

1.6 VIRTUALIZACION A NIVEL DEL SISTEMA OPERATIVO

Antes de entrar a explicar con detenimiento cómo funciona este tipo de virtualización hay que puntualizar una diferencia importante y es la forma en la que son denominados los huéspedes creados en estos casos. Para ello se utilizará el término ***servidor virtual*** **más que** ***máquina virtual***. La diferencia entre ambos conceptos en la terminología empleada en virtualización a nivel del sistema operativo es sencilla: **un *servidor virtual* hace referencia a un tipo de huésped más ligero, que no precisa de un *kernel propio* sino que lo comparte con el sistema anfitrión que lo aloja. También son conocidos en muchos ámbitos como *containers* (contenedores).**

En este tipo de virtualización **los *servidores virtuales* son creados sobre una única instancia de un sistema operativo** –compartiendo un mismo núcleo- manteniéndolos como sea oportuno **aislados, independientes**.

Uno de sus rasgos principales es, como podemos ver, la **ausencia de una capa intermedia de virtualización**. **Esto proporciona, prácticamente a partes iguales, tanto ventajas como desventajas**. Por un lado puede presentar un rendimiento en la ejecución de los *servidores virtuales* muy próximo al que experimentarían de forma nativa reduciendo por ejemplo el consumo de memoria por *servidor virtual* u optimizando su entrada y salida debido a que un solo *kernel* controla el acceso a los dispositivos y recursos, como hemos dicho anteriormente, además de presentar soluciones de virtualización mucho más ligeras con las que podremos tener un número de *servidores virtuales* superior al número de *máquinas virtuales completas* que tendríamos por ejemplo al implementar virtualización de plataforma (en primer lugar, por no necesitar un *kernel propio*). En cambio, la ausencia de esta capa de virtualización intermedia implica que haya que realizar diversas modificaciones en el núcleo o sistema operativo anfitrión, sobre todo para hacer *creer* a los *servidores virtuales* que son ejecutadas en

un entorno exclusivo, por lo que un requisito indispensable que el núcleo o *kernel* del sistema anfitrión se pueda modificar (como sabemos, esto no es posible en sistemas operativos privativos). Además, al compartir el mismo núcleo con el servidor físico anfitrión tenemos la principal desventaja de que un fallo en el *kernel* puede provocar la caída de la totalidad de los *servidores virtuales* alojados (único punto de fallo). **Por lo general los *servidores virtuales***:

- **No pueden correr sistemas operativos que difieran del instalado en el sistema anfitrión, aunque en algunos casos es posible que ejecuten diferentes versiones de la misma distribución o incluso distintas distribuciones Linux**. Por ejemplo, FreeBSD jails permite ejecutar diferentes versiones de FreeBSD sobre un único *kernel* FreeBSD, por lo tanto permitiendo tener diferentes versiones de aplicaciones, procesos, librerías, etc. Otro ejemplo: Linux V-Server, FreeVPS y OpenVZ pueden ejecutar diferentes distribuciones Linux en los *servidores virtuales*, aunque siempre compartiendo el mismo *kernel*.

- **Las librerías, herramientas, utilidades,… que ejecuten los *servidores virtuales* deben estar compilados para el mismo juego de instrucciones y hardware que utiliza el sistema anfitrión**.

El modo de operar en el que se basa este modelo para virtualizar es el implementado por el concepto *chroot (change root)*. Durante el proceso de arranque del sistema, el *kernel* puede hacer uso de *sistemas de ficheros root* como los que proporcionan *discos RAM de arranque* o *sistemas de ficheros RAM de arranque* para la carga de drivers y otras tareas de inicialización del sistema. El *kernel* entonces tiene la posibilidad de cambiar a otros *sistemas de ficheros root* utilizando el comando *chroot* para montarlos como *sistemas de ficheros root* finales, y continuar así el proceso de inicialización y configuración desde estos sistemas de ficheros.

Partiendo de esta base, la virtualización a nivel del sistema operativo extiende el concepto *chroot* permitiendo al sistema iniciar *servidores virtuales*, que trabajan de manera aislada e independiente dentro de los límites de su propio *sistema de ficheros root*, lo que **aporta bastante seguridad en lo que a accesos a sistemas de ficheros *ajenos* se refiere,** por ejemplo, en el caso de que uno de los *servidores virtuales* se encontrara infectado por un virus. Si ocurriera que un *servidor virtual* se viera comprometido, esto afectaría únicamente a su propio *sistema de ficheros root.* Este *nivel* de seguridad *extra* que aportan los *servidores virtuales* en soluciones como **FreeBSD´s chroot jails, Linux V-Server, o FreeVPS** ha sido experimentado durante años por ISPs para proporcionar a usuarios acceso a sus propios ***servidores privados virtuales***, sobre los cuales pueden tener relativamente control total, sin ceder ni una oportunidad para vulnerar la seguridad.

Así, **uno de los mayores usos en la actualidad de la virtualización a nivel del sistema operativo es la consolidación de servidores**. Aunque muchos de ellos ya han sido nombrados, como ejemplos representativos de este modelo podemos citar **OpenVZ**, **Linux V-Server**, **Virtuozzo, FreeBSD´s chroot jails, Free VPS, y Solaris Zones, o Solaris Containers**.

1.6.1 Arquitectura

En la figura 2.21 podemos apreciar de manera gráfica la **arquitectura general de virtualización a nivel de sistema operativo** descrita en el apartado anterior. **Los elementos distintivos de este tipo de virtualización son**:

- **La ausencia de una capa intermedia de virtualización** dedicada especialmente para este propósito.

- **La creación de *servidores virtuales* completamente aislados en área de usuario, que comparten el mismo *kernel* con el sistema anfitrión** –son construidos sobre un mismo sistema operativo.

- **Existencia de ciertas modificaciones en el sistema operativo de la máquina** para la manipulación y gestión de la infraestructura virtual de servidores.

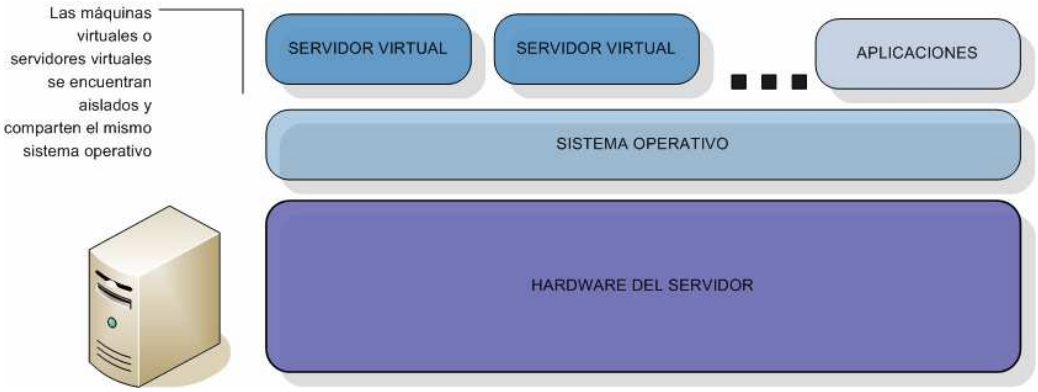


Figura 2.21 Arquitectura general de virtualización a nivel del sistema operativo.

1.6.2 OpenVZ

Es considerada **una de las mejores opciones a la hora de consolidar servidores Linux sobre anfitriones Linux**, y su popularidad no ha parado de crecer desde hace un par de años. A grandes rasgos su modo de operar es el mismo que el del resto de soluciones dentro de este grupo: permite la creación y mantenimiento de *servidores virtuales* aislados al introducir importantes modificaciones a nivel del *kernel* Linux anfitrión, creando una completa infraestructura virtual que *parece* real para los sistemas hospedados.

Dentro de la terminología OpenVZ **los *servidores virtuales* son también a menudo llamados *servidores privados virtuales* o *vps*, *entornos virtuales* (*virtual environments* en inglés) o incluso *contenedores***; en la mayoría de las ocasiones podremos leer que se trata de una solución *basada en contenedores*.

OpenVZ **incluye un completo conjunto de herramientas para la administración y monitorización de los *servidores virtuales*** que logran hacer que la mayoría de las tareas puedan ser completadas con relativa facilidad. Las más importantes son:

- **vzctl.** Se trata de un interfaz de alto nivel para manejar a nivel de comandos los *vps*. Es sencillo por ejemplo crear e iniciar un *servidor virtual* con el uso de tan sólo dos comandos: vzcreate y vzctl start. Otro comando, vzctl set, es usado para cambiar el valor de diferentes parámetros de los *vps*, algunos de ellos incluso con el *vps* ejecutándose, algo que no es posible con otras soluciones que implementan emulación o paravirtualización.

- **Templates y vzpkg.** Las *templates* facilitan el despliegue rápido (incluso en apenas segundos) de infraestructuras virtuales, ya que se trata de imágenes preestablecidas de *vps*. Los templates más concretamente son paquetes, y los ***template cache*** son archivos (*tarballs*) de un entorno ***chrooted*** con estos paquetes instalados. Vzpkg

tools, por su parte, es un conjunto de herramientas que facilita la creación de *template cache*, soportando por ejemplo repositorios basados en *rpm* y *yum*.

OpenVZ (http://wiki.openvz.org/Main_Page) es software libre, y es distribuido bajo la licencia GNU GPL. El alcance de este proyecto no queda ahí, sino que **es la base de otra importante solución de virtualización a nivel del sistema operativo: Parallels Virtuozzo Containers**, comercializada por Parallels, que por lo tanto también se dedica a apoyar OpenVZ. Experimenta las mismas ventajas y desventajas que he comentado en el apartado anterior; en términos más generales, se puede decir que OpenVZ es una solución que aprovecha muy bien y explota las ventajas que ofrece aplicar virtualización: mejor utilización de los servidores, evitando conflicto entre servicios, etc.

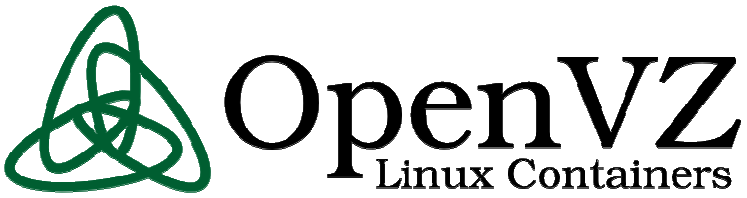


Figura 2.22 Logotipo de OpenVZ: virtualización a nivel del sistema operativo.

Las modificaciones introducidas en el *kernel OpenVZ* introducen las siguientes funcionalidades:

- Virtualización y aislamiento. Permite que existan múltiples *servidores privados virtuales* que comparten un mismo *kernel* de manera segura. Cada *vps* dispone de su propio conjunto de *recursos* (CPU, memoria, espacio en disco, dispositivos de red,…) y *objetos* (archivos, librerías, aplicaciones, usuarios, grupos, procesos, configuración de red, dispositivos, memoria compartida…).

- Administración de recursos. De especial importancia debido a la existencia finita de recursos, se ocupa de limitar (y en muchos casos garantizar) el acceso a una cantidad determinada de recursos como procesador, memoria, espacio de disco,.. por *servidor privado virtual*.

- Checkpointing. Nos ayuda a llevar a cabo procesos de *congelación* del estado de los *servidores privados virtuales*, pudiendo salvarlo en disco, para después si lo deseamos restaurarlo. Esto, como podemos imaginar nos permitirá llevar a cabo *migración en caliente* de los *vps* entre diferentes servidores físicos. Esta migración es apreciada por parte del cliente como un *retraso en la comunicación* más que como tiempo de caída, al tener que migrar también las comunicaciones de red.

Para finalizar con OpenVZ podemos citar cuáles son los escenarios más habituales en los que ha sido y sigue siendo utilizado. Sin duda, el más común e importante de todos es la consolidación de servidores.

- Consolidación de servidores.

- Escenarios que requieren gran seguridad. El aislamiento de los *vps* garantiza que si alguna aplicación de uno de ellos sufre un agujero de seguridad, no comprometerá al resto.

- Hosting. Sin duda la virtualización a nivel de sistema operativo es la más utilizada por ISPs para ofrecer a sus clientes completos sistemas virtualizados.

- Desarrollo de software y prueba.
- Educación.

1.6.3 Linux V-Server

Linux V-Server (http://linux-vserver.org) es otra de las soluciones de virtualización a nivel de sistema operativo de gran importancia. De forma análoga a lo que hacen las otras, en este caso el *kernel* Linux es utilizado para la **creación de múltiples *Virtual Private Servers* *(VPS, servidores privados virtuales)* independientes en área de usuario**. Este aislamiento del espacio de usuario es conseguido gracias a **diferentes modificaciones del núcleo y las estructuras internas de datos, sumado a la implementación de conceptos como routing segmentado, cuotas extendidas, *chroot y contexto***. *Chroot y contexto* permiten encerrar y **aislar complemente los procesos, herramientas, aplicaciones, sistema de ficheros… de un *servidor privado virtual* en una especie de *contenedor*** (*container*, como es conocido en la mayoría de la bibliografía), haciendo que éstos sean ejecutados solamente en el entorno del *vps*. Para el arranque inicial el núcleo define un contexto denominado *por defecto*, existiendo también un tipo especial de contexto que se llama *espectador* para la administración y monitorización de la totalidad de los procesos en ejecución.

Figura 2.23 Logotipo de Linux-V-Server.

Linux V-Server no sólo permite el aislamiento de los *servidores privados virtuales*, sino que además es posible especificar y asignar **límites de uso y manipulación de los recursos** disponibles como espacio en disco, consumo de memoria y procesador, adaptadores de red, etc. Si es deseable o entra dentro de nuestras necesidades, **los distintos *servidores virtuales* pueden comunicarse con facilidad haciendo uso de mecanismos de carpetas compartidas**. Las ventajas y desventajas al usar Linux V-Server son las mismas que he comentado anterior de manera general para todas las soluciones de este tipo.

Linux V-Server es la más antigua de las soluciones de virtualización a nivel del sistema operativo para Linux, estando soportado desde el núcleo 2.4 y en muy distintas arquitecturas: **x86, x86_64, SPARC, PowerPC, MIPS, o ARM**. Por último, comentar que **no se debe confundir Linux V-Server con Linux Virtual Server**; el segundo proyecto nada tiene que ver con la virtualización, y sí con el establecimiento de *clústeres* para balanceo de carga en servidores Linux proporcionando una única interfaz.

1.7 VIRTUALIZACION A NIVEL DEL KERNEL

En este modelo de virtualización **no se requiere la presencia de un *hipervisor*, sino que es el *kernel* de Linux el que ejecuta las máquinas virtuales de igual forma a como lo hace con otros procesos en el espacio de usuario**. Tanto las librerías como las aplicaciones y sistemas operativos que corren en las máquinas virtuales deben estar compilados para el mismo hardware y conjunto de instrucciones que el *kernel* del sistema anfitrión que las ejecuta,

compilado para la máquina física sobre la que corre, siempre y cuando no hagamos uso de virtualización asistida por hardware.

A continuación vamos a ver en detalle dos soluciones que a lo largo de los años han destacado por su eficiencia dentro de esta categoría: **KVM** y **User-mode Linux**. Viendo cómo operan, entenderemos en mayor medida en qué consiste la *virtualización a nivel del kernel.*

1.7.1 Arquitectura

En la figura 2.24 podemos ver de manera general la **arquitectura presentada por las soluciones de virtualización a nivel del *kernel* Linux**. Es importante observar que en este caso es **el *kernel* Linux del sistema anfitrión el que actúa como hipervisor** en la mayoría de los casos, en otros como veremos como ocurre en el caso especial de KVM lo hace incluyendo un módulo especial para ese propósito de manejar las máquinas virtuales.

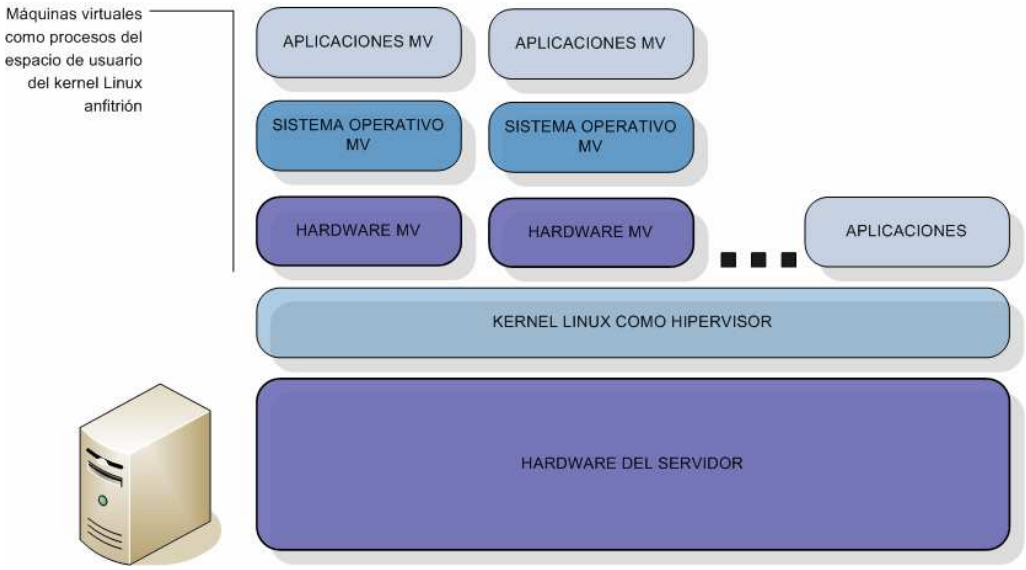


Figura 2.24 Arquitectura general de virtualización a nivel del kernel Linux.

Las máquinas virtuales son ejecutadas en el espacio de usuario del sistema anfitrión, como procesos totalmente independientes. **Que puedan albergar o no sistemas operativos no modificados depende de si se dispone de virtualización asistida por hardware** en los procesadores con Intel VT o AMD-V (en algunos casos, como KVM, el uso de estos procesadores será obligado); en caso de que no sea así, los huéspedes al completo deberán encontrarse compilados para la misma arquitectura que el *kernel* del sistema anfitrión.

1.7.2 User-mode Linux

User-mode Linux (http://user-mode-linux.sourceforge.net/) ha sido **soportado en el *mainline* del *kernel* Linux desde hace bastante tiempo, necesitando ser activado y recompilado antes de poder usarse**. User-mode Linux, o *UML* como también es conocido, no requiere de software administrativo de manera separada para ejecutar o administrar las máquinas virtuales, sino que pueden ser ejecutadas directamente desde la línea de comandos.

Figura 2.25 Logotipo de User-mode Linux.

UML permite al sistema operativo Linux ejecutar otros sistemas operativos Linux en el espacio de usuario, como si se tratara de procesos en el sistema operativo anfitrión, por lo que deben estar compilados para tal efecto. **De esta forma, varios núcleos Linux pueden ejecutarse en el contexto de un único núcleo, pudiendo incluso alojar a otros núcleos de forma anidada**. Los cambios introducidos en la recompilación para usar UML permiten la virtualización de los dispositivos, es decir, que las máquinas virtuales puedan compartir su acceso a dispositivos de bloque (sistemas de ficheros, CD-ROMs, disqueteras…), tarjetas de red, consolas, hardware de audio y vídeo, etc. **Típicamente los sistemas de ficheros de las máquinas virtuales en UML son contenidos en una imagen,** un único fichero, aunque es posible disponer de sistemas de ficheros compartidos haciendo uso de tecnologías que lo permitan, como *NFS* (*Network File System*). **El acceso por parte de las máquinas virtuales a los recursos hardware disponibles es totalmente configurable**, pudiendo asignar solamente los dispositivos que consideremos estrictamente necesarios.

El uso principal de UML a lo largo de los años ha sido para proporcionar entornos de desarrollo al nivel del *kernel* y prueba, sobre todo por muchos proyectos *open source*. Es una forma muy segura de depurar el desarrollo del *kernel*, distribuciones y procesos Linux, de ejecutar y probar código que probablemente contenga *bugs* sin llegar a temer por nuestra instalación principal. Esto es debido fundamentalmente a que UML permite hacer creer a las máquinas virtuales que disponen de un hardware que puede llegar a ser completamente diferente al disponible físicamente en la máquina anfitriona. **Otros populares usos de UML son los siguientes:**

- **Alojamiento de servidores virtuales.**
- **En entornos educativos-académicos.**
- **Sandbox:** como entorno de supervisión de determinados proyectos y procesos.

1.7.3 Kernel-based Virtual Machine

Kernel-based Virtual Machine o ***KVM*** (http://www.linux-kvm.org/page/Main_Page), que fue introducido a partir del *mainline* 2.6.20 del *kernel* de Linux, **hace uso de un módulo en el *kernel* del sistema anfitrión (*kvm.ko*) que proporciona la infraestructura de virtualización**. **Requiere soporte de virtualización por hardware en el procesador** (Intel VT o AMD-V) porque hace uso también de un módulo específico del procesador (*kvm-intel.ko* o *kvm-amd.ko*) y aunque está previsto que en el futuro no sea así, **aún requiere de un proceso *Qemu* ligeramente modificado como contenedor para la visualización y ejecución de las máquinas virtuales**. **El paradigma presentado por KVM es de especial importante**; la mayoría de las empresas en la actualidad, incluso también fue reconocido por VMware, consideran que las técnicas de virtualización *viajan* en este sentido y seguirán el camino que

anda KVM. Sin duda el paso más importante en lo que a ello respecta lo ha dado Red Hat cambiando la estrategia de virtualización presente en sus distribuciones de Xen a KVM. Por su inclinación a ***exprimir al máximo*** **las características del conjunto de instrucciones de virtualización de los nuevos procesadores de Intel y AMD** es por lo que se espera que obtenga un rendimiento excelente en la ejecución de las máquinas virtuales, cercano a su rendimiento nativo.

Figura 2.26 Logotipo de Kernel-based Virtual Machine.

Creado y mantenido por *Quramnet*, el proyecto KVM es completamente *open source*. Al hacer uso de virtualización asistida por las tecnologías de virtualización de los procesadores Intel y AMD es posible crear y configurar máquinas virtuales con sistemas operativos no modificados. De hecho, es impresionante observar la **larga lista de sistemas operativos invitados que son soportados** actualmente por KVM en su web oficial (http://www.linux-kvm.org/page/Guest_Support_Status), como se puede ver en la figura 2.27.

Figura 2.27 Estado del soporte de sistemas operativos invitados en la web de KVM.

En la figura 2.28 queda recogida la **arquitectura presentada por KVM**, basándonos en la vista para las soluciones de virtualización a nivel del *kernel* de manera general. **Los sistemas operativos de las máquinas virtuales invitadas son ejecutados en el espacio de usuario del núcleo Linux anfitrión y se comunican a través de un proceso Qemu** -ligeramente modificado- **con el módulo KVM** que ha sido integrado en el *kernel* para así obtener acceso al hardware virtualizado.

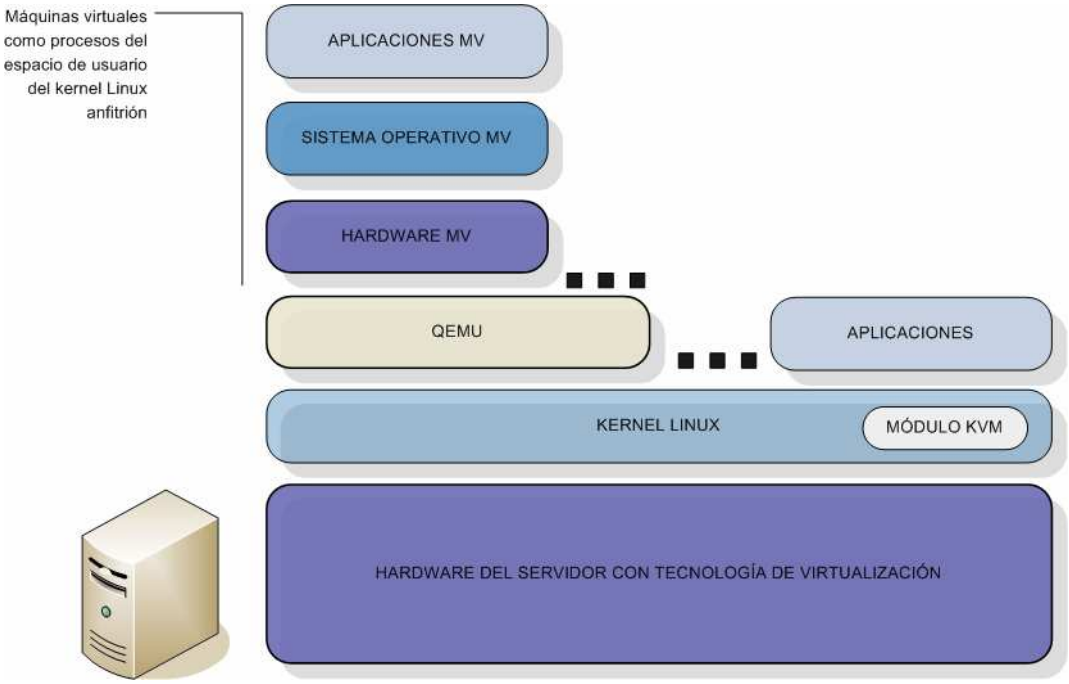


Figura 2.28 Arquitectura de virtualización presentada por KVM.

El módulo KVM introduce un nuevo modo de ejecución en el núcleo Linux adicional a los modos *kernel* y *user* que de por sí aporta el *kernel* de forma estandarizada, llamado *guest*, utilizado para la ejecución de todo el código generado por las máquinas virtuales el cual no haga uso de instrucciones de entrada y salida, usando el modo normal *user* para ello.

Otra característica muy importante y destacable de KVM es que **proporciona funcionalidades que permiten implementar de manera sencilla *migración de máquinas virtuales*, tanto de manera *offline* como *en caliente*, y la toma y recuperación de imágenes o *snapshots* de las máquinas virtuales**. Las máquinas pueden ser incluso migradas entre diferentes procesadores (de AMD a Intel o viceversa), aunque no lógicamente de 64 a 32 bits.

La importancia de KVM va más allá de su uso como virtualizador: **ha sido la primera tecnología de virtualización que ha pasado a formar parte del núcleo Linux**. Las razones por las que eso ocurrió fue su limpia implementación, que no requiere la inclusión de un *hipervisor* y por lo tanto es una solución puramente Linux. Uno de los principales beneficios de KVM, por ejemplo frente a Xen, es que **no requiere modificación del *kernel* de Linux**, siendo eso mismo **lo que produce que no pueda obtener los rendimientos experimentados por las técnicas de paravirtualización**.

2 Sistemas de Almacenamiento

En este segundo apartado del capítulo abordaremos el requisito fundamental de disponer de un **espacio compartido para el almacenamiento de las máquinas virtuales** de nuestra infraestructura. El almacenamiento, junto a los diferentes tipos de sistemas de ficheros disponibles, nos **aportará cualidades muy importantes como la flexibilidad, escalabilidad, alta disponibilidad o alto rendimiento**.

Lo habitual cuando tratamos de crear una infraestructura virtual de consolidación de servidores es que las máquinas virtuales se almacenen en una única localización o en varias localizaciones pero haciendo posible que todo este **almacenamiento separado sea visto como**

un único recurso a compartir por los servidores anfitriones. En cualquiera de los casos es imprescindible que garanticemos la disponibilidad de este almacenamiento, ya que contiene las imágenes y configuración de las máquinas virtuales; de lo contrario éstas no serán accesibles.

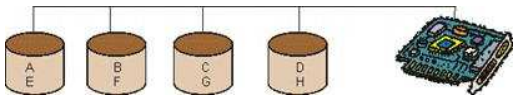
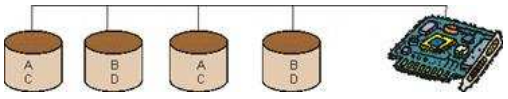
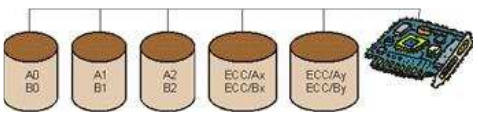
Nuestro objetivo en las siguientes páginas es abordar desde un punto de vista teórico cuáles son las soluciones más utilizadas para proporcionar este requisito y las tecnologías más utilizadas en este tipo de escenarios, además de los protocolos y configuración básica de almacenamiento en las que se basan.

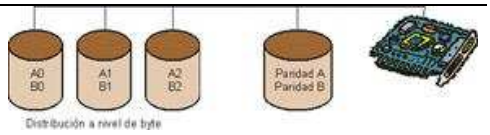
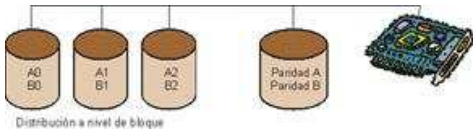
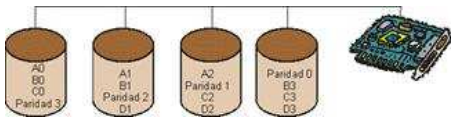
2.1 RAID

Los *sistemas RAID* (*Redundant Array of Independent Disks*) son **grupos de discos independientes** –si ocurriera un fallo en alguno de ellos, el resto seguiría su funcionamiento correctamente- **que operan de manera conjunta por lo que pueden ser vistos como un único sistema de almacenamiento**.

Existen **seis niveles RAID (desde el 0 hasta el 5) que difieren en la configuración del conjunto de discos para la expansión de los datos que contienen**. Dependiendo de las necesidades y el escenario en el que debamos usar RAID elegiremos una u otra configuración, teniendo en cuenta y compensando gasto y velocidad. En la tabla 2.3 quedan recogidos los seis tipos de configuración RAID definidos por un grupo investigación de la Universidad de California en Berkeley, oficialmente en 1988 por David A. Patterson, Garth A. Gibson y Randy H. Katz.

Tabla 2-3. Configuraciones RAID

Tipo	Descripción	Esquema
RAID 0	Disco con bandas sin tolerancia al error. No hay redundancia (la información es distribuida entre todos los dispositivos) lo que mejora la capacidad de procesamiento. ***Creación de bandas en disco.***	
RAID 1	Reflejo y duplexing. Existe replicación de todos los datos de una unidad en otra, lo que provoca que haya tolerancia a fallos. Mejor rendimiento con operaciones secuenciales. Coste elevado al dedicar una unidad completa a replicación. ***Disco espejo***.	
RAID 0+1	Reflejo de discos con bandas. Obtiene redundancia y rendimiento al combinar RAID 0 y RAID 1. Mejor opción para lectura y escritura.	
RAID 2	***Código Hamming EEC***. Utiliza códigos Hamming para detección y corrección de errores, ya proporcionados por defecto por unidades SCSI.	

RAID 3	***Transferencia en paralelo con paridad***. Almacena la paridad de los datos distribuidos (a nivel *byte*) en una unidad adicional. La distribución en bandas a nivel de *byte* requiere hardware adicional.	
RAID 4	***Discos de información independientes con paridad compartida***. Los datos son distribuidos a nivel de *bloque* en diferentes unidades, almacenando en otra información de paridad para recuperación. Escritura peor rendimiento al actualizar paridad.	
RAID 5	***Discos de información independientes con paridad distribuida***. Parecido al nivel anterior resolviendo el cuello de botella de la paridad. Aún así ahora es la lectura un poco más lenta.	

Las configuraciones RAID más utilizadas son RAID 1, RAID 0+1 y RAID 5 (dependiendo de la gama de nuestro servidor y uso de los datos a almacenar) ya que para los escenarios más habituales proporcionan buenos resultados en cuanto a rendimiento y velocidad. Siempre es recomendable mantener sistema operativo, datos, registros y ficheros *log* de un servidor en sistemas de almacenamiento diferentes para mejorar el rendimiento al mismo tiempo que aumentamos la seguridad y resistencia ante los ataques que podamos sufrir.

El grupo de discos del sistema RAID es manipulado y operado por una *controladora RAID*, proporcionando acceso a nivel físico. Todos los dispositivos unidos al RAID estarán disponibles como si de un solo disco se tratara, pudiendo ser particionado, crear volúmenes lógicos sobre él, eliminarlos, crear distintos tipos de sistemas de ficheros, montarlos… **Algo muy interesante y que soportan sistemas operativos como GNU/Linux es el RAID software**, donde ya no es necesaria una controladora RAID hardware ya que un elemento software proporciona todas las funcionalidades que ésta ofrecía. Este tipo de controladoras software son muy eficientes, si bien disponen de un rendimiento un poco menor que las controladoras RAID hardware tradicionales. Sin embargo nos permiten obtener ventaja de otros aspectos como la gran portabilidad de la que disponen entre todos los sistemas GNU/Linux en los que se encuentre el software instalado y se soporten las mismas interfaces de disco físico (IDE, ATA, SATA, SAS…), no como las controladoras RAID hardware que suelen diferir bastante cuando se cambia de fabricante.

2.2 LVM2

LVM2 (*Logical Volume Manager o administrador de volúmenes lógicos*, http://sourceware.org/lvm2/) **es un sistema para la administración completa de discos con un gran potencial y flexibilidad**, ampliamente utilizado en servidores con sistema operativo GNU/Linux, sustituyendo a otras herramientas como *fdisk*. Entre las **operaciones más importantes que es posible realizar con LVM2** están, independientemente del sistema de ficheros:

- **Redimensionado de *grupos lógicos***.

- **Redimensionado de *volúmenes lógicos***.
- **Toma de instantáneas de lectura y escritura**.
- **RAID 0 de *volúmenes lógicos***.

LVM2 no implementa RAID 1 o RAID 5, para lo que tendremos que hacer uso de software propio RAID para estas configuraciones. Los ***volúmenes lógicos*** pueden ser discos duros completos, una parte de un disco duro y varios discos duros o bien partes juntas de varios discos duros. Los *volúmenes lógicos*, que serán formateados y sobre los cuales se instalarán los sistemas de ficheros, agrupan ***volúmenes físicos*** creados a partir de particiones físicas de disco o discos completos, y éstos a su vez, se engloban en ***grupos lógicos***. Cada *volumen físico* puede ser asignado solamente a un *volumen lógico.* En la figura 2.29 podemos apreciar mejor cómo son ubicados los diferentes niveles LVM2.

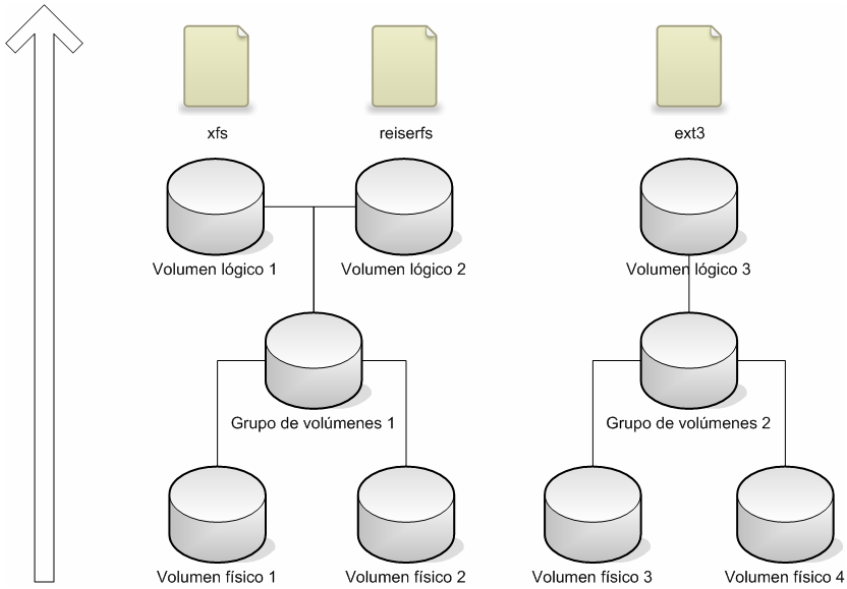


Figura 2.29 Representación de los niveles que forman LVM2.

La escalabilidad que presentan este tipo de sistemas es alta; por ejemplo, es fácil aumentar el almacenamiento disponible añadiendo un nuevo disco duro para posteriormente redimensionar un volumen lógico sumándole el nuevo espacio. **LVM2 se sitúa entre el acceso a disco y el sistema de ficheros** (normalmente sistemas de ficheros cuyo tamaño pueda cambiar), por lo que éstos no ven los discos directamente sino los *volúmenes lógicos* creados por LVM2.

2.3 SISTEMAS DE FICHEROS DISTRIBUIDOS

Otra tecnología importante son los ***sistemas de ficheros distribuidos o sistemas de ficheros en red***, como ***NFS*** (*Network File System,* **protocolo ubicado en la capa de aplicación del modelo TCP/IP para la administración de sistemas de ficheros distribuidos por defecto en la mayoría de los sistemas de tipo Unix y GNU/Linux**). **Disponen de su propia funcionalidad para la creación y gestión de volúmenes lógicos posibilitando la compartición de sistemas de ficheros instalados en ellos, su almacenamiento o impresoras**

a través de la red entre varios sistemas heterogéneos ya que proporcionan mecanismos de bloqueo para sincronizar escrituras simultáneas.

AFS** (Andrew File System), **GFS** (Global File System)* y ***SMB** (Server Message Block,* también conocido como ***CIFS) **son otros sistemas de este tipo bien conocidos** y que también permiten este tipo de operaciones buscando transparencia y escalabilidad en la compartición. Los *sistemas de ficheros distribuidos* como *AFS* y *GFS* proporcionan acceso a los volúmenes lógicos a nivel de *sistema de ficheros*, lo que los hace bastante similares a los dispositivos *NAS* (*Network-Attached Storage*), que analizaremos en el apartado *2.5 NAS*.

NFS fue el pionero de todos ellos en el año 1984 cuando comenzó a ser comercializado por Sun Microsystems con el objetivo y compromiso de mantener en su diseño independencia del sistema operativo, protocolo de capa de transporte y máquina física. Actualmente en su versión 4 (http://www.nfsv4.org/) incluye seguridad *Kerberos*, integra el uso de cortafuegos, permite crear listas de control de acceso y las operaciones pueden incluir una descripción del estado. **Todas las operaciones implementadas en NFS son síncronas para garantizar la integridad de los datos**; esto quiere decir que hasta que la operación no es finalizada completamente el servidor no la devolverá al cliente. **Estos sistemas trabajan bajo el paradigma *cliente-servidor*,** en el que en concreto varios clientes remotos acceden a la información compartida en red por un servidor, consiguiendo que ésta aparezca como local.

Entre las ventajas del uso de sistemas de ficheros distribuidos destaca el **disponer de toda la información localizada de forma centralizada** en un único servidor por lo que no es necesaria la replicación de esta información cuando más de un usuario trabaja con ella. En cambio, como podemos imaginar, esta ventaja puede convertirse en un grave problema si el servidor fuese comprometido. Además de sistemas de ficheros es posible compartir también entre los equipos de la LAN diferentes dispositivos de almacenamiento: CD/DVD-ROM, memorias y discos duros externos, etc.

No podemos finalizar el presente apartado sin comentar brevemente **dos relevantes implementaciones de dos de estos protocolos citados: *OpenAFS***, **implementación *open source* de AFS y sobre todo *Samba*, en su caso del protocolo de archivos compartidos de Microsoft Windows SMB conocido posteriormente como CIFS**.

Las características ofrecidas por la aplicación **Samba** (http://www.samba.org/), distribuida bajo la licencia GNU GPL, son muy importantes debido a que **permite que equipos Unix, GNU/Linux o Mac OS actúen como servidores o clientes en redes Microsoft Windows**. Es posible llevar a cabo procesos de validación de usuarios actuando como un *controlador principal de dominio* o incluso como un dominio *Active Directory* en redes Windows, compartir directorios, impresoras y servir sus colas y autenticar usuarios.

OpenAFS (http://www.openafs.org/) por su parte una solución, escrita en lenguaje C, que **aboga por ser multiplataforma, escalable, segura, independiente de la localización y transparente en sus operaciones**. Nació cuando IBM liberó el código fuente de AFS y lo puso a disposición de la comunidad para su desarrollo y mantenimiento.

Figura 2.30 OpenAFS y Samba: implementaciones open source de protocolos de sistemas de ficheros distribuidos/en red.

2.4 DRBD

DRBD (Distributed Replicated Block Device, http://www.drbd.org/*)* es un **software de replicación de dispositivos de *bloque***, es decir, discos duros, volúmenes, particiones, etc., con el que es posible crear discos duros espejo (RAID 1) cuya ubicación es diferente en distintos servidores a través de la red. La replicación del contenido de los dispositivos puede llevarse a cabo **en tiempo real y de forma ininterrumpida**; ello nos permite por ejemplo que los distintos servidores que mantienen réplicas del original observen y tengan acceso en todo momento a las máquinas virtuales almacenadas y su estado actualizadas. Además esta replicación es totalmente **transparente a las aplicaciones** que almacenan los datos, no son conscientes de la copia que está teniendo lugar.

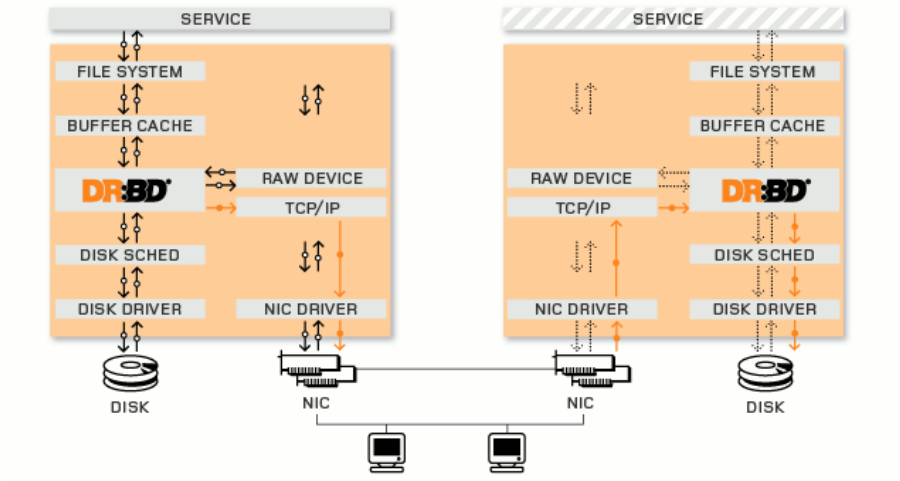


Figura 2.31 Esquema de funcionamiento de DRBD, comprendido como un sistema RAID 1 en red.

Como contraprestaciones del uso de DRBD tenemos por ejemplo la **no posibilidad de establecimiento de diversas localizaciones** como origen de la copia de manera simultánea: sólo puede haber un único modelo de copia. Además, **en el caso de la replicación de máquinas virtuales el rendimiento experimentado puede que no sea el esperado** sino disponemos de conexiones de red de alta velocidad, ya que las máquinas virtuales suelen tener que consumir un espacio aproximado en disco de 2Gb.

DRBD puede realizar la **réplica de manera síncrona o asíncrona**. En la primera la aplicación origen de la copia sería notificada de que ha sido realizada con éxito solamente cuando ésta haya concluido completamente. En la segunda, por el contrario, la aplicación origen

no es consciente del final de la copia hasta que la réplica de los datos ha sido propagada a todos los equipos receptores. En DRBD los recursos, a los que se dotará de redundancia, pueden tener un **papel primario o secundario en cuanto a la disponibilidad del almacenamiento**: uno primario puede ser usado tanto para lectura como para escritura mientras que uno secundario se limitará solamente a recibir actualizaciones del recurso primario. No hay que confundir estos términos con los de **papel activo o pasivo, relativos a la disponibilidad de la aplicación**.

El uso habitual de DRBD en virtualización de plataforma tiene que ver con la creación de un disco compartido para poder implementar *migración en caliente* de máquinas virtuales y obtener alta disponibilidad en los servicios virtualizados. Esto se consigue por ejemplo en Xen creando un **disco virtual compartido** por varias máquinas en el que son almacenadas, normalmente en volúmenes lógicos diferentes, las máquinas virtuales y sus ficheros de configuración.

A continuación veremos dos tecnologías que son confundidas en la mayoría de los casos, *NAS y SAN*, ya que el servicio que proporcionan básicamente es el mismo: almacenamiento compartido en una red y que es accesible por un conjunto de servidores con todo lo que ello conlleva (establecimiento de políticas de mantenimiento, recuperación y copias de seguridad…). A pesar de ello veremos a continuación cómo su forma de operar difiere lo que nos puede llevar a elegir una u otra en función de nuestro presupuesto y necesidades.

2.5 NAS

Los sistemas *NAS (Network-Attached Storage o almacenamiento adjunto a la red)* son **servidores con hardware y software especializado**. Permiten **acceder de forma remota y compartida a sus sistemas de ficheros** utilizando diferentes protocolos. **La conexión usada es TCP/IP** (Ethernet, FDDI, ATM) por lo que los protocolos normalmente usados para el acceso al medio compartido son NFS y CIFS (*Microsoft Common Internet File System*). En una NAS los datos son identificados por archivo y permite transferir metadatos, es decir, información relativa a los datos como es el propietario, fecha de creación, permisos sobre el archivo… además, la seguridad y políticas aplicables al recurso son gestionadas por el propio NAS, lo que permite que los recursos montados sean visibles desde diferentes sistemas operativos. El cliente solicita los ficheros al NAS, el cual los procesa localmente. NAS trabaja a **nivel de fichero** (generalmente de pequeño tamaño) y no de bloque, lo que permite ahorrar tiempo en las transferencias y ancho de banda de la red.

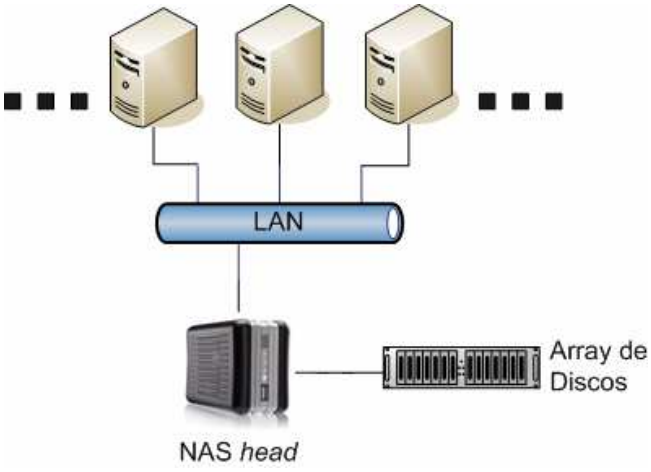


Figura 2.32 Topología NAS.

Un sistema NAS puede tratarse de un sistema específico dedicado a tal efecto (*NAS hardware*) **o también de** una forma más general puede ser considerado un sistema de este tipo **un servidor que comparte sus unidades de red ejecutando un *sistema operativo NAS***. Suelen tener de todos modos más de un dispositivo de almacenamiento, normalmente dispuesto siguiendo algún sistema RAID (*Redundant Arrays of Independent Disks*). El **NAS *head*** es la interfaz de la que disponen los clientes para acceder al almacenamiento compartido; dispone de dirección de red propia y es administrable vía web.

Como acabamos de decir en el párrafo anterior es posible **configurar nuestro sistema NAS sin tener que adquirir un dispositivo diseñado para este objetivo**. Para ello existe una gran variedad de **distribuciones libres basadas en GNU/Linux y FreeBSD** que podemos instalar y configurar en nuestro ordenador personal o servidor. Incluso en ocasiones no es necesario que dispongamos de equipos con altas prestaciones ya que existen distribuciones *LiveCD*, ejecutables desde una memoria USB o desde uno de los discos que forman parte del almacenamiento. **Normalmente lo que hacen estos *sistemas operativos NAS* es ejecutar el software *Samba* y los protocolos FTP y NFS**.

Algunos ejemplos importantes de este tipo de sistemas operativos son **FreeNAS** (http://freenas.org/doku.php?lang=es), **OpenNAS** (http://opennas.codigolivre.org.br/) o **NASLite** (http://www.serverelements.com/naslite.php).

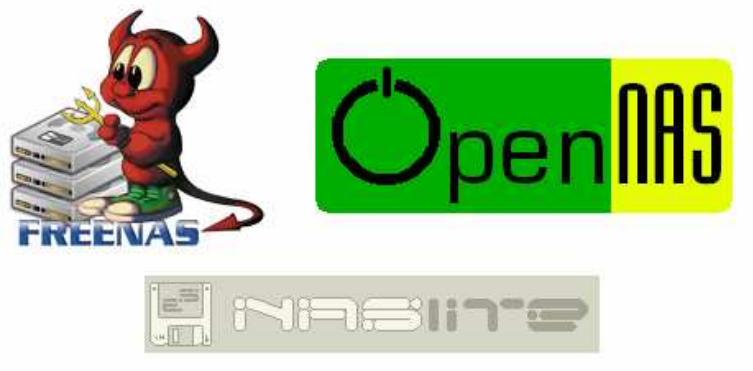


Figura 2.33 Distribuciones NAS más importantes.

2.6 SAN

Una *SAN (Storage Area Network o red de almacenamiento)* es un tipo de **red especializada para permitir acceso de forma rápida, segura y fiable por parte de servidores a recursos de almacenamiento compartidos**, es decir, nos ofrece la posibilidad de adhesión de dispositivos remotos de almacenamiento tales como arrays de discos, librerías de cintas, dispositivos ópticos… a servidores, de manera que el servidor crea que éstos se encuentran instalados localmente. Para ello **se sirven de canales de fibra** (protocolo *Fibre Channel*, **aunque recientemente también las hay basadas en la tecnología *iSCSI***, no tan rápida pero sí menos costosa) que dotan a las comunicaciones entre los servidores y la red de almacenamiento de una gran velocidad y rendimiento, por lo tanto, los servidores deben estar preparados para este tipo de conexiones. En los sistemas SAN **las copias y transferencias son a nivel de *bloque*** -más lentas- de ahí que sea necesario tal hardware en su topología. El acceso al almacenamiento es a bajo nivel lo que lo diferencia de otros modos de almacenamiento en red.

La seguridad de los datos es gestionada en los servidores que montan los recursos, por lo que el estado de estos dependerá del tipo de sistema operativo que lo monta y un mismo volumen no suele poder ser gestionado por dos sistemas operativos diferentes.

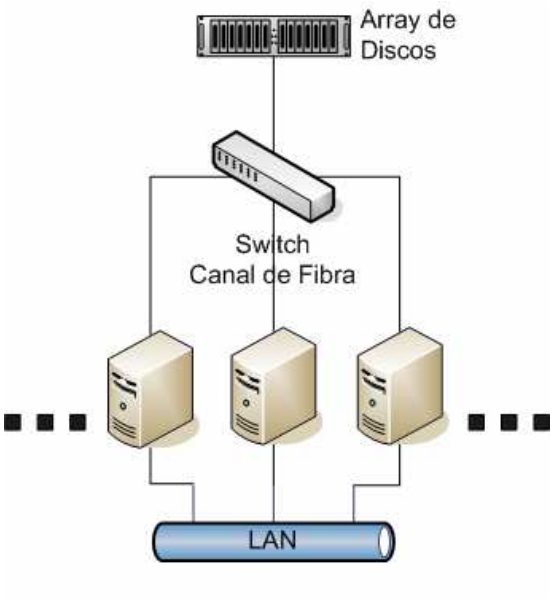


Figura 2.34 Topología SAN.

Como podemos ver el hardware necesario en este tipo de red de almacenamiento hace que su **precio sea más elevado que en el caso de utilizar un sistema NAS**, llegando a ser prohibitivo para pequeñas y medianas empresas: conexiones de fibra óptica para cada servidor, switch de canal de fibra… para obtener una mayor velocidad en nuestra red, mientras que en los sistemas NAS es suficiente con conexiones de 1Gb, algo más habitual en la actualidad. De cualquier modo es previsible que la velocidad de las conexiones de red que usamos hoy en día siga aumentando.

El **modelo de arquitectura centralizado** usado en las SANs permite llevar a cabo tareas administrativas de una manera sencilla al mismo tiempo que presenta una escalabilidad y fiabilidad excelentes. Sin embargo debemos prestar especial atención para asegurar su funcionamiento correcto, de lo contrario si se comprometiera el sistema las consecuencias serían desastrosas.

2.7 TECNOLOGÍAS RECIENTES

Novedosas tecnologías en el sector del *almacenamiento* han aparecido recientemente. Este es el caso de ***iSCSI*** (*Internet Small Computer Systems Interface*), **que permite el uso de *SCSI* sobre redes TCP/IP** –con el requisito del uso de un controlador de red especial- o ***AoE*** (*ATA over Ethernet*) **que soporta acceso a dispositivos *ATA* sobre conexiones Ethernet convencionales**. Ambos permiten el acceso también a dispositivos de almacenamiento disponibles en la red a **nivel de *bloque*** pero a un menor coste, siendo de toda forma recomendable su uso en redes con un ancho de banda grande.

Internet SCSI o iSCSI es un protocolo situado en la capa de transporte según el modelo de pila TCP/IP, como *canal de fibra* (*Fibre Channel*) o *SCSI Parallel Interface*. Su uso ha ido aumentando progresivamente debido a la implantación cada vez mayor de redes *Gigabit Ethernet*. **Al permitir crear redes de almacenamiento con un coste menor iSCSI se está**

consolidando cada vez más como una alternativa viable al uso de SANs (*canal de fibra*). Sólo requiere cualquier tipo de interfaz TCP/IP para la transmisión de datos (por ejemplo Ethernet) lo que lo dota de una **gran compatibilidad**; aunque esto también es visto por muchos como una contraprestación ya que deberá compartir el canal de transmisión con el tráfico TCP/IP restante por lo que se puede ver perjudicado el rendimiento final, aunque esto no es tan importante si usamos *Gigabit Ethernet* por ejemplo en nuestra red.

ATA over Ethernet o AoE es un protocolo de almacenamiento en red *ligero* cuyo diseño tiene el objetivo de proporcionar un **acceso sencillo y de alto rendimiento a dispositivos de almacenamiento ATA en redes Ethernet**. Al igual que iSCSI permite así la creación de SANs de **bajo coste**. No depende de las capas superiores a Ethernet, como IP o TCP, y por lo tanto no es accesible desde Internet u otras redes IP; esto provoca que sea un protocolo enormemente *ligero*, fácil de implementar y seguro.

3 Comparativa y selección de una solución

Después del recorrido teórico realizado en los dos apartados anteriores de este capítulo por las soluciones y tecnologías de mayor relevancia en ***virtualización de plataforma* y *sistemas de almacenamiento*** nos disponemos finalmente a sentar las bases de lo que será el desarrollo práctico del proyecto fin de carrera.

Antes de decidirnos por un esquema práctico a implementar u otro es necesario recordar cuáles son los **objetivos prácticos que se han fijado** para el desarrollo del proyecto y cuáles son los **requisitos fundamentales que deben estar presentes para alcanzarlos**.

El objetivo general es la **implantación de una infraestructura virtual** sobre la cual serán estudiadas las operaciones fundamentales sobre virtualización relativas a creación, gestión, administración y monitorización de máquinas virtuales. Sobre esta infraestructura estableceremos **consolidación de servidores de telefonía IP**, con el software libre *Asterisk* instalado, cuyo rendimiento será medido en diversos escenarios de producción (real sin virtualización, virtual, alta disponibilidad/alto rendimiento y *migración de máquinas virtuales*) lo que nos permitirá extraer conclusiones interesantes al respecto.

Por tanto podemos listar los **requisitos a cubrir para el desarrollo práctico del proyecto**, **y por ende a tener en cuenta en la selección de una solución** que integre de la forma correcta virtualización y almacenamiento:

- Hacer uso de ***software libre* que opere en sistemas operativos GNU/Linux**.

- **Uso del software para el establecimiento de centralitas de telefonía IP *Asterisk* (versión 1.4.28)**. *Asterisk* puede ser instalado y configurado sin problemas en las últimas versiones de las distribuciones GNU/Linux más utilizadas, como Red Hat, Fedora, CentOS o Debian.

- **La instalación básica de *Asterisk* no necesita parches o software adicional. Aún así es recomendable instalar el paquete *asterisk-addons***, que contiene módulos adicionales, para la misma versión de *Asterisk*. Otros paquetes son necesarios también en escenarios más complejos que los que se presentan en este proyecto, tales como *zaptel* (con módulos del *kernel* para dar soporte a tarjetas de comunicaciones analógico-digitales, y utilidades de configuración y diagnóstico) o *libpri* (librería encargada para dar señalización de primario a *zaptel*).

- ***Asterisk* necesita más recursos de procesamiento que de memoria para casos con alta carga de trabajo**. En cuanto a almacenamiento en nuestro caso no necesitaremos mucho, ya que *Asterisk* no hará uso aplicaciones que consuman grandes cantidades como bases de datos.

- Tanto las máquinas virtuales como *Asterisk* serán sometidos a pruebas de estrés, por lo que **es requerida gran capacidad de procesamiento y memoria, y no tanto de almacenamiento** (a penas se establecerán unas pocas máquinas virtuales y no habrá aplicaciones como bases de datos) **en los servidores anfitriones**. Las conexiones de red entre los servidores físicos son *Gigabit Ethernet*; aunque parezca lo contrario, la velocidad de las conexiones de red puede llegar a imponer una gran limitación si no es suficiente cuando se realizan transferencias del estado de máquinas virtuales o el almacenamiento usado por éstas es muy grande.

- El objetivo es ofrecer servicios de consolidación de servidores de telefonía IP. **Los escenarios que se presentan no precisan hardware adicional o *drivers* especiales en este proyecto, salvo los propios sistemas anfitriones**.

- **Tener la posibilidad de experimentar con *virtualización asistida por hardware*** (procesadores Intel VT y AMD-V), por lo que los servidores anfitriones deben disponer de este soporte.

- La solución elegida debe **encontrar un buen equilibrio entre coste y rendimiento**.

- **El *software de virtualización* debe ser lo suficientemente potente como para ser instalado y configurado en entornos empresariales, soportando y ofreciendo funcionalidades avanzadas** como gestión y monitorización de anfitriones remotos o *migración de máquinas virtuales*.

- **El almacenamiento compartido entre los servidores físicos debe ser lo más eficiente y sencillo posible, manteniendo un coste bajo.**

Para hacer frente a estos objetivos y requisitos vamos a seleccionar una solución eligiendo primero un *submodelo* de los que han sido presentados en este capítulo dentro de *virtualización de plataforma* y el software respectivo, y después un sistema *de almacenamiento* que apoye el desarrollo de las máquinas virtuales, sobre todo de cara a soportar alta disponibilidad, alto rendimiento y *migración de las máquinas virtuales*.

3.1 SELECCION DE UN MODELO Y SOLUCION DE VIRTUALIZACION DE PLATAFORMA

En la tabla *2.4 Comparativa de submodelos de virtualización de plataforma* quedan resumidas **las ventajas e inconvenientes de los distintos *submodelos* de *virtualización de plataforma*** entre los que vamos a elegir uno para el desarrollo de nuestra infraestructura virtual. **Algunas de las razones a las que se ha atendido para valorar un aspecto como positivo o negativo son: si se trata de *software libre*, los escenarios de uso más habituales, soporte a *virtualización por hardware*, el rendimiento medio experimentado, la presencia de funcionalidad avanzada y de características como seguridad, escalabilidad, flexibilidad o facilidad de uso**.

Tabla 2-4. Comparativa de submodelos de *virtualización de plataforma*

Submodelo	Aspectos positivos	Aspectos negativos
Sistemas operativos invitados	• Uso de una *aplicación de virtualización*. • Configuración y uso simple. • Idónea en entornos de escritorio. • Con Intel VT o AMD-V actuará como virtualizador puro. • Gran variedad de soluciones. • Con emulación puede ejecutar sistemas operativos privativos.	• Su potencial es limitado en escenarios con gran exigencia; carece de funciones extra. • No hace uso de *hipervisor* y en ocasiones depende de la emulación: pérdida de rendimiento.
Emulación	• Es posible la emulación de cualquier máquina; apoya a otros modelos. • Ideal para entornos de prueba y depuración. • Uso en ámbitos educativos y lúdicos. • Facilidad de instalación y configuración. • Es posible ejecutar instancias de sistemas operativos privativos.	• Suele ofrecer un rendimiento bastante bajo, dependiendo de de la aplicación emulada, arquitectura, etc. • No es recomendable su uso en producción. • Gran dificultad para llevar a cabo emulación de alto nivel. • Prácticamente no hay soporte para operaciones complejas.
Virtualización completa	• Uso de *hipervisor*. • Optimización del rendimiento con *virtualización hardware*. • Buen rendimiento en el aislamiento y compartición de equipos entre usuarios. • Gran diversidad de aplicaciones adicionales. • Escalabilidad y seguridad. • Consolidación de servidores efectiva.	• Incluye código en el *hipervisor* para emulación cuando sea necesario: rendimiento menor que con *paravirtualización*. • Su integración con infraestructuras *cloud* está aún en evolución. • Soporte hardware para virtualización obligatorio cuando queramos hacer uso de sistemas operativos no modificables. • Las soluciones más importantes *no* son *open source*.
Paravirtualización	• Uso de *hipervisor*. • No incluye código para emulación en el *hipervisor.* • Modificaciones en los sistemas operativos invitados: rendimiento sea cercano al nativo, eliminando la necesidad de captura de instrucciones privilegiadas en el *hipervisor*. • Con *virtualización hardware* el rendimiento es optimizado y se pueden ejecutar sistemas operativos privativos. • La solución más potente y extendida en uso a lo largo de los años es *open source*: Xen. • Xen es ligero y experimenta baja sobrecarga. • Fuerte aislamiento entre dominios. • Funcionalidad extendida	• Hay que realizar modificaciones en los sistemas operativos invitados, aunque con ello se obtiene un mejor rendimiento. • Los SOs invitados deben ser modificables (salvo que tengamos disponible Intel VT o AMD-V en el procesador). • Los proyectos que amplían la funcionalidad de Xen están aún en proceso. • En ocasiones el proceso de instalación y configuración puede presentarse bastante complejo.

	disponible. • Escalabilidad.	
Virtualización a nivel del sistema operativo	• *Servidores virtuales* más ligeros al no precisar de un *kernel propio*, por lo que es posible disponer de un mayor número de ellos. • Ausencia por completo de una *capa de virtualización* que permite obtener un rendimiento cercano al nativo: un solo *kernel* controla el acceso a dispositivos y recursos. • Gran seguridad basada en *chroot*. • Soluciones ampliamente utilizadas en consolidación de servidores, *hosting*, desarrollo de software y prueba.	• La ausencia de la *capa de virtualización* implica que haya que realizar modificaciones en el sistema operativo anfitrión para establecer el entorno virtual, por lo que debe ser modificable. • Un fallo en el *kernel* puede provocar la caída de todos los *servidores virtuales.* • Los *servidores virtuales* no pueden correr sistemas operativos diferentes en esencia al anfitrión, además sus librerías y utilidades deben estar compiladas para el mismo juego de instrucciones.
Virtualización a nivel del *kernel*	• El *kernel* de Linux actúa como *hipervisor*. • Soluciones *open source*. • Las máquinas virtuales son ejecutadas en área de usuario como procesos totalmente independientes. • Usos principales: desarrollo y prueba al nivel del *kernel*, alojamiento de *servidores virtuales*, entornos educativos. • Rendimiento cercano al nativo por ejemplo con KVM. • Soporte de una infinidad de sistemas operativos invitados. • Operaciones avanzadas disponibles. • KVM por ejemplo no requiere modificación en el *kernel Linux*, al ser integrado como módulo.	• Salvo que usemos Intel VT o AMD-V, tanto las librerías como las aplicaciones de las máquinas virtuales deben estar compiladas para el mismo conjunto de instrucciones que el *kernel* del sistema anfitrión. • KVM no requiere modificar el *kernel*, pero precisamente por eso su rendimiento es menor que el obtenido con *paravirtualización*, por ejemplo.

Se podrían haber comparado más exhaustivamente las posibilidades que ofrecen soluciones concretas dentro de cada submodelo en cuanto a *procesador anfitrión* y *procesador invitado* soportados, *sistema operativo anfitrión e invitado* soportados, licencia, soporte *migración de máquinas virtuales,…* por ejemplo, pero se no se ha considerado necesario ya que no existen grandes restricciones ni requisitos en cuanto a ello, salvo los comentados anteriormente.

Teniendo en cuenta la comparativa presentada en la tabla anterior y los requisitos que hay que abordar en el proyecto **la *paravirtualización* es escogida para el desarrollo de la infraestructura virtual del proyecto como modelo a seguir, y dentro de ella Xen como solución, debido a que**:

- **El rendimiento obtenido por las soluciones de *paravirtualización* es excelente**, permitiendo que las máquinas virtuales sean ejecutadas prácticamente de forma nativa, como si se tratase de equipos físicos.

- **Hace uso de *hipervisor*, con las ventajas que ello supone** de cara a la gestión y administración de la infraestructura. Además **es un *hipervisor* más ligero, descargado de tareas con penalizaciones en el rendimiento** como la interceptación de instrucciones privilegiadas.

- Es posible su uso en conjunción con **soporte hardware para virtualización de los procesadores Intel VT y AMD-V**, comportándose entonces como *virtualización pura nativa*.

- **Las desventajas en *paravirtualización* no lo son tanto**. El que en un principio solo podamos instanciar máquinas virtuales con sistemas operativos modificables es algo que desde el punto de vista GNU/Linux es un compromiso totalmente razonable debido a las mejoras obtenidas en el rendimiento respecto a otros modelos.

- **Xen, máximo exponente de la *paravirtualización*, es una de las soluciones más extendidas en uso en la actualidad, incluso en consolidación de servidores**.

- **Que Xen sea *software libre* distribuido bajo la licencia GNU GPL proporciona una gran flexibilidad** en el uso de la herramienta, desarrollada y mantenida por una activa y extensa comunidad de desarrolladores. **Xen puede ser utilizado en las últimas versiones de las distribuciones GNU/Linux más utilizadas.**
- **Xen permite desarrollar infraestructuras virtuales seguras, eficientes, escalables y dinámicas**.

- **Xen proporciona funcionalidades *extra* avanzadas** como administración de colas de servidores anfitriones remotos y *migración en caliente de máquinas virtuales*.

- **Existe una gran variedad de herramientas administrativas y de apoyo auxiliares para Xen**: interfaces gráficas, utilidades para la automatización de procesos, sitios web con máquinas virtuales preconfiguradas, etc.

- **Xen ha mostrado buenos resultados en relación a su integración con sistemas de almacenamiento compartidos**: NFS, sistemas de ficheros replicados, NAS, etc.

Con un modelo y solución de *virtualización de plataforma* elegidos nos disponemos a continuación finalmente a comparar y seleccionar un tipo de **medio de almacenamiento compartido para la localización de las máquinas virtuales**.

3.2 SELECCION DE UNA SOLUCION DE ALMACENAMIENTO COMPARTIDO

La tabla *2.5 Sistemas de almacenamiento* contiene de forma resumida cuáles son **las principales ventajas e inconvenientes que presentan las técnicas de almacenamiento** explicadas anteriormente en este capítulo. Es fundamental elegir correctamente una de ellas para la implementación de nuestra infraestructura virtual ya que **el sistema de almacenamiento elegido alojará las imágenes de disco y ficheros de configuración de nuestras máquinas virtuales, siendo éste accesible y compartido por los servidores físicos anfitriones** de los que dispongamos.

Tabla 2-5. Sistemas de almacenamiento

Submodelo	Ventajas	Inconvenientes
RAID	Flexibilidad (6 configuraciones diferentes a usar según nuestras necesidades). Posibilidad de RAID software.	Diferente rendimiento, coste, redundancia… dependiendo de la configuración implementada.
LVM2	Gran potencial y flexibilidad. RAID 0. Transparencia de cara al sistema de ficheros.	No es posible implementar RAID 1 o 5.
Sistemas de ficheros distribuidos o en red (NFS, CIFS…)	Escalabilidad, flexibilidad, transparencia. Información localizada de forma centralizada. Bajo coste y sencillez para su implantación.	Compatibilidad dependiendo del protocolo utilizado. El servidor como único punto de fallo.
Sistemas de ficheros replicados (DRBD)	Transparencia. Replicación en tiempo real y de manera ininterrumpida. Posibilita *migración de máquinas virtuales* al crear un disco duro virtual compartido. Proporciona alta disponibilidad.	Replicación a *nivel de bloque*. No admite diversas localizaciones como origen de la copia. Puede experimentar bajo rendimiento al replicar máquinas virtuales.
NAS	Conexiones TCP/IP (Ethernet, ATM…) lo que permite usar NFS y CIFS (Samba): compatibilidad. Hace uso de RAID. Acceso a *nivel de fichero*. Existencia de distribuciones *NAS (NAS software).*	Alto coste cuando sean usados *NAS hardware.*
SAN	Rápido, seguro, fiable. Administración centralizada.	Alto coste (uso de *Fibre Channel*): conexiones de fibra óptica para cada servidor, *switch* de canal de fibra. Único punto de fallo. Acceso a *nivel de bloque.* Poca compatibilidad.

Las características que deben primar en el almacenamiento que queremos establecer en los escenarios a desarrollar en el proyecto **deben ser fundamentalmente cuatro**: **sencillez, eficiencia, bajo coste, y buen rendimiento**. **Teniendo en cuenta los objetivos del proyecto y la información presentada en la tabla anterior se decide usar NFS como sistema de almacenamiento compartido para las máquinas virtuales**: es fácil de instalar y configurar, desde hace años se considera un estándar para los sistemas de ficheros en red, es escalable, flexible y transparente para los equipos. Además su coste, como es de sobra sabido, es bajo.

CAPITULO 3

XEN

1 Introducción

En los capítulos anteriores estudiamos junto al resto de soluciones de virtualización qué es Xen y de forma general cómo trabaja, quedando clara su importancia en cuanto a funcionalidad, flexibilidad y alto rendimiento. De esta manera fue elegida como la herramienta que más se adapta a las necesidades del proyecto fin de carrera para la construcción de nuestra infraestructura virtual de servidores.

Ahora es el momento de apreciar todo lo comentado con anterioridad desde un punto de vista práctico; **examinar y probar con detenimiento las principales funcionalidades que describen el éxito de Xen, instalando y configurando con gran flexibilidad un *hipervisor* ligero y rápido para desarrollar sobre él máquinas virtuales haciendo uso tanto de *paravirtualización* como de *virtualización completa soportada por hardware***. Antes de comenzar con la instalación y configuración de Xen listaremos los conceptos fundamentales en su terminología y recordaremos brevemente la arquitectura interna que presenta el sistema en su versión 3.0, con la que trabajaremos.

Para trabajar con Xen es necesario **conocer los términos y definiciones** que son usados para referirnos a los conceptos más importantes en su tecnología:

- ***VM (Virtual Machine).*** Una *máquina virtual* es en Xen el entorno virtualizado formado por un sistema operativo y las aplicaciones que los usuarios ejecutan sobre él.

- ***PVM (Paravirtualized Virtual Machine).*** Una máquina virtual *paravirtualizada* es aquella que es ejecutada por Xen en un entorno virtual con *paravirtualización* y por tanto en la que solamente es posible ejecutar sistemas operativos modificables. Gracias a las modificaciones llevadas a cabo en la máquina virtual ésta es

consciente del proceso de virtualización liberando de cargas *extra* al *hipervisor*, mejorando notablemente el rendimiento global.

- ***HVM (Hardware Virtual Machine).*** Una *máquina virtual hardware* puede al contrario que una *PVM* ejecutar sistemas operativos no modificables ya que habita en un entorno completamente virtualizado haciendo uso para ello de *soporte hardware en el procesador*.

- ***VMM (Virtual Machine Monitor).*** Es una forma equivalente de nombrar al *hipervisor (monitor de máquinas virtuales)*. Es el software que se ejecuta directamente sobre el hardware permitiendo el establecimiento y mantenimiento de toda la infraestructura virtual incluyendo las máquinas virtuales.

- ***Host.*** Sistema *anfitrión* que proporciona todos los recursos físicos y ambiente para la ejecución de máquinas virtuales.

- ***Guest.*** Sistema *invitado* que se ejecuta en una máquina virtual.

- ***Domain.*** Un dominio es una instancia de una máquina virtual, es decir, una máquina virtual con unos parámetros o características determinadas que la diferencian del resto. De todas formas, dominio y máquina virtual suelen ser utilizados de forma indistinta.

- ***dom0.*** Es el primer *dominio* en el sistema Xen y dispone de los privilegios y *drivers* necesarios para el acceso directo al hardware disponible en el servidor anfitrión. Su sistema operativo es ejecutado por el propio Xen y tiene un carácter administrativo con utilidades y herramientas que le permiten crear y operar sobre el resto de dominios.

- ***domU.*** Resto de dominios existentes en el sistema Xen, configurados, puestos en marcha y administrados por *dom0*. Por ejemplo si dispusiéramos de un servidor Xen con cuatro sistemas *invitados* tendríamos como *dominios dom0* (idéntico al resto de *dominios* salvo por el acceso privilegiado al hardware del servidor), *dom1*, *dom2* y *dom3*.

En la figura 3.1 queda recogida la **arquitectura Xen 3.0 incluyendo dos *dominios invitados***. Los *dominios* o *máquinas virtuales* son ejecutados de forma paralela (en el ejemplo dos dominios) sobre un único sistema físico anfitrión. Cuando se inicia Xen en primer lugar arranca el *hipervisor*, que corre directamente sobre el hardware subyacente, el cual a su vez es el responsable de iniciar el *dominio dom0* sobre el que se ejecuta el sistema operativo anfitrión. Alcanzado este punto el *hipervisor* comprueba la tabla de páginas y se encarga de la asignación de recursos a los nuevos dominios. **Cada *dominio* interactúa con el *hipervisor* mediante *hiperllamadas*** (*hypercalls*, consistentes en señales software); el *hipervisor* responde a las *hiperllamadas* mediante el envío de eventos. **Una *hiperllamada* es a una llamada al sistema (*syscall*) como el *hipervisor* al sistema operativo**.

Entonces de esta forma **es posible la creación de nuevos *dominios domU* mediante las herramientas y utilidades disponibles en *dom0***. Estas herramientas trabajan comunicándose con una ***interfaz de control* en el *hipervisor***. Todas las peticiones de acceso al hardware por parte de los *dominios domU* son realizadas desde su *interfaz cliente de drivers de dispositivo (front-end device drivers)* a la *interfaz servidora de drivers de dispositivo* del *dom0*, el cual las *pasará* al hardware. Esto último es posible gracias a que los sistemas operativos invitados *saben* que están siendo ejecutados en dominios sobre el *hipervisor Xen.*

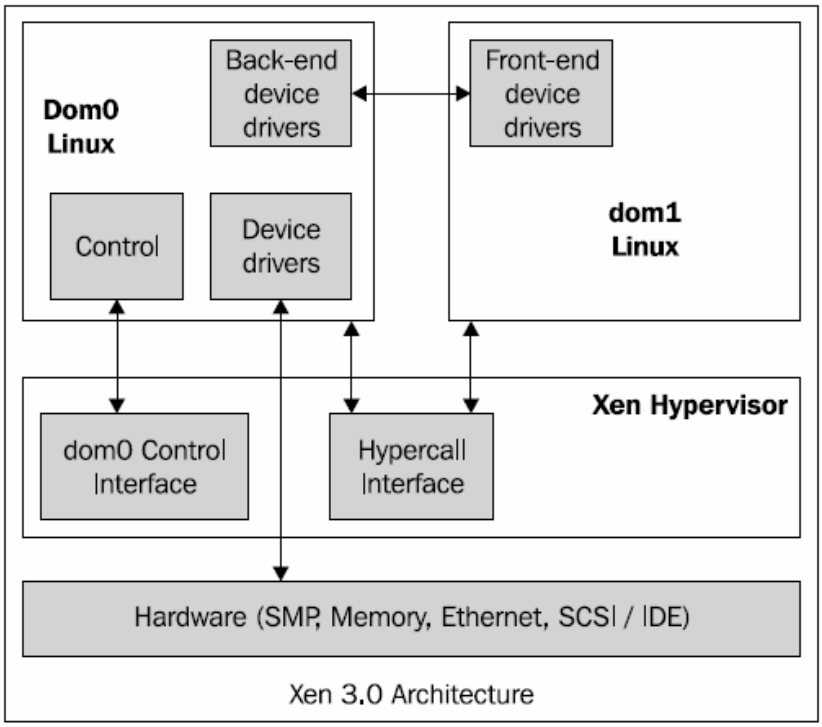


Figura 3.1 Arquitectura Xen 3.0.

2 Instalación y configuración inicial

A continuación veremos paso a paso cómo instalar Xen haciendo uso de paquetes disponibles en repositorios. **El código y los ejemplos que se muestran han sido realizados en la última versión estable de la distribución Debian GNU/Linux sobre la arquitectura x86_64 (AMD64), *Lenny***, aunque se pueden realizar de forma análoga en otras distribuciones adecuando los contenidos. Xen se encuentra disponible en forma de paquetes binarios para cualquier distribución, aunque se considera que dispone de una mayor estabilidad y seguridad su instalación haciendo uso de sistemas empaquetados.

La instalación de Xen consiste en tres fases elementales en las que se instalan diferentes elementos de la infraestructura de virtualización:

- **El *hipervisor***.
- **El *núcleo o kernel Xen modificado*** que trabajará coordinado con el *hipervisor*.
- **Utilidades y aplicaciones que consideremos necesarias** para trabajar con virtualización: creación de máquinas virtuales, provisionamiento, monitorización, etc.

Al final también **se** presenta de forma básica cómo configurar Xen para alojar máquinas virtuales en un sistema Linux, con el objetivo de una mejor comprensión así como iniciar al lector para que pueda configurarlo según sus necesidades. Además analizaremos con mayor detalle los archivos de configuración de Xen en el siguiente apartado.

Antes de nada resulta de utilidad comentar las ***dependencias* requeridas por Xen en su instalación**, **resueltas de forma automática para el usuario**; ello nos permitirá entender algunas funcionalidades sobre las que trabaja Xen. **Para la instalación de Xen v3.3 son necesarias las siguientes dependencias, utilidades y aplicaciones complementarias**:

- **Un sistema física anfitrión en el que estemos ejecutando una distribución Linux con gestor de arranque *GRUB***.

- **El paquete *iproute2***. Se trata de una colección de utilidades para el control de las actividades de *networking* en TCP/IP, como la configuración IPv4 o IPv6 de una interfaz de red o el control del tráfico.

- **El conjunto de utilidades *bridge-utils***. Permiten configurar el *bridge Ethernet Linux 2.4*, utilizado para la conexión de varias interfaces de red a la misma salida a la vez.

- **El sistema Linux *hotplug*** –ya incluido desde hace tiempo **en las últimas distribuciones como *udev***. *udev* es el gestor de dispositivos que usa el *kernel Linux* en su versión 2.6. Controla los ficheros de dispositivo en el directorio */dev*.

- ***bridge-utils.*** Paquete que contiene utilidades que nos permitirán configurar un *puente Ethernet Linux*, necesario para conectar varios adaptadores de red.

- ***libxenstore.*** Librería Xen para las comunicaciones *XenStore*: espacio compartido entre dominios para el almacenamiento de información.

- ***xen-utils.*** Se trata de un conjunto de utilidades de espacio de usuario para administrar sistemas virtualizados Xen.

- ***xen-utils-common.*** Archivos comunes para las utilidades que contiene el paquete *xen-utils.*

- ***xenstore-utils***. Grupo de utilidades para la manipulación del *XenStore*.

Xen puede ser ejecutado en architecturas x86 necesitando un procesador *P6* o posterior (por ejemplo Pentium Pro, Celeron, Pentium II-III-IV, Xeon, AMD Athlon, AMD Duron). **También son soportadas máquinas multiprocesador así como *Hyperthreading (SMT)*, y arquitecturas *IA64* o *PowerPC*.**

Xen de 32 bits soporta también por defecto las extensiones de direccionamiento físico de Intel (*Intel PAE*) lo que permite a máquinas x86 de 32 bits direccionar hasta 64Gb de memoria física (en el caso de no usar *PAE* sólo podremos direccionar hasta 4Gb). Xen también soporta plataformas x86/64 como *Intel EM64T* y *AMD Opteron* las cuales pueden direccionar hasta 1Tb de memoria física actualmente.

A pesar de todo **Xen *traspasa* la mayoría de las cuestiones de soporte hardware al sistema operativo invitado ejecutado por *dom0***, como por ejemplos los *drivers de dispositivo.* **Por sí mismo Xen sólo contiene el código requerido para la detección e iniciación de procesadores secundarios, *routing* de interrupciones y ejecutar la enumeración del bus PCI**. Esta aproximación permite aumentar la compatibilidad con la mayoría de dispositivos hardware soportados por Linux. Por defecto el *kernel Xen* soporta también la mayoría del hardware de red y disco para servidores, aunque esto siempre es configurable para añadir soporte para hardware adicional.

2.1 INSTALACION DEL HIPERVISOR

Como ya sabemos el *hipervisor* en una de las partes más importantes, sino la más importante, en el proceso interno de las soluciones de *paravirtualización* como Xen. **El *hipervisor Xen* se ejecuta directamente sobre el hardware**, esto es, es de tipo *bare metal* o *native*. **Actúa como *capa de virtualización*** mostrando a las máquinas virtuales y administrando los recursos del hardware disponible. **Al realizar modificaciones en los sistemas operativos de los dominios no es necesaria la emulación de hardware en el *hipervisor Xen*, lo que le permite eliminar la necesidad de captura de instrucciones conflictivas y por tanto mejorar su rendimiento**. Además, y esto es muy importante, **es el encargado de iniciar el dominio privilegiado *dom0***, en el cual corre el sistema operativo anfitrión.

En este primer paso para la instalación de Xen instalamos su *hipervisor*, conocido como *Xen Hypervisor*, al mismo tiempo que vemos que **son instaladas una serie de utilidades de gestión y control que son utilizadas por el *dominio administrativo* de Xen *dom0***. En primer lugar nos registramos en el sistema con cuenta de administrador *root* para poder realizar todas las operaciones:

```
usuario@deb1:~$ su root
```

Ahora ya con privilegios de administrador estamos en disposición de proseguir con la instalación. En primer lugar **buscamos el paquete *xen-hypervisor*** para ver las diferentes versiones y opciones disponibles según nuestra distribución y arquitectura del procesador:

```
deb1:/home/usuario# aptitude search xen-hypervisor
```

De entre las opciones que veamos en la salida de la búsqueda **elegimos la última versión** que se adecúa mejor a nuestro caso, en concreto el paquete ***xen-hypervisor-3.2-1-amd64*** ya que disponemos de una arquitectura de 64 bits (x86_64 o AMD64), **y realizamos su instalación**:

```
deb1:/home/usuario# aptitude install xen-hypervisor-3.2-1-amd64
```

Una vez estos los paquetes relativos a utilidades adicionales y dependencias han sido descargados y configurados, a excepción de ***xen-utils*** ya que lo hará en última instancia, **el instalador realiza las siguientes operaciones**:

```
Searching for GRUB installation directory ... found: /boot/grub
Searching for default file ... found: /boot/grub/default
Testing for an existing GRUB menu.lst file ... found:
/boot/grub/menu.lst
Searching for splash image ... none found, skipping ...
Found kernel: /boot/vmlinuz-2.6.26-2-amd64
Updating /boot/grub/menu.lst ... done
```

El proceso de instalación busca el directorio del gestor de arranque (*GRUB*) para comprobar que funciona correctamente. Antes de **actualizarlo con una entrada para al *hipervisor*** de Xen muestra también el *kernel* actualmente disponible en el directorio */boot*. **El instalador termina con la instalación y configuración del paquete *xen-utils***.

2.2 INSTALACION DEL KERNEL XEN

Una vez instalado el *hipervisor Xen* **el siguiente paso consiste en la instalación del *núcleo* o *kernel Xen***, también fundamental ya que se trata del ***kernel* del dominio administrativo *dom0***, responsable de las actividades más importantes en la infraestructura virtual.

Para disponer del *kernel Xen* deberemos descargar e instalar la versión adecuada de dos paquetes muy importantes:

- ***linux-modules***. Paquete binario que proporciona los módulos cargables precompilados para el *kernel Linux Xen.* Lo elegiremos en función del tipo de arquitectura del equipo en el que queremos instalarlo así como de versión del *kernel Linux Xen* y la distribución que vayamos a instalar.

- ***linux-image***. Paquete binario que contiene la imagen de una versión del *kernel Linux Xen* para una determinada arquitectura del procesador y distribución Linux. Será usado como *kernel* para los dominios *dom0* y *domU*.

Es recomendable antes de elegir una versión tanto de *linux-modules* como de *linux-image* estar completamente seguro de la estabilidad de los mismos de acuerdo con la arquitectura de nuestro procesador y la distribución Linux que estemos usando. **Procedemos para descargar e instalar el paquete *linux-modules-2.6.26-2-xen-amd64_2.6.26-19lenny2_amd64.deb*** de los repositorios de Debian de la siguiente forma:

```
deb1:/home/usuario/Desktop# wget http://security.debian.org/debian-security/pool/updates/main/l/linux-2.6/linux-modules-2.6.26-2-xen-amd64_2.6.26-19lenny2_amd64.deb

deb1:/home/usuario/Desktop# dpkg -i linux-modules-2.6.26-2-xen-amd64_2.6.26-19lenny2_amd64.deb
```

De la misma manera **operamos con el paquete *linux-image-2.6.26-2-xen-amd64_2.6.26-19lenny2_amd64.deb***:

```
deb1:/home/usuario/Desktop# wget http://security.debian.org/debian-security/pool/updates/main/l/linux-2.6/linux-image-2.6.26-2-xen-amd64_2.6.26-19lenny2_amd64.deb

deb1:/home/usuario/Desktop# dpkg -i linux-image-2.6.26-2-xen-amd64_2.6.26-19lenny2_amd64.deb
```

2.3 INSTALACION DE LIBC6-XEN

Este último y tercer paso **tan solo hay que realizarlo cuando instalamos Xen en arquitecturas i386. *libc6-xen* es un conjunto de librerías compartidas *glibc6* en su versión para Xen**.

```
deb1:/home/usuario/Desktop# aptitude search libc6-xen
```

Para su **instalación** procedemos como siempre, ejecutando el comando correspondiente dependiendo de la distribución Linux que estemos usando:

```
deb:/home/usuario/Desktop# aptitude install libc6-xen
```

Esta instalación significa que no habrá que deshabilitar */lib/tls* o eliminarlo como muchas guías de instalación de Xen sugieren.

2.4 CONFIGURACION DEL ARRANQUE

Una vez realizada la instalación de Xen es necesario seguir una serie de pasos antes de reiniciar el sistema y comenzar a utilizarlo:

- **Paso 1: Comprobación de los ficheros para el arranque de Xen.** Comprobamos en primer lugar, y si todo fue bien no habrá ningún problema, que en el directorio */boot* después de la instalación realizada tenemos disponibles tres archivos en referencia a Xen:

```
deb1:/boot# ls
config-2.6.26-2-amd64      initrd.img-2.6.26-2-amd64
System.map-2.6.26-2-amd64      vmlinuz-2.6.26-2-xen-amd64
config-2.6.26-2-xen-amd64  initrd.img-2.6.26-2-amd64.bak
System.map-2.6.26-2-xen-amd64  xen-3.2-1-amd64.gz
grub                       initrd.img-2.6.26-2-xen-amd64
vmlinuz-2.6.26-2-amd64
```

Los tres son especialmente importantes: el *kernel Xen* (*xen-3.2-1-amd64.gz*) **y su ejecutable** (*kernel* comprimido *vmlinuz-2.6.26-2-xen-amd64*) **y un *disco RAM inicial* asociado** (*initrd.img-2.6.26-2-xen-amd64*), que es un sistema de archivos temporal que utilizará el *kernel* para la carga de módulos o modificaciones durante el inicio del sistema antes de que el sistema raíz sea montado.

- **Paso 2: Configuración del arranque del dominio *dom0*. Lo hacemos editando el fichero de configuración del demonio *xend, xend-config.sxp***, encargado de la ejecución de herramientas administrativas relacionadas con los dominios**.** Este fichero se encuentra **en el directorio */etc/xen*** y es quizás el más importante de cara a caracterizar el comportamiento de Xen:

```
deb1:/etc/xen# ls
scripts  xend-config.sxp  xend-config-xenapi.sxp  xend-pci-
permissive.sxp  xend-pci-quirks.sxp
```

En el siguiente apartado será cuando veamos con mayor detenimiento el contenido de este fichero, en el cual ahora sólo realizaremos unas pocas **modificaciones con el objetivo de garantizar el arranque del dominio *dom0***. Así, **descomentamos las siguientes cuatro líneas**:

```
(network-script network-bridge)

(vif-script vif-bridge)

(dom0-min-mem 196)

(dom0-cpus 0)
```

- **En las dos primeras líneas establecemos que el script que utilizaremos para la configuración de la red será *bridge*, así como la interfaz virtual que también será de tipo *bridge***. El modo *bridge* permite que una conexión de red se comporte de forma exactamente igual a como lo haría si estuviera conectada directa a la red física. Tanto el modo *bridge* como el resto de modos disponibles son estudiados en el apartado *3. Configuración de la red.*

- **En la tercera línea escribimos el mínimo de memoria que reservaremos para el dominio *dom0***. Se trata de un parámetro muy importante porque en casos en los que no se incluya un mínimo aceptable podemos arriesgarnos a no disponer de memoria suficiente para el arranque de *dom0* –por ejemplo cuando muchos dominios consuman el resto de los recursos de memoria.

- Finalmente **en la última línea configuraremos el número de procesadores o *cores* a los que tendrá acceso para la computación de sus actividades *dom0;*** al escribir 0, tendrá acceso a todos los disponibles.

- **Paso 3: Configuración de la entrada en el menú de arranque de sistemas *GRUB* para el *kernel Xen* y por ende del dominio *dom0***. Para ello localizamos el fichero *menu.lst* en el directorio */boot/grub*:

```
deb1:/boot/grub# ls
default     e2fs_stage1_5  grub.conf     menu.lst
minix_stage1_5     stage1  xfs_stage1_5
device.map  fat_stage1_5   jfs_stage1_5  reiserfs_stage1_5
stage2
```

Editamos en él la entrada que fue creada durante la instalación de Xen con los valores adecuados para la partición raíz, ruta del *kernel* y los parámetros y módulos que necesitemos incluir para su lanzamiento, por ejemplo:

```
title          Xen 3.2-1-amd64 / Debian GNU/Linux, kernel
2.6.26-2-xen-amd64

root           (hd0,0)

kernel         /boot/xen-3.2-1-amd64.gz dom0_mem=196M

module         /boot/vmlinuz-2.6.26-2-xen-amd64 root=/dev/sda1
ro console=tty0

module         /boot/initrd.img-2.6.26-2-xen-amd64
```

El parámetro del *kernel dom0_mem* es muy importante porque nos ayudará a garantizar que el dominio *dom0* dispondrá del mínimo de memoria.

Concluida esta serie de pasos para la configuración del arranque de Xen y el dominio administrativo *dom0* ya podemos reiniciar el sistema y seleccionar la entrada creada en el menú de arranque para comenzar a trabajar con ellos.

3 Configuración básica e interfaz administrativa de dom0

Antes de crear dominios invitados en nuestro sistema Xen es preciso **una vez instalado *dom0* conocer cómo podemos acceder a su configuración y saber cómo administrar de forma básica en él el resto de dominios *domU***. Es por ello que en el presente apartado **estudiamos la configuración del demonio *xend* mediante el fichero *xend-config.sxp*** para lo primero **y la interfaz administrativa de *dom0*, *xm***, para lo segundo.

3.1 XEND Y EL FICHERO XEND-CONFIG.SXP

El demonio *xend*, escrito en python, **es el responsable de la ejecución de las funciones de administración del sistema relacionadas con las máquinas virtuales en Xen**. **Es el punto central para el control de todos los recursos virtualizados** de los que disponen los dominios por lo que es un requisito indispensable que se encuentre en ejecución (como usuario *root*, ya que necesita acceso a funciones de administración del sistema privilegiadas) para poder poner en marcha y administrar las máquinas virtuales.

Para poder arrancar automáticamente *xend* durante el inicio del sistema disponemos del script */etc/init.d/xend*. Haremos uso si lo consideramos necesario de herramientas como *chkconfig* o *ntsysv* para ello. De todas formas **es posible gestionar su estado desde línea de comandos** también como es habitual.

Para **iniciar, detener, reiniciar o ver el código de estado de *xend*** lo haremos como sigue:

```
# xend start|stop|restart|status
```

Para configurar el comportamiento de *xend* modificamos el fichero */etc/xenxend-config.sxp*. Podemos ver información completa sobre los distintos parámetros y opciones en la página *man* de *xend-config.sxp* (figura 3.2) y el propio fichero, ampliamente documentado.

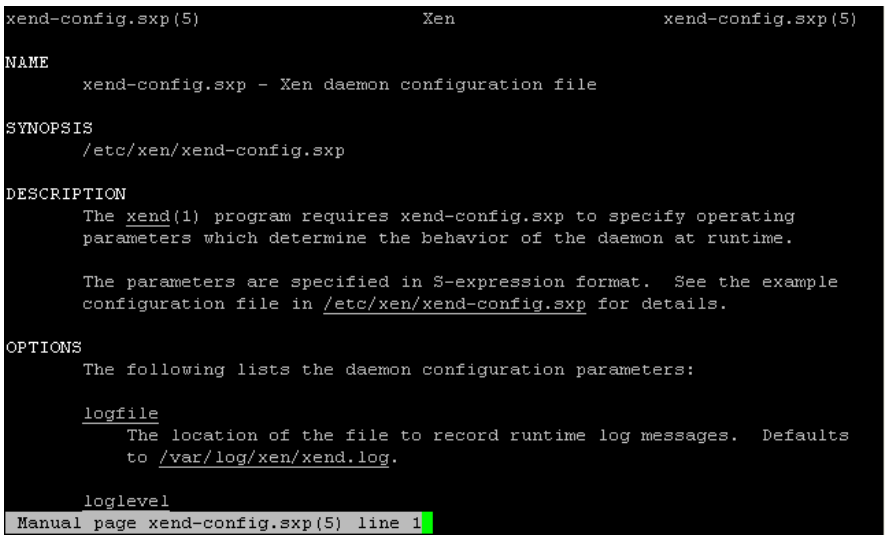


Figura 3.2 Página man de xend-config.sxp.

Tanto una interfaz HTTP como una *API (Application Programming Interface)* de *sockets* en el dominio Unix (necesario como utilidad de bajo nivel para la interfaz administrativa

de *dom0 xm*) se facilitan para las comunicaciones con *xend*, permitiendo a usuarios remotos ejecutar comandos.

Veamos los aspectos más importantes de la configuración de Xen en el fichero *xend-config.sxp*, cuyas líneas comentaremos o descomentaremos en función de las características que queramos dejar presentes:

- ***logfile.*** Fichero donde se almacena la información generada como *log* en tiempo de ejecución de *xend.* **Por defecto el fichero se encuentra en el directorio */var/log/xen/xend.log***.

- ***loglevel.*** Sirve para filtrar los mensajes que aparecerán en el archivo *log* por debajo del nivel especificado: *DEBUG* (por defecto), *INFO, WARNING, ERROR, o CRITICAL*.

- ***xend-http-server.*** Escribimos *'yes'* o *'no'* dependiendo de si queremos que *xend* inicie el servidor HTTP para permitir este tipo de comunicaciones. Por defecto se encuentra deshabilitado.

- ***xend-unix-server.*** Análoga a la opción anterior pero para el servidor de *sockets* tipo Unix. Se encuentra habilitado por defecto ya que su uso es obligatorio por las herramientas *cliente* que operan sobre los dominios, como por ejemplo *xm.*

- ***xend-relocation-server.*** Aquí podemos habilitar bajo nuestra responsabilidad (por defecto no lo está) el *servidor de reubicación* en *xend* para permitir la funcionalidad de *migración de dominios* entre diferentes máquinas físicas anfitrionas.

- ***xend-unix-path.*** Se trata de la ubicación del *socket* Unix que es utilizado por las herramientas administrativas para comunicarse con el *servidor Unix* de *xend.* Por defecto el directorio es: ***/var/lib/xend/xend-socket***.

- ***xend-port.*** Puerto de escucha que es utilizado para administrar el servidor HTTP de *xend.* Por defecto es el puerto **8000**.

- ***xend-relocation-port.*** Puerto que usa el *servidor de reubicación* cuando éste es habilitado al implementar *migración de dominios.*

- ***xend-address.*** Dirección en la que *xend* escucha para establecer conexiones HTTP si el servidor de este tipo está habilitado. Si escribimos *localhost* no será posible establecer ninguna conexión; si por el contrario escribimos *''* permitimos conexiones en cualquier interfaz de red.

- ***xend-relocation-address.*** Opción análoga a la anterior pero para el *servidor de reubicación* usado para *migración de dominios,* si se encuentra habilitado.

- ***console-limit.*** Límite del *buffer* de la consola del servidor en kilobytes. Por defecto tiene un valor de 1024, y es establecido por dominio, algo que es necesario para prevenir que un solo dominio logre saturar la consola del servidor mediante el envío masivo de información.

- ***network-script.*** Es el nombre del script ubicado en el directorio ***/etc/xen/scripts*** que será ejecutado para **establecer el entorno de red de la infraestructura virtual** y por tanto el rol que jugará *dom0* en él. Puede ser cualquier de los scripts disponibles, aunque **lo habitual es que sea *network-bridge, network-nat o network-route.*** En el apartado *3.3 Topologías de redes virtuales Xen* podemos ver en qué consiste cada uno de los esquemas de red establecidos por estos scripts.

- ***vif-script.*** Es el nombre del **script** (de los que se encuentran en */etc/xen/scripts*) **que es utilizado para configurar las interfaces de red virtuales cuando sean creadas y también destruidas**. Lo normal es que el tipo de interfaz virtual creada sea del mismo tipo de configuración de red establecido en *network-script.* Puede que el tipo de interfaz sea sobrescrito en los archivos de configuración de los dominios.

- ***dom0-min-mem.*** Especificamos la c**antidad mínima de memoria en megabytes reservada para *dom0***, pudiendo dejar el resto de memoria libre para ser asignada a los *dominios domU*. Es importante conocer cuál puede ser esta cantidad para evitar problemas a la hora de arrancar *dom0.* Si escribimos 0 la memoria de *dom0* no será cambiada al mínimo automáticamente: no especificaremos mínimo luego siempre usa el máximo disponible.

- ***dom0-cpus.*** Es el **número de procesadores o *CPUs* a las cuáles tiene acceso permitido *dom0* en sistemas *SMP*** *(multiprocesamiento simétrico,* con dos o más procesadores*).* Si escribimos el valor 0 le permitimos acceder a todos los procesadores que haya físicamente instalados.

Hay otras muchas opciones que nos permiten configurar otros aspectos avanzados. **A continuación podemos ver un ejemplo del fichero de configuración de Xen, *xend-config.sxp***:

```
# -*- sh -*-

#
# Xend configuration file.
#

# This example configuration is appropriate for an installation that
# utilizes a bridged network configuration. Access to xend via http
# is disabled.

# Commented out entries show the default for that entry, unless otherwise
# specified.

#(logfile /var/log/xen/xend.log)
#(loglevel DEBUG)


# The Xen-API server configuration.  (Please note that this server is
# available as an UNSUPPORTED PREVIEW in Xen 3.0.4, and should not be relied
# upon).
#
# This value configures the ports, interfaces, and access controls for the
# Xen-API server.  Each entry in the list starts with either unix, a port
# number, or an address:port pair.  If this is "unix", then a UDP socket is
# opened, and this entry applies to that.  If it is a port, then Xend will
# listen on all interfaces on that TCP port, and if it is an address:port
# pair, then Xend will listen on the specified port, using the interface
with
# the specified address.
#
# The subsequent string configures the user-based access control for the
# listener in question.  This can be one of "none" or "pam", indicating
either
```

```
# that users should be allowed access unconditionally, or that the local
# Pluggable Authentication Modules configuration should be used.  If this
# string is missing or empty, then "pam" is used.
#
# The final string gives the host-based access control for that listener. If
# this is missing or empty, then all connections are accepted.  Otherwise,
# this should be a space-separated sequence of regular expressions; any host
# with a fully-qualified domain name or an IP address that matches one of
# these regular expressions will be accepted.
#
# Example: listen on TCP port 9363 on all interfaces, accepting connections
# only from machines in example.com or localhost, and allow access through
# the unix domain socket unconditionally:
#
#   (xen-api-server ((9363 pam '^localhost$ example\\.com$')
#                   (unix none)))
#
# Optionally, the TCP Xen-API server can use SSL by specifying the private
# key and certificate location:
#
#                   (9367 pam '' /etc/xen/xen-api.key /etc/xen/xen-api.crt)
#
# Default:
#   (xen-api-server ((unix)))


#(xend-http-server no)
#(xend-unix-server no)
#(xend-tcp-xmlrpc-server no)
#(xend-unix-xmlrpc-server yes)
#(xend-relocation-server no)

#(xend-unix-path /var/lib/xend/xend-socket)


# Address and port xend should use for the legacy TCP XMLRPC interface,
# if xen-tcp-xmlrpc-server is set.
#(xen-tcp-xmlrpc-server-address 'localhost')
#(xen-tcp-xmlrpc-server-port 8006)

# SSL key and certificate to use for the legacy TCP XMLRPC interface.
# Setting these will mean that this port serves only SSL connections as
# opposed to plaintext ones.
#(xend-tcp-xmlrpc-server-ssl-key-file  /etc/xen/xmlrpc.key)
#(xend-tcp-xmlrpc-server-ssl-cert-file /etc/xen/xmlrpc.crt)


# Port xend should use for the HTTP interface, if xend-http-server is set.
#(xend-port            8000)

# Port xend should use for the relocation interface, if xend-relocation-
server
# is set.
#(xend-relocation-port 8002)

# Address xend should listen on for HTTP connections, if xend-http-server is
# set.
# Specifying 'localhost' prevents remote connections.
# Specifying the empty string '' (the default) allows all connections.
#(xend-address '')
#(xend-address localhost)

# Address xend should listen on for relocation-socket connections, if
# xend-relocation-server is set.
# Meaning and default as for xend-address above.
#(xend-relocation-address '')

# The hosts allowed to talk to the relocation port.  If this is empty (the
# default), then all connections are allowed (assuming that the connection
# arrives on a port and interface on which we are listening; see
# xend-relocation-port and xend-relocation-address above).  Otherwise, this
# should be a space-separated sequence of regular expressions.  Any host
with
# a fully-qualified domain name or an IP address that matches one of these
# regular expressions will be accepted.
#
# For example:
```

```
#  (xend-relocation-hosts-allow '^localhost$ ^.*\\.example\\.org$')
#
#(xend-relocation-hosts-allow '')

# The limit (in kilobytes) on the size of the console buffer
#(console-limit 1024)

##
# To bridge network traffic, like this:
#
# dom0: ----------------- bridge -> real eth0 -> the network
#                            |
# domU: fake eth0 -> vifN.0 -+
#
# use
#
(network-script network-bridge)
#
# Your default ethernet device is used as the outgoing interface, by
default.
# To use a different one (e.g. eth1) use
#
# (network-script 'network-bridge netdev=eth1')
#
# The bridge is named xenbr0, by default.  To rename the bridge, use
#
# (network-script 'network-bridge bridge=<name>')
#
# It is possible to use the network-bridge script in more complicated
# scenarios, such as having two outgoing interfaces, with two bridges, and
# two fake interfaces per guest domain.  To do things like this, write
# yourself a wrapper script, and call network-bridge from it, as
appropriate.
#
(network-script network-dummy)

# The script used to control virtual interfaces.  This can be overridden on
a
# per-vif basis when creating a domain or a configuring a new vif.  The
# vif-bridge script is designed for use with the network-bridge script, or
# similar configurations.
#
# If you have overridden the bridge name using
# (network-script 'network-bridge bridge=<name>') then you may wish to do
the
# same here.  The bridge name can also be set when creating a domain or
# configuring a new vif, but a value specified here would act as a default.
#
# If you are using only one bridge, the vif-bridge script will discover
that,
# so there is no need to specify it explicitly.
#
(vif-script vif-bridge)


## Use the following if network traffic is routed, as an alternative to the
# settings for bridged networking given above.
#(network-script network-route)
#(vif-script     vif-route)


## Use the following if network traffic is routed with NAT, as an
alternative
# to the settings for bridged networking given above.
#(network-script network-nat)
#(vif-script     vif-nat)


# Dom0 will balloon out when needed to free memory for domU.
# dom0-min-mem is the lowest memory level (in MB) dom0 will get down to.
# If dom0-min-mem=0, dom0 will never balloon out.
(dom0-min-mem 196)

# In SMP system, dom0 will use dom0-cpus # of CPUS
# If dom0-cpus = 0, dom0 will take all cpus available
(dom0-cpus 0)
```

```
# Whether to enable core-dumps when domains crash.
#(enable-dump no)

# The tool used for initiating virtual TPM migration
#(external-migration-tool '')

# The interface for VNC servers to listen on. Defaults
# to 127.0.0.1  To restore old 'listen everywhere' behaviour
# set this to 0.0.0.0
#(vnc-listen '127.0.0.1')

# The default password for VNC console on HVM domain.
# Empty string is no authentication.
(vncpasswd '')

# The VNC server can be told to negotiate a TLS session
# to encryption all traffic, and provide x509 cert to
# clients enalbing them to verify server identity. The
# GTK-VNC widget, virt-viewer, virt-manager and VeNCrypt
# all support the VNC extension for TLS used in QEMU. The
# TightVNC/RealVNC/UltraVNC clients do not.
#
# To enable this create x509 certificates / keys in the
# directory /etc/xen/vnc
#
#  ca-cert.pem       - The CA certificate
#  server-cert.pem   - The Server certificate signed by the CA
#  server-key.pem    - The server private key
#
# and then uncomment this next line
# (vnc-tls 1)

# The certificate dir can be pointed elsewhere..
#
# (vnc-x509-cert-dir /etc/xen/vnc)

# The server can be told to request & validate an x509
# certificate from the client. Only clients with a cert
# signed by the trusted CA will be able to connect. This
# is more secure the password auth alone. Passwd auth can
# used at the same time if desired. To enable client cert
# checking uncomment this:
#
# (vnc-x509-verify 1)

# The default keymap to use for the VM's virtual keyboard
# when not specififed in VM's configuration
#(keymap 'en-us')

# Script to run when the label of a resource has changed.
#(resource-label-change-script '')
```

3.2 XEN MANAGER (XM)

Xen Manager (xm) **es la principal herramienta para la administración en Xen**. **Hace uso del demonio *xend* para poder comunicarse con el *hipervisor Xen,*** por lo que es necesario disponer de privilegios administrativos para ejecutar sus comandos.

***xm* se encuentra diseñado para funcionar de forma asíncrona** por lo que es posible que al ejecutar un comando tarde un tiempo en interactuar con el sistema. Para comprobar la finalización o no de los mismos hay disponibles otros comandos con funciones de monitorización como vemos más adelante. **El formato general para invocar un *comando xm* es el siguiente**:

```
# xm subcomando [opciones] [argumentos] [variables]
```

Las **opciones y argumentos siempre dependerán del comando que queramos ejecutar, mientas que las variables se especifican mediante declaraciones del tipo *variable***

= ***valor*** que sobrescribirán cualquier valor que haya en los archivos de configuración usados. Para acceder fácilmente a la lista de subcomandos de *xm* podemos ejecutar:

```
deb1:~# xm help
```

En la tabla 3.1 podemos ver un **resumen con los subcomandos disponibles** en la interfaz administrativa *xm*.

Tabla 3-1. Subcomandos *xm*

Subcomando	Breve descripción
console	Inicia la consola serie de un dominio.
create	Crea un dominio basado en archivo de configuración.
destroy	Apaga un dominio inmediatamente.
new	Añade un dominio al administrador de dominios *xend*.
delete	Elimina un dominio del administrador de dominios *xend*.
domid	Obtiene el identificador de un dominio a partir de su nombre.
domname	Obtiene el nombre de un dominio a partir de su identificador.
dump-core	Interrumpe el procesamiento por parte de un dominio.
list	Lista información sobre todos los dominios o uno concreto.
mem-max	Establece la cantidad de memoria máxima reservada para el dominio.
mem-set	Establece la cantidad de memoria actual en uso para el dominio.
migrate	Migra un dominio entre dos anfitriones diferentes.
pause	Pausa la ejecución de un dominio.
reboot	Reinicia un dominio.
rename	Cambia el nombre de un dominio.
restore	Restaura un dominio partiendo de su estado previamente guardado.
resume	Reanuda un dominio administrado por *xend*.
save	Guarda el estado de un dominio de forma que es posible restaurarlo después.
shutdown	Apaga un dominio.
start	Inicia un dominio administrado por *xend*.
suspend	Suspende un dominio administrado por *xend*.
sysrq	Envía *sysrq* al dominio.
trigger	Envía un *trigger* al dominio.
top	Monitoriza el sistema anfitrión y los dominios en tiempo real.
unpause	Finaliza la pausa de un dominio.
uptime	Muestra el tiempo que han estado ejecutándose todos los dominios o un dominio concreto.
vcpu-list	Lista los *procesadores virtuales* para todos los dominios o un dominio concreto.
vcpu-pin	Establece los procesadores que un *procesador virtual.*
vcpu-set	Establece el número de *procesadores virtuales* a los que tiene acceso un dominio.
debug-keys	Envía claves de depuración a Xen.
dmesg	Permite leer y/o limpiar el búfer de mensajes de Xen.
info	Muestra información sobre el host Xen.
log	Muestra el *log* realizado por Xen.
serve	Permite realizar proxy *xend* XMLRPC sobre *stdio*.
sched-credit	Obtiene/establece los parámetros para el planificador basado en créditos.
sched-sedf	Obtiene/establece los parámetros EDF.
block-attach	Crea un nuevo *dispositivo de bloque virtual*.
block-detach	Elimina un *dispositivo de bloque virtual* en un dominio.
block-list	Lista los *dispositivos de bloque virtuales* para un dominio.
block-configure	Permite editar la configuración de un *dispositivo de bloque.*
network-attach	Crea un nuevo *adaptador de red virtual*.
network-detach	Elimina un *adaptador de red virtual* en un dominio.
network-list	Lista los *adaptadores de red virtuales* para un dominio.
vtpm-list	Lista los *dispositivos TPM virtuales*.
vnet-list	Lista las *redes virtuales (vnets).*
vnet-create	Crea una nueva *red virtual (vnet)* a partir de un archivo de configuración.

vnet-delete	Eliminar una *red virtual (vnet).*
labels	Lista las etiquetas de un determinado tipo para una política activa.
addlabel	Añade una etiqueta de seguridad a un dominio.
rmlabel	Elimina una etiqueta de seguridad de un dominio.
getlabel	Muestra la etiqueta de seguridad para un dominio o recurso.
dry-run	Permite comprobar si un dominio tiene acceso a sus recursos.
resources	Muestra información para cada recurso etiquetado.
dumppolicy	Muestra información acerca del estado ACM del *hipervisor.*
setpolicy	Establece la política para el sistema.
resetpolicy	Establece la política para el sistema con los valores por defecto.
getpolicy	Obtiene la política del sistema.
shell	Lanza un *shell* interactivo.

Para obtener **más información** sobre los diferentes subcomandos es recomendable ver la página *man* de *xm* (véase la figura 3.3) y también la página *man* de *xmdomain.cfg*.

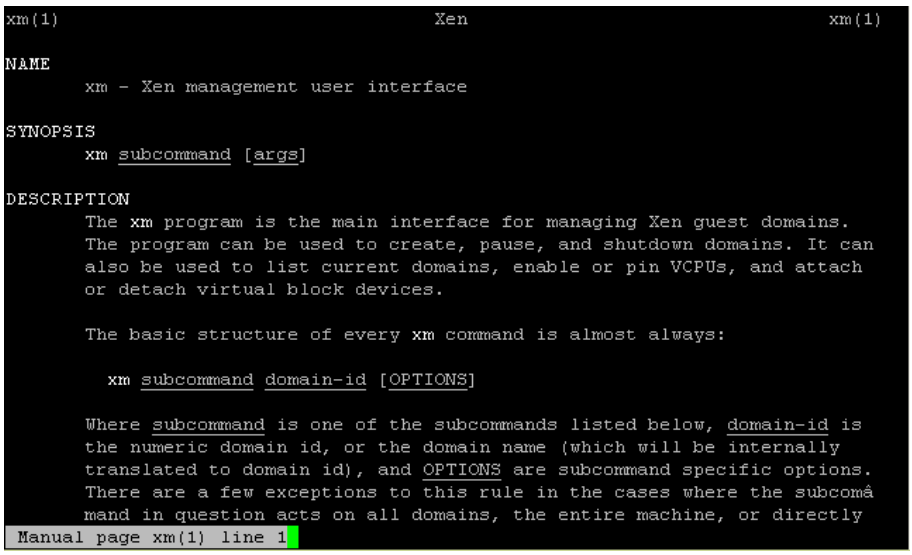


Figura 3.3 Página man de xm.

Existe una larga lista de comandos para la manipulación de aspectos avanzados de nuestro sistema como por ejemplo el mantenimiento de *redes virtuales* o la creación de políticas de seguridad. Son especialmente interesantes también los **comandos que nos permiten realizar una planificación de la carga de los dominios.** Xen dispone de un sistema planificador de CPU basado en créditos con el que es posible realizar un balanceado de carga de los dominios a través de los procesadores físicamente disponibles. **Se asigna un peso para marcar la prioridad y un límite de consumo de CPU (expresado en porcentaje) para cada dominio y el sistema se encarga automáticamente de planificar su ejecución**. Debemos recordar que es posible restringir las CPUs a las que tienen acceso los dominios mediante el comando *xm vcpu-pin*.

De todos los **subcomandos de *xm*** disponibles comentamos a continuación los que se consideran **de mayor importancia e imprescindibles para comenzar a utilizar Xen**. Se presentan clasificados en **tres categorías: administrativos básicos, para operar sobre recursos de los dominios y de monitorización**.

3.2.1 Comandos administrativos básicos

- ***xm console.*** Nos permite **acceder a la consola del dominio** que especifiquemos, bien mediante su nombre o bien mediante su identificador.

```
# xm console <dom_name>

# xm console <dom_id>
```

- ***xm create.*** Es el comando utilizado para la **creación de nuevos dominios basándonos en su archivo de configuración**. Además podemos especificar en la propia llamada al comando pares *variable-valor* para añadir características al dominio o para sobrescribir las que lo hayan sido en su configuración. Es necesario escribir la ruta completa del fichero de configuración del dominio a no ser que éste se encuentre en el directorio */etc/xen*.

```
# xm create config_dom.cfg [variable1=valor1
variable2=valor2…]

# xm create /camino/al/archivo/config.cfg [variable1=valor1
variable2=valor2… ]

# xm create –c config_dom.cfg [variable1=valor1
variable2=valor2…]
```

Como vemos en el tercer ejemplo podemos utilizar el ***flag –c*** equivalente a iniciar el dominio y acceder posteriormente a su consola con *xm console*. Si no lo escribimos el dominio solamente será iniciado y estará disponible para Xen (véase la figura 3.4).

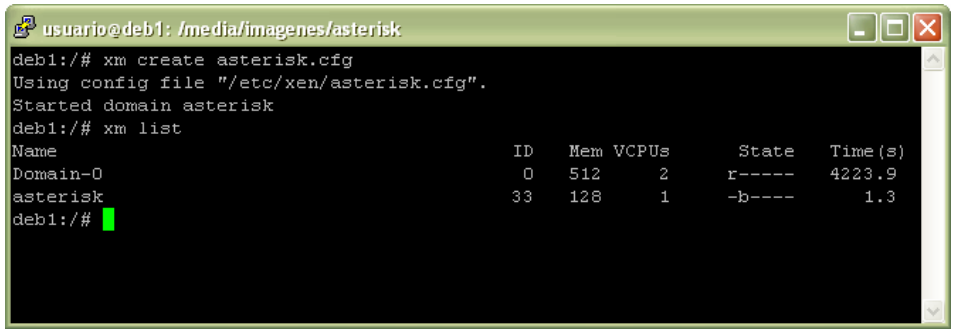


Figura 3.4 Creación de un dominio con xm create. Comprobación con xm list.

- ***xm destroy.*** Se utiliza cuando queremos **finalizar la actividad de un dominio de una forma *limpia***, apagándolo por completo. Su resultado es igual al que obtendríamos por ejemplo ejecutando *poweroff* en un equipo. Al igual que otras muchas operaciones puede ser aplicada sobre el dominio facilitando su nombre o su identificador.

```
# xm destroy <dom_name>

# xm destroy <dom_id>
```

- ***xm domid.*** Permite obtener el **identificador de un dominio a partir de su nombre**.

```
# xm domid <dom_name>
```

- ***xm domname.*** Es el subcomando opuesto al anterior: devuelve el **nombre de un dominio a partir de su identificador** (véase la figura 3.5).

```
# xm domname <dom_id>
```

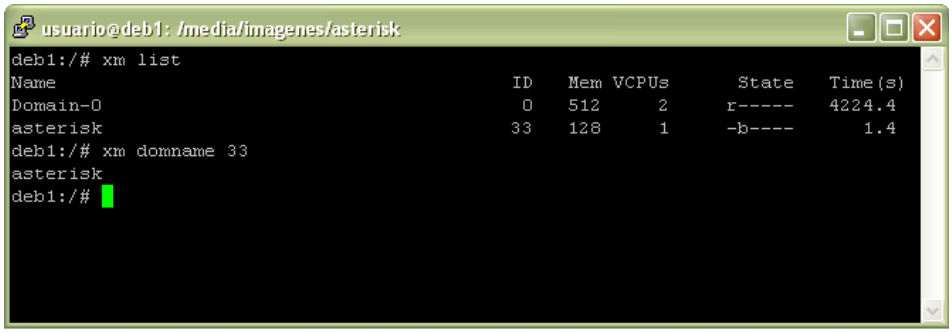


Figura 3.5 Obtención del nombre de un dominio sabiendo su identificador con xm domname.

- ***xm rename.*** Permite **cambiar el nombre de un dominio** aunque éste se encuentre en ejecución.

```
# xm rename <dom_name> <dom_name2>

# xm rename <dom_id> <dom_name2>
```

- ***xm pause.*** Permite **pausar un dominio que se encuentre en ejecución**, paralizando por completo su estado; ello no conlleva, como podríamos pensar, que el dominio libere los recursos que esté consumiendo en este instante.

```
# xm pause <dom_name>

# xm pause <dom_id>
```

- ***xm unpause.*** Con *unpause* podemos **devolver un dominio al estado en ejecución** en el que se encontraba antes pausarlo.

```
# xm unpause <dom_name>

# xm unpause <dom_id>
```

- ***xm migrate.*** Permite realizar la ***migración de dominios*** **desde el sistema anfitrión en que es ejecutado a otro**. Para migrar un dominio es necesario indicar el identificador o nombre del dominio a migrar y la dirección IP del sistema anfitrión destino. **El sistema anfitrión destino**:

    - debe estar corriendo el demonio *xend.*

- debe estar ejecutando la misma versión de Xen que el sistema origen.
- debe disponer del puerto TCP para migración abierto y aceptar conexiones desde el equipo origen.
- debe haber recursos disponibles para la acogida y ejecución del dominio: memoria, CPU, disco, etc.

La sintaxis de *xm migrate* es:

```
# xm migrate <dom_id> <host> [OPCIONES]

# xm migrate <dom_name> <host> [OPCIONES]
```

El subcomando *xm migrate* acepta al invocarlo **dos opciones**:

- ***-l/--live***. **Para usar *migración en caliente***: el dominio será movido entre los dos sistemas anfitriones sin tener que ser *pausado*.
- ***-r/--resource.*** Se usa para **fijar la máxima cantidad en megabytes permitidas para la *migración del dominio***. Así podremos asegurar que el enlace de red no es saturado con el tráfico generado por la migración.

Para la implementación de esta funcionalidad además es necesario realizar alguna configuración en los sistemas Xen de los *hosts* que intervienen, optimizar el rendimiento mediante el uso de un medio compartido para el almacenamiento de las máquinas virtuales… Todo ello es explicado en el capítulo que regoce la implementación del proceso de *migración de dominios* en Xen.

- ***xm save.*** Con él podemos **guardar el estado actual en ejecución de un dominio en un fichero** para su posterior restauración en el mismo o en otro equipo anfitrión. Una vez que se guarda el dominio éste deja de ejecutarse por lo que consecuentemente libera todos los recursos de los que disponía asignados. Tanto el dominio como su estado pueden ser restaurados haciendo uso del comando *xm restore* y el archivo de estado.

```
# xm save <dom_id> archivo-estado-dominio

# xm save <dom_name> archivo-estado-dominio
```

En algunas ocasiones presenta limitaciones propias de este tipo de operaciones, como también las tiene la hibernación de equipos; por ejemplo, al guardar el estado del dominio es posible que las conexiones TCP establecidas caduquen debido a que el protocolo TCP usa *timeouts* para ellas.

- ***xm restore.*** **Permite la restauración de un dominio y su estado completo haciendo uso de un archivo de estado** en el que se almacenó con *xm save.*

```
# xm restore archivo-estado-dominio
```

- ***xm reboot.*** **Permite reiniciar un dominio**, actuando como si fuera ejecutado el comando *reboot* en la propia consola del dominio. El comportamiento del dominio

cuando es reiniciado puede ser configurado en su archivo de configuración correspondiente mediante la opción *on_reboot*.

```
# xm reboot [OPCIONES] <dom_id>

# xm reboot [OPCIONES] <dom_name>
```

Como vemos además **es posible especificar dos *flags* con opciones para *xm reboot***:

- ***-a/--all***. **Reiniciará todos los dominios** en ejecución en el sistema.
- ***-w/--wait***. Es usado **para que el subcomando no termine hasta que el reinicio se haya completado en el dominio**, algo que puede consumir bastante tiempo dependiendo de los procesos a parar. Si no se especifica, el subcomando devolverá inmediatamente su salida, lo que no quiere decir que el dominio haya completado el reinicio.

- ***xm shutdown.* Permite apagar el dominio**. Tomará más o menos tiempo dependiendo de la cantidad y carga de los servicios que se encuentren ejecutando en él. Como ocurre con otros subcomandos, el comportamiento del dominio cuando éste es ejecutado puede especificarse mediante el parámetro *on_shutdown* del archivo de configuración del dominio. Admite las mismas opciones vistas para *xm reboot*.

```
# xm shutdown [OPCIONES] <dom_id>

# xm shutdown [OPCIONES] <dom_name>
```

3.2.2 Comandos para administrar los recursos asignados a los dominios

Hay un conjunto de **subcomandos de *xm* que son realmente útiles para manipular los recursos que tienen asignados los dominios en tiempo de ejecución** –de lo contrario realizaríamos modificaciones en el fichero de configuración de los mismos. A continuación **veremos básicamente los que son relativos a la manipulación de memoria y procesadores**, **aunque también hay subcomandos para *dispositivos virtuales de bloques (VBDs, Virtual Block Devices), adaptadores de red virtuales*, etc**.

- ***resources.*** Con *xm resources* podemos **listar todos los recursos que han sido previamente etiquetados** en el archivo global de etiquetas de recursos.

```
# xm resources
```

- ***dry-run.* Permite comprobar que los recursos que han sido asignados al dominio en su archivo de configuración se encuentran accesibles**. Cada uno de los recursos es listado junto a la decisión final tomada sobre su estado (véase la figura 3.6). Por defecto toma los ficheros del directorio */etc/xen*; en caso contrario escribiremos la ruta completa del fichero.

```
# xm dry-run config_dom.cfg

# xm dry-run /camino/al/archivo/config.cfg
```

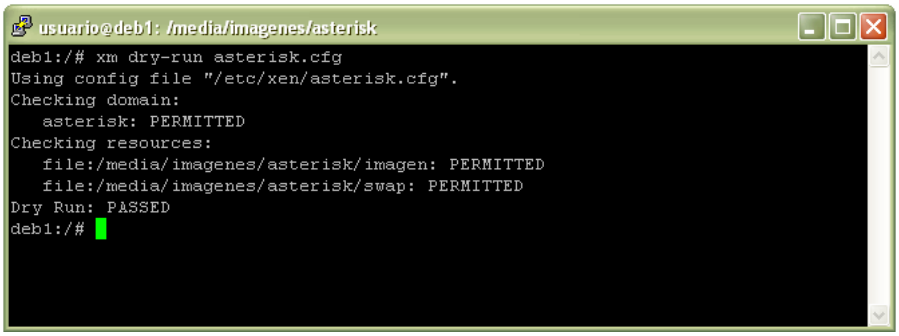


Figura 3.6 Comprobación del acceso a los recursos del dominio con xm dry-run.

- ***xm mem-max.*** **Permite establecer la cantidad máxima de memoria en megabytes reservada para el dominio**. Es probable que no se corresponda con la cantidad actual de memoria en uso para el dominio porque éste puede reducirla para que así el sistema operativo anfitrión disponga de ella.

```
# xm mem-max <dom_id> <mem>

# xm mem-max <dom_name> <mem>
```

- ***xm mem-set.*** Con él es posible **establecer la cantidad actual en uso para el dominio**. Normalmente se utiliza para fijar un mínimo en uso garantizado del que no pueda bajar el dominio automáticamente.

```
# xm mem-set <dom_id> <mem>

# xm mem-set <dom_name> <mem>
```

- ***xm vcpu-list.*** **Muestra un listado de todas las *vcpus* disponibles** en el sistema (véase la figura 3.7) o en un dominio específico.

```
# xm vcpu-list <dom_id>

# xm vcpu-list <dom_name>

# xm vcpu-list
```

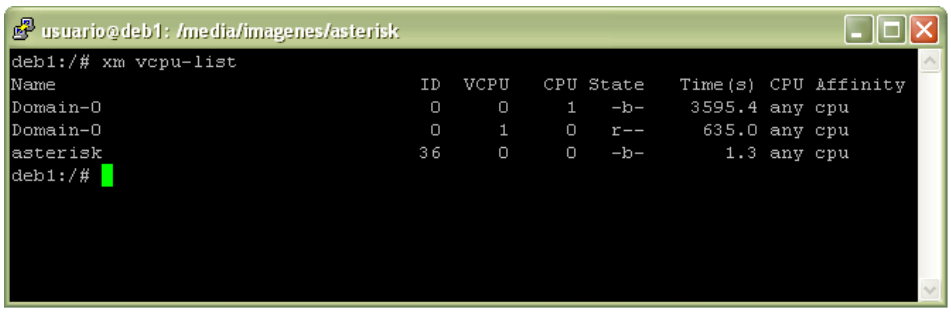


Figura 3.7 Listado de procesadores virtuales de todos los dominios con xm vcpu-list.

- ***xm vcpu-pin.*** **Permite determinar sobre qué CPUs físicas puede ser mapeada una CPU virtual (*vcpu*) del dominio**. Lo normal es que se utilice para restringir el

acceso de las *vcpus* sólo a las CPUs físicas que necesitemos y así delimitar su carga. La palabra clave *all* puede ser usada para aplicar la lista de CPUs a *todas las vcpus.*

```
# xm vcpu-pin <dom_id> vcpu cpus

# xm vcpu-pin <dom_name> vcpu cpus
```

- ***xm vcpu-set.*** **Permite establecer un número de CPUs virtuales a utilizar en el dominio**. Como máximo pueden ser establecidas las indicadas al inicio del dominio en su archivo de configuración o como variable de *xm create.*

```
# xm vcpu-set <dom_id> vcpu-count

# xm vcpu-set <dom_name> vcpu-count
```

Tener múltiples *vcpus* pertenecientes al mismo dominio mapeadas en la misma CPU física es muy probable que nos lleve a experimentar problemas en el rendimiento. Es mejor utilizar en este caso *vcpu-set* para separar el ámbito de acción de las *vcpus* y así asegurarnos de que cada una es ejecutada en una CPU física diferente.

En el caso de realizar operaciones intensas de entrada/salida es mejor dedicar determinados *cores* o CPUs para ejecutar *dom0*, eliminando la posibilidad de que otros dominios puedan hacer uso de ella. Si por ejemplo el dominio necesita procesar muchos datos lo mejor es permitir que acceda a todos los *hyperthreads* de las CPUs físicas.

3.2.3 Comandos de monitorización básicos

De los subcomandos existentes en la interfaz *xm* para monitorizar los dominios de nuestro sistema Xen **comentamos a continuación solamente *uptime***, ya que el resto son estudiados en profundidad en la sección *5. Monitorización de dominios* debido a su importancia: ***xm list* para el listado de dominios con información sobre el consumo actual de recursos, *info, log,* y *dmesg* para la obtención de información acerca del dominio *dom0* y *top* para la monitorización en tiempo real detallada de la totalidad de los dominios de la infraestructura**.

- ***xm uptime.*** Con él **mostramos en pantalla para todos los dominios** (nombre e identificador) **disponibles en el sistema su *uptime* o tiempo que ha transcurrido desde que fueron iniciados por última vez** (véase la figura 3.8).

```
# xm uptime
```

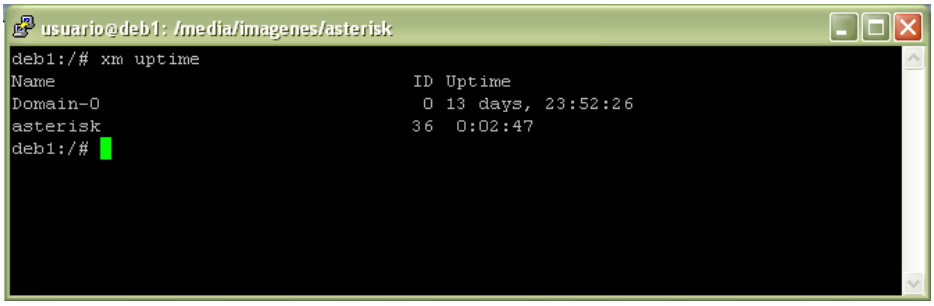


Figura 3.8 Tiempo que ha transcurrido desde que fueron iniciados los dominios con xm uptime.

3.1 TOPOLOGIAS DE REDES VIRTUALES EN XEN

A continuación veremos las principales **topologías de red virtuales estándar que pueden ser establecidas en nuestro sistema Xen: *bridge o puente, route o encaminador* y *NAT***. A partir de ellas es posible si lo deseamos implementar configuraciones más complejas, como por ejemplo una con dos *puentes*, *vlans*, o las llamadas *virtual networks* en las que los dominios *domU* comparten una red virtual con *dom0*.

De manera general **la configuración de todas ellas es similar habilitando los correspondientes scripts para la configuración de las conexiones de *dom0* y *domUs* en el archivo de configuración de *xend*** (*xend-config.sxp*). Todos estos scripts se encuentran en el directorio */etc/xen/scripts*. En lo que respecta a los *domUs* siempre es posible sin embargo sobrescribir la configuración de sus adaptadores de red mediante las opciones recogidas en el archivo de configuración del propio dominio.

Xen crea por defecto ocho *pares de interfaces Ethernet virtuales conectadas* para el uso por parte de *dom0* (véase la figura 3.9). Cada par es visto como dos interfaces Ethernet conectadas por un cable Ethernet interno: *veth0* conectada a *vif0.0*, *veth1* conectada a *vif0.1*… y así hasta *veth7* conectada a *vif0.7*. Cada par puede ser configurado dando a *veth#* direcciones IP y MAC, y luego conectando *vif0.#* a un *puente*.

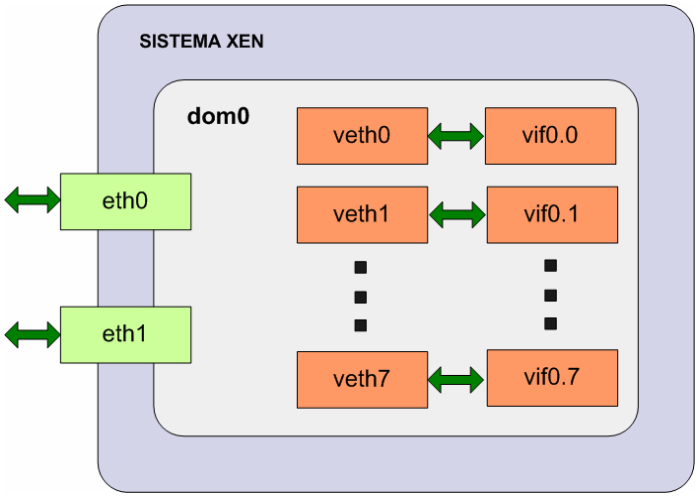


Figura 3.9 Pares Ethernet virtuales creados por defecto en Xen.

Cada adaptador de red virtual es nombrado de la forma *vifd.n* donde *d* es el identificador del dominio y *n* el del adaptador. **Cada vez que un nuevo dominio *domU* es iniciado Xen le asigna un nuevo identificador de dominio, y además crea un nuevo *par de interfaces Ethernet virtuales conectadas*, con un *extremo* en el *domU* y otro en el *dom0*. Cada *domU Linux* ve el dispositivo como *eth0*** (si dispone de más adaptadores, *eth1, eth2…*), **mientras que el *extremo* en *dom0* toma la nomenclatura *vif<id#>.0*** (*vif<id#>.1, vif<id#>.2,…* si hay más).

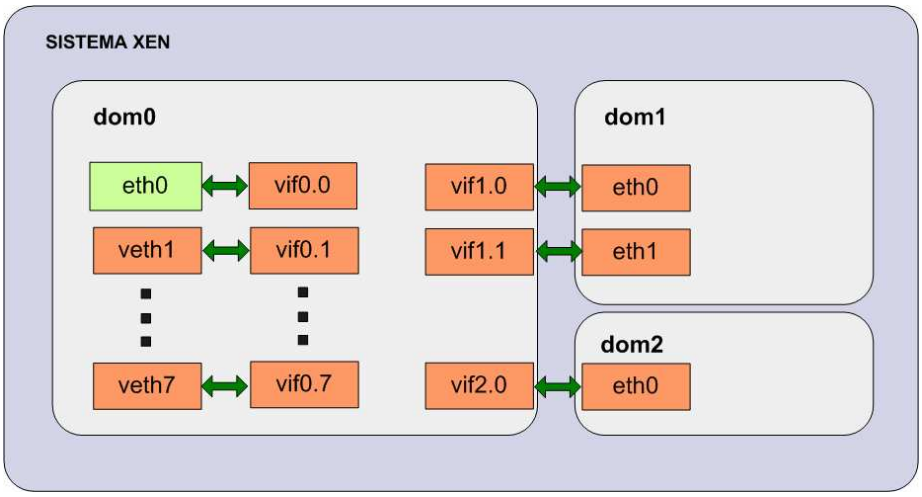


Figura 3.10 Pares de interfaces Ethernet conectadas para los dominios domU.

Cuando el dominio *domU* es apagado todas su interfaces virtuales Ethernet son eliminadas por Xen.

Las interfaces de red virtuales en los *dominios Xen* también disponen de direcciones MAC Ethernet: por defecto Xen selecciona una dirección aleatoria para ella, que es distinta para cada instancia de dominio. Sin embargo si queremos establecer una dirección MAC fija para un adaptador de un dominio lo podemos hacer mediante la variable *mac* en la opción *vif* del archivo de configuración de dominio (véase el apartado siguiente *4.1 Ficheros de configuración de dominios*). El rango de direcciones MAC dentro de 00:16:3e:xx:xx:xx es reservado para el uso por Xen por lo que es recomendado su uso. Aún así son direcciones válidas las que tienen el formato XY:XX:XX:XX:XX:XX con Y = {2, 6, A o E} y X = {dígito hexadecimal} , asegurando que se trata de una dirección *unicast* y que mantenemos un rango de direcciones *localmente asignado.*

Los **tipos de interfaces virtuales disponibles en Xen son**:

- ***Bridge o puente***. La conexión se realiza mediante un *puente* haciendo creer al dominio que se encuentra conectado directamente a la red física.

- ***Route o encaminador***. En este caso el dominio privilegiado *dom0* actúa como *router* y encamina todo el tráfico de red para el resto de dominios *domU*.

- ***NAT***. Igual que el tipo anterior, con la diferencia que en esta ocasión el dominio *dom0* además de enrutar el tráfico de red realizará funciones *NAT*, compartiendo su dirección de red con el resto de dominios.

A continuación veremos en qué consiste cada uno de ellos y aprenderemos como fácilmente configurar una interfaz de red virtual en un dominio.

3.1.1 Bridge o puente

En este tipo de configuración, la usada por defecto por Xen, **los dominios serán conectados a la red mediante lo que se llama una conexión *bridged***, o lo que es lo mismo, **con sus adaptadores virtuales creyendo que son conectados de forma directa a la red física través de un *puente o bridge*, por lo que los dominios aparecer en la red como equipos individuales** (véase la figura 3.11).

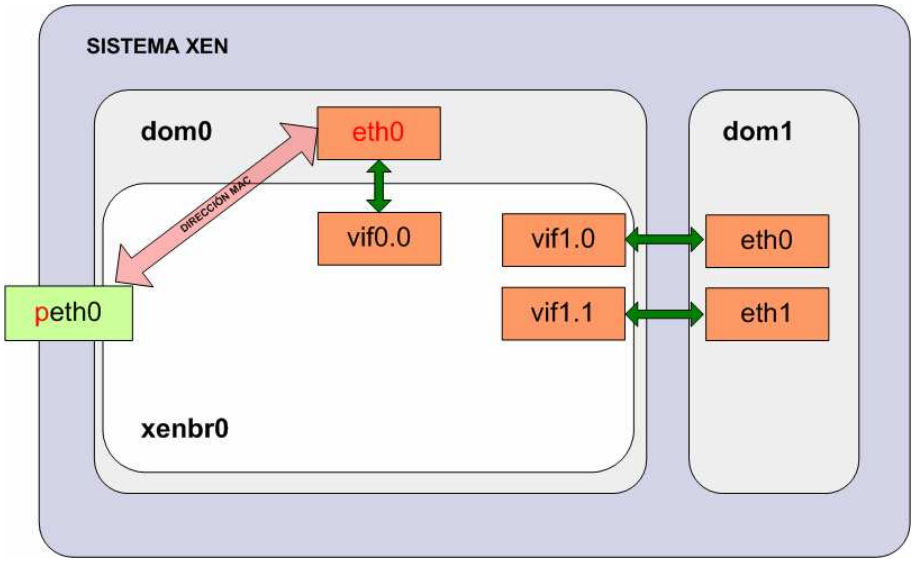


Figura 3.11 Configuración tipo Bridge o puente.

Los paquetes llegan al hardware, son manejados por el driver Ethernet de *dom0* y aparecen en *peth0*, el cual los pasa al *puente xenbr0* (todo ello a nivel Ethernet). Entonces es el *puente* el encargado de distribuirlos como si se tratara de un *switch* entre las diferentes interfaces virtuales *vifX.Y* dependiendo de la dirección MAC del receptor. La interfaz virtual *vif* pasa el paquete a Xen, el cual devuelve el paquete al dominio correspondiente (de igual forma ocurre para *dom0* que tiene enlazada *vif0.0* a *eth0*). Finalmente el dominio destino dispone de dirección IP, por lo que es posible implementar un filtrado aquí.

Para utilizar la configuración tipo *bridge* para la red virtual se descomentarán las siguientes dos líneas en el fichero de configuración *xend-config.sxp*:

```
# Red de la infraestructura virtual y dom0
(network-script network-bridge)

# Red de los domUs
(vif-script vif-bridge)
```

Con ello indicamos a *xend* que **el script *network-bridge* es el encargado de la configuración de red tipo *puente* para la infraestructura virtual**, **lo que conlleva también la configuración de red de *dom0***, e indicamos que **el script usado para la configuración de las interfaces virtuales también tipo *puente* de los *domU* es *vif-bridge***.

Resulta útil separar la interfaz física de la del dominio *dom0*, ya que de esta manera se podrá establecer un filtrado en *dom0* que no afecte al tráfico del resto de dominios. Como podemos ver en este tipo de configuración *dom0* es un dominio como otro cualquiera, con su interfaz de red virtual conectada al *puente*; **el resto de dominios disponen de la misma configuración de red y pertenecen al mismo segmento de red que *dom0*. El comportamiento general es visto como varios equipos reales conectados a un *switch* con la interfaz *peth0* como la conexión de salida del mismo**. El uso de la configuración *bridged* es útil por ejemplo cuando nos encontramos en una red local y tenemos a nuestra disposición cuantas direcciones IP queramos dentro del mismo rango.

3.1.2 Route o encaminador

Este tipo de configuración provoca que ***dom0* se comporte como enrutador y encamine todo el tráfico del resto de dominios** (véase la figura 3.12). Así, **hay que configurar en los dominios invitados *domU* la dirección IP de *dom0* como puerta de enlace predeterminada**. Crea un enlace punto a punto entre *dom0* y cada uno de los dominios *domU;* añade las rutas a cada uno de los dominios en la tabla de rutas de *dom0*, por lo que cada *domU* debe tener una dirección de red estática. DHCP por tanto no funcionará en este escenario ya que la ruta no es creada y por lo tanto los mensajes DHCP no llegan a los dominios.

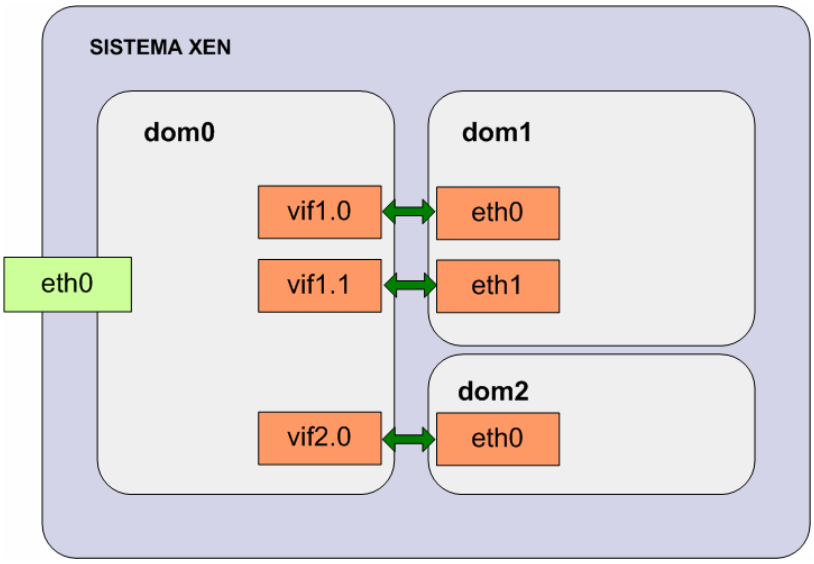


Figura 3.12 Configuración tipo Route o encaminador y NAT.

Los scripts utilizados para la configuración de los dominios son *network-route* y *vif-route* en sustitución de *network-bridge* y *vif-bridge* respectivamente, por lo que descomentaremos las siguientes dos líneas en el fichero de configuración *xend-config.sxp*:

```
# Red de la infraestructura virtual y dom0
(network-script network-route)

# Red de los domUs
(vif-script vif-route)
```

Cuando los paquetes del tráfico de los dominios viajan hacia el exterior a través de *eth0* tienen la misma dirección MAC para cualquiera que sea la dirección IP. Esto se puede cambiar **habilitando el comportamiento *proxy ARP*** *(Address Resolution Protocol)* **para el router en**

el script *network-route*, donde podemos realizar toda la configuración avanzada que estimemos conveniente. Cuando disponemos de varias direcciones IP dentro del mismo rango optamos bien por utilizar una configuración *bridge* o bien una configuración *route*, dependiendo de si el equipo que estamos conectando acepta o no diferentes direcciones MAC para las diferentes IPs. En el primer caso operamos a nivel de enlace de la pila TCP/IP y en el segundo a nivel de red.

3.1.3 NAT

Mediante este tercer tipo de configuración **es posible incluir en *dom0* la funcionalidad *NAT (Network Address Translation)* por la que actuando como enrutador podrá realizar el mapeado de las direcciones privadas de los dominios en su dirección pública y viceversa en los campos origen y destino de los paquetes cuando éstos se comuniquen con el exterior**. La figura que muestra este tipo de configuración es la misma que para el tipo *route*, 3.12. La configuración de este tipo de escenario sigue las mismas pautas que los anteriores; haremos uso de los scripts facilitados para ello. Así ahora **descomentamos las líneas que contienen la especificación de los scripts *network-nat* y *vif-nat***:

```
# Red de la infraestructura virtual y dom0
(network-script network-nat)

# Red de los domUs
(vif-script vif-nat)
```

De esta forma haremos que ***dom0* se encuentra conectado a dos subredes diferentes**: la primera típicamente pública en *eth0* y la segunda con direcciones privadas en las interfaces virtuales del tipo *vif<id#>.n*. Ahora **las direcciones de red de los dominios *domU* no se encuentran en el mismo rango que la dirección *externa* de *dom0***.

Para obtener **más información** de las configuraciones de red es recomendable la lectura de los manuales de usuario y desarrollador de Xen así como del siguiente enlace dentro de la *wiki* del grupo *XenSource*: http://wiki.xensource.com/xenwiki/XenNetworking.

4 Definición y creación de dominios domU

A continuación ilustraremos el proceso tanto de definición como de creación de dominios *domUs* o máquinas virtuales. **La definición de los dominios se realiza mediante archivos de configuración con extensión *.cfg***, cuya sintaxis estudiaremos; **la creación puede ser implementada siguiendo diferentes metodologías** dependiendo, por ejemplo, del tipo de sistema operativo o distribución Linux que se instalará en el dominio. Prácticamente la totalidad del contenido del apartado es enfocado a *dominios paravirtualizados*; sólo en el apartado *4.2.3 Hardware Virtual Machines* estudiaremos como caso especial *dominios completamente virtualizados.*

4.1 FICHEROS DE CONFIGURACION

La interfaz administrativa *xm* utiliza ficheros de configuración python para la definición de los dominios. Cada uno de estos ficheros de configuración dispone de **diferentes opciones configurables** para el dominio, algunas de ellas son totalmente necesarias otras en cambio son consideradas adicionales.

Por defecto los ficheros de configuración de dominios, cuya extensión es *.cfg*, son tomados por el comando *xm create* del directorio */etc/xen*; en el caso de encontrarse en

cualquier otra ubicación deberemos especificar la ruta total al archivo. **En el caso de que queramos iniciar los dominios en el arranque del sistema podemos almacenarlos en el directorio *./etc/xen/auto***; el script ***/etc/init.d/xendomains*** buscará en este directorio archivos de configuración de dominios a iniciar/detener. Por tanto este script debe ser instalado (normalmente esto se hace al mismo tiempo que *xend*), y para habilitarlo en arranque podemos ejecutar:

```
deb1:/ update-rc.d xendomains defaults 21 20
```

El script */etc/init.d/xendomains* necesita del archivo de configuración ***/etc/default/xendomains***; en este fichero podemos cambiar el directorio que contiene los archivos de configuración a arrancar/detener automáticamente editando la línea con el parámetro ***XENDOMAINS_AUTO***:

```
XENDOMAINS_AUTO=/etc/xen/auto
```

Todas las opciones que son configuradas en este fichero se especifican mediante la sintaxis *nombre_opcion* = *valor* como vamos a ver a continuación. Si un parámetro no se incluye en el fichero de configuración toma el valor por defecto asignado por Xen. **Las opciones o parámetros más habituales para la construcción de dominios Xen son las siguientes**:

- ***kernel.*** El valor escrito es la **ruta completa a la imagen del *kernel* que utilizará *dom0* para arrancar el dominio**. Lo habitual es que el *kernel* se encuentre copiado en el directorio ***/boot*** del dominio *dom0.* Por ejemplo:

  ```
  kernel = "/boot/vmlinuz-2.6.26-2-xen-amd64"
  ```

- ***ramdisk.*** Permite indicar **la ruta completa para la localización del disco RAM de inicio para el dominio** en el caso de que lo necesitemos (en muchas distribuciones Linux no es necesario si usamos un *kernel por defecto*). Estos discos son usados en Linux para la carga de módulos durante la puesta en marcha del sistema operativo.

  ```
  ramdisk = "/boot/initrd.img-2.6.26-2-xen-amd64"
  ```

- ***memory.*** La opción *memory* es una de las más importantes ya que **nos permite asignar la cantidad de memoria en megabytes para el dominio**. Es muy importante asignar suficiente memoria al dominio ya que de lo contrario lo más probable es que no pueda iniciarse; también hay que tener presente la cantidad total de memoria asignada en total a los dominios (incluyendo *dom0*) que nunca debe ser igual o superior a la disponible físicamente en el servidor por lo que es recomendable dar cierto margen en este sentido al servidor.

  ```
  memory = 2048
  ```

- ***name.*** **Nombre único** que queremos asignar al dominio. Será usado por los comandos administrativos y la totalidad de las aplicaciones de gestión para identificar al dominio. Como podemos imaginar, no es posible crear dos dominios con el mismo nombre. Por ejemplo:

```
name = "asterisk"
```

- ***root.*** **Sirve para indicar al dominio el dispositivo que contendrá el sistema de ficheros raíz** por lo que debe estar incluido en el archivo *fstab* de configuración del sistema de ficheros. Equivale al parámetro *root* que puede ser facilitado a un *kernel Linux.*

```
root = "/dev/sda1 ro"
```

- ***nics.*** **Número de adaptadores de red** a asignar al dominio. Si no es especificado su valor por defecto es 1.

```
nics = 2
```

- ***disk.*** Es una **lista de los *dispositivos de bloques virtuales*** (*VBDs, virtual block devices*) **que serán exportados al dominio**. De esta manera indicaremos al dominio cuáles serán sus discos virtuales (imagen del sistema de ficheros, partición para intercambio…) y su ubicación real. Los dispositivos son listados en un array de la siguiente forma:

```
disk = [‘dispositivo1’, ‘dispositivo2’, ‘dispositivo3’,... ]
```

Para especificar un dispositivo es necesario indicar tres términos:

```
‘backend-dev,frontend-dev,modo’
```

  - ***backend-dev***. Es el **dispositivo del lado del *dominio servidor – dom0*** que será exportado al *dominio cliente (invitado) – domU.* **Puede ser especificado siguiendo dos sintaxis diferentes**:
    - ***phy:dispositivo.*** En lugar de dispositivo escribiremos el dispositivo físico que queramos exportar al dominio. Esto lo podemos hacer de forma simbólica, por ejemplo *sda1*, o bien escribiendo su representación hexadecimal, por ejemplo 0x301 para *hda1*.
    - ***file://path/to/file.*** Permite exportar el fichero especificado como dispositivo *loopback.*
  - ***frontend-dev*****.** Permite inficar **cómo aparece el dispositivo en el dominio al exportarlo**. Como dijimos anteriormente, puede ser citado en forma simbólica o hexadecimal.
  - ***modo.*** **Modo de acceso al dispositivo por parte del dominio**: ***r*** para sólo lectura y ***w*** para lectura/escritura. También podemos escribir ***w!*** para los casos en los que exportemos discos como lectura/escritura y actualmente estén montados.

Un **ejemplo** de uso de la opción *disk* es:

```
disk =
['file:/media/imagenes/asterisk_lenny/imagen,sda1,w','file:/m
edia/imagenes/asterisk_lenny/swap,sda2,w']
```

- ***vif.*** *Virtual Ethernet Interface (interfaz virtual Ethernet).* Esta opción será la usada para **establecer la configuración de los adaptadores de red del dominio**, creados mediante un script. Por defecto contiene una cadena vacía por cada una de las interfaces de red, por lo que sólo tendremos que escribir los valores que deseamos cambiar para cada una de ellas –el resto serán asignados por *xend*-:

```
 vif = [‘adaptador1’, ‘adaptador2’,... ]
```

Los diferentes atributos que son configurables para cada adaptador son los siguientes (en la forma *atributo=valor*): *mac, type, bridge, ip, script, backend, vifname, rate, model y accel*. Aún así los que habitualmente son especificados aquí son ***mac*** y ***bridge.*** En el siguiente ejemplo asignamos al primer adaptador de red una dirección MAC y un *puente* (*bridge*) al que conectarla:

```
vif = ['mac=1A:1B:1C:2A:2B:2C, bridge=xen-br0']
```

En el caso de no especificar dirección MAC Xen asigna al dominio una aleatoriamente. En cuanto al *puente*, si no es especificado, Xen conecta el adaptador al primero que encuentre. Cuando no se especifica un **nombre para la interfaz de red** Xen asigna uno de la forma *vifd.n* donde *d* es el identificador del dominio y *n* el del adaptador.

- ***dhcp.*** Si queremos **obtener la configuración de red a través de DHCP** entonces escribimos *yes* como valor de esta opción. Y si queremos establecer la dirección IP de forma manual escribimos *no*.

```
dhcp = "no"
```

- ***ip.*** La **dirección IP** puede ser incluida bien como parte de la opción *vif* vista anteriormente o bien como una opción separada; la primera forma la utlizamos cuando disponemos de más de un adaptador de red en el dominio.

```
ip = '192.168.2.10’
```

- ***netmask.*** Permite **fijar la máscara de subred de la dirección IP del adaptador de red** del dominio. Al igual que la opción *ip* puede ser especificada o no junto a la opción *vif* dependiendo del número de interfaces de red disponibles en el dominio.

```
netmask = '255.255.255.0'
```

- ***gateway.*** Permite **establecer la puerta de enlace predeterminada** para la salida de la conexión de red del dominio. Esta opción es usada de la misma forma que las dos anteriores, *ip* y *netmask.*

```
gateway = ‘192.168.2.1’
```

- ***cpu.*** Permite especificar el **número de procesador en el que se iniciará el dominio**, comenzando por 0 y hasta el número de procesadores disponibles menos 1. Por defecto contiene el valor -1, lo que significará que Xen será libre para tomar cualquiera de los procesadores disponibles.

```
cpu = 0
```

- ***cpus.*** Permite establecer un **listado de los procesadores donde se puede ejecutar el dominio**. La sintaxis de la lista permite establecer rangos, procesadores específicos y también utilizar negaciones usando el carácter ^. Por ejemplo:

```
cpus = “0-3,5,^1”
```

  Esto quiere decir que el dominio se puede ejecutar en los procesadores 0, 2, 3 y 5.

- ***vcpus.*** Permite **establecer el número de procesadores virtuales que se asignan al dominio**. Para poder usar este parámetro el *kernel Xen* debe estar compilado con soporte SMP. Su valor por defecto es 1.

```
vcpus = 2
```

Las siguientes opciones nos permiten configurar el comportamiento del dominio ante determinados eventos, como por ejemplo el apagado o reinicio del mismo:

- ***on_shutdown.*** Es usada **para configurar el comportamiento en el caso de realizar un apagado desde dentro del propio dominio o desde *dom0* con el comando *xm shutdown.* Hay cuatro valores disponibles** para este parámetro:

  - ***destroy.*** *Limpia* por completo el estado del dominio y lo apaga tal y como lo debería hacer un equipo físico en condiciones normales.

  - ***restart.*** El dominio se reinicia. Es equivalente al reinicio de un equipo real.

  - ***preserve.*** El estado del dominio no es *limpiado* del todo. Es útil en condiciones de fallo en el dominio ya que permite recoger evidencias de los errores que se hayan producido.

  - ***rename-restart.*** No se limpia el estado del dominio al mismo tiempo que uno nuevo es creado con su misma configuración. El dominio antiguo es renombrado en base al nombre que poseía originalmente, ahora mantenido por el nuevo dominio, y libera los recursos que tiene asignados luego pueden ser capturados por el nuevo. Un nuevo identificador también se geneera para el dominio anterior ya que el nuevo posee el original.

  Por ejemplo, podemos elegir la opción *destroy*:

```
on_shutdown = destroy
```

- ***on_reboot.*** Es similar a la anterior pero en este caso **para la acción de reinicio**, **bien invocado por el propio dominio o en *dom0* con el comando *xm restart***. Dispone de los mismos cuatro valores que la opción *on_shutdown.*

```
on_reboot = restart
```

- ***on_crash.*** Se utiliza para **configurar la opción a llevar a cabo cuando el dominio pase a un estado inválido**. Para esta opción también es posible especificar los mismos cuatro valores que en los casos anteriores.

```
on_crash = restart
```

Para obtener información completa sobre la totalidad de las opciones disponibles en los archivos de configuración de dominios y los valores que aceptan, así como los valores que son asignados por Xen en el caso de no especificarlos podemos ejecutar el comando:

```
xm create --help_config
```

4.2 EJEMPLO

A continuación vemos un ejemplo de configuración de un dominio, *asterisk.cfg*:

```
kernel = "/boot/vmlinuz-2.6.26-2-xen-amd64"

ramdisk = "/boot/initrd.img-2.6.26-2-xen-amd64"

memory = 2048

name = "asterisk"

disk =
['file:/media/imagenes/asterisk_lenny/imagen,sda1,w','file:/media/im
agenes/asterisk_lenny/swap,sda2,w']

root = "/dev/sda1 ro"

dhcp = "no"

vif = ['mac=1A:1B:1C:2A:2B:2C']

netmask = '255.255.255.0'

gateway = '150.214.153.1'

ip = '150.214.153.233'

broadcast = '150.214.153.255'

on_poweroff = 'destroy'

on_reboot = 'restart'

on_crash = 'restart'

vcpus = 2

extra = 'console=hvc0 xencons=tty'
```

Aunque una configuración *por defecto* funciona sin problemas, es importante resaltar que la complejidad a la hora de editar nuestro fichero de configuración depende de los requisitos que establezcamos para el dominio, sobre todo en lo que se refiere a ***red, almacenamiento y sistemas de ficheros***, por lo que muchas veces es necesario conocer en mayor profundidad estos aspectos. Esto ocurre por ejemplo cuando queremos cambiar la topología de la red virtual (mediante el uso de los scripts *route* o *nat* en lugar del usado por defecto *bridge*, como vimos en el apartado anterior *3.3 Topologías de redes virtuales Xen*) o cuando necesitamos asignar algún dispositivo PCI al dominio *de manera exclusiva*, para lo cual habrá que hacer uso de una sintaxis extendida de algunas de las opciones ya vistas.

Cuando queremos definir *dominios completamente virtualizados*, esto es, *dominios HVM (Hardware Virtual Machines)* que haciendo uso de soporte de virtualización en el procesador pueden ejecutar sistemas operativos no modificados, **debemos presentar algunas modificaciones y diferentes opciones en el archivo de configuración del dominio aunque de manera general el proceso de configuración y puesta en marcha es análogo al usado para *dominios paravirtualizados***. Esto lo veremos en el apartado para la definición y creación de este tipo de dominios *4.2.3 Hardware Virtual Machines (HVM).*

4.3 CREACION DE DOMINIOS DOMU

A la hora de crear un nuevo *dominio Xen* es bastante útil realizar una serie de tareas dentro de una estrategia con el objetivo de hacerlo de la forma más eficiente posible, abarcando los diferentes aspectos esenciales del proceso. Estas tareas son:

1. **Elegir y proporcionar el medio de almacenamiento para el dominio**: ficheros, particiones, volúmenes lógicos,… Como veremos en los procedimientos que siguen en este apartado, hemos utilizado imágenes en ficheros.

2. **Instalar los archivos para el sistema operativo del dominio en el medio de almacenamiento**. La instalación se puede realizar siguiendo diferentes procedimientos: con herramientas que lo automatizan como *debootstrap*, con una imagen ISO, desde CD-ROM, o incluso simplemente un sistema de ficheros simple que contenga todos los archivos necesarios. En nuestro caso haremos uso fundamentalmente de la herramienta *debootstrap* y *Xen-Tools*, que hace uso también de *debootrap* y *rinse*.

3. **Crear el archivo de configuración que especifica los parámetros que utiliza Xen para crear y configurar el dominio**, tal y como hemos estudiado en el apartado anterior. Como veremos algunas herramientas como *Xen-Tools* pueden incluso crear este fichero de forma automática.

4. **Crear el dominio mediante la interfaz de administración *xm*** Utilizamos el comando *xm create* haciendo uso del archivo de configuración y el medio de almacenamiento configurado (con sistema operativo instalado) creados en los apartados anteriores.

Aún así también **es posible optar por la opción de obtener *kernels domU*, imágenes de disco o incluso dominios completos preconfigurados con la distribución GNU/Linux que queramos desde algunos sitios web en Internet**. El más importante de ellos es sin duda **http://stacklet.com/**. Es la web que ha sustituido a la que hasta hace bien poco era la más importante en esta cuestión, http://jailtime.org/. Podemos encontrar todo tipo de imágenes de sistemas operativos –CentOS, Fedora, Debian, Gentoo, Mandriva, Slackware, Ubuntu - no sólo para *Xen 3* sino también para otras soluciones como Qemu o VMDK, además de *kernels Xen* para nuestros sistemas operativos invitados. Todas las descargas de dominios incluyen tanto las

imágenes de disco correspondientes con el sistema operativo que queramos preinstalado como los archivos de configuración para poder iniciar rápidamente el dominio con el comando *xm create*.

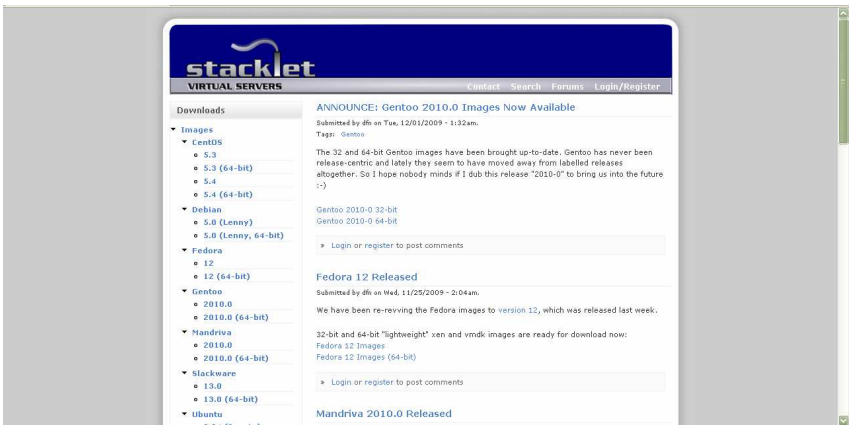


Figura 3.13 Página inicial http://stacklet.com/.

Otra web muy interesante pero que proporciona imágenes de sistemas operativos para *Qemu* es *oszoo* (http://www.oszoo.org/wiki/index.php/Main_Page). Cuando hagamos uso de dominios basándonos en ficheros preconfigurados descargados de sitios web debemos leer atentamente las instrucciones que proporcionan y las características del dominio que descargamos –arquitectura, versión del sistema operativo, cambios recientes que se han introducido... Lo más probable en estos casos es que tengamos que reconfigurar algunos aspectos del mismo relativos sobre todo a cuestiones de *red* y así adaptar el dominio a la funcionalidad que esperamos de él.

En los siguientes apartados mostraremos el proceso de creación de dominios sin hacer uso de imágenes preconfiguradas y sí de la estrategia presentada antes, ya que ello nos permite comprender mejor el proceso. Lo haremos creando el medio de almacenamiento a mano y con *Xen-Tools*, e instalando el sistema operativo con *debootrap* (sin y con *Xen-Tools*) y *rinse* (*Xen-Tools*). Finalmente crearemos un dominio *HVM (Hardware Virtual Machine)* presentando en qué difiere el proceso de creación en este caso del de los *dominios paravirtualizados*.

4.3.1 "A MANO" APOYADA POR DEBOOTSTRAP

En este primer método de creación de dominios lo hacemos creando las imágenes de disco usando comandos Linux clásicos y *debootstrap* para la instalación básica y rápida de una distribución basada en Debian GNU/Linux. **El fichero de configuración del dominio también es creado a mano** por nosotros mismos en lugar de utilizar una herramienta que nos permita hacerlo de forma automática, como veremos en apartados posteriores.

***Debootstrap* permite la instalación de distribuciones basadas en Debian** (el propio Debian o Ubuntu) **en sistemas de ficheros**. Si queremos instalar distribuciones de otro tipo como huésped (por ejemplo RPM, como Red Hat, CentOS o Fedora Core) hay otras utilidades disponibles como es el caso de *Rinse*, aunque con la limitación de que la creación del *domU* debe realizarse en un anfitrión (*dom0*) Debian.

Creación de la imagen del dominio

El primer paso para la creación de un dominio es la creación de una **imagen que actua como partición de instalación**; **utilizaremos**, por motivos de comodidad aunque en términos de rendimiento no sea óptimo, **un fichero que contiene la imagen del dominio**. También se puede utilizar una partición real para instalarlo o *volúmenes lógicos.*

Para ello **creamos un directorio** (por ejemplo */media/imagenesk*) **en el cual se almacenan todas las imágenes de los discos duros de los dominios**; como si se tratase de un repositorio. **Dentro de él creamos un directorio para cada uno de los dominios**; por ejemplo si nuestra primera máquina virtual se llama *asterisk1*:

```
deb1:/media# mkdir imagenes

deb1:/media# cd imagenes

deb1:/media/imagenes# mkdir asterisk1

deb1:/media/imagenes# ls
asterisk1
```

Dentro del directorio del dominio creamos, por ejemplo, **un fichero de 2GB para la imagen de disco del mismo, y otro de 128MB para *swap***. Si deseamos disponer de más discos para el dominio lo hacemos exactamente de la misma forma. Esto lo hacemos **haciendo uso del comando Linux *dd***:

```
deb1:/media/imagenes/asterisk1# dd if=/dev/zero of=imagen bs=1024k
count=2048
2048+0 records in
2048+0 records out
2147483648 bytes (2,1 GB) copied, 49,4488 s, 43,4 MB/s

deb1:/media/imagenes/asterisk1# dd if=/dev/zero of=swap bs=1024k
count=128
128+0 records in
128+0 records out
134217728 bytes (134 MB) copied, 2,99811 s, 44,8 MB/s
```

Donde *if=/dev/zero* se utiliza para que el fichero generado esté vacío, *of=imagen* permite indicar el fichero de salida, *bs=1024k* es el tamaño de bloque y *count* es el tamaño total del fichero expresado en megabytes. Si ejecutamos el comando *ls -la* vemos que se han generado correctamente los dos ficheros con el tamaño indicado:

```
deb1:/media/imagenes/asterisk# ls -la
total 2230416
drwxr-xr-x 2 root root       4096 ene 11 19:45 .
drwxr-xr-x 6 root root       4096 feb  9 17:44 ..
-rw-r--r-- 1 root root 2147483648 ene 11 19:46 imagen
-rw-r--r-- 1 root root  134217728 ene 11 19:47 swap
```

Ahora que disponemos de los ficheros que servirán de particiones de disco y swap a la máquina virtual les damos el formato necesario, en este caso *ext3* para el fichero que contendrá el sistema operativo y *swap* para el de intercambio, mediante el comando *mkfs*:

```
deb1:/media/imagenes/asterisk1# mkfs.ext3 imagen

deb1:/media/imagenes/asterisk1# mkswap swap
```

Instalación manual de Linux (Plantilla)

Es momento de realizar la instalación de una distribución (en este caso hemos elegido una básica de Debian) **en la imagen de disco que tenemos para el dominio**. Es ahora cuando **nos ayudamos de la utilidad *debootstrap*** para ello por lo que necesitamos instalarla:

```
deb1:/media/imagenes/asterisk1# aptitude install debootstrap
```

Si queremos conocer qué otras distribuciones es posible instalar de forma básica utilizando *debootstrap* podemos acceder a la información proporcionada por *man*. Como se comentó anteriormente, **vamos a instalar la distribución Debian GNU/Linux, más concretamente en su versión estable *etch*. Lo hacemos ejecutando *debootstrap* de la siguiente manera**, que como se puede observar es muy sencilla, ya que simplemente indicamos la arquitectura (i386), la distribución a instalar, la imagen donde se realizará la instalación, y el *mirror* del cual obtendremos el sistem. **Como paso previo es necesario montar la imagen de nuestro sistema de ficheros *ext3*, por ejemplo en el directorio */mnt/disk* que nos creamos para tal efecto**:

```
deb1:/media/imagenes/asterisk1# mkdir /mnt/disk

deb1:/media/imagenes/asterisk1# mount -o loop
/media/imagenes/asterisk1/imagen /mnt/disk

deb1:/media/imagenes/asterisk1# debootstrap --arch i386 etch
/mnt/disk http://ftp.rediris.es/debian
```

Configuración de la imagen del dominio

Con el sistema base instalado podemos establecer temporalmente la raíz de la imagen de disco del dominio como la raíz del sistema de ficheros usando el comando *chroot* sobre el directorio en el que tenemos montada la imagen, algo que puede llegar a ser **muy importante de cara a establecer una configuración común y así crear múltiples dominios ya que esta imagen podrá ser utilizada en la creación de otras máquinas virtuales a posteriori**.

```
deb1:/# chroot /mnt/disk /bin/bash

deb1:/# ls
bin  boot  dev  etc  home  initrd  lib  lost+found  media  mnt  opt
proc  root  sbin  srv  sys  tmp  usr  var
```

Situados en la raíz de la imagen podemos hacer todo tipo de instalaciones, configuraciones, etc.,… que consideremos necesarias y constituyan un dominio modelo que vayamos a replicar en el futuro. En primer lugar actualizamos las fuentes y las claves públicas de éstas:

```
deb1:/# aptitude update
```

Previo a la instalación del *kernel Xen* (como sabemos, y salvo que usemos virtualización por hardware en el procesador debemos modificar los sistemas operativos de los

dominios invitados) y **debido a que el dominio tiene una distribución y una arquitectura de procesador i386 debemos instalar el paquete *libc6-xen***, la librería compartida adecuada para trabajar con un *kernel Xen*:

```
deb1:/# aptitude install libc6-xen
```

El siguiente paso que debemos realizar es la instalación del *kernel Xen* para el dominio. **La imagen del *kernel* debe copiarse (junto con su disco RAM de inicio, en el caso de necesitarlo) en el directorio */boot* del sistema de ficheros del dominio *dom0*, porque Xen lo necesitará para cargarlo al arrancar el dominio *domU***. El contenido de este directorio antes de realizar la instalación y copia del *kernel* de la máquina virtual es:

```
deb1:/# exit

deb1:/# cd /boot

deb1:/boot# ls
config-2.6.26-2-amd64      initrd.img-2.6.26-2-amd64
System.map-2.6.26-2-amd64      vmlinuz-2.6.26-2-xen-amd64
config-2.6.26-2-xen-amd64  initrd.img-2.6.26-2-amd64.bak
System.map-2.6.26-2-xen-amd64  xen-3.2-1-amd64.gz
grub                       initrd.img-2.6.26-2-xen-amd64  vmlinuz-
2.6.26-2-amd64
```

Instalamos la imagen del *kernel Xen* en el sistema de ficheros del dominio que estamos configurando. Para que se comunique con el hipervisor instalamos el paquete *linux-xen-image* (concretamente su versión para i686, ya que es la arquitectura que hemos establecido en el dominio):

```
deb1:/# chroot /mnt/disk /bin/bash

deb1:/# aptitude install linux-image-xen-686

deb1:/# exit
```

Con el *kernel Xen* del *domU* instalado, en este caso ha sido la versión del *kernel Xen 2.6.18-6*, **copiamos los ficheros que han sido generados en el directorio */boot* del dominio en el directorio */boot* original del *dom0***, ya que no disponemos de ellos al ser un núcleo diferente y es necesario para poder arrancar la máquina virtual:

```
deb1:/mnt/disk# cp /mnt/disk/boot/vmlinuz-2.6.18-6-xen-686 /boot/ -R

deb1:/mnt/disk# cp /mnt/disk/boot/initrd.img-2.6.18-6-xen-686 /boot/
-R
```

Ya de vuelta en el sistema raíz original comprobamos el contenido del directorio ***/boot*** **que debe disponer de dos núcleos Xen diferentes**: *2.6.26-2-xen-amd64* para el *dom0* y *2.6.18-6-xen-686* para el *domU* que estamos configurando:

```
deb1:/mnt/disk# cd /boot/

deb1:/boot# ls
config-2.6.26-2-amd64          System.map-2.6.26-2-amd64
config-2.6.26-2-xen-amd64      System.map-2.6.26-2-xen-amd64
grub                           vmlinuz-2.6.18-6-xen-amd64
initrd.img-2.6.18-6-xen-amd64  vmlinuz-2.6.26-2-amd64
```

```
initrd.img-2.6.26-2-amd64       vmlinuz-2.6.26-2-xen-amd64
initrd.img-2.6.26-2-amd64.bak   xen-3.2-1-amd64.gz
initrd.img-2.6.26-2-xen-amd64
```

Con el *kernel* de la máquina virtual también instalado **podemos continuar con la configuración de la imagen a nuestro gusto accediendo de nuevo a ella mediante el comando *chroot***. Por ejemplo, instalamos el paquete ***ssh*** para poder acceder de forma remota al dominio:

```
deb1:/# aptitude install ssh
```

También instalamos el paquete ***ifrename*** ya que es de gran utilidad, **ya que si no lo hacemos cada vez que reiniciemos el dominio cambiará su dispositivo de red** (primero eth0, luego eth1, etc). *ifrename* nos permite para una dirección MAC concreta, renombrar su dispositivo de red.

```
deb1:/# aptitude install ifrename
```

Continuamos con la personalización del dominio por ejemplo editando el fichero ***/etc/iftab* para definición de las interfaces de red de que dispone**. Lo creamos si no existe con una línea para indicar que la interfaz con esa dirección MAC (la asignada a la interfaz de red virtual) será el dispositivo *eth0*:

```
deb1:/# vi /etc/iftab
```

```
eth0 mac 0A:0B:0C:1A:1B:1C
```

También si lo deseamos creamos las **entradas básicas en el fichero */etc/fstab* para definición del sistema de ficheros**. La imagen del sistema de ficheros la vamos a montar en lo que para el sistema virtualizado será */dev/sda1*, y la partición de intercambio *swap* en */dev/sda2*:

```
deb1:/# vi /etc/fstab
```

```
/dev/sda1 /     ext3  errors=remount-ro       0     1
/dev/sda2 none  swap  sw                      0     0
proc            /proc proc  defaults          0     0
```

Como último paso de configuración podemos editar también la configuración de red en el fichero ***/etc/network/interfaces***. En este caso por simplicidad estableceremos que el interfaz *eth0* del dominio sea configurado por el servidor DHCP de nuestra red:

```
deb1:/# vi /etc/network/interfaces
```

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
```

Cuando consideremos que disponemos de la imagen del dominio completamente configurada según nuestras necesidades, salimos de ella volviendo al sistema de ficheros raíz original del dominio *dom0* y la desmontamos del directorio */mnt/disk*.

```
deb1:/# exit

deb1:/boot# umount /mnt/disk/
```

Como vimos en el apartado *6.1 Fichero de configuración de dominios* es posible realizar muchas de estas modificaciones, como la definición de las interfaces de red del dominio, el sistema de ficheros o el nombre del sistema en el fichero */etc/hostname*, desde el propio fichero de configuración del dominio *domU*, evitando tener que estar abriendo y editando ficheros de configuración en las máquinas virtuales continuamente. Posteriormente en esta memoria tendremos acceso a ejemplos de ficheros de configuración de dominios –más concretamente cuando realicemos pruebas de rendimiento sobre dominios- en los que se han caracterizado estos aspectos de esta forma, ya que el objetivo en este momento es crear máquinas virtuales sencillas.

Alcanzado este punto sólo resta crear el fichero de configuración del dominio *asterisk1* que utilizará Xen y el dominio *dom0* para su puesta en marcha.

Creación del fichero de configuración del dominio

A continuación se incluye un **ejemplo de fichero de configuración que puede ser creado para el dominio *asterisk1*** con el que hemos estado trabajando hasta el momento; se trata de un archivo con extensión ***.cfg*** y que debe ser ubicado en el directorio ***/etc/xen***:

asterisk1.cfg:

```
kernel = "/boot/vmlinuz-2.6.18-6-xen-686"

ramdisk = "/boot/initrd.img-2.6.18-6-xen-686"

memory = 128

name = "asterisk1"

disk =
['file:/media/imagenes/asterisk1/imagen,sda1,w','file:/media/imagene
s/asterisk1/swap,sda2,w']

root = "/dev/sda1 ro"

vif = ['mac=0A:0B:0C:1A:1B:1C']
```

Analicemos un poco los parámetros que hemos incluido en este sencillo ejemplo:

- ***kernel.*** Indicamos la ruta completa a la imagen del *kernel* que se usa para arrancar el dominio, que como sabemos debe estar copiada en en el directorio */boot* del dominio *dom0*.

- ***ramdisk.*** Indicamos la ruta completa al disco RAM de inicio con los módulos que necesitamos usar en la puesta en marcha del dominio. Como en nuestro caso el sistema de ficheros es *ext3*, y el *kernel* de Debian lo trae compilado como módulo debemos hacer uso de él.

- ***memory.*** Especificamos la cantidad de memoria en megabytes asignada al dominio. En nuestro caso son 128Mb, sabiendo que el dominio *dom0* dispone de

196MB como mínimo y teniendo en mente el total de memoria disponible en el equipo para asignar a las máquinas virtuales.

- ***name.*** Especificamos con este parámetro el nombre que queremos darle al dominio. En nuestro ejemplo es *asterisk1*.

- ***disk.*** Este parámetro nos permite indicar los discos virtuales a los que tiene acceso el dominio, es decir, qué dispositivos de almacenamiento tiene el sistema virtualizado y su ubicación real. Como indicamos anteriormente en *fstab*, la imagen del sistema de ficheros se corresponde con el dispositivo *sda1* y la partición de intercambio *swap* con el dispositivo *sda2*. Ambas particiones han sido especificadas como ficheros y se le otorga al dominio acceso de lectura y escritura (carácter *w*).

- ***root.*** Es igual que el parámetro pasado a un *kernel Linux* común, indicando qué dispositivo del dominio contiene el sistema de ficheros raíz (debe ser un dispositivo incluido en el fichero *fstab*). En nuestro caso es el dispositivo */dev/sda1* con permisos de sólo lectura.

- ***vif.*** Este parámetro es muy importante ya que nos proporciona el medio de creación de dispositivos de red virtuales para el dominio. Podemos especificar cómo se conectará el interfaz, dirección IP, dirección física, etc. Por simplicidad (configuraciones más complejas son estudiadas más adelante) asignamos solamente la misma dirección MAC que incluimos antes en el fichero *iftab*, por ejemplo como ocurren en el caso en el que desconocemos cómo es la configuración de red posterior.

Como hemos podido ver se ha configurado un conjunto simple de parámetros con la idea de ilustrar de una manera rápida la **sencillez en la configuración de dominios con Xen**. Para ver el resto de parámetros que es posible añadir/configurar y algunas de las opciones más importantes podemos ver el contenido del apartado anterior *6.1 Fichero de configuración de dominios* o bien remitirnos al lector a la página *man* de *xmdomain.cfg* para tener acceso a toda la información disponible (véase la figura 3.14).

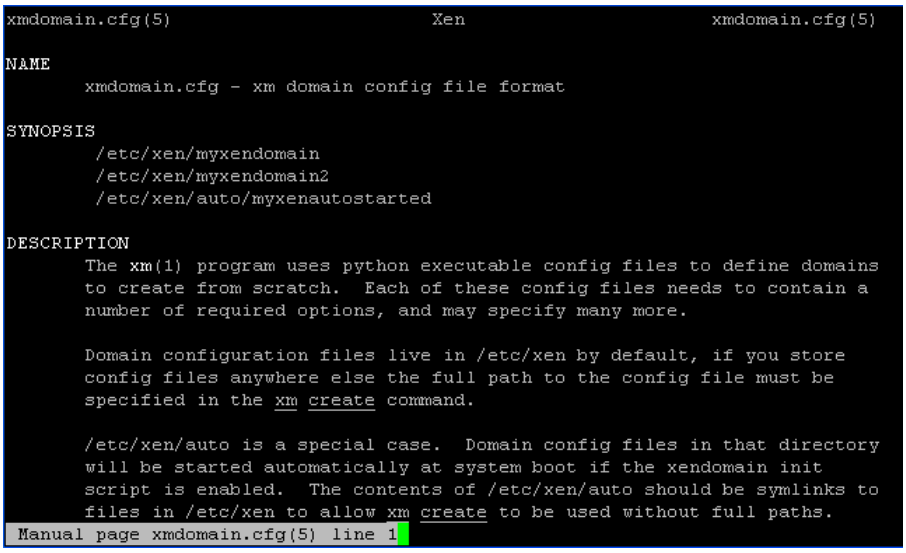


Figura 3.14 Inicio de la página man de xmdomain.cfg.

Recomendación final

Antes de poner en marcha el nuevo dominio que acabamos de configurar es totalmente recomendable copiar los ficheros */media/imagenes/asterisk1/imagen* y */media/imagenes/asterisk1/swap* **en otro directorio creado** (por ejemplo */media/imagenes/base*) **para almacenar estas imágenes como imágenes modelo o base para su uso posterior** en la creación de otros dominios basados en Debian *etch* y la configuración que hemos realizado:

```
deb1:/# cp /media/imagenes/asterisk1/imagen /media/imagenes/base/ -R

deb1:/# cp /media/imagenes/asterisk1/swap /media/imagenes/base/ -R

deb1:/media/imagenes/base# ls -la
total 4460824
drwxr-xr-x 2 root root       4096 ene 12 20:33 .
drwxr-xr-x 6 root root       4096 feb  9 17:44 ..
-rw-r--r-- 1 root root 2147483648 ene 11 21:03 imagen
-rw-r--r-- 1 root root 2147483648 ene 12 20:33 imagen_asterisk_lenny
-rw-r--r-- 1 root root  134217728 ene 11 21:03 swap
-rw-r--r-- 1 root root  134217728 ene 12 20:33 swap_asterisk_lenny
```

Arrancando la nueva máquina virtual

Para arrancar el nuevo dominio que acabamos de configurar, con la imagen del sistema de ficheros en */mnt/disk* ya desmontada, **ejecutamos el comando de creación de dominios *xm create* en el dominio privilegiado *dom0***. Como vimos anteriormente **podemos incluir el flag *–c* para acceder directamente a la consola de la máquina virtual**; si no lo hacemos ésta es puesta en marcha en modo *background.*

```
deb1:/# xm create asterisk1.cfg -c
```

```
deb1:/# xm create asterisk1.cfg
Using config file "/etc/xen/asterisk1.cfg".
Started domain asterisk1
```

Una vez que el dominio ha sido iniciado no debemos olvidar entrar en su consola si no usamos el flag *–c* al invocar su arranque (haciendo uso del comando ***xm console***) para **registrarnos como usuario *root* y cambiar su contraseña mediante el comando *passwd***. Desde este momento el dominio es accesible desde *dom0* y podemos comenzar a hacer uso de él como si un equipo independiente se tratara.

Creación de una segunda máquina virtual (utlizando la Plantilla)

A continuación vamos a **crear una segunda máquina virtual basándonos en la primera que acabamos de poner en marcha en las páginas anteriores**. Esto nos servirá de ejemplo sencillo para ilustrar algunas de las importantes mejoras y ventajas que puede llegar a

aportar la virtualización en procesos como la puesta en marcha de equipos, aprovisionamiento, o recuperación ante desastres.

Como vamos a ver el proceso va ser realmente **sencillo al contar con las imágenes de disco tanto del sistema de ficheros como de intercambio almacenadas previamente** en el directorio */media/imagenes/base*. Éstas pueden ser reutilizadas ya que queremos crear un nuevo dominio que dispondrá de la misma configuración que la realizada anteriormente.

```
deb1:/media/imagenes/base# ls -la
total 4460824
drwxr-xr-x 2 root root       4096 ene 12 20:33 .
drwxr-xr-x 6 root root       4096 feb  9 17:44 ..
-rw-r--r-- 1 root root 2147483648 ene 11 21:03 imagen
-rw-r--r-- 1 root root 2147483648 ene 12 20:33 imagen_asterisk_lenny
-rw-r--r-- 1 root root  134217728 ene 11 21:03 swap
-rw-r--r-- 1 root root  134217728 ene 12 20:33 swap_asterisk_lenny
```

Copiamos las imágenes base en un nuevo directorio */media/imagenes/asterisk2* creado para este segundo dominio que queremos crear:

```
deb1:/media/imagenes# mkdir asterisk2

deb1:/media/imagenes# cp /media/imagenes/base/imagen
/media/imagenes/asterisk2 -R

deb1:/media/imagenes# cp /media/imagenes/base/swap
/media/imagenes/asterisk2/ -R
```

Una vez copiados los ficheros de las imágenes podemos montar el disco para así realizar los cambios oportunos en los archivos de configuración para establecer valores que son personalizados para el nuevo dominio.

```
deb1:/media/imagenes/asterisk2# mount -o loop
/media/imagenes/asterisk2/imagen /mnt/disk/

deb1:/media/imagenes/asterisk2# chroot /mnt/disk/ /bin/bash

deb1:/# ls
bin  boot  dev  etc  home  initrd  lib  lost+found  media  mnt  opt
proc  root  sbin  srv  sys  tmp  usr  var
```

Debemos aclarar que **las modificaciones que llevamos a cabo no son necesarias si en cambio especificamos estos valores para el dominio como parámetros en su archivo de configuración**; como para el dominio anterior se hizo directamente en los archivos propios del dominio continuamos ahora así también –posteriormente se verán los archivos de configuración para crear dominios huéspedes con mayor complejidad. La primera modificación que realizaremos será para establecer el **nombre de nuestra nueva máquina**, *asterisk2*, en el fichero */etc/hostname*:

```
deb1:/etc# vi hostname
```

La segunda modificación será para indicar en el archivo */etc/iftab* la **dirección MAC de la interfaz de red virtual** de nuestra segunda máquina, en nuestro caso asignaremos 1A:1B:1C:2A:2B:2C.

```
deb1:/etc# vi iftab
```

```
eth0 mac 1A:1B:1C:2A:2B:2C
```

Guardados los cambios, desmontamos la imagen de disco que estábamos manipulando y copiamos nuestro fichero de configuración para nuestro dominio original *asterisk1.cfg* en otro cuyo nombre será *asterisk2.cfg* para el nuevo, editamos los nuevos valores que corresponden y lo guardamos en el mismo directorio */etc/xen/*:

```
deb1:/media/imagenes/asterisk2# umount /mnt/disk/
```

asterisk2.cfg:

```
kernel = "/boot/vmlinuz-2.6.18-6-xen-686"

ramdisk = "/boot/initrd.img-2.6.18-6-xen-686"

memory = 128

name = "asterisk2"

disk =
['file:/media/imagenes/asterisk2/imagen,sda1,w','file:/media/imagene
s/asterisk2/swap,sda2,w']

root = "/dev/sda1 ro"

vif = ['mac=1A:1B:1C:2A:2B:2C']
```

Una vez completados todos los pasos, algo que puede llevarnos tan sólo unos pocos minutos, **podemos arrancar esta segunda máquina virtual** de igual forma a como lo hicimos con la primera, **con el comando *xm create* en *dom0***. Después **podemos comprobar tanto con el comando *xm list* como con *xm top* (*xentop*) que ha sido creada satisfactoriamente**:

xm list:

```
deb1:/etc/xen# xm list
Name                                ID   Mem VCPUs      State   Time(s)
Domain-0                             0   128     1     r-----    121.5
asterisk1                            1   128     1     -b----      4.9
asterisk2                            2   128     1     -b----      4.6
```

xm top:

```
xentop - 17:21:26   Xen 3.2-1
3 domains: 1 running, 2 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 514492k total, 410184k used, 104308k free    CPUs: 1 @ 1800MHz
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%)
VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR SSID
 asterisk1 --b---          4    0.0     131072   25.5     131072      25.5
1    1        1       95    2        0      739      186   32
 asterisk2 --b---          4    0.0     131072   25.5     131072      25.5
1    1        1        5    2        0      701       58   32
  Domain-0 -----r        117    1.9     131072   25.5   no limit       n/a
1    4        0        0    0        0        0        0   32
```

Obviamente también podemos **acceder a la consola de la segunda máquina virtual** que se ha creado y puesto en marcha **haciendo uso de *xm console***, así como conectarnos

mediante ***ssh*** tanto a la primera como a *dom0*. La primera vez que accedemos **nos registramos como usuario *root* y cambiamos la contraseña con *passwd***:

```
deb1:/etc/xen# xm console asterisk2
XENBUS: Device with no driver: device/console/0
Freeing unused kernel memory: 148k freed
Loading, please wait...
Begin: Loading essential drivers... ...
Done.

.
.
.

asterisk2 login: root
Linux asterisk2 2.6.18-6-xen-686 #1 SMP Sat Dec 27 13:17:00 UTC 2008
i686

.
.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

asterisk2:~# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Establecemos una conexión *ssh* desde el segundo dominio (*asterisk2*) al primero creado previamente (*asterisk1*) sabiendo que éste tiene como dirección de red 192.168.20.102. Como podemos ver la conexión es realizada exactamente igual que si estuviéramos operando con equipos disponibles físicamente, no hay diferencia alguna:

```
asterisk2:~# ssh 192.168.20.102
The authenticity of host '192.168.20.102 (192.168.20.102)' can't be
established.
RSA key fingerprint is
f7:c5:a7:a3:4a:68:76:9d:6c:6f:a7:b2:4e:c0:d2:ca.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.20.102' (RSA) to the list of
known hosts.
root@192.168.20.102's password:
Last login: Tue May 26 15:25:58 2009
Linux asterisk1 2.6.18-6-xen-686 #1 SMP Sat Dec 27 13:17:00 UTC 2008
i686

The programs included with the Debian GNU/Linux system are free
software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
asterisk1:~#
```

4.3.2 USANDO LA HERRAMIENTA XEN-TOOLS

Como hemos podido experimentar en el apartado anterior hemos llegado a crear un dominio huésped en Xen construyendo “a mano” las imágenes de disco necesarias para el mismo; ahora, para agilizar el proceso haremos uso de un conjunto de scripts disponibles en la

herramienta *Xen-Tools*. ***Xen-Tools*** **simplificará bastante la parte inicial del proceso de creación de un dominio *domU* y ello nos permitirá lograr optimizar el tiempo que consume**. Seguiremos la misma línea de trabajo que en el caso anterior: primero crearemos un dominio para después crear otro reutilizándolo.

Como estamos comprobando una de las mayores ventajas que aporta el uso de **Xen** como solución de virtualización es que **utiliza dominios que son directamente sistemas de ficheros (en particiones o imágenes) y que por tanto pueden modificarse montándolos en un directorio de nuestro sistema de ficheros**. Esto nos permite en la mayoría de los casos como hemos hecho hasta ahora **crear una imagen con el método que consideremos oportuno y utilizar ésta como patrón para crear todos los nuevos dominios** que sean similares con sólo pequeñas modificaciones.

Xen-Tools (http://www.xen-tools.org/software/xen-tools/) **es una herramienta que incluye diversos scripts escritos en *perl* para facilitar y automatizar la creación de los dominios *domU* en anfitriones que disponen de Debian GNU/Linux como sistema operativo, creando de forma rápida las imágenes de los discos virtuales de los mismos y sus archivos de configuración *Xen* para el arranque con *dom0***. Como limitación tenemos que decir que sólo permite instalar algunas de las distribuciones GNU/Linux soportando instalaciones vía *deboostrap*, vía *rinse* o *rpmstrap*, o incluso haciendo uso de copias de imágenes previamente creadas:

```
#
#  Currently supported and tested distributions include:
#
# via Debootstrap:
#
#  Debian:
#   sid, sarge, etch, lenny.
#
#  Ubuntu:
#   edgy, feisty, dapper.
#
# via Rinse:
#   centos-4, centos-5.
#   fedora-core-4, fedora-core-5, fedora-core-6, fedora-core-7
#
#
```

Con *Xen-Tools* **es posible crear y arrancar nuevos dominios en cuestión de minutos** especificando de una manera sencilla ciertos parámetros que nos permiten configurar por ejemplo los adaptadores de red virtuales, el conjunto de particiones de las que dispondrá, etc. También permite por ejemplo instalar de forma automática *openSSH*.

Xen-Tools **por tanto tiene como dependencia *debootstrap***, que ya fue instalado anteriormente y que nos permite instalar distribuciones GNU/Linux basadas en Debian, como haremos también en esta ocasión. **Instalamos *Xen-Tools***, cuyo instalador reconoce e instala también a su vez de forma automática el conjunto de utilidades y librerías de las que depende para operar:

```
deb1:/# aptitude install xen-tools
```

Una vez instalada la herramienta podemos accedemos al fichero de configuración */etc/xen-tools/xen-tools.conf* en el cual se encuentran todos los parámetros configurables para la creación de las imágenes de disco para los dominios *domU*. Así, **la práctica común es incluir en este fichero todos los parámetros compartidos por los diferentes dominios** que

queramos crear para posteriormente al invocar la creación de los mismos añadir los parámetros específicos que restan.

El fichero *xen-tools.conf* se encuentra ampliamente comentado por lo que resulta bastante sencillo configurar estos parámetros. **En la tabla 3.2 vemos cómo podemos especificar algunos de los parámetros con un ejemplo sencillo para crear la imagen de un dominio** similar a los creados anteriormente.

Tabla 3-2. Parámetros más utilizados en el fichero de configuración *xen-tools.conf*

Nombre	Descripción	Ejemplo
dir	Permite indicar el directorio donde se crearán las imágenes de disco de los dominios, dentro de un directorio *domains*.	`dir = /media/imagenes/`
install-method	Establece el método de instalación de la distribución para el dominio: *debootstrap*, *rinse-rpmstrap…*	`install-method = debootstrap`
size	Permite especificar el tamaño de disco virtual para el dominio. No hay que olvidar el total de disco disponible en el sistema anfitrión.	`size = 1Gb`
memory	Permite indicar la cantidad de memoria de la que dispondrá el dominio. Hay que tener presente la cantidad de memoria físicamente instalada en el sistema anfitrión, qué cantidad necesita el dominio *dom0* como mínimo y cuánta memoria queremos reservar para otras máquinas virtuales.	`memory = 128Mb`
swap	Establece el tamaño de la partición de intercambio del dominio.	`swap = 128Mb`
fs	Indica el tipo de sistema de ficheros que queremos instalar en la imagen de disco del dominio (normalmente *ext3*).	`fs = ext3`
dist	Especifica la distribución Linux a instalar dependiendo del método de instalación especificado en *install.method*.	`dist = etch`
image	Podemos elegir si crear la imagen asignando el espacio total de disco desde el principio (tipo *full*) o bien hacerlo de manera que vaya creciendo en tamaño a medida que lo vaya requiriendo (tipo *sparse*).	`image = sparse`
dhcp	Si el dominio obtiene la información de red a través de DHCP (1). Si asignamos la dirección de red de forma manual escribimos 0 y usamos la opción *–ip* al lanzar *xen-create*-image y los parámetros *gateway*, *netmask y broadcast*.	`dhcp = 1`
kernel	Especifica la ruta completa al *kernel Xen* modificado que utiliza el dominio para arrancar (normalmente en */boot*).	`kernel = /boot/vmlinuz-2.6.18-6-xen-686`
initrd	Contiene la ruta a un disco RAM de inicio si necesitamos la carga de módulos en el inicio del dominio (normalmente en */boot*).	`initrd = /boot/initrd.img-2.6.18-6-xen-686`
mirror	Se utiliza para especificar el *mirror* del cual obtendremos la réplica de la distribución que vamos a instalar.	`mirror = http://ftp.us.debian.org/debian/`

Existen además otros parámetros que también pueden ser útiles por ejemplo para establecer la arquitectura a usar por el instalador (***arch***) o las opciones para los sistemas de ficheros:

```
ext3_options    = noatime,nodiratime,errors=remount-ro
ext2_options    = noatime,nodiratime,errors=remount-ro
xfs_options     = defaults
reiser_options = defaults
```

Una vez que hayamos configurado el fichero *xen-tools.conf* a nuestro gusto estamos en disposición de proceder con la creación de nuevos dominios.

La situación ideal y en la que los tiempos se reducen aún más en la creación de la imagen es cuando creamos un dominio con idéntica versión de sistema operativo que el *dom0*, ya que *debootstrap* copiará los paquetes del *dom0* al *domU* y configurará el nuevo dominio en lugar de utilizar el *mirror* remoto que le hayamos especificado. En este ejemplo *dom0* se encuentra ejecutando Debian *lenny* como sistema operativo y en *domU* vamos a instalar Debian *etch* vía *debootstrap*, luego no es nuestro caso.

Creamos entonces una primera imagen –que podrá ser reutilizada posteriormente para otros dominios- **basándonos en la configuración realizada de *Xen-Tools* con el comando *xen-create-image*, al cual indicamos por línea de comandos como parámetros los valores que consideramos particulares para el dominio que nos ocupa**, como son el nombre del equipo (***--hostname*** *asterisk3*) o la dirección física de la tarjeta de red (***--mac*** *2A:2B:2C:3A:3B:3C*):

```
deb1:/etc/xen-tools# xen-create-image --hostname asterisk3 --mac
2A:2B:2C:3A:3B:3C

General Information
--------------------
Hostname       :  asterisk3
Distribution   :  etch
Partitions     :  swap            128Mb (swap)
                  /               1Gb   (ext3)
Image type     :  sparse
Memory size    :  128Mb
Kernel path    :  /boot/vmlinuz-2.6.18-6-xen-686
Initrd path    :  /boot/initrd.img-2.6.18-6-xen-686

Networking Information
----------------------
IP Address     : DHCP [MAC: 2A:2B:2C:3A:3B:3C]


Creating partition image:
/media/imagenes//domains/asterisk3/swap.img
Done

Creating swap on /media/imagenes//domains/asterisk3/swap.img
Done

Creating partition image:
/media/imagenes//domains/asterisk3/disk.img
Done

Creating ext3 filesystem on
/media/imagenes//domains/asterisk3/disk.img
Done
Installation method: debootstrap
Done
```

```
Running hooks
Done

No role scripts were specified.  Skipping

Creating Xen configuration file
Done
All done


Logfile produced at:
    /var/log/xen-tools/asterisk3.log
```

El resto de parámetros que pueden ser facilitados al comando ***xen-create-image*** son básicamente los expuestos anteriormente para el fichero *xen-tools.conf*, y se pueden consultar accediendo a la página *man* del comando (véase la figura 3.15).

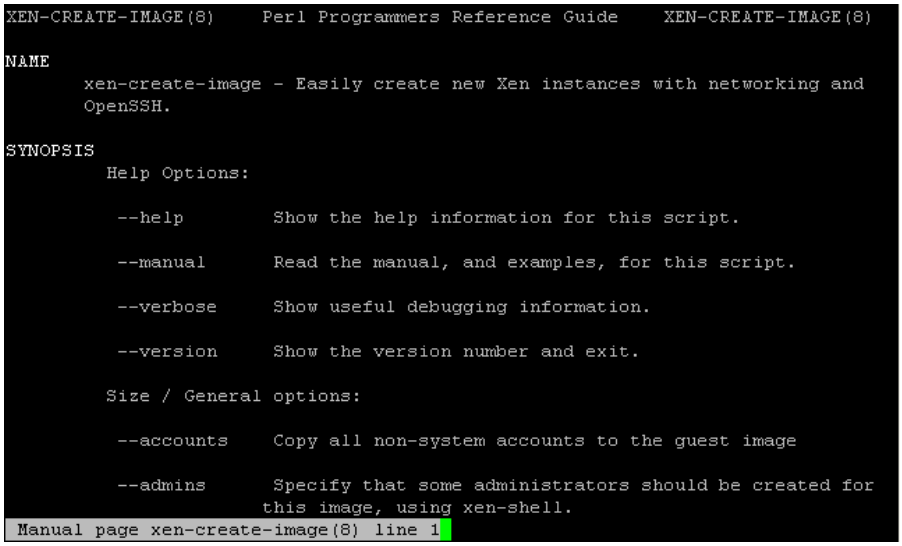


Figura 3.15 Inicio de la página man de xen-create-image.

En la salida del comando *xen-create-image* se pueden identificar las distintas operaciones o etapas de las que consta: muestra información general y de red acerca de la configuración, crea y formatea la partición *swap* de intercambio, crear y formatea también la partición de disco – sistema de ficheros (*ext3*), el sistema operativo se instala mediante *debootstrap*, se ejecutan *hooks* (técnicas de inserción de código en las llamadas del sistema para alterarlas) y **finalmente crea el archivo de configuración para el arranque del nuevo dominio con el nombre que hemos indicado** (*nombre_del_dominio.cfg*) **en el directorio */etc/xen*.**

Con el archivo de configuración y las imágenes de disco del dominio *asterisk3* estamos en disposición de iniciarlo tal y como lo hicimos en el apartado anterior -hasta este punto puede llegar a mejorar el rendimiento de todo el proceso gracias al uso de *Xen-Tools*. Antes de nada **guardamos las imágenes creadas** y que se encuentran en el directorio */media/imagenes/domains/asterisk3* **como modelo para futuros dominios similares** que queramos crear en el directorio */media/imágenes/base/xen-tools.*

```
deb1:/# cd /media/imagenes/domains/asterisk3/

deb1:/media/imagenes/domains/asterisk3# ls
disk.img  swap.img

deb1:/media/imagenes/domains/asterisk3# ls /etc/xen
asterisk1.cfg   asterisk2.cfg   asterisk3.cfg  xend-config.sxp
xend-config-xenapi.sxp   xend-pci-quirks.sxp
scripts        xend-pci-permissive.sxp

deb1:/media/imagenes/domains/asterisk3# mkdir
/media/imagenes/base/xen-tools

deb1:/media/imagenes/domains/asterisk3# cp
/media/imagenes/domains/asterisk3/disk.img /media/imagenes/base/xen-
tools/ -R

deb1:/media/imagenes/domains/asterisk3# cp
/media/imagenes/domains/asterisk3/swap.img /media/imagenes/base/xen-
tools/ -R
```

Antes de proceder a iniciar la máquina virtual, además de guardar las imágenes creadas, **debemos echar un vistazo al archivo de configuración creado por *Xen-Tools* para comprobar el trabajo realizado por la herramienta y que los valores de los parámetros son los deseados**.

asterisk3.cfg (generado por *Xen-Tools*):

```
#
# Configuration file for the Xen instance asterisk3, created
# by xen-tools 3.9 on Tue May 26 19:22:23 2009.
#

#
#  Kernel + memory size
#
kernel      = '/boot/vmlinuz-2.6.18-6-xen-686'
ramdisk     = '/boot/initrd.img-2.6.18-6-xen-686'
memory      = '128'

#
#  Disk device(s).
#
root        = '/dev/sda2 ro'
disk        = [

'file:/media/imagenes//domains/asterisk3/swap.img,sda1,w',

'file:/media/imagenes//domains/asterisk3/disk.img,sda2,w',
              ]


#
#  Hostname
#
name        = 'asterisk3'

#
#  Networking
#
dhcp        = 'dhcp'
vif         = [ 'mac=2A:2B:2C:3A:3B:3C' ]

#
#  Behaviour
#
```

```
on_poweroff = 'destroy'
on_reboot   = 'restart'
on_crash    = 'restart'
```

Es interesante ver cómo **los últimos tres parámetros incluidos** (*on_poweroff, on_reboot, on_crash*) **permiten establecer el comportamiento del dominio ante eventos determinados** (apagado, reinicio y estado inválido respectivamente).

Si queremos antes de iniciar la máquina virtual podemos **montar su imagen y establecer su sistema raíz como el actual para realizar cualquier tipo de configuración** tal y como hemos hecho en otras ocasiones. **Ponemos en marcha el nuevo dominio ejecutando el comando *xm create* con el archivo de configuración del mismo**:

```
deb1:/etc/xen# xm create asterisk3.cfg
Using config file "./asterisk3.cfg".
Started domain asterisk3
```

Accedemos a la consola del dominio con *xm console* y nos registramos como usuario *root* para cambiar la contraseña con el comando *passwd*:

```
deb1:/etc/xen# xm console asterisk3
Begin: Loading essential drivers... ...
Done.

.
.
.

asterisk3 login: root
Linux asterisk3 2.6.18-6-xen-686 #1 SMP Sat Dec 27 13:17:00 UTC 2008
i686

.
.
.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

asterisk3:~# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Podemos ahora por ejemplo actualizar los repositorios del sistema operativo ejecutando ***aptitude update*** para así comprobar su salida al exterior:

```
asterisk3:~# aptitude update
```

El dominio ha sido creado con éxito. Si ejecutamos cualquiera de las herramientas de monitorización o listado de dominios de Xen, por ejemplo ***xm list*, el nuevo dominio aparecerá como disponible**.

```
deb1:/# xm list
Name                              ID   Mem VCPUs      State   Time(s)
Domain-0                           0   128     1     r-----    713.1
asterisk3                          7   128     1     -b----      4.1
```

Creación de una segunda máquina virtual

La opción más sencilla pero al mismo tiempo más costosa para la creación de un nuevo dominio basado en otro con *Xen-Tools* es utilizando el mismo fichero de configuración de la herramienta *xen-tools.conf* y ejecutando de nuevo el comando *xen-create-image* con los parámetros que sean diferentes para el nuevo dominio (en nuestro ejemplo, --*hostname* y --*mac*). Al ejecutar de nuevo este comando las imágenes del dominio serán creadas de nuevo por lo que se consume más tiempo que reutilizando una imagen guardada anteriormente en la creación de otro dominio.

Otra opción mucho más útil y eficaz consiste en una vez creadas y guardadas las imágenes de un primer dominio con *Xen-Tools*, copiarlas en un nuevo directorio para el nuevo dominio y editar el contenido del fichero de configuración que fue generado también por *Xen-Tools* con los valores de los parámetros que queremos incluir. Con las nuevas imágenes copiadas y el nuevo fichero de configuración iniciamos el nuevo dominio con *xm create* como lo solemos hacer.

Como se puede concluir el proceso de creación y puesta en marcha de dominios basándonos en imágenes ya preconfiguradas y almacenadas en repositorios **es por lo general un proceso muy sencillo y rápido**, limitándonos a tener que reconfigurar solamente los parámetros que sean únicos o diferenciadores para los nuevos dominios –además de instalar nuevos servicios o aplicaciones si es necesario. Además, **si necesitamos crear de nuevo las imágenes de disco para el dominio, herramientas como *Xen-Tools* nos permiten hacerlo también de una manera fácil y automatizada**. **Procesos administrativos como la recuperación de sistemas ante fallos, aprovisionamiento de dominios, instalación y configuración de equipos… se ven enormemente optimizados y mejorados gracias a la forma de operar y las características de la virtualización**.

Creación de dominios con instancias de sistemas operativos de tipo RPM

Si queremos instalar otro tipo de distribuciones, por ejemplo de tipo RPM, y no basadas en sistemas Debian/Ubuntu **podemos hacer uso de otro tipo de aplicaciones que también aparecen integradas con *Xen-Tools*** de igual forma como lo está *debootrap*. **Se trata de *rinse*** (http://www.xen-tools.org/software/rinse/) y ***rpmstrap***. Su utilización no es presentada aquí ya que la forma de operar es análoga a la anterior, eligiendo el método de instalación en el archivo *xen-tools.conf* para la configuración de *Xen-Tools*:

```
install-method = rinse
```

Después en la opción *dist* del mismo fichero seleccionamos la distribución a instalar, por ejemplo CentOS o Fedora –en el propio archivo *xen-tools.conf* tenemos una lista de las distribuciones disponibles. **Aunque existen otras herramientas, por ejemplo *cobbler*** (https://fedorahosted.org/cobbler/) **o *virt-install*** (basado en ***libvirt***, http://libvirt.org/), **no deja de ser paradójico que la forma más rápida de instalar una distribución RPM en una máquina virtual sea con *rinse* sobre Debian, es decir, con una distribución que no usa RPM como formato nativo**. La gran diferencia entre utilidades como *virt-install* y *cobbler* y otras como *Xen-Tools (deboostrap, rinse)* es que el suministro de las máquinas virtuales se basa en el propio instalador de la distribución con unos valores predeterminados en forma de fichero *kickstart*, realizando una instalación *desatendida*, algo que por lo general es más lento.

Aparte de estos métodos de instalación *Xen-Tools* también incluye otros como *copy* o *tar* que pueden llegar a ser incluso más rápidos en la puesta en marcha de las máquinas virtuales. El primero realiza una instalación copiando un directorio que contiene una instalación previa y el segundo lo hace descomprimiendo una imagen creada con anterioridad. **No podemos olvidar también como otro recurso adicional la posibilidad de la descarga de máquinas virtuales preconfiguradas de sitios de Internet**, como vimos al comienzo de esta sección.

4.3.3 VIRT-INSTALL

A continuación vamos a presentar otra herramienta utilizada para la creación de dominios invitados en Xen, pero que **además de ser utilizada en Debian GNU/Linux también es posible operar con ella en distribuciones como Fedora Core**. En el código que sigue así como los ejemplos presentados la versión utilizada ha sido **Fedora Core 8**.

virt-install es una **utilidad en línea de comandos** que permite la creación de máquinas virtuales en Xen de una manera sencilla. **Es instalada junto a Xen y junto a la utilidad para la creación y monitorización de dominios Xen *virt-manager***, por lo que para instalarla si no disponemos de ella debemos instalar *virt-manager*:

```
[root@localhost /]# yum install virt-manager
```

El único requisito presentado por *virt-manager* para funcionar correctamente es ***VNC* si es que necesitamos acceso a la consola gráfica de los dominios *domU***. En el proceso de creación de un nuevo dominio podemos especificar si el acceso al mismo será vía texto o gráficamente. Para comprobar si *vnc* se encuentra instalado en nuestro sistema podemos ejecutar:

```
[root@localhost /]# rpm –q vnc
```

En el caso de que el resultado sea que no se encuentra instalado, lo podemos hacer fácilmente ejecutando como usuario *root*:

```
[root@localhost /]# yum install vnc
```

Entonces ***VNC* es usado por otra herramienta** también instalada junto a *virt-manager* y *virt-install*, ***virt-viewer***, para mostrarnos gráficamente la consola del dominio creado.

Para iniciar *virt-install* es suficiente con ejecutar registrado como usuario *root* el siguiente comando:

```
[root@localhost /]# virt-install
```

***virt-install* admite una gran cantidad de parámetros y valores para la configuración del nuevo dominio** en la propia llamada a la herramienta; para consultarlos es recomendable visitar la página *man* de la misma (véase la figura). **Si no escribimos ningún parámetro al invocar *virt-install* la aplicación se inicia y comienza a realizarnos preguntas con el objetivo de configurar las características del nuevo dominio que queremos crear**:

- *¿Cuál es el nombre de la máquina virtual?* Se trata del nombre que damos al dominio y que es utilizado para su monitorización por *dom0* y las herramientas administrativas que usemos en nuestra infraestructura virtual.

- *¿Qué cantidad de memoria RAM se asigna (en megabytes)?* Cantidad de memoria RAM que tendrá disponible el dominio.
- *¿Qué es usado como disco (camino al fichero)?* Aquí especificamos la ruta completa al fichero o partición de disco que será usada para contener el sistema operativo del dominio.
- *¿Qué tamaño tiene el disco (en gigabytes)?* Tamaño del disco a crear para el dominio especificado en la pregunta anterior en gigabytes.
- *¿Desea soporte gráfico para la máquina virtual?* Si deseamos habilitar o no el acceso mediante *vnc* a la consola gráfica del dominio.
- *¿Cuál es la localización del instalador?* Especificamos la localización del instalador y los ficheros del sistema operativo del dominio; esto se puede hacer indicando una dirección FTP, HTTP, NFS, una imagen ISO, o un dispositivo local CD/DVD-ROM. Por ejemplo, si queremos instalar FC8 podemos indicar la dirección HTTP http://archives.fedoraproject.org/pub/archive/fedora/linux/releases/8/Fedora/i386/os/ en el árbol de instalación de Fedora con raíz en http://archives.fedoraproject.org/pub/archive/fedora/.

En la figura 3.16 podemos apreciar un ejemplo de creación de un dominio *domU* con Fedora Core 8 como sistema operativo contestando las preguntas anteriores formuladas por *virt-install.*

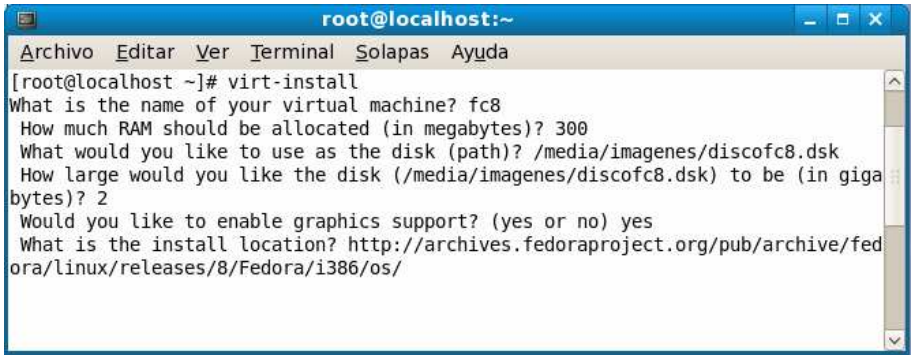


Figura 3.16 Ejemplo de creación de un domU FC8 con virt-install.

Finalizadas las preguntas comienza la instalación del sistema operativo tomándolo del medio de instalación origen que hayamos facilitado, bien mediante consola gráfica con *virt-viewer* y *vnc* (véase la figura 3.17) o bien en modo texto en el terminal (según la figura 3.18) según se haya configurado también con *virt-install*.

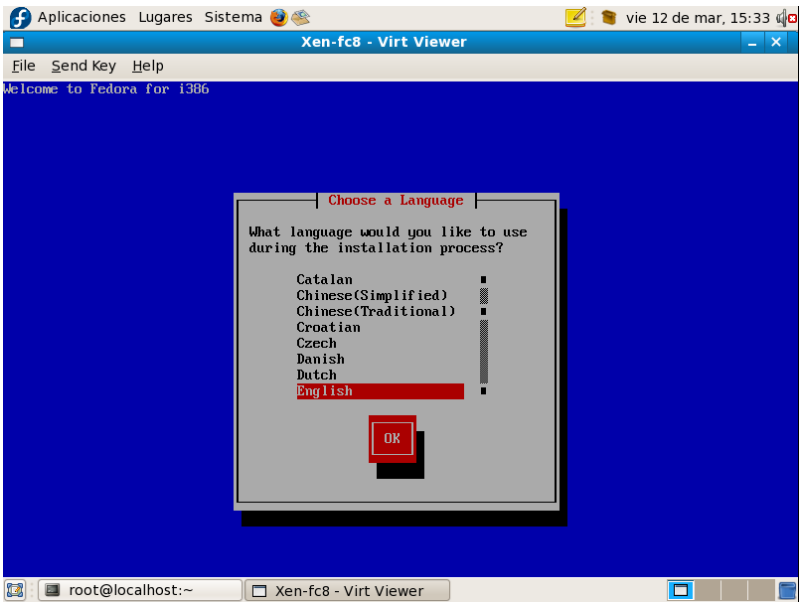


Figura 3.17 Instalación de FC8 en un nuevo dominio mostrada gráficamente por virt-viewer.

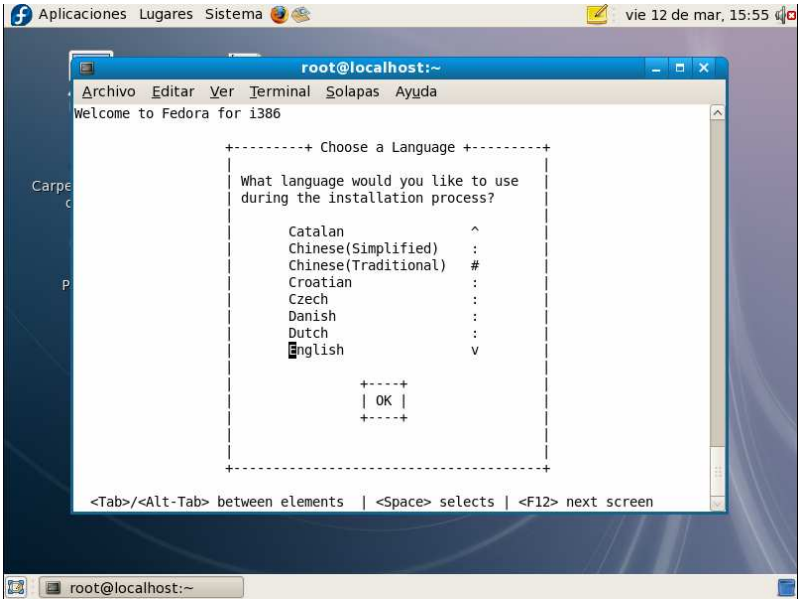


Figura 3.18 Instalación en modo texto de FC8 en un nuevo dominio.

Una vez completada la instalación del sistema operativo el nuevo dominio puede ser administrado como cualquier otro haciendo uso de las herramientas administrativas y de monitorización que se encuentren instaladas en *dom0*, las cuales veremos en la sección *5. Monitorización de dominios*. Los dominios creados por *virt-install*, al igual que ocurre con los creados mediante la herramienta gráfica *virt-manager*, pasan a ser inmediatamente administrados por el demonio *xend*, por lo que **su configuración es almacenada en el directorio *<u>/var/lib/xend/domains</u>* en ficheros con nombre el *UUID* del dominio**. Estos ficheros utilizan una sintaxis para describir la configuración y comportamiento de los dominios diferente a la presentada anteriormente para los archivos de configuración *.cfg*, similar a la salida presentada al ejecutar el comando *xm list –long* para la obtención de información acerca de un dominio.

4.3.4 HARDWARE VIRTUAL MACHINES (HVMs)

Hasta ahora todo lo que ha sido estudiado ha sido enfocado en la dirección de *dominios paravirtualizados*, en los que el sistema operativo es modificado para que tome parte del proceso de virtualización junto al *hipervisor Xen*, obteniendo un rendimiento superior final por ello. Para finalizar con la documentación acerca de la definición y creación de *dominios Xen* veremos cómo hacerlo para **el caso en el queramos disponer de *máquinas virtuales hardware (HVM, Hardware Virtual Machines)***, que como ya sabemos **pueden correr sistemas operativos no modificados (por ejemplo, Microsoft Windows) valiéndose del soporte en el procesador para virtualización**.

El proceso para la definición y creación de este tipo de *dominios completamente virtualizados* es similar al que hemos visto para *dominios paravirtualizados*: proporcionamos los medios de almacenamiento y sistema operativo para el dominio, creamos el archivo de configuración que usa Xen para arrancarlo y finalmente haciendo uso de la interfaz *xm* lo iniciamos y operamos sobre él. **Las diferencias que encontramos son en el modo de configuración del dominio mediante el uso de parámetros diferentes/adicionales en su archivo *.cfg* y en la instalación del sistema operativo sin modificar desde una imagen ISO o leyendo desde CD-ROM**.

Es importante de cara a la creación del archivo de configuración del *domU* tener claro los recursos físicos a los que tendrá acceso; por ejemplo si crearemos la imagen de los discos duros virtuales con el comando *dd* –como hicimos en los ejemplos anteriores-, si daremos acceso al dispositivo CDROM o una imagen del mismo para la carga del sistema operativo… Las entradas *disk* en el archivo de configuración para determinar el acceso a dispositivos de almacenamiento en *dom0* son exactamente iguales para *dominios paravirtualizados y completamente virtualizados.*

A continuación podemos ver un **archivo de configuración estándar para *dominios completamente virtualizados* en Xen**:

```
import os, re

arch = os.uname()[4]

if re.search(‘64`, arch):
    arch_libdir = ‘lib64`
else:
    arch_libdir = ‘lib’

kernel = “/usr/lib/xen/boot/hvmloader”

builder=’hvm’

memory = 512

shadow_memory = 8

name = “nombre_dominio”

acpi=1

apic=1

vif = [ ‘type=ioemu, bridge=xenbr0` ]

disk = [ ‘file:/ruta/a/la/imagen/de/disco.img,hda,w’,
‘phy:/dev/hdb,hdc:cdrom,r’ ]
```

```
device_model = '/usr/' + arch_libdir + '/xen/bin/qemu-dm'
boot="dc"
sdl=0
vnc=1
vncconsole=1
vncpassword=''
stdvga=0
serial='pty'
usb=1
usbdevice='tablet'
```

La primera parte del fichero es un **bloque de código escrito en Python que importa dos módulos para la identificación de la arquitectura en la que está siendo ejecutado Xen y el directorio de librerías usadas por Xen** (32 o 64 bits). Estas líneas serán siempre comunes y no deberían ser alteradas para todas las configuraciones de *dominios HVM*.

Además, el fichero contiene las siguientes entradas:

- ***kernel.*** **Sirve para definir el código inicial que es cargado en el inicio del dominio**. En este ejemplo es un cargador de *máquinas virtuales hardware* que sirve para la emulación de una BIOS x86 *pura* sobre la que es ejecutado el arranque del sistema operativo en función de la opción *boot*.

- ***builder.*** Esta opción contiene el **nombre de la función que es aplicada para la construcción del *dominio domU***.

- ***memory.*** Permite establecer la **cantidad de memoria en megabytes asignada inicialmente para el *dominio HVM***.

- ***shadow_memory.*** Es utilizado para establecer la **cantidad de memoria en megabytes reservada para la tabla interna usada para *mapear* las páginas de memoria de *domU*** en las páginas de memoria de *dom0.* Normalmente es una cantidad que debería ser igual a 2Kb por Mb de memoria en el dominio más unos pocos Mb por *vcpu.* Por lo general 8Mb son suficientes.

- ***name.*** **Nombre que queremos asignar al dominio**. Es conveniente que al tratarse de un *dominio completamente virtualizado* incluyamos en su nombre alguna referencia a ello.

- ***acpi.*** **Permite especificar si *ACPI*** *(Advanced Power and Control Interface, interfaz avanzada de control y energía)* **se encontrará habilitada** en la máquina virtual. Un valor de 1 así lo indicará.

- ***apic.*** Al igual que en la opción anterior escribiremos un 1 **si queremos habilitar en el dominio *APCI*** *(Advanced Programmable Interrupt Controller, controlador avanzado programable de interrupciones).*

- ***vif*. Contiene una lista de cadenas que especifica los adaptadores de red que se establecen en el dominio**. En el ejemo la interfaz de red usa el dispositivo emulador ***ioemu*** de Qemu en lugar del driver de red virtual de Xen, ***netfront***. Además incluimos que la máquina virtual seaañadida al *puente xenbr0*, el *puente Xen* por defecto.

- ***disk*.** Es quizás la parte más importante para la configuración de un *dominio HVM*. Aquí **identificamos los dispositivos de almacenamiento virtuales del dominio** asociados a los reales disponibles localmente en el anfitrión (discos, CD/DVD…). Cada uno de los dispositivos usa la nomenclatura que fue comentada anteriormente en este mismo capítulo: **nombre del dispositivo en *dom0*, nombre del dispositivo en *domU*, y la información sobre los privilegios de acceso**. Los nombres de los dispositivos en la configuración de *HVMs* deben hacer referencia a dispositivos completos y no a particiones de dispositivos.

- ***device_model*. Modo real para la emulación de instrucciones** usado por los *dominios completamente virtualizados*. En este caso es la ruta completa al *modelo de dispositivo binario Qemu (**qemu-dm**).*

- ***boot*. Indicamos el orden de preferencia para el arranque en los distintos dispositivos del sistema**. *HVM* usa este orden para arrancar desde esos dispositivos, donde ***c*** representa el disco duro virtual y ***d*** el dispositivo DVD/CD-ROM (también se podría usar ***a*** para un disquete).

A continuación **disponemos de una serie de opciones que definirán el acceso a la consola gráfica del sistema operativo, pudiendo seleccionar entre soporte *SDL* *(Linux Simple DirectMedia Layer)* o *VNC* *(Virtual Network Computing)***. Escribimos un 1 en la opción que queramos usar, escribiendo en la otra un 0. Tanto *SLD* como *VNC* suelen funcionar bastante bien, aunque *VNC* presenta algunas ventajas; por ejemplo más flexibilidad en términos de acceso remoto desde otros sistemas en la misma red local o a través de Internet, o que al cerrar el visor *VNC* el sistema continúa ejecutándose permitiendonos conectar de nuevo en el futuro (con *SDL* el dominio será apagado).

- ***sdl*.** Permite habilitar (1) o deshabilitar (0) el **acceso mediante *SDL* a la consola gráfica del *domU HVM*.**

- ***vnc*.** Permite habilitar (1) o deshabilitar (0) **el acceso mediante *VNC* a la consola gráfica del *domU HVM*.**

- ***vncconsole*.** Por defecto Xen no inicia la *consola VNC* automáticamente cuando el *dominio domU* es puesto en marcha. Para tener acceso a ella de forma automática escribimos el valor 1 en esta opción del fichero.

- ***vncpassword*.** Permite establecer una contraseña para las conexiones con la consola del dominio invitado mediante *VNC*.

- ***stdvga*.** Para especificar el **uso de gráficos *VGA* estándar** (1) **o *Cirrus Logic*** (0).

- ***serial*.** Es usada para **redigirir la salida de la consola serie del *dominio HVM* a una *pseudo-TTY***, lo que hace posible que podamos recoger información del arranque de consola conectándonos a */dev/pty/<id#>* (*<id#>* es el identificador del

dominio) con el comando *xm* o bien con cualquier otro emulador de terminales (como *minicom*).

- ***usb.*** **Permite añadir dispositivos USB al *dominio HVM* haciendo uso de los comandos de la interfaz *xm*** propios, por ejemplo *xm block-attach* (para dispositivos de almacenamiento *USB*) o *network-attach* (para dispositivos *USB* de red).

- ***usbdevice.*** Finalmente esta opción generalmente es necesaria para **evitar problemas con el manejo del puntero del ratón USB en la consola del *domU HVM***. Aunque el valor recomendado es *tablet*, que fuerza el seguimiento del movimiento absoluto del ratón dentro de la consola *HVM*, existen otros como por ejemplo *mouse* que sigue el movimiento relativo y hace uso de técnicas estándar como la aceleración.

De esta forma obtenemos un fichero de configuración con las opciones mínimas requeridas para establecer un *dominio HVM* en Xen; otras opciones se encuentran disponibles para una mayor caracterización del entorno del *domU*. Una vez que el archivo de configuración es escrito para el nuevo dominio lo guardamos con extensión *.cfg* en el directorio de acceso para este tipo de ficheros por defecto para Xen */etc/xen.*

A continuación **podemos iniciar el nuevo dominio mediante el comando *xm create* de la manera habitual** como lo hemos hecho hasta ahora. Si hemos configurado el acceso a la consola gráfica con *VNC* entonces ésta se inicia unos instantes después de haber creado el dominio; siempre podemos **acceder a ella igualmente haciendo uso de un *cliente VNC* como por ejemplo *vncviewer, TightVNC o RealVNC*. Por defecto el puerto al que realizamos la conexión es 127.0.0.1:5900+<id#>** (siendo <id#> el identificador del dominio). Por ejemplo si el dominio al que queremos conectarnos tiene como identificador 11 lo haremos mediante el *cliente VNC* a la dirección 127.0.0.1:5911. De todas formas siempre es recomendable comprobar en qué puerto se encuentra el servicio ejecutando el comando:

```
deb1# netstat -ndap
```

No debemos olvidar la configuración *VNC* en *dom0*, estableciendo en la opción *vnc-listen* del archivo de configuración *xend-config.sxp* de *xend* las direcciones desde las cuales es posible establecer conexión (127.0.0.1 para locales, 0.0.0.0 para todas direcciones). Después de hacer cambios reiniciamos el demonio *xend*:

```
deb1# /etc/init.d/xend restart
```

Para ilustrar el proceso completo desarrollamos a continuación un ejemplo para la creación de un *dominio domU completamente virtualizado* sobre el que es ejecutado como sistema operativo invitado **Microsoft Windows XP**.

4.3.5 Ejemplo de creación de una HVM: Windows XP en Xen

Como paso previo **debemos comprobar si nuestro procesador dispone de soporte hardware para virtualización en el procesador**, incluyendo la tecnología Intel-VT si se trata de un procesador Intel o AMD-V si es AMD. Para ver las características de nuestro procesador ejecutamos el siguiente comando:

```
deb1# less /proc/cpuinfo
```

En el caso de procesadores Intel debe tener presente la característica *vmx*, y en el caso de AMD *svm*. Si el procesador no muestra soporte para virtualización, es posible que lo tenga y que no esté habilitado. Para comprobarlo reiniciamos el equipo y comprobamos en la configuración de la BIOS del sistema si disponemos de alguna opción para permitir *soporte a virtualización.* En el caso de que exista lo habilitamos y volvemos a ejecutar el comando y así comprobar que se encuentra operativo, ya que de lo contrario no podemos desarrollar *dominios completamente virtualizados.* Aún así en ciertas distribuciones han aparecido problemas debido a que el *flag vmx* de Intel sigue sin mostrarse. Podemos comprobar sin embargo que se encuentra activo mediante los comandos *xm info* o *xm dmesg*; en uno de los mensajes grabados por Xen mostrados en salida de éste último podemos ver que se encuentra activo:

```
deb1:~# xm dmesg
(XEN) Xen version 3.2-1 (Debian 3.2.1-2) (waldi@debian.org) (gcc
version 4.3.1 (Debian 4.3.1-2) ) Sat Jun 28 09:32:18 UTC 2008
(XEN) Command line: dom0_mem=196M
.
.
.
 (XEN) Detected 3000.101 MHz processor.
(XEN) HVM: VMX enabled
(XEN) CPU0: Intel(R) Xeon(R) CPU           E3110  @ 3.00GHz stepping
0a
(XEN) Booting processor 1/1 eip 8c000
(XEN) CPU1: Intel(R) Xeon(R) CPU           E3110  @ 3.00GHz stepping
0a
(XEN) Total of 2 processors activated.
.
.
.
```

A continuación **comenzamos a preparar la instalación de Windows XP cuyo primer requisito es el espacio en disco**. Creamos el almacenamiento del disco virtual para el invitado usando un disco físico o utilizando una imagen de disco. Como hicimos en los ejemplos anteriores creamos una imagen de disco de 8Gb usando el comando ***dd*** en un directorio */media/imágenes/winxp-HVM* creado para el almacenamiento el dominio:

```
deb1:/media/imagenes# mkdir winxp-HVM

deb1:/media/imágenes# cd winxp-HVM

deb1:/media/imagenes/winxp-HVM# dd if=/dev/zero of=disk.img bs=1024k
count=8192
8192+0 records in
8192+0 records out
8589934592 bytes (8,6 GB) copied, 235,512 s, 36,5 MB/s

deb1:/media/imagenes/winxp-HVM# ls -la
total 8396820
drwxr-xr-x 2 root root       4096 feb  9 17:45 .
drwxr-xr-x 6 root root       4096 feb  9 17:44 ..
-rw-r--r-- 1 root root 8589934592 feb  9 17:49 disk.img
```

Después de unos minutos el almacenamiento de 8Gb para nuestro nuevo dominio habrá sido creado. Una vez que el disco o imagen de disco está disponible **el siguiente paso es la creación del fichero de configuración del *dominio HVM Xen***. **Tomando como base el ejemplo de fichero presentado antes lo adaptamos para nuestro nuevo dominio *winxp-HVM***:

```
import os, re

arch = os.uname()[4]

if re.search('64', arch):
    arch_libdir = 'lib64'
else:
    arch_libdir = 'lib'

kernel = "/usr/lib/xen/boot/hvmloader"

builder='hvm'

memory = 512

shadow_memory = 8

name = "winxp-HVM"

acpi=1

apic=1

vif = [ 'type=ioemu, bridge=eth0' ]

disk = [ 'file:/media/imagenes/winxp-HVM/disk.img,hda,w',
'phy:/dev/scd0,hdc:cdrom,r' ]

device_model = '/usr/' + arch_libdir + '/xen/bin/qemu-dm'

boot="dc"

sdl=0

vnc=1

vncunused=1

stdvga=0

serial='pty'

usb=1

usbdevice='tablet'
```

Como podemos observar hemos caracterizado el fichero de configuración *HVM* básico con la cantidad de memoria necesaria, el nombre del dominio y los dispositivos de almacenamiento a los que queremos que acceda –especificando como disco principal la imagen de disco que hemos creado en la página anterior y acceso al dispositivo lector de DVDs/CD-ROMs. La prioridad de los dispositivos en el arranque es fijada en este orden: DVD/CD-ROM, disco. El acceso a la consola gráfica del dominio ha sido configurado mediante *VNC*, añadiendo la opción ***vncunused*** con *valor 1* para que sea asignado un puerto libre para el servidor *VNC* del dominio a partir de 5900.

Podemos comprobar que el dominio tiene acceso a los recursos que le han sido asignados mediante el uso del comando *xm dry-run* registrados como usuario privilegiado *root* ya que es necesario siempre para la ejecución de comandos *xm* (véase la figura 3.19).

Figura 3.19 Comprobando el acceso a los recursos por parte del nuevo dominio HVM.

Una vez que hemos verificado el dominio y que tiene el acceso permitido a sus recursos **podemos iniciarlo con el comando *xm create*:**

```
deb1:/# xm create winxp-HVM.cfg
Using config file "/etc/xen/winxp-HVM.cfg".
Started domain winxp-HVM
```

También se puede apreciar en el archivo de configuración anterior como **el nombre del *puente* al que conectamos el adaptador de red del *dominio HVM* es *eth0*** y no *xenbr0* como cabe esperar (*puente por defecto* en Xen). Es recomendable siempre para evitar problemas comprobar valores para algunas opciones que asumimos *por defecto*, por ejemplo para el nombre del *puente* lo podemos de la siguiente forma:

```
deb1:/var/lib/xen# brctl show
bridge name     bridge id               STP enabled     interfaces
eth0            8000.002481b2482c       no              peth0
                                                        tap0
                                                        vif75.0
```

En el caso de que surjan problemas al iniciar el dominio es de gran ayuda examinar los mensajes que contienen los siguientes ficheros *log*, ubicados en el directorio */var/log/xen*:

- ***xend.log*,**
- ***xend-debug.log*,**
- ***xen-hotplug.log*,**
- ***qemu-dm<#id>.log***, donde <#id> es el identificador del dominio que hemos iniciado.

En éste último se almacenan los **eventos relacionados con la *emulación qemu* de dispositivos realizada al crear dominios HVM**, y es ahí donde se observó un problema para poner en marcha los *scripts de red* al iniciar el dominio ya que el *puente de conexión* no era identificado correctamente al no proporcionar el nombre adecuado.

Ejecutando *xm list* o *xm top* (véase la figura 3.20) **vemos que efectivamente el dominio ha sido creado con éxito**:

```
deb1:~# xm list
Name                                ID   Mem VCPUs      State   Time(s)
Domain-0                             0   196     2     r-----    195.0
```

```
winxp-HVM                                    12   512     1      ------       0.0
```

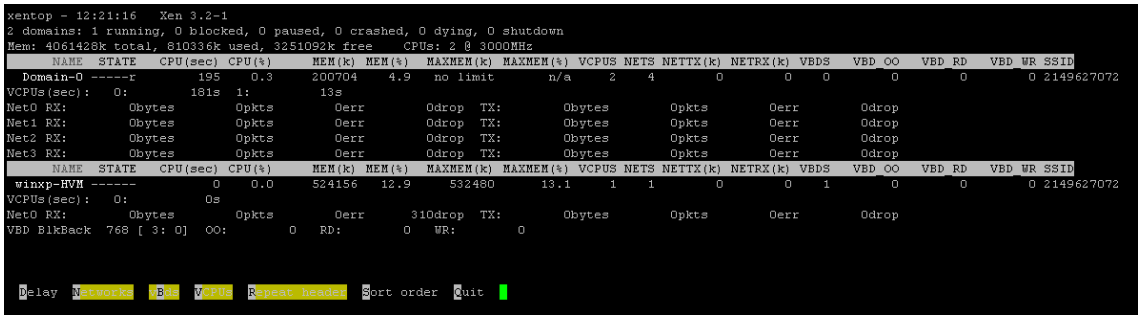


Figura 3.20 Nuevo dominio winxp-HVM monitorizado con xm top.

Lo normal es que si ha sido configurado el acceso *VNC* de forma automática tras su puesta en marcha al transcurrir unos instantes aparezca en nuestra pantalla. Si no lo hace, **comprobamos cuál es el puerto que en el que ha sido alojado el servidor *VNC* para el nuevo dominio**:

```
deb1:/media/imagenes/winxp-HVM# ps -ef | grep vnc
root     12159  7720  5 20:04 ?        00:00:00 /usr/lib64/xen/bin/qemu-dm -
d 76 -domain-name winxp-HVM -vnc 127.0.0.1:0 -vncunused -vcpus 1 -boot dc -
serial pty -acpi -usb -usbdevice tablet -net
nic,vlan=1,macaddr=00:16:3e:37:f3:62,model=rtl8139 -net
tap,vlan=1,bridge=eth0 -M xenfv
root     12374 27636  0 20:04 pts/2    00:00:00 grep vnc
root     13796 27886  0 17:52 ?        00:00:00 [vncviewer] <defunct>
root     27886     1  0 16:10 ?        00:00:00 vncviewer -log *:stdout:0 -
listen 5501
```

Ya sólo **queda iniciar la conexión mediante un cliente *VNC* en la dirección 127.0.0.1:0** como sigue:

```
deb1:/media/imagenes/winxp-HVM# vncviewer 127.0.0.1:0
```

Así nos conectamos a **la consola gráfica del *dominio HVM* creado, que muestra como podemos ver en la figura 3.21 el comienzo del proceso de instalación de Windows XP iniciado desde la unidad DVD/CD-ROM**:

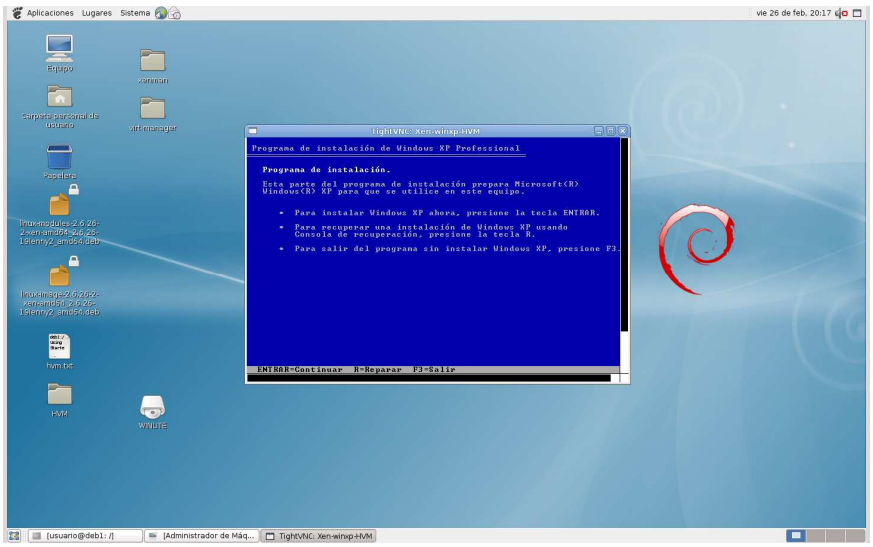


Figura 3.21 Inicio de la instalación de Windows XP en la conexión VNC al dominio winxp-HVM.

Es posible que la conexión no sea establecida debido a que no se encuentran instalados algunos **paquetes y librerías necesarios para ello**, aunque lo normal es que hayan sido instalados como dependencias anteriormente. Estos son:

- ***libsdl1.2debian-all* y *libsdl1.2-dev* para conexiones *sdl***,
- ***libvncserver-dev*** (http://libvncserver.sourceforge.net/) **para conexiones *VNC***.

5 Monitorización de dominios

En este último apartado del capítulo profundizamos en las **actividades de monitorización en Xen y las posibilidades que de forma básica nos pueden proporcionar las utilidades disponibles para ello**. Comenzamos examinando los distintos subcomandos de la interfaz administrativa *xm* que nos permiten obtener información del estado del dominio *dom0: xm info*, *xm log* y *xm dmesg*. Seguiremos viendo subcomandos que con los que podremos ver el estado del resto de dominios *domU* (*xm list*) y monitorizar el consumo que experimentan de los distintos recursos (*xm top*). Por último serán presentadas herramientas gráficas para este tipo de actividades, tanto propias del sector de virtualización (*XenMan de Convirt y Virt-Manager*) como generales (*Nagios y MRTG*).

5.1 OBTENIENDO INFORMACION DE DOM0

Existe un **conjunto de comandos de la interfaz administrativa *Xen Manager (xm)* que nos permiten obtener información el dominio privilegiado *dom0***, el dominio más importante ya que es el encargado de la administración y monitorización de las actividades del resto de dominios. Por ello es importante conocerlos y saber qué nos pueden aportar; los comandos que analizamos en este apartado son tres: ***xm info, xm log* y *xm dmesg***, que forman parte de un subconjunto de comandos utilizados en la interfaz de administración *xm* para el host anfitrión. Así, si necesitamos más información sobre su uso podemos consultar la página *man* de *xm* (véase la figura 3.22).

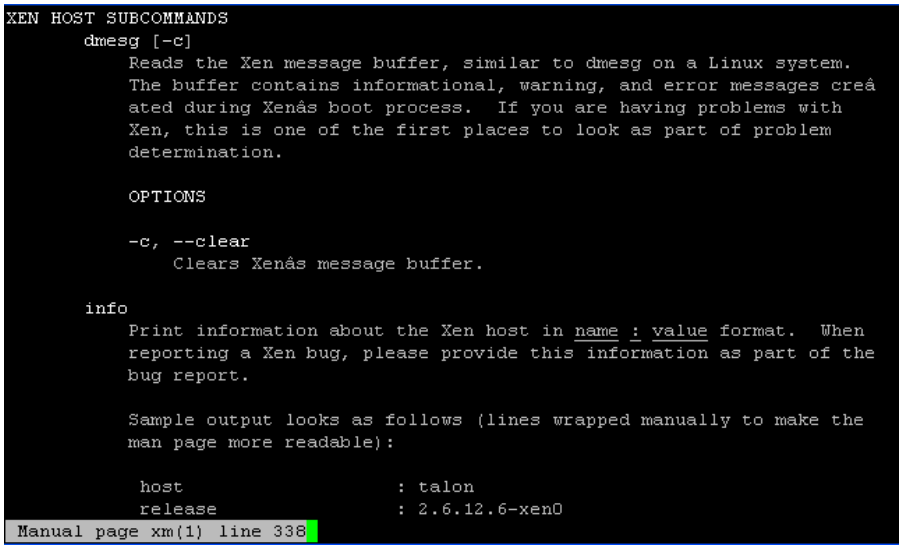


Figura 3.22 Sección de subcomandos para el host anfitrión en la página man de xm.

Los comandos que nos permiten obtener información concreta sobre el sistema de virtualización Xen son:

- ***xm info.*** **Muestra información general sobre el *host anfitrión Xen*** (véase la figura 3.23). La información es presentada en pares atributo/valor, siendo bastante sencillo interpretarla por lo general; los campos que pueden entrañar alguna dificultad en su interpretación son explicados en la página *man* del comando. Por ejemplo, ***free_memory*** o ***xen_caps***, que informan sobre la memoria disponible sin asignar a Xen ni a ningún dominio actualmente y la versión y arquitectura de Xen, respectivamente.

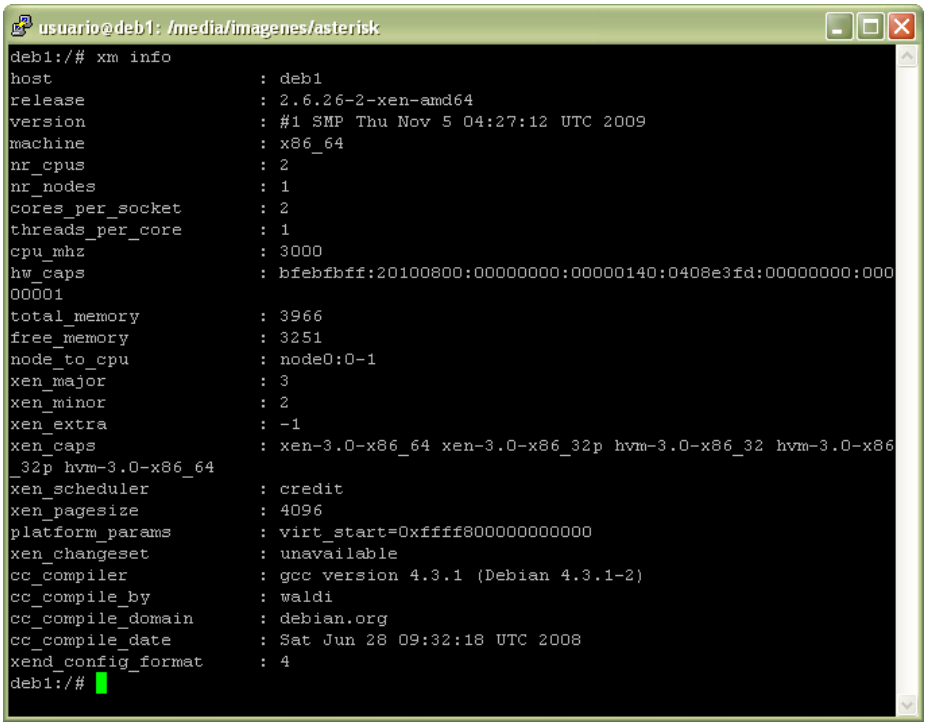


Figura 3.23 Ejecución del subcomando xm info.

- ***xm log***. **Muestra el *log* que ha ido generando el demonio *xend***. El log se almacena en el fichero ***xend.log*** en el directorio */var/log*.

```
deb1:~# xm log

.
.
.

 [2010-01-27 21:07:45 17088] DEBUG (XendDomainInfo:1542)
Removing vbd/2050
[2010-01-27 21:07:45 17088] DEBUG (XendDomainInfo:590)
XendDomainInfo.destroyDevice: deviceClass = vbd, device =
vbd/2050
[2010-01-27 21:07:45 17088] DEBUG (XendDomainInfo:1542)
Removing console/0
[2010-01-27 21:07:45 17088] DEBUG (XendDomainInfo:590)
XendDomainInfo.destroyDevice: deviceClass = console, device =
console/0
```

- ***xm dmesg.*** Permite la lectura del búfer de mensajes de Xen de forma similar a como lo hace el comando *dmesg* en un sistema Linux. Este búfer contiene mensajes informativos, de advertencia y de error generados en el proceso de arranque de Xen por lo que es muy útil su ejecución cuando aparecen problemas en Xen. Con la opción *–c* podemos vaciar el búfer de mensajes.

```
deb1:~# xm dmesg
(XEN) Xen version 3.2-1 (Debian 3.2.1-2) (waldi@debian.org)
(gcc version 4.3.1 (Debian 4.3.1-2) ) Sat Jun 28 09:32:18 UTC
2008
(XEN) Command line: dom0_mem=196M
(XEN) Video information:
(XEN)  VGA is text mode 80x25, font 8x16
(XEN)  VBE/DDC methods: none; EDID transfer time: 0 seconds
(XEN)  EDID info not retrieved because no DDC retrieval
method detected

.
.
.

(XEN) *** LOADING DOMAIN 0 ***
(XEN)  Xen  kernel: 64-bit, lsb, compat32
(XEN)  Dom0 kernel: 64-bit, lsb, paddr 0x200000 -> 0x631918
(XEN) PHYSICAL MEMORY ARRANGEMENT:
(XEN)  Dom0 alloc.:   000000010a000000->000000010c000000
(41984 pages to be allocated)
(XEN) VIRTUAL MEMORY ARRANGEMENT:
(XEN)  Loaded kernel: ffffffff80200000->ffffffff80631918
(XEN)  Init. ramdisk: ffffffff80632000->ffffffff81931c00
(XEN)  Phys-Mach map: ffffffff81932000->ffffffff81994000
(XEN)  Start info:    ffffffff81994000->ffffffff819944a4
(XEN)  Page tables:   ffffffff81995000->ffffffff819a6000
(XEN)  Boot stack:    ffffffff819a6000->ffffffff819a7000
(XEN)  TOTAL:         ffffffff80000000->ffffffff81c00000
(XEN)  ENTRY ADDRESS: ffffffff80200000
(XEN) Dom0 has maximum 2 VCPUs
(XEN) Initrd len 0x12ffc00, start at 0xffffffff80632000
(XEN) Scrubbing Free RAM:
.....................................done.
(XEN) Xen trace buffers: disabled
(XEN) Std. Loglevel: Errors and warnings
(XEN) Guest Loglevel: Nothing (Rate-limited: Errors and
warnings)
(XEN) Xen is relinquishing VGA console.
(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to
switch input to Xen)
(XEN) Freed 104kB init memory.
(XEN) traps.c:1996:d0 Domain attempted WRMSR 000000000000019a
from 00000000:00000002 to 00000000:00000000.
(XEN) traps.c:1996:d0 Domain attempted WRMSR 000000000000019a
from 00000000:00000002 to 00000000:00000000.
```

5.2 XM LIST

***List* es un subcomando de la interfaz administrativa del usuario *xm* que resulta de gran utilidad pues permite listar de manera sencilla y concisa todos los dominios existentes en nuestra infraestructura virtual**; nos mostrará todos los *dominios U* administrados por el *dom0* así como el propio *dom0.*

En un principio puede parecer que suministra poca información y ahí es en realidad donde vemos la virtud de esta utilidad: permite observar de un vistazo el estado de las máquinas virtuales. Por ejemplo:

```
deb1:/# xm list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0   128     1     r-----    290.4
asterisk1                                    3   128     1     -b----      4.6
asterisk2                                    4   128     1     -b----      4.1
```

Los diferentes campos que muestra *xm list* para cada dominio son los siguientes:

- ***Name.*** **Muestra el nombre de cada dominio**. *Domain-0* es el dominio *dom0.*

- ***ID.*** Es el **identificador del dominio**. Se trata de un número que permite la identificación de forma única y unívoca de cada dominio. Tanto el nombre como el identificador pueden ser usados indistintamente para invocar comandos que actúen sobre los dominios.

- ***Mem.*** Muestra la **cantidad de memoria en megabytes a la cual es deseable** (en el caso del *dom0* es la mínima especificada en su configuración) **que tenga acceso el dominio**.

- ***VCPUs.*** Es el **número de procesadores virtuales mostrados al dominio y por tanto número de procesadores que pueden ser usados por éste**. Los núcleos o *CPUs* reales pueden ser compartidos por los diferentes dominios.

- ***State.*** Permite ver el **estado actual en el que se encuentra el dominio**. En *Xen 3.2-1* hay seis estados posibles, cuya descripción se incluye en la tabla 3.3.

Tabla 3-3. Estados de un dominio Xen

Estado	Descripción
r – running	El dominio se encuentra actualmente ejecutándose en una CPU
b – blocked	El dominio se encuentra bloqueado luego no se está ejecutando o no es posible que pueda estarlo. Que un dominio esté bloqueado puede ser causado debido a una espera de entrada/salida o a que el dominio no tenga ninguna operación restante que llevar a cabo.
p – paused	El dominio ha sido pausado, normalmente por el administrador haciendo uso del comando *xm pause*. Un dominio pausado seguirá consumiendo los recursos que le han sido asignados, como memoria, pero no será usado en las planificaciones realizadas por el *hipervisor Xen*.
s – shutdown	Apagado, por lo que para poder usado debe ser arrancado de nuevo.
c – crashed	El dominio ha sufrido algún tipo de problema y ha terminado de forma violenta. Normalmente este estado tiene lugar solamente cuando un dominio no ha sido configurado para reiniciarse en caso de fallo.
d – dying	Un domino se encuentra en proceso de apagado o fallo, aún por culminar.

Es interesante una vez que conocemos los diferentes estados en los que puede encontrarse un dominio Xen ver esquemáticamente las posibles transiciones entre ellos, qué puede provocar que un dominio pase de un estado a otro. **El conjunto de los estados y las transiciones entre estados de los *dominios Xen* es llamado *ciclo***

de vida del dominio y ha quedado recogido de manera simplificada en la figura 3.24.

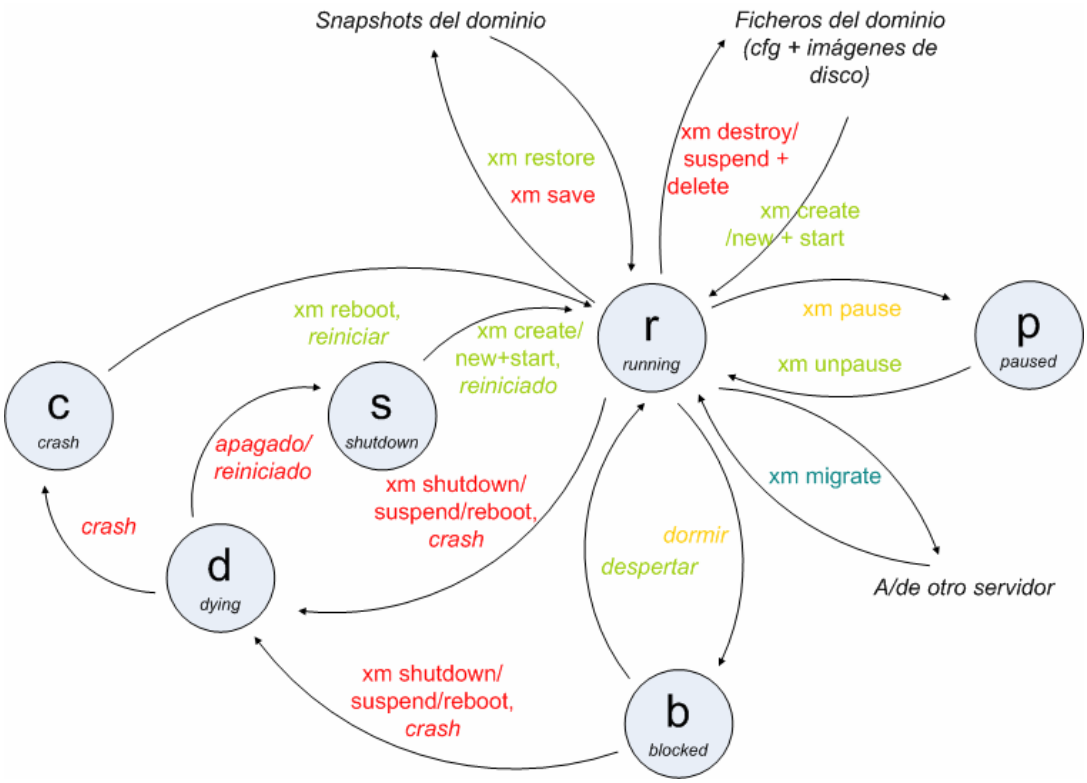


Figura 3.24 Ciclo de vida de un dominio Xen.

Todas las transiciones son identificadas con los comandos usados por los administradores en la interfaz *xm* salvo en los casos de reinicios automáticos al fallar y las transiciones *dormir y despertar* que tienen lugar automáticamente cuando el dominio deja de operar y vuelve a ello, respectivamente. Desde la interfaz administradora del dominio *dom0* es posible llevar a cabo todas las actividades que necesitamos: crear y eliminar dominios, pausarlo o devolverlo a ejecución si estaba en pausa, apagarlo, reiniciarlo,… incluso como sabemos operaciones de carácter avanzado como son la toma y recuperación de imágenes del estado en ejecución de dominios o su migración entre diferentes servidores físicos anfitriones.

- ***Time(s).*** Muestra el **tiempo total que ha estado ejecutándose el dominio en segundos**, contabilizado por Xen.

Como siempre para obtener más información sobre estos campos o cualquier otro aspecto relacionado con algún comando de la interfaz *xm* podemos **consultar la página *man* de *xmdomain.cfg*. En el caso de *xm list* también es posible la adición del flag *–long* para mostrar una salida con información más detallada**, con toda la información conocida de todos los dominios existentes:

```
deb1:/# xm list --long
```

Si lo que queremos es obtener información extensa para un solo dominio podemos escribir el comando de la siguiente forma (véase también la figura 3.25):

```
deb1:/# xm list --long nombre_dominio
```

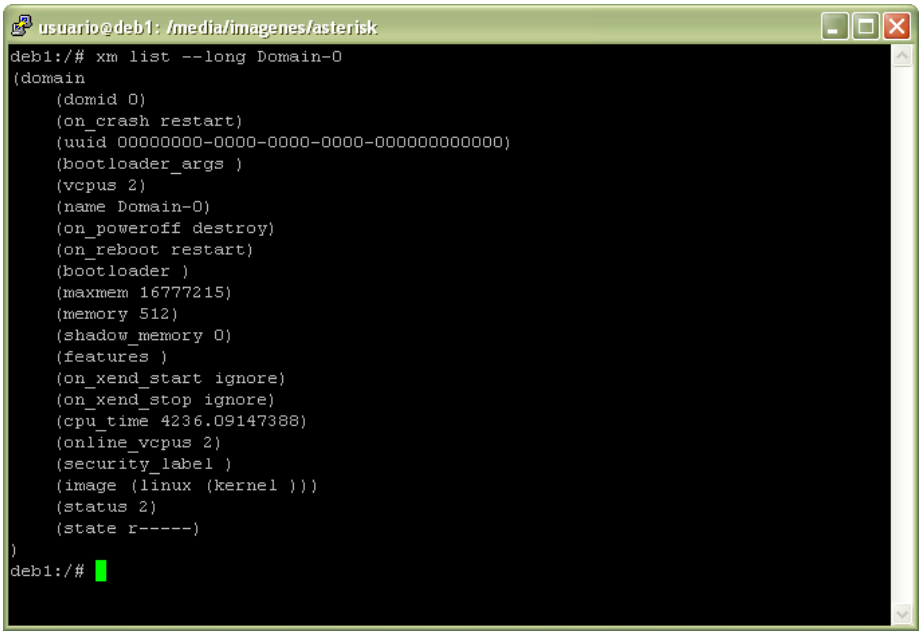


Figura 3.25 Información mostrada por xm list –long para el dominio dom0 .

5.3 XM TOP (XENTOP)

El comando *xm list* es muy útil en muchos casos pero en otros pierda su utilidad debido a que **no proporciona información y monitorización en tiempo real de los dominios**: es ejecutado y muestra la información de la que dispone sobre los dominios en ese instante de tiempo.

Para monitorizar en tiempo real el estado de los dominios de nuestra infraestructura virtual así como para obtener información más detallada del consumo que están experimentando de los diferentes recursos disponemos del comando *xm top* (también ejecutable como *xentop*), con una interfaz análoga al comando *top* de Linux.

En la figura 3.26 podemos ver una captura tomada de la ejecución de *xm top – xentop* monitorizando dos dominios (*asterisk1* y *asterisk2*) a parte del dominio *dom0.*

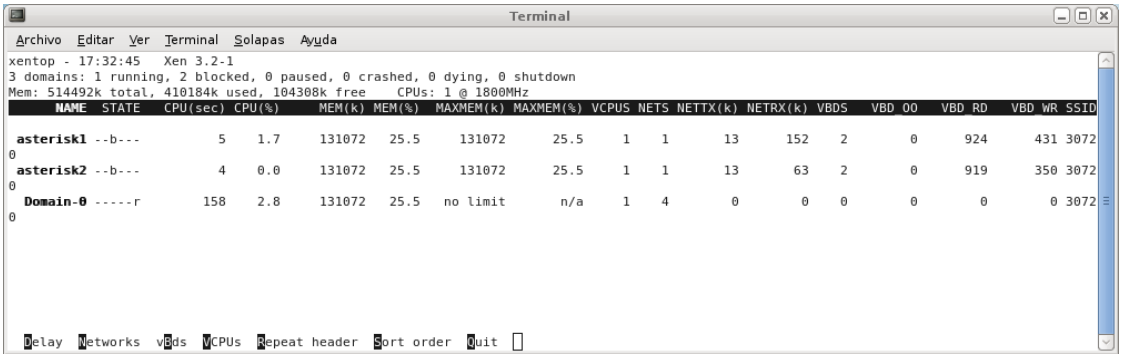


Figura 3.26 Monitorización de dominios con xm top - xentop.

La información más importante que proporciona *xm top* consiste en:

- ***Información general***. Hora actual, versión de Xen, número total de dominios, número de dominios para cada posible estado,… así como la cantidad total de memoria, libre y en uso, el número de CPUs y su velocidad.

- ***Información para cada dominio***. **Ofrece información para la identificación de los dominios y también indicadores del consumo que están experimentando en tiempo real** tanto de memoria como de CPU e interfaces de red. Los más importantes son:

    - ***Name y State***. **Nombre y estado del dominio**. Iguales a los vistos para el comando *xm list.*

    - ***CPU(sec), CPU(%) y VCPUs.*** **Los dos primeros nos proporcionan el tiempo total en segundos así como el porcentaje de uso de CPU por parte del dominio; *VCPUs* indica el número de *CPUs virtuales* que han sido asignadas al dominio**. Junto a los indicadores de memoria son los más importantes. Es importante reseñar que habrá un 100% de CPU disponible por cada CPU asignada al dominio; esto significa por ejemplo que si a una máquina virtual Xen han sido asignadas 2 CPUs entonces su consumo podrá llegar como máximo a un 200%. Los procesadores pueden ser compartidos por diferentes dominios lo que provoca que compitan unos con otros por su uso; si una de las dos CPUs anteriores es compartida por dos dominios entonces la suma del consumo de ambos de esa CPU no podrá superar el 100%.

    - ***MEM(k), MEM(%), MAXMEM(k) y MAXMEM(%).*** Estos indicadores nos proporcionan **información sobre la cantidad de memoria asignada y máxima en kilobytes del dominio y el porcentaje de consumo que ello representa sobre el total disponible físicamente en el servidor físico**. Con la memoria no ocurre lo mismo que con los procesadores: la memoria es asignada de forma exclusiva a cada dominio y una vez que ello sucede no puede ser usada por otro dominio.

    - ***NETS, NETTX(k) y NETRX(k).*** El valor ***NETS* indica el número de adaptadores de red** configurados en el dominio. ***NETTX y NETRX* muestran respectivamente el total de información transmitida y recibida** por las interfaces de red en kilobytes.

xm top* además proporciona en su interfaz una serie de opciones que nos permiten manipular la visualización de la monitorización en función de nuestras necesidades**. ***Delay (D) es quizás el más útil debido a que es usado para especificar el tiempo de muestreo de la información. Si pulsamos ***N (Networks)*** *o* ***V (VCPUs)*** ampliamos la información mostrada para las actividades de red y CPU respectivamente, pudiendo ver detalladamente el consumo de tráfico por *red virtual* creada en el primer caso y el consumo de procesamiento por cada *CPU virtual* en el segundo. De igual forma la opción ***B (VBDs)*** amplía la información mostrada para los *dispositivos de bloques virtuales* de cada dominio.

Para organizar mejor la visualización de los contenidos tenemos las opciones ***Repeat Header y Sort order.*** La primera sirve para repetir la cabecera de atributos mostrados para cada dominio mientras que la segunda nos permite cambiar el atributo por el que ordenar cada uno de los registros mostrados. Para salir de la herramienta pulsamos ***Q (Quit)***. En las siguientes imágenes (figuras 27 a 29) se muestran seleccionadas algunas de estas opciones.

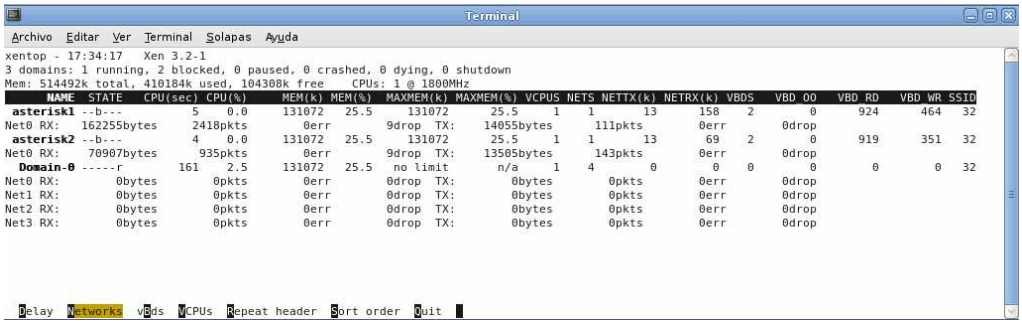


Figura 3.27 Monitorización de dominios con xm top – xentop y la opción N (Networks).

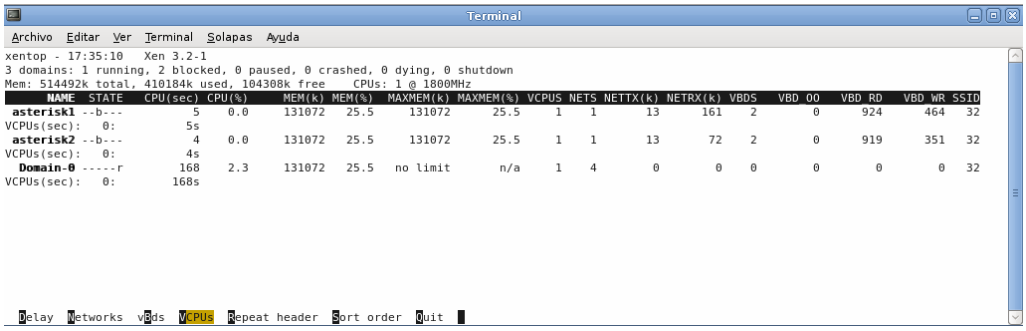


Figura 3.28 Monitorización de dominios con xm top – xentop y la opción V (VCPUs).

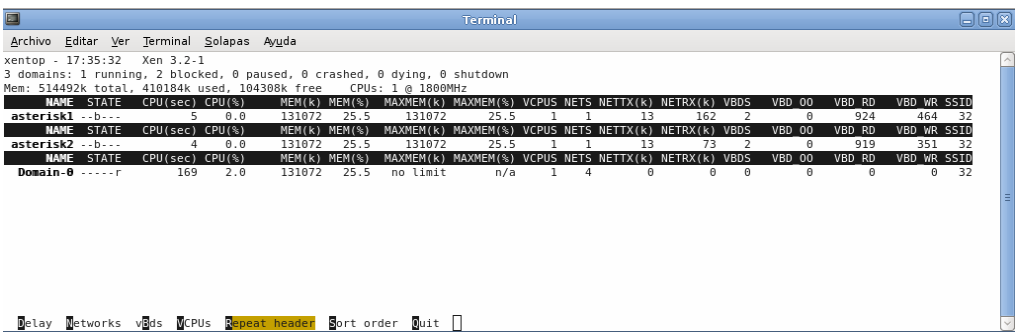


Figura 3.29 Monitorización de dominios con xm top – xentop y la opción R (Repeat header).

5.4 HERRAMIENTAS GRAFICAS

Además de las herramientas básicas de monitorización incluidas en el propio Xen existen otras que deben ser tenidas en cuenta y que normalmente debido a sus interfaces gráficas pueden hacer que la manipulación de nuestra infraestructura virtual sea más agradable. A continuación presentaremos brevemente **cuatro de estas herramientas; dos de ellas son propias del sector de la virtualización (*XenMan/Convirt, Virt-Manager*) por lo que son estudiadas con mayor detalle, mientras que las otras dos pueden ser usadas ya que una vez iniciados los dominios éstos pueden ser monitorizados como si de equipos reales se tratara**.

5.4.1 XenMan de Convirt

Convirt (*Controlling Virtual Systems*, http://www.convirture.com/) **es un proyecto con licencia GPL anteriormente llamado *XenMan* que facilita un entorno gráfico muy eficiente** para la gestión y administración de nuestra plataforma de virtualización. La **consola de administración *XenMan*** es el primer producto dentro de este proyecto. ***XenMan* permite una gestión completa del ciclo de vida de una plataforma Xen centralizando la administración de todo el entorno en una única consola segura**. Además lo hace facilitando y simplificando bastante todos los procesos, los cuales pueden ser realizados tan sólo con unos pocos clics.

Entre las características más importantes que posee se encuentran:

- ***La posibilidad de gestionar múltiples servidores***. Además del servidor local (si se instala en local) nos proporciona acceso a la gestión de servidores con máquinas virtuales de forma remota.

- ***Consola centralizada***. En una sola consola podemos ver el rendimiento, la configuración y administrar nuestros servidores y las máquinas virtuales que contienen rápidamente. No sólo permite observar todo esto, sino que también podemos actuar sobre las máquinas virtuales directamente desde la propia consola.

- ***Seguridad***. *XenMan* mantiene comunicaciones seguras con los servidores debido al uso de *túneles SSH*.

- ***Repositorio de imágenes***. Permite almacenar imágenes de los sistemas operativos de nuestros dominios con el fin futuro de distribuirlas a lo largo de nuestra infraestructura. Así, un aprovisionamiento transparente para el usuario es posible.

- ***Fácil, gratuito y libre***.

***XenMan* debe ser instalado en la máquina desde la cual queremos realizar todas las operaciones**, normalmente un equipo cliente con *servidor X*, aunque también es posible que queramos hacerlo localmente para administrar la plataforma virtual de nuestro servidor local.

A continuación mostramos el **proceso de instalación de *XenMan* sobre la distribución Debian GNU/Linux**. En primer lugar debemos preparar Xen para **cubrir las necesidades de conectividad de *XenMan* editando la siguiente línea en el fichero de configuración del *demonio xend xend-config.sxp***:

```
(xend-tcp-xmlrpc-server yes)
```

Además **debemos disponer del directorio */etc/xen/auto*** –que como sabemos almacena archivos de configuración para los dominios que serán arrancados/apagados automáticamente en la puesta en marcha/apagado del sistema anfitrión-, de lo contrario debemos crearlo:

```
deb1:~/downloads# mkdir /etc/xen/auto
```

También es necesario asegurar que las comunicaciones mediante *SSH* como *root* al servidor son posibles. A continuación **instalamos desde repositorios la librería *python-paramiko***; es una librería de cliente *SSH* que *XenMan* utiliza para la administración de forma remota:

```
deb1:~/downloads# apt-get install python-paramiko
```

Ya estamos en disposición de **comenzar con la instalación propiamente dicha de *XenMan 6.0***. En primer lugar **descargamos el fichero comprimido con las fuentes** de la herramienta de la web http://sourceforge.net; **descomprimimos el fichero y accedemos al directorio creado**:

```
deb1:~/downloads# wget
http://ovh.dl.sourceforge.net/sourceforge/xenman/xenman-0.6.tar.gz

deb1:~/downloads# tar –xzf xenman-0.6.tar.gz

deb1:~/downloads# cd xenman-0.6
```

Después **ejecutamos el script *mk_image_store*** para crear una copia local del almacén de imágenes, que contiene todas las imágenes de dominios preparadas para ser instaladas:

```
deb1:~/downloads/xenman-0.6# sh ./mk_image_store
```

El siguiente paso consiste en crear un enlace débil para solventar un problema que presenta *XenMan* en Debian, y es que necesita acceder las *librerías Python* instaladas por Xen pero en Debian éstas se encuentran ubicadas en un directorio diferente al esperado:

```
deb1:~/downloads/xenman-0.6# ln –s /usr/lib/xen-3.2-1/lib/python
/usr/lib/python
```

Finalmente solucionamos también un *bug* presente en la versión de *XenMan* que estamos instalando para Debian (ocurre lo mismo en Ubuntu) para el cual existe el parche *xenman_0.6_xen_3.1_patch_v1.txt* que aplicamos en el directorio en el que está localizado el fichero *XenDomain.py*, */camino/a/xenman-0.6/src:*

```
deb1:~/downloads/xenman-0.6# cd src

deb1:~/downloads/xenman-0.6/src# patch < ./xenman_0.6_xen_3.1_patch_v1.txt
```

El parche puede ser descargado en el *link*
http://sourceforge.net/tracker/index.php?func=detail&aid=1738580&group_id=168929&atid=848458.

Ya **podemos iniciar *XenMan* como usuario *root* si ha sido instalado localmente y necesitamos acceder al servidor local o por el contrario como cualquier otro usuario desde el directorio** */camino/a/**xenman-0.6***:

```
deb1:~/downloads/xenman-0.6# ./XenMan
```

En la figura 3.30 podemos ver el **estado inicial que presenta la aplicación al ser arrancada por primera vez**. En la parte izquierda podemos ver una cola de servidores con máquinas virtuales, a la que por defecto es añadido el servidor *localhost*, y una lista con las imágenes de sistemas operativos huésped que contiene el almacén de imágenes de *XenMan*. Si pulsamos sobre un servidor podemos ver en la parte central la información relativa al mismo así como un resumen de su estado. Como es habitual disponemos en la parte superior de una barra de menús y otra de herramientas.

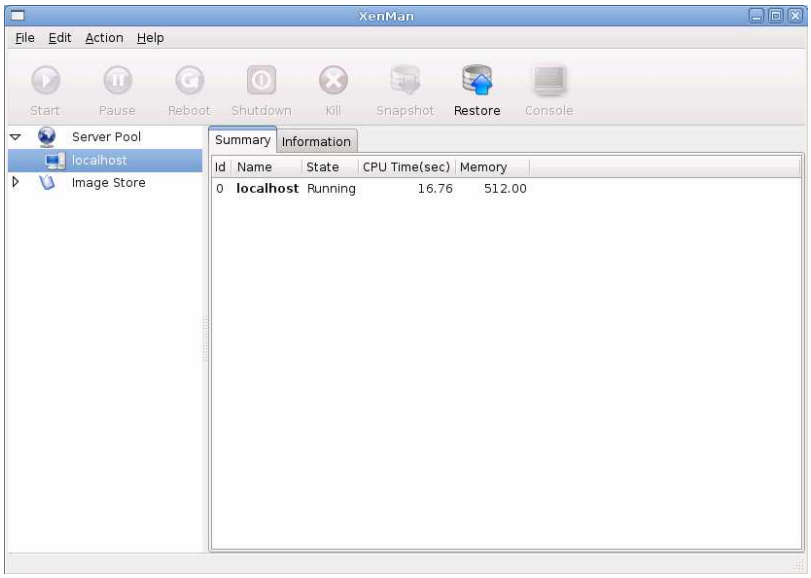


Figura 3.30 Estado inicial de la herramienta XenMan 6.0.

En la barra de menús disponemos de cuatro menús principales que nos darán acceso a todas las operaciones posibles a realizar con *XenMan*:

- ***File***. Permite dar de alta nuevos servidores a la cola, aprovisionamiento de máquinas virtuales, abrir y cargar archivos de configuración de nuevos dominios, iniciar/detener todos los dominios al mismo tiempo, eliminar un dominio/servidor/archivo de configuración, y salir de la aplicación.

- ***Edit***. Permite configurar el dominio editando su fichero de configuración o cambiando los parámetros del mismo en un formulario.

- ***Action***. En él disponemos de las diferentes acciones para cada dominio, iguales a las mostradas en la barra de herramientas: *iniciar, pausar, reiniciar, apagar, eliminar, tomar una imagen, restaurar,* o *acceder a la consola.*

- ***Help***. Desde aquí tenemos acceso a la ayuda proporcionada por *XenMan* y a diversa información sobre la versión instalada.

En cuanto a la barra de herramientas, hay ocho botones para un acceso rápido a las operaciones más habituales con dominios (algunas de ellas deshabilitadas para los dominios *dom0 – servidores* ya que, por ejemplo, no tiene sentido apagarlo o iniciarlo), las mismas que existen en el menú *Action.*

Al añadir archivos de configuración de máquinas virtuales ya las tenemos disponibles bajo el servidor al que pertenecen. En la figura 3.31 es posible apreciar cómo dos dominios han sido añadidos e iniciados en nuestro servidor local, reflejado también en su estado. Además, la lista de imágenes en el almacén ha sido desplegada:

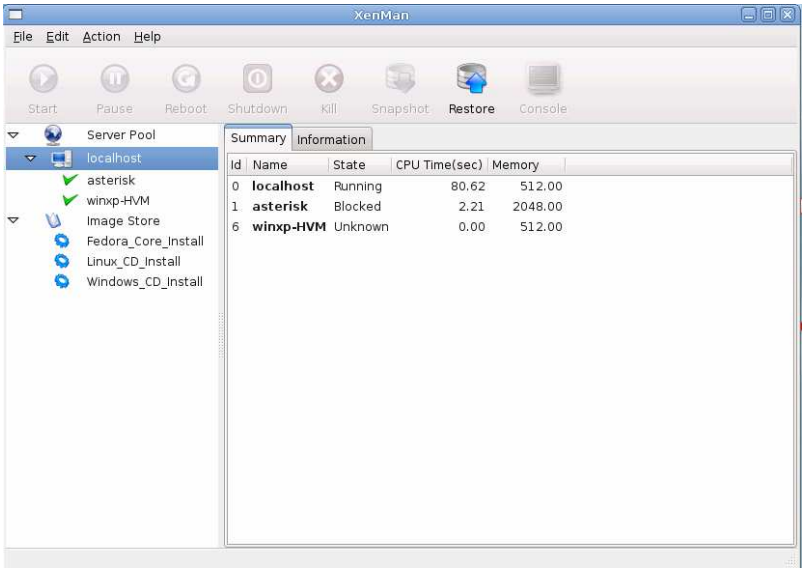


Figura 3.31 Consola general de XenMan con dos dominios añadidos e iniciados.

Si hacemos clic con el botón derecho del ratón sobre un dominio accedemos a las operaciones más habituales además de a su configuración, bien editando directamente el fichero de configuración (véase la figura 3.32) o bien cambiando los parámetros en formularios presentados en diversas pestañas (figura 3.33).

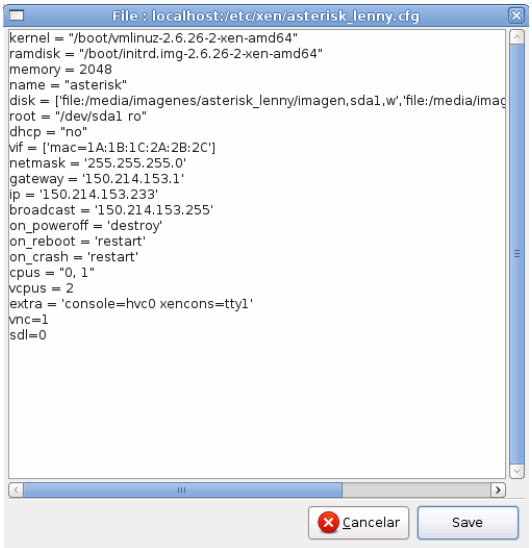


Figura 3.32 Edición del fichero de configuración de un dominio con XenMan.

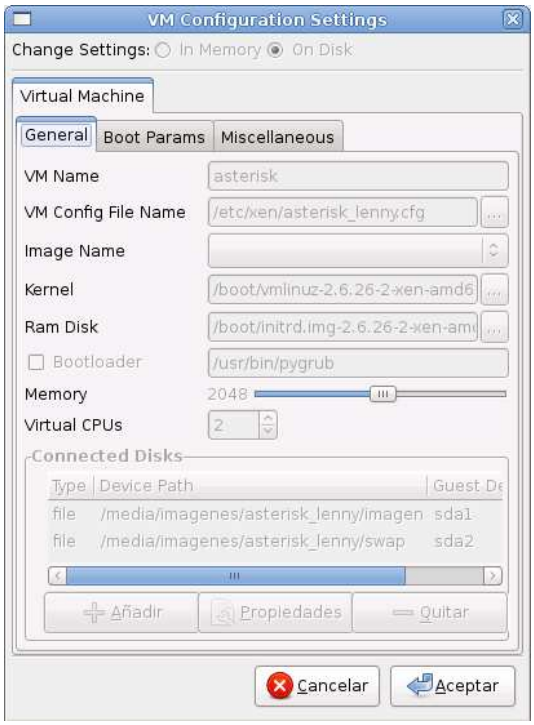


Figura 3.33 Edición de la configuración de un dominio en VM Configuration Settings.

Para terminar con *XenMan* es importante saber que **en el directorio** */camino/a/**xenman-0.6/doc*** **disponemos de diversa documentación tanto para su instalación como para la configuración de *XenMan* en el fichero** */camino/a/xenman-0.6/**xenman.conf*** **y así adaptarlo completamente a nuestras necesidades**.

5.4.2 Virtual Machine Manager

Virtual Machine Manager o ***virt-manager*** (http://virt-manager.et.redhat.com/) **es una aplicación escrita en Python y distribuida bajo licencia GPL que consiste en una interfaz gráfica de usuario para la administración de máquinas virtuales, no sólo en Xen, sino también en Qemu o KVM**. Cuando la ejecutamos podemos acceder a un listado de los dominios actualmente activos en nuestra plataforma, así como su rendimiento y consumo de recursos como procesador y memoria en tiempo real. Además **dispone de asistentes gráficos para la creación de nuevos dominios o la edición de la configuración de los ya existentes** (*hardware virtual*, recursos asignados al dominio…). **También incluye un *cliente VNC embebido*** de forma que podemos acceder a las consolas gráficas de los dominios de una forma sencilla.

Figura 3.34 Página web de Virtual Machine Manager: http://virt-manager.et.redhat.com/.

Es recomendable visitar su sitio web ya que proporciona mucha información para los procesos de instalación y configuración, así como interesantes tutoriales relacionados con las actividades que permite y una *wiki*. Veamos a continuación cómo instalar *Virt-Manager*; antes **instalamos la utilidad *virtinst*, de la que hará uso para implementar muchas de sus operaciones, y que a su vez incluye tres herramientas**:

- ***Virt Install:*** una herramienta en línea de comandos para proporcionar sistemas operativos a los dominios, ofreciendo una API a *Virt-Manager* para su asistente gráfico de creación de máquinas virtuales.

- ***Virt Clone:*** utilidad también en línea de comandos para la clonación de sistemas invitados inactivos, copiando las imágenes de disco y definiendo una configuración con nuevo nombre, UUID y dirección MAC.

- ***Virt Image:*** para la instalación de sistemas operativos en los dominios basándonos en imágenes predefinidas.

Antes de instalar *virtinst* en su versión 0.500.2 (que es la requerida como mínimo para la versión de *Virt-Manager* que queremos instalar, 0.8.3) **debemos tener instalada la librería *libvirt*** (versión 0.2.1 o superior), **el paquete para el lenguaje *python*** (versión 2.4 o 2.5) **y la herramienta *virt-viewer*** (0.0.1 o superior, interfaz para la interacción con la consola gráfica del sistema virtualizado). **Cumplidos estos requisitos procedemos con la descarga de *virtinst* de la página de *Virt-Manager*, lo descomprimimos, accedemos al directorio creado en la descompresión y ejecutamos el script *python setup.py* para realizar la instalación**:

```
deb1:~/downloads# wget http://virt-
manager.et.redhat.com/download/sources/virtinst/virtinst-
0.500.2.tar.gz

deb1:~/downloads# tar xfvz virtinst-0.500.2.tar.gz

deb1:~/downloads# cd virtinst-0.500.2

deb1:~/downloads/virtinst-0.500.2# python setup.py install
```

Con *virtinst* instalado ya podemos **instalar *Virt-Manager 0.8.3***. Para ello **descargamos el código fuente de la web oficial de la herramienta** (vista anteriormente), **el cual descomprimimos para después acceder al directorio que ha sido creado**:

```
deb1:~/downloads# wget http://virt-
manager.et.redhat.com/download/sources/virt-manager/virt-manager-
0.8.3.tar.gz

deb1:~/downloads# tar xfvz virt-manager-0.8.3.tar.gz

deb1:~/downloads# cd virt-manager-0.8.3
```

Dentro del directorio **configuramos la instalación ejecutando *./configure*, para después compilar (*make*) e instalar (*make install*)**:

```
deb1:~/downloads/virt-manager-0.8.3# ./configure

deb1:~/downloads/virt-manager-0.8.3# make

deb1:~/downloads/virt-manager-0.8.3# make install
```

Tanto *virtinst* como *Virt-Manager* han sido creados por y para RedHat y Fedora, aunque como es indicado en la documentación propia de las aplicaciones no debe haber problema para su instalación en otras distribuciones. En el caso concreto de Debian, por ejemplo hubo que renombrar el directorio */usr/lib/xen-3.2-1* como */usr/lib/xen*, directorio estándar usado por aplicaciones como *Virt-Manager* para las librerías Xen de usuario (como consecuencia de este cambio además tuvo que ser reinstalado el paquete *xen-utils*). Finalmente para **satisfacer las necesidades de conectividad de *Virt-Manager* con Xen cambiamos la siguiente línea en el fichero de configuración del demonio *xend xend-config.sxp***:

```
(xen-unix-server yes)
```

Reiniciamos el demonio *xend* (*/etc/init.d/xend restart*), y ya podemos arrancar *Virt-Manager*, para lo que basta con ejecutar el siguiente comando como usuario privilegiado *root*:

```
# virt-manager
```

En la figura 3.35 podemos ver cómo se presenta la aplicación inicialmente, mostrando como único dominio registrado *dom0* ejecutándose en el servidor *localhost*.

Figura 3.35 Aspecto inicial de la aplicación Virt-Manager.

Si hacemos clic en el botón *Nueva* de la barra de herramientas accedemos a una de las grandes posibilidades de *Virt-Manager* como es el **asistente de creación de dominios** (véase la figura 3.36).

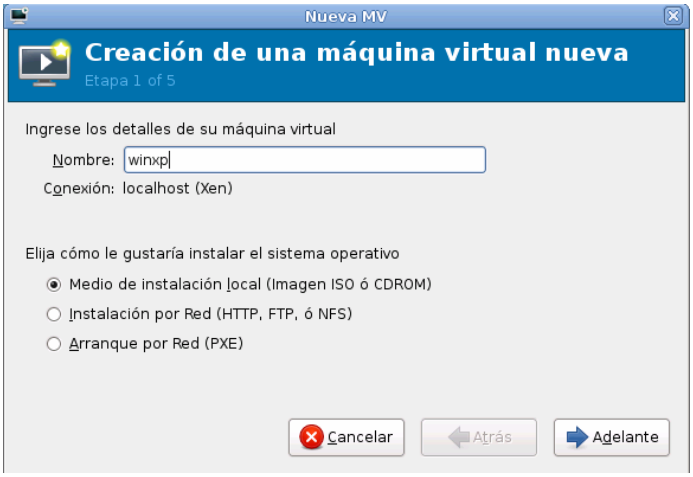


Figura 3.36 Paso inicial en el asistente para la creación de una máquina virtual nueva.

El asistente nos guia por todo el proceso de creación, en el que deberemos dar un nombre al dominio, elegir el tipo y versión así como el procedimiento de instalación del sistema operativo, seleccionar cuál será el almacenamiento para el dominio (pudiendo crear uno nuevo –completo desde el inicio o dinámico- o utilizar uno ya existente –por ejemplo creado con *dd*), configurar los parámetros del adaptador de red, el tipo de virtualización a aplicar y arquitectura. Si durante el asistente además elegimos caracterizar aún más la configuración **antes de la instalación podemos**, como se muestra en la figura 3.34, **configurar detalladamente el dominio al completo** (*acpi* y *apic* en la máquina, reloj, procesador, memoria, opciones de arranque, disco, interfaces de red, opciones para las consolas serie y gráfica…).

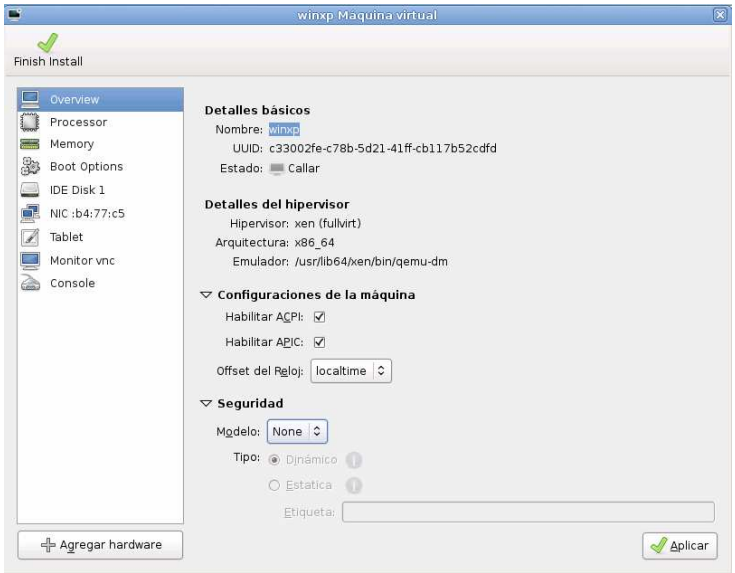


Figura 3.37 Configuración del dominio justo antes de finalizar la instalación.

Además, **si lo consideramos necesario podemos añadir nuevos dispositivos hardware a nuestro nuevo dominio para lo cual también disponemos de otro asistente** (véase la figura 3.38). Esta operación también es posible una vez creado el dominio. **Una vez creado el dominio éste se encuentra disponible en la consola central de administración**, al igual por ejemplo que otros dominios que hayan sido iniciados con la interfaz administrativa *xm* de Xen, **junto a gráficos sobre consumo actual de CPU y tráfico de de E/S de disco y red si**

así lo hemos configurado (ver figura 3.39). Desde esta consola podemos realizar todas las operaciones posibles sobre las máquinas virtuales haciendo uso de los menús y los botones de la barra de herramientas.

Figura 3.38 Asistente para la adición de nuevo hardware virtual al dominio.

Figura 3.39 Gráficos de rendimiento y consumo actual en la consola Virt-Manager.

Haciendo un repaso de las diferentes **opciones disponibles en los menús del *Administrador de Máquinas Virtuales*** resumiremos la funcionalidad que éste ofrece:

- ***Archivo***. **Permite la restauración de máquinas virtuales guardadas, añadir una nueva conexión** (a un nuevo *hipervisor Xen*, Qemu o KVM en otro servidor), cerrar y salir.

- ***Editar***. **Desde aquí podemos ver los detalles del equipo anfitrión-*dom0*** (rendimiento, redes virtuales, almacenamiento, interfaces de red,… véase la figura 3.40), **los detalles y configuración de la máquina virtual seleccionada** (figura 3.41) **y las preferencias en la configuración del administrador** (tiempo para la actualización de los estados, numero de muestras a mantener en el historial, si habilitar sondeo de estadísticas de E/S de disco y red, detalles sobre las consolas de las maquinas virtuales o la retroalimentación de las diferentes operaciones al ser ejecutadas).

- ***Vista***. **Permite seleccionar los diferentes gráficos a mostrar en la consola principal** junto al listado de dominios: utilización cd CPU, E/S de disco, E/S de red.

- ***Ayuda***. Ver los **detalles de la versión** *de Virt-Manager* instalada (0.8.3): créditos, licencia, etc.

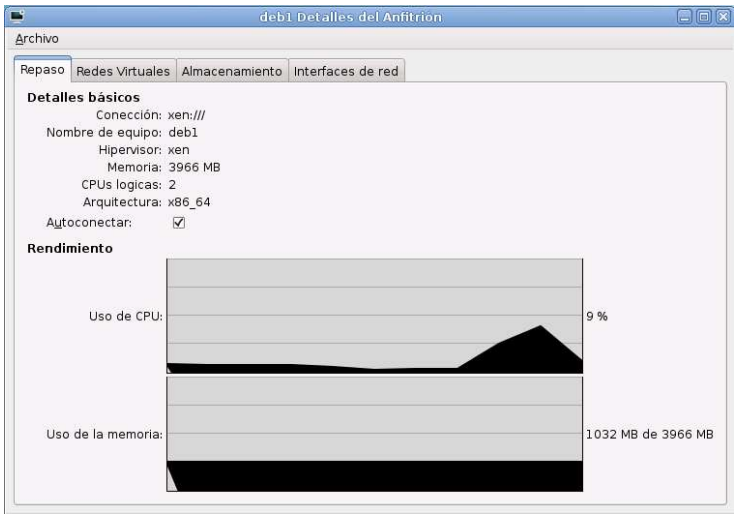


Figura 3.40 Detalles del equipo anfitrión.

En cuanto a la barra de herramientas, los botones que recoge nos permiten acceder de una manera rápida y con pocos *clics* a manipular el estado de los dominios: *Nueva, Abrir, Ejecutar, Pause y Apagar (Reiniciar, Apagar y Forzar apagado).* Si pulsamos **en *Abrir* accedemos a una ventana que recoje los detalles de la máquina virtual** (véase la figura 3.41) **y la edición de su configuración o bien su consola gráfica o serie** (figura 3.42).

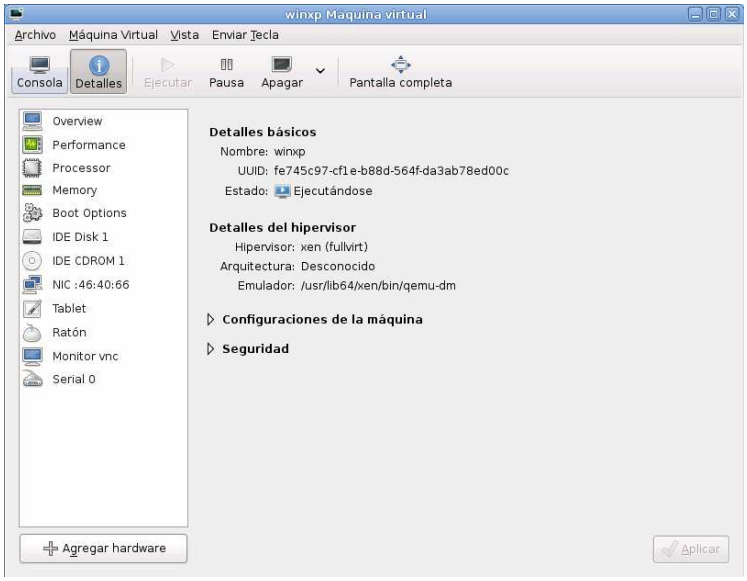


Figura 3.41 Detalles y configuración de la máquina virtual.

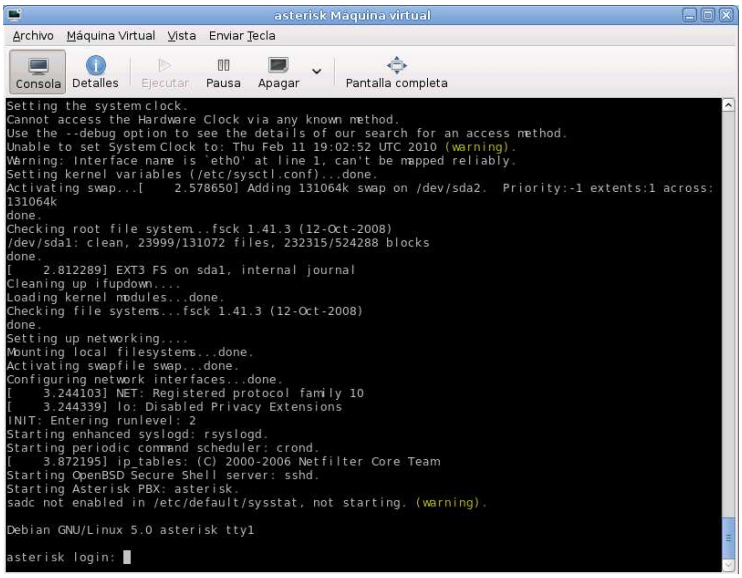


Figura 3.42 Consola serie de una máquina virtual en Virt-Manager.

Además de la barra de herramientas con las operaciones ya comentadas a realizar sobre la máquina virtual (con botones adicionales para visualizar la consola en pantalla completa y cambiar entre la vista consola y detalle) **aparecen en esta ventana nuevos menús que nos brindan acceso a las siguientes tareas**:

- ***Archivo***. Desde aquí podemos **volver al administrador** (consola principal de la aplicación), cerrar y salir.

- ***Máquina Virtual***. ***Ejecutar, Pausa, Apagar, Guardar, Clonar,*** e incluso operaciones avanzadas como ***Migrar*** **y** ***Tomar foto***.

- ***Vista***. **Permite permutar la visualización de la ventana entre consola y detalles** (desde ésta última configuraremos la máquina virtual, pudiendo añadir nuevo hardare), **establecerla en pantalla completa, escalar el monitor, cambiar entre consola serie y gráfica** (Serial 0 y gráfica VNC), y mostrar/ocultar la barra de herramientas.

- ***Enviar Tecla***. Permite **enviar al dominio la pulsación de determinadas teclas especiales**, como las *teclas F*, *retroceso*, *suprimir*, *imprimir pantalla…*

Para finalizar con *Virt-Manager* recordamos **y comprobamos que la conexión mediante un cliente VNC a la consola gráfica de un dominio correctamente configurado es posible en la dirección 127.0.0.1:5900** (usada por defecto al crear un dominio en Xen y *Virt-Manager*). En la figura 3.43 podemos apreciar una conexión VNC a la consola gráfica de un *dominio HVM* creado con *Virt-Manager* con el sistema operativo Windows XP instalado.

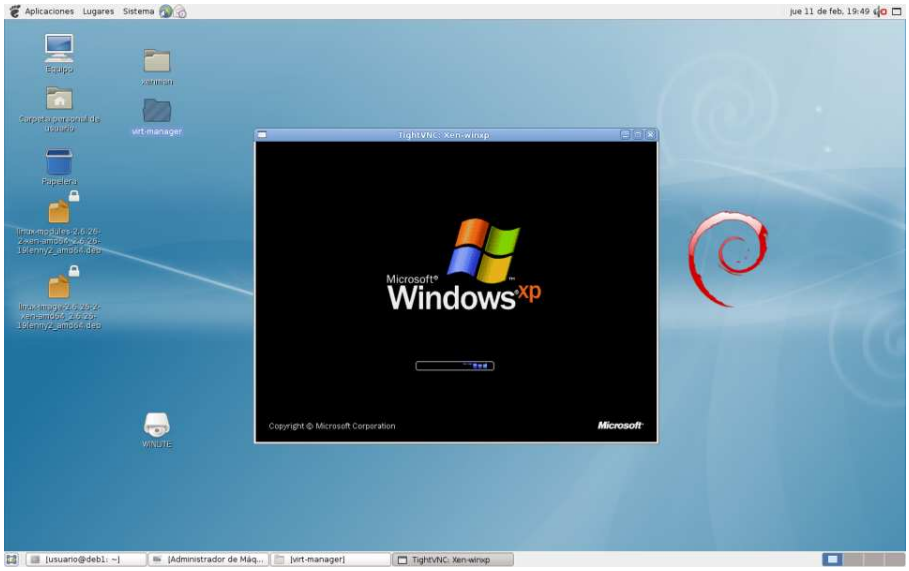


Figura 3.43 Conexión VNC a la consola gráfica de un dominio HVM creado con Virt-Manager.

5.4.3 Nagios

Una vez que un dominio es arrancado podemos operar sobre él como si se tratara de un equipo físicamente disponible, por lo que también es posible utilizar herramientas de monitorización generales (no diseñadas especialmente para virtualización) contra ellos. **Tanto *Nagios* como *MRTG*** (en el apartado siguiente) **son dos herramientas fundamentales en el campo de la monitorización** y es por ello que son presentadas a continuación.

Nagios (http://www.nagios.org), **anteriormente llamando *Netsaint*, es un sistema libre de monitorización completa de redes, equipos y servicios**. Podemos acceder a información en tiempo real de todo tipo, por ejemplo sobre servicios de red parados o sobre el consumo de los recursos en sistemas hardware (carga del procesador, uso de memoria, disco, estado de los puertos de comunicaciones, tarjetas de red…). Permite además operar independientemente del sistema operativo del sistema objetivo así como monitorizar de forma remota mediante el uso de *túneles SSH o SSL cifrados*.

Figura 3.44 Web http://www.nagios.org/.

Con *Nagios* es posible definir la generación de alertas y notificaciones así como manejadores de eventos para cuando éstos tengan lugar y por lo tanto resolverlos de forma proactiva. Como es de esperar *Nagios* facilita el estado de la red en tiempo real a través de su **interfaz web, mostrando informes, gráficos de comportamiento de los sistemas que estemos monitorizando, y listados de las notificaciones emitidas, alertas, problemas, registros, etc.** Debido a todas estas características que posee y aunque recientemente suene fuerte el nombre ***Icinga***, *Nagios* **es considerada en la actualidad como la herramienta libre de monitorización para sistemas Unix/Linux más completa**.

5.4.4 MRTG

MRTG* o *Multi Router Traffic Grapher (http://oss.oetiker.ch/mrtg/) **es una herramienta libre distribuida bajo licencia GPL** escrita en C y Perl por Tobias Oetiker y Dave Rand **que se utiliza para la supervisión de la carga de tráfico de interfaces de red en diferentes equipos**. Permite operar sobre sistemas tanto Unix/Linux como Windows o incluso *Netware*. **MRTG está basado en *SNMP* *(Simple Network Management Protocol)* para la recolección de la información relativa al servidor o *router***, por lo que es indispensable contar en el sistema monitorizado con *SNMP* correctamente configurado y operativo.

Figura 3.45 Logotipo de MRTG.

Puede generar **diferentes tipos de informes y gráficos embebidos en páginas HTML** que proveen de una representación visual idónea para la supervisión del tráfico con respecto al tiempo (véase la figura 3.43). Es **ejecutado como un *demonio* o bien planificado con *cron***, tomando muestras de valores con intervalos de tiempo especificados en su archivo de configuración mediante la ejecución de determinados scripts. En las últimas versiones toda la información procesada por *MRTG* es almacenada en una base de datos a partir de la cual son generados los informes y gráficas.

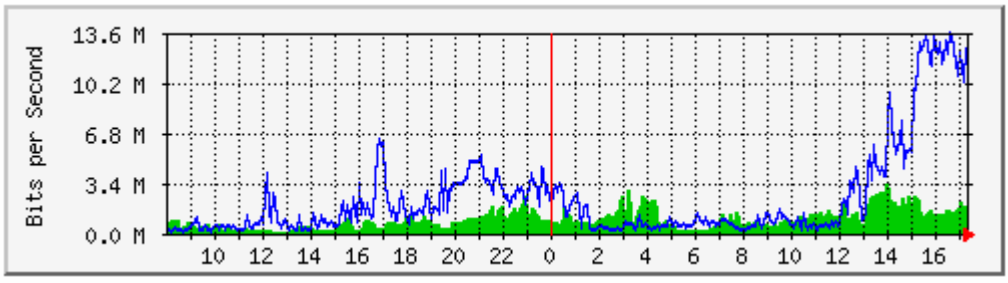


Figura 3.46 Gráfico tipo para la evolución del tráfico de interfaces en tiempo real generado por MRTG.

Como podemos ver *MRTG* se convierte en una herramienta especialmente interesante para el caso en el que dispongamos de **dominios con servicios que registran una gran actividad en sus interfaces de red**.

CAPITULO 4

INTRODUCCION A LA TELEFONIA IP Y ASTERISK

En este capítulo cuarto comenzamos a conocer en qué consisten los servicios de **telefonía IP**. La voz IP o *VoIP (Voice over IP)* permite el transporte de tráfico de voz y video sobre redes IP; para ello, los contenidos de audio y vídeo que intercambian los locutores son transportados divididos en pequeños fragmentos en origen, para ser reemsamblados después en destino. La telefonía IP está tomando un protagonismo especial en los últimos años, gracias como no a la explosión de Internet, debido a las relevantes ventajas que proporciona su uso respecto a los sistemas de telefonía y centralitas tradicionales, con las que es compatible.

Como hemos comentado con anterioridad en este mismo documento uno de los objetivos del presente proyecto es la virtualización de servicios de telefonía IP (consolidación de servidores centralita VoIP). De todas las soluciones de telefonía IP existentes en la actualidad una destaca sobre el resto por su excelente funcionalidad y eficiencia: ***Asterisk***. *Asterisk* nació de la necesidad de disponer de un sistema telefónico *software* de bajo coste, lo que supuso una revolución en el mundo de las comunicaciones, hasta ese momento basados en sistemas *hardware*, menos flexibles y escalables.

Sirva este capítulo como pequeña introducción a esta tecnología y su personificación en el uso de *Asterisk*.

1 La telefonía IP

La telefonía IP (*Voice over Internet Protocol*), nacida de la necesidad y la utilidad de hacer uso de una sola red para la transmisión de datos en lo que a telefonía se refiere, es la **comunicación que viaja a través de la red, tanto de audio como de video, permitiendo un ahorro elevado tanto en costes de comunicación como en costes de mantenimiento de las infraestructuras de telefonía**, **teniendo al mismo tiempo grandes posibilidades de escalabilidad, portabilidad, e integración con las redes locales internas y recursos de las empresas e Internet.**

Como vamos a ver a continuación, la mayoría de las ventajas de las que nos podemos beneficiar al usar telefonía IP nacen del hecho de que sea **una *implementación software* de centralitas de telefonía**. El abanico de características y posibilidades es muy amplio, lo que unido al bajo coste hace que disponga de una gran atención e interés.

La inquietud por la tecnología que hoy es conocida mundialmente como *VoIP* comenzó durante la década de los noventa, aunque también existen publicaciones anteriores, cuando investigadores de varias instituciones se interesaron en este tipo de comunicaciones, en la transmisión de datos de audio y video sobre redes IP. A grandes rasgos la comunicación que tiene lugar en *VoIP* consiste en la fragmentación de datos de audio y vídeo, su transmisión, y la unión de esos fragmentos en destino.

Desde la aparición en el año 1995 del primer *teléfono software* la evolución que ha vivido esta tecnología ha tenido algunos altibajos. Al principio, sobre todo porque las redes de transmisión de datos no disponían de un ancho de banda suficiente, no tuvo ningún éxito a nivel comercial. Fue al final de esta década de los noventa cuando al posibilitar el uso de teléfonos tradicionales en entornos *VoIP* un gran número de empresas se lanzaron a la comercialización de productos y servicios *VoIP*. El porcentaje de uso de la *VoIP* en términos de tráfico de voz aumentaba paulatinamente, en parte impulsado por la creación de las primeras plataformas empresariales para trabajar con *VoIP* desarrolladas por una empresa como Cisco Systems. Todo ello, unido a la gran mejora experimentada en las redes de transmisión de datos y la eclosión de Internet, garantizaba los requisitos fundamentales en cuanto a calidad y fiabilidad de las comunicaciones de voz.

Se desarrollaron entonces gracias al impulso vivido por la *VoIP* numerosas soluciones de centralitas telefónicas software, entre las que destaca sin duda alguna *Asterisk*, que vemos en el siguiente apartado de este capítulo.

La telefonía IP también puede plantear diversas desventajas con su uso, aunque la importancia y número de éstas no es comparable a la importancia y número de ventajas que presenta.

Los **principales beneficios** que podemos obtener al aplicar *VoIP* son*:*

- ***Reducción de costes***: los costes que tendremos que asumir usando *VoIP* son realmente bajos fundamentalmente debido al uso de Internet como medio de transporte de las comunicaciones. El pago de llamadas quedará reducido al pago de la conexión a Internet de la que dispongamos. Los costes asociados también a la creación, desarrollo y mantenimiento de la infraestructura telefónica de la empresa se verán disminuidos de manera importante. Además, también podemos disfrutar del uso de *teléfonos software* en lugar de *hardware*, más caros.

- ***Escalabilidad****:* el servicio de *VoIP* dispone de una gran escalabilidad, tanto si hablamos de los servidores de telefonía como del número de usuarios soportados

por el servicio. Sobre todo es debido a la posibilidad de instalar el servicio de forma distribuida (también es posible realizarlo de forma centralizada), más flexible y dispuesto a asumir futuras mejoras tecnológicas.

- ***Portabilidad**:* la telefonía IP también es un servicio altamente portable al mismo nivel como lo pueden ser otros que trabajan sobre Internet. La *VoIP* no limita la movilidad de sus usuarios, permitiendo el uso de sus servicios dondequiera que se encuentren. Por ejemplo, un empleado puede realizar llamadas, revisar su buzón de voz, o acceder al servicio de fax virtual a través de la intranet de su empresa desde su domicilio.

- ***Integración con redes locales y recursos existentes e Internet**: VoIP* trabaja sobre las redes TCP/IP, luego su integración con cualquier red local de una empres u organización es sencilla. Ello aporta beneficios considerables al eliminar las redes telefónicas tradicionales: no se necesita extender la red telefónica de la empresa al disponer ya de conexiones de red *Ethernet*.

- ***Mayor eficiencia***. La telefonía IP permite aprovechar mejor el ancho de banda disponible, permitiendo que al mismo tiempo que tenemos una conversación podamos utilizar el canal de comunicación para otros objetivos. Esto es más cierto cuando al aplicar compresión o eliminación de redundancia en la transmisión de voz.

- ***Características y prestaciones diferenciadoras***: la *VoIP* ofrece funcionalidades y dispone de características y servicios que la sitúan en posición ventajosa en el mercado de la telefonía y con los cuales no puede competir de forma alguna la telefonía tradicional, ya que aumenta el número de servicios y mejora de forma abrumadora la calidad y fiabilidad de los mismos. Buzones de voz, salas de conferencias (sin límite de participantes, grupos de personas…), distribución y gestión automática de llamadas, contestador automático, interacción con la red de telefonía tradicional, reconocimiento de llamadas, creación de números virtuales, transmisión de diferentes tipos de datos (video, imágenes… al trabajar en redes de paquetes), uso de fax virtuales (mejorando la calidad, a bajo coste, sin máquina fax), *IVR* o *respuesta interactiva de voz*… son algunas de las características más destacables.

  Además, la *VoIP* no se encuentra limitada al transporte de voz al basarse en una red de paquetes, por lo que puede manejar otros tipos de datos y comunicaciones: vídeo, texto, etc.

- ***Flexibilidad**:* las soluciones *VoIP* en general ofrecen una gran flexibilidad tanto en su configuración como en uso, sobre todo si hablamos de soluciones *software libre* como *Asterisk*, en cuyo caso podremos configurar y adaptar la solución perfectamente a las necesidades que tengamos y los objetivos que queramos alcanzar. Por ejemplo, unas de las características más importantes de *Asterisk* es la posibilidad que tenemos de crear y configurar un completo *plan de marcado* o *dial plan*, pudiendo establecer complejas y eficientes configuraciones de la lógica a seguir en el procesado de llamadas.

  Además, la red sobre la que trabaja *VoIP* es completamente flexible, no es necesario disponer de un escenario o topología de red determinado.

La telefonía IP puede ser configurada de forma avanzada asociada a otras utilidades para dotar de características como tolerancia a fallos, alta disponibilidad –

disponibilidad ininterrumpida- **o balanceo de carga** –para proporcionar alto rendimiento- **al servicio**. Dejando a un lado estas posibles configuraciones de mayor complejidad, y si el servicio es configurado de forma centralizada o distribuida, presentamos en la Figura 4.1 la arquitectura general que puede presentar la *VoIP*.

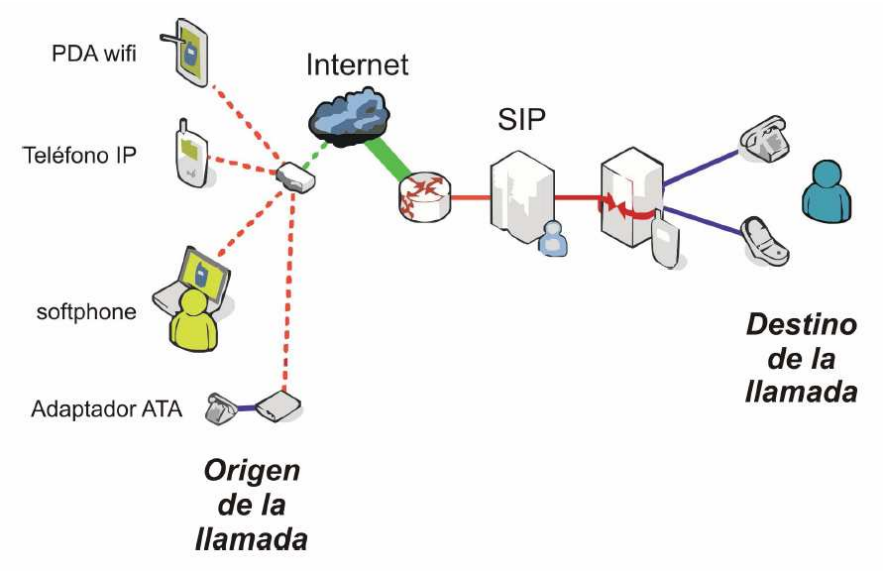


Figura 4.1. Arquitectura VoIP genérica.

En ella podemos apreciar la existencia de diferentes elementos fundamentales para poder comunicarnos y que constituyen **el origen y destino de las comunicaciones, como teléfonos IP (hardware), *softphones* o *teléfonos software* en computadores**, **adaptadores *ATA*** (para permitir la conexión de teléfonos tradicionales a la red IP), **o también por ejemplo *PDAs* con conexión inalámbrica**. Otros elementos son de extrema importancia en el funcionamiento del servicio *VoIP*, como **el protocolo *SIP* encargado de iniciar y finalizar las llamadas, y la entidad B2BUA (*back-to-back user agent*) intermediaria que procesa las comunicaciones *VoIP* y las retransmite a destino.**

La funcionalidad ofrecida por los **teléfonos IP** es mejorada a gran velocidad, pudiendo incluso disponer de teléfonos con transmisión de vídeo en tiempo real, con movilidad IP, para la realización de videoconferencias, etc. Disponen de al menos una conexión RJ-45 para poder ser conectados a la red (*proxy*, *router*, otro teléfono IP…). Los hay incluso con un *router* o *switch* integrado, dependiendo de la gama (existen de gama baja, media y alta).

Figura 4.2. Teléfono IP de la casa Linksys.

Los ***softphones* o *teléfonos software*** nos permiten hacer uso de la *VoIP* en dispositivos como ordenadores y PDAs, tan sólo disponiendo de una tarjeta de audio y micrófono instalados. Existen clientes *VoIP* para diferentes protocolos, aunque el más usado de todos es *SIP* (*Session Initiation Protocol*), por ejemplo por los conocidos *teléfonos software X-Lite* y *Zoiper* (este último también puede trabajar sobre *IAX*).

Figura 4.3. Softphone X-Lite.

Los ***adaptadores ATA*** *(Analog Telephone Adaptor)* o adaptadores de teléfonos analógicos son los dispositivos que llevan a cabo la conversión de las señales de las comunicaciones analógicas a un protocolo manejable por la *VoIP*. También realizan la conversión inversa: pasan la señal digital a analógica para que los teléfonos tradicionales la pueda procesar.

Para que la comunicación entre los diversos dispositivos sea realizada con éxito debe mediar un determinado **protocolo de señalización**. Esto, en las redes analógicas para telefonía tradicional consiste en una reserva del medio previa a la comunicación. En las redes de conmutación de paquetes el objetivo es análogo, garantizando al mismo tiempo diversos parámetros de calidad. Los protocolos de señalización más utilizados en conmutación de paquetes son ***H323 y SIP***; este último lo vemos en el siguiente apartado.

1.1 EL PROTOCOLO SIP

El protocolo *SIP (Session Initiacion Protocol o* protocolo de inicio de sesiones*)* es el protocolo de señalización a nivel de aplicación encargado de realizar la **iniciación, modificación y terminación de sesiones multimedia interactivas** (comunicaciones de vídeo, audio, mensajería, etc.). La última revisión de *SIP* fue realizada en el RC 3261 de Junio de 2002. Su integración es completa con otros protocolos destinados a la administración de sesiones multimedia, como *RVSP (Resource Reservation Protocol* o protocolo para la reserva de recursos*), RTP (Real-Time Protocol* o protocolo para tiempo real*) o RTSP (Real-Time Streaming Protocol* o protocolo para *streaming* en tiempo real*).*

Cuatro tareas determinan la gran eficiencia en la aplicación de este protocolo:

- **Localización de usuarios.** La movilidad del usuario no es limitada al permitir conocer su ubicación en todo momento. Los usuarios son identificados para un determinado dominio mediante el ***URI** (Uniform Resource Identifier)* o **dirección *SIP*** asociada, que puede ser de la siguiente forma:

```
sip:usuario@dominio[:puerto]

sip:usuario@direccionIP[:puerto]
```

  El **dominio** es el nombre del **proxy SIP** que conoce la dirección de red para el terminal del usuario (por ejemplo, *ual.es* es el dominio del usuario *100@ual.es*, y conoce su dirección de red).

- **Negociación de los parámetros de la comunicación.** Puertos a usar, *códecs* o algoritmos de codificación y decodificación de audio, tipo de tráfico, etc.

- **Disponibilidad del usuario.** Antes de establecer la comunicación es posible conocer si un usuario se encuentra preparado para la misma o no.

- **Gestión de la comunicación.** La comunicación puede ser manipulada en todo momento: es posible modificar la sesión, finalizarla, conocer su estado, etc.

Para poder establecer una comunicación mediante *SIP* es necesario que varios elementos estén presentes en la misma:

- ***Agentes de usuario o User Agents.*** Trabajan con la señalización *SIP*.

  - ***UAC: User Agent Client.*** Realiza *peticiones SIP* y recibe *respuestas SIP* del *UAS* (por ejemplo un teléfono IP).

  - ***UAS: User Agent Server.*** Acepta *peticiones SIP* realizadas por el *UAC* y envía a éste las *respuestas SIP* correspondientes. Un teléfono IP también es un ejemplo de *UAS*, ya que acepta peticiones de inicio de comunicación enviadas otro teléfono IP que actúa como *UAC*. Un *servidor Proxy SIP* también es un *UAS*.

- ***Servidor Proxy.*** Se encarga de reenviar las *peticiones SIP* de un *UAC* a un *UAS* y las *respuestas SIP* correspondientes del *UAS* al *UAC*. Para ello traduce las direcciones *SIP* de *usuario@dominio* a *usuario@direccionIP*.

- ***Location Server.*** Acepta peticiones de registro por parte de los *UAC*. Guarda toda la información relativa a la localización física del *UAC* para permitir así su rápida

localización (traducciones *usuario@direccionIP* para las direcciones *SIP usuario@dominio*).

- ***Redirect Server.*** Su funcionamiento es similar al de los *proxies SIP*, con la diferencia de que al terminar la traducción de la dirección *SIP* informa al *UAC* que realiza la *petición SIP* para que sea él mismo quien la envíe al *UAS* destino.

- ***Back-to-back User Agent (B2BUA).*** Como dijimos anteriormente, es una entidad que media para el procesamiento de *peticiones SIP* entrantes por lo que actúa como un *UAS*, y responde a las mismas actuando como un *UAC* generando una nueva *petición SIP* a enviar a partir de la *petición SIP* original.

Los mensajes SIP intercambiados durante una comunicación *SIP* entre los diferentes elementos que acabamos de ver pueden ser categorizados en ***peticiones y respuestas***. Existen seis ***peticiones SIP*** diferentes: **INVITE** (para establecer una comunicación)**, ACK** (para el reconocimiento de la *respuesta SIP 200 OK* que acepta el inicio de una comunicación)**, BYE** (para finalizar la comunicación)**, CANCEL** (permite cancelar una petición anterior en progreso)**, OPTIONS** (para que un *UAC* realice una solicitud de cierta información a un *UAS*) **y REGISTER** (las envían los *UAC* a un servidor de localización para informar de la posición actual). Cada petición tiene asociada una ***respuesta SIP*** enumerada con un código desde 100 a 699, agrupadas en ***grupos de respuestas*****:**

- ***1xx***. Indican el **estado temporal de la comunicación**.
- ***2xx***. Para informar sobre el **éxito de una *petición SIP*.** La más conocida es la correspondiente al éxito en el establecimiento de la comunicación con INVITE: *200 OK*.
- ***3xx***. Para informar de que **una *petición SIP* ha de ser reenviada a otro *UAS***.
- ***4xx***. **Errores en el *cliente SIP***.
- ***5xx***. **Errores en el *servidor SIP***.
- ***6xx***. **Errores generales**.

Para terminar, y antes de comenzar a hablar de *Asterisk*, en la Figura 4.4 podemos ver un ejemplo de establecimiento de comunicación mediante *SIP* entre dos terminales.

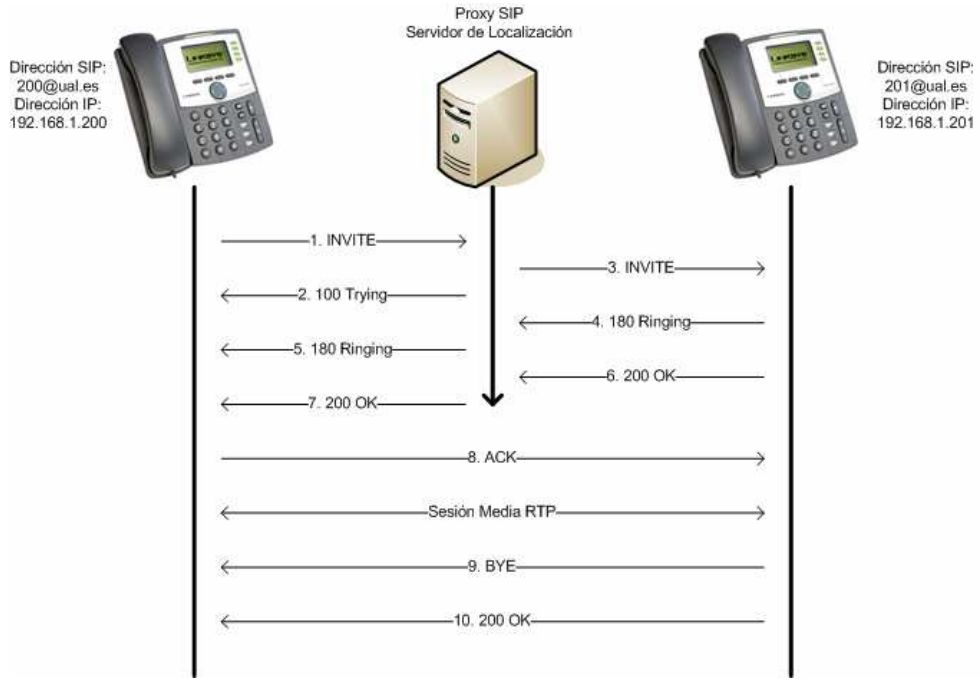


Figura 4.4. Proceso de establecimiento de llamada entre dos terminales.

2 Asterisk

Asterisk es la primera *PBX (Private Branch Exchange o central secundaria privada automática*-central telefónica*) software* ideada y desarrollada por Mark Spencer. Al tratarse de *software* la flexibilidad y escalabilidad de esta solución es superior a la de las tradicionales centralitas *hardware*. *Asterisk,* además, al confluir con el proyecto *Zapata Telephony* logró unir la telefonía tradicional y la *VoIP*.

Asterisk (http://www.asterisk.org/) es distribuido como *software libre* bajo la licencia *GPL (General Public License)*. Las funcionalidades que brinda esta solución, soportada por *Digium,* han provocado que sea actualmente el producto más extendido en diferentes ámbitos.

Figura 4.5. Logos de Asterisk y Digium: la solución de telefonía IP por excelencia.

Uno de los puntos fuertes de *Asterisk*, además de ser *software libre*, es su **arquitectura modular**. Esto permite que la aplicación sea completamente adaptable a las necesidades del usuario que ampliar su funcionalidad sea sencillo. Es decir, podemos habilitar o deshabilitar diferentes módulos si vamos a usarlos o no, pudiendo dedicar recursos a otras tareas si lo requerimos, y añadir o programar nuevos módulos para *Asterisk*. Los módulos que *Asterisk* debe cargar son especificados en el archivo ***modules.conf***; para ver los que actualmente se han cargado podemos ejecutar la orden siguiente en el intérprete en línea de comandos *CLI* de *Asterisk:*

```
localhost*CLI> module show
```

Figura 4.6. Página Web de Asterisk: http://www.asterisk.org/.

Asterisk, que a fecha de redacción de esta memoria se encuentra en la versión 1.6.1, proporciona **características que le brindan un gran potencial**; por ejemplo, buzón de voz, salas de conferencias, distribución y gestión automática de llamadas, contestador automático, interacción con la red de telefonía tradicional, reconocimiento de llamadas, posibilidad de creación de números virtuales, entre otras. Todo ello, y sobre todo al bajo coste que supone, hace pensar que reemplazará, sino en su totalidad prácticamente, a la telefonía tradicional implantada en empresas, instituciones, hogares…

2.1 INSTALACION

A continuación vamos a realizar la instalación paso a paso de *Asterisk* en la distribución Debian GNU/Linux 5 Lenny. Para la instalación de *Asterisk* en cualquier otra distribución Linux no debemos tener problema alguno operando de forma análoga, usando los comandos equivalentes a los expuestos.

Existen diferentes paquetes disponibles dependiendo del tipo de instalación que queramos realizar; por ejemplo, si tan sólo queremos instalar *Asterisk* (sin ninguna interacción con líneas de telefonía analógicas o digitales) es suficiente el paquete *asterisk* (y posiblemente el paquete *asterisk-addons*, que proporciona módulos adicionales a la instalación básica de *Asterisk*). Éste es el tipo de instalación que vemos a continuación. Para el uso de tarjetas de comunicación analógica disponemos además del paquete *zaptel*, mientras que para tarjetas de comunicaciones digitales tenemos *libpri*.

Comenzamos por tanto con la instalación de las dependencias necesarias. Para ello ejecutamos el comando *apt-get install*:

```
apt-get install gcc-4.1 libncurses5-dev libssl-dev zlib1g-dev
libnewt-dev libusb-dev build-essential
```

Las dependencias instaladas se corresponden con las utilidades de compilación (*gcc*), *make*, librerías y cabeceras de C, y las librerías de desarrollo *OpenSSL*. Descargamos ahora la versión 1.4.28 del **paquete *asterisk*** desde la sección oficial de descargas en la Web

(http://downloads.asterisk.org/pub/telephony/asterisk/) que aunque no es la última sí es la más estable y probada:

```
# wget
http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-
1.4.28.tar.gz
```

Una vez descargado el paquete, lo descomprimimos con la utilidad *tar* e ingresamos en el directorio creado:

```
# tar xzf asterisk-1.4.28.tar.gz
# cd asterisk-1.4.28
```

Realizamos la configuración de la instalación:

```
# ./configure
```

Después ejecutamos el comando *make menuselect*. Así podemos ver los módulos que van a ser instalados y elegir los que queramos incluir, eliminar los que no, etc. Si vemos un módulo marcado con una 'X' quiere decir que hace falta alguna librería adicional para poder realizar su instalación (se nos indica cuál); instalada la librería faltante podemos volver a ejecutar *configure* y el módulo aparecerá marcado con un '*', por lo que podemos incluirlo sin problemas:

```
# make menuselect
```

Finalmente compliamos *Asterisk*, lo instalamos, lo incluimos en el arranque del sistema operativo e instalamos para terminar un conjunto de ficheros de ejemplo y así obtener una configuración inicial válida (aunque limitada) para poder a trabajar con *Asterisk*:

```
# make

# make install

# make config

# make samples
```

A la instalación del paquete *asterisk* sumamos **la instalación del paquete que contiene módulos adicionales *asterisk-addons***. Procedemos de manera similar; primero descargamos las fuentes del paquete para la versión de *Asterisk* que hemos instalado anteriormente:

```
# wget
http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-
addons-1.4.10.tar.gz
```

Descomprimimos e ingresamos en el directorio creado en la descompresión:

```
# tar xzf asterisk-addons-1.4.10.tar.gz
# cd asterisk-addons-1.4.10
```

Para terminar configuramos, compilamos e instalamos el paquete:

```
# ./configure

# make
```

```
# make install
```

Para comprobar que hemos instalado *Asterisk* correctamente podemos por ejemplo acceder a la interfaz de línea de comandos para la administración de *Asterisk*, *CLI*. Para ello antes debemos iniciar el servicio ejecutando el *script* disponible en el directorio */etc/init.d/* tras la instalación:

```
# /etc/init.d/asterisk start
```

Iniciado correctamente *Asterisk*, ejecutamos el comando *asterisk –r* para acceder al *CLI*, seguido de tantas letras *v* como nivel de detalle queramos en la salida mostrada en la manipulación del mismo:

```
# asterisk -rvvv
Asterisk 1.4.28, Copyright (C) 1999 - 2009 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show
warranty' for details.
This is free software, with components licensed under the GNU
General Public
License version 2 and other licenses; you are welcome to
redistribute it under
certain conditions. Type 'core show license' for details.
=========================================================================
=====
  == Parsing '/etc/asterisk/asterisk.conf': Found
  == Parsing '/etc/asterisk/extconfig.conf': Found
Connected to Asterisk 1.4.28 currently running on deb1 (pid = 25485)
Verbosity was 0 and is now 3
deb1*CLI>
```

2.2 PUESTA EN MARCHA

Una vez que *Asterisk* ha sido compilado e instalado una completa estructura de directorios se encuentra disponible para dar soporte a las distintas partes y funcionalidades de *Asterisk*. De entre todos los directorios disponibles destacan los siguientes:

- */etc/asterisk/*. Almacena los ficheros de configuración para *Asterisk.* Por ejemplo, el fichero *asterisk.conf* en el que es posible indicar la ubicación resto de directorios.

- */usr/lib/asterisk/modules/*. Contiene los archivos binarios correspondientes a los módulos que han sido incluidos y compilados junto a *Asterisk*.

- */var/lib/asterisk/*. Incluye diferentes ficheros y directorios de gran importancia con datos que usa *Asterisk*. Entre ellos, la base de datos de *Asterisk astdb*, de tipo Berkeley DB2.

Para la manipulación del estado del servicio podemos hacer uso como en la mayoría de los casos del *script* instalado en el directorio */etc/init.d/* al hacer *make config* y que permite iniciar de forma automática *Asterisk*:

```
# /etc/init.d/asterisk start

# /etc/init.d/asterisk stop

# /etc/init.d/asterisk restart
```

También lo podemos iniciar de forma manual de la forma siguiente, habiendo ingresado como usuario *root*, en modo *background*:

```
# asterisk
```

O también en primer plano, por lo que accedemos directamente a la interfaz de línea de comandos *CLI*:

```
# asterisk –c

CLI>
```

Para conectarnos a la consola de *Asterisk* si éste ha sido iniciado en segundo plano lo hacemos añadiendo el *flag –r* al comando *asterisk*, teniendo la posibilidad de añadir también tantas letras *v* como nivel de *verbose* (información detallada sobre el funcionamiento de *Asterisk*) y tantas letras *d* como nivel de *debug* (información para la depuración a bajo nivel del funcionamiento de *Asterisk*). Por ejemplo:

```
# asterisk -rvvvd
```

Si queremos abandonar *Asterisk* procedemos de forma diferente si lo hemos iniciado en primer o segundo plano. Para abandonar (y detener) *Asterisk* al haberlo iniciado en primer plano lo hacemos así:

```
CLI> stop now
```

En cambio, cuando lo hemos iniciado en segundo plano al abandonar *Asterisk* no es detenido:

```
CLI> quit
```

Tanto si arrancamos *Asterisk* en primer plano usando *–c* como si lo arrancamos en segundo plano y posteriormente nos conectamos a la instancia del mismo usando *–r* accedemos a su **intérprete de comandos *CLI*** *(Command Line Interpreter).* En la tabla 4.1 podemos ver los comandos más utilizados.

Tabla 4-1. Comandos más utilizados en el intérprete de comandos *CLI* de *Asterisk*

Finalidad	Comando
Mostar información sobre las aplicaciones	`CLI> core show applications` `CLI> core show application <nombre>`
Mostrar información sobre las funciones	`CLI> core show functions` `CLI> core show function <nombre>`

Mostrar, habilitar y deshabilitar módulos	`CLI> module show`
	`CLI> module load <nombre>`
	`CLI> module unload <nombre>`

Mostrar los usuarios SIP/IAX	`CLI> sip show peers`
	`CLI> sip show users`
	`CLI> iax2 show peers`
	`CLI> iax2 show users`

Mostrar los canales activos	`CLI> core show channels`
	`CLI> sip show channels`
	`CLI> iax2 show channels`

Iniciar, detener y reiniciar *Asterisk*	`# asterisk`
	`# asterisk –c`
	`# asterisk –r`
	`CLI> stop now`
	`CLI> stop when convenient`
	`CLI> stop gracefully`
	`CLI> restart now`

	CLI> restart when convenient CLI> restart gracefully
Acceder la ayuda	CLI> help CLI> help sip CLI> help sip show CLI> help sip show peers

2.3 CONFIGURACION BÁSICA

En este último apartado del capítulo vamos a ver los aspectos más fundamentales de la configuración de *Asterisk*. Lo vamos a hacer de forma práctica, aprendiendo cómo es posible configurar un dispositivo *SIP* (por ser considerado como un estándar para la *VoIP* y ser el que vamos a utilizar en el desarrollo del proyecto, presentado antes en este mismo capítulo) así como un *dialplan* o *plan de marcado* básico.

La configuración de los dispositivos *SIP* se realiza en el fichero *sip.conf*. La estructura general de este fichero es la siguiente:

```
[general]
parametro1=valor1
parametro2=valor2

[nombre_de_usuario]
parametro3=valor3

[nombre_de_usuario]
parametro4=valor4
```

La sección *general* contiene valores para los parámetros que son aplicables a todos los usuarios *SIP* de forma global. Por ejemplo, el puerto a usar en las comunicaciones (5060), las interfaces en las que escuchar peticiones, etc. Después, en la sección dedicada para cada usuario *SIP*, se definen los valores específicos para algunos de los parámetros. Existen tres **tipos de usuarios *SIP***:

- ***peer***. Usuarios que reciben llamadas de *Asterisk*.
- ***user***. Usuarios que envían llamadas a *Asterisk*.
- ***friend***. Usuarios que envía y reciben llamadas a/ de *Asterisk*.

Lo normal es que el tipo *friend* sea utilizado para los teléfonos y el tipo *peer* para los proveedores de telefonía IP. Veamos un pequeño ejemplo para la definición de dos usuarios introduciendo algunos parámetros y valores:

```
[general]
port=5060
bindaddr=0.0.0.0

[eugenio]
type=friend
username=eugenio
secret=123
context=desde-usuarios
host=dynamic
callerid= Eugenio <2000>

[carmen]
type=friend
username=carmen
```

```
secret=321
context=desde-usuarios
host=dynamic
callerid= Carmen <2001>
```

En la sección *general:*

- ***port.*** Puerto a usar en las comunicaciones *SIP*.
- ***bindaddr.*** Dirección en la que *Asterisk* escucha peticiones (con 0.0.0.0 indicamos todas las interfaces).

En los usuarios:

- ***type***. Tipo de usuario (*peer, user o friend*).
- ***username***. Nombre de usuario.
- ***secret***. Contraseña a utilizar para la autenticación.
- ***context***. Permite indicar el contexto aplicable al usuario (más adelante vemos en qué consiste un contexto).
- ***host***. Dirección IP del usuario. Si especificamos *dynamic* estamos indicando que el usuario debe registrarse y obtener una dirección IP.
- ***callerid***. Es el identificador usado para el usuario, mostrado en el terminal del receptor cuando éste realiza una llamada.

Cada vez que introducimos modificaciones en el fichero *sip.conf* es necesario volver a cargar su configuración en el intérprete de comandos *CLI*, ejecutando la siguiente sentencia:

```
CLI> sip reload
```

Si existe algún error en los cambios introducidos se nos advierte de ello en este momento. Como vimos en la tabla 4.1, si queremos consultar en cualquier momento un listado (incluyendo nombres de usuario, *host*, dirección IP, puerto, estado, etc.) de los usuarios *SIP* podemos ejecutar los comandos ***show sip users*** y ***show sip peers*** en el *CLI*.

Un ejemplo de *dialplan* o *plan de marcado* para la prueba de las comunicaciones entre los dos usuarios anteriores puede ser el siguiente, definiendo el procesado de las llamadas en el contexto *desde-usuarios* al que pertenecen:

```
[general]

[desde-usuarios]
exten => 2000,1,Dial(SIP/eugenio)
exten => 2001,1,Dial(SIP/carmen)
```

Así, gracias a este diminuto *dialplan* podemos llamar al usuario *eugenio* marcando la extensión 2000, y al usuario *carmen* marcando la 2001. Para ello podemos configurar cualquier teléfono IP o *softphone* que soporte *SIP* (*X-Lite, Zoiper* que es gratuito,…).

El *dialpan* o *lógica de marcado* es la parte fundamental de *Asterisk*, donde indicamos un conjunto de instrucciones que nos permiten procesar las llamadas que recibe la centralita IP, es decir, cuando un usuario marca una extensión a la que quiere llamar el *dialplan* su llamada es procesada de acuerdo a las instrucciones que contiene para ese número tecleado. A continuación vemos los conceptos básicos para empezar a conocer cómo es posible configurar un *dialplan*.

El *dialplan* es especificado en el fichero ***extensions.conf***, el cual incluye **contextos, extensiones, prioridades y etiquetas, aplicaciones y funciones**:

- ***Extensiones.*** Las extensiones números o caracteres alfanuméricos que se permiten marcar a los usuarios (diferentes a los terminales). A las extensiones le son asociadas acciones e instrucciones, ejecutadas por orden de prioridad. La sintaxis usada para la definición de extensiones es como sigue:

```
Exten => número de extensión, prioridad,
aplicación(argumentos)
```

El número de una extensión puede ser especificado como tal como un patrón cuando queremos hacer referencia a un grupo de extensiones. En la tabla 4.2 podemos ver los diferentes caracteres especiales que es posible utilizar para la definición de patrones o variables en las extensiones (siempre deben empezar con guión bajo _).

Tabla 4-2. Caracteres especiales para la definición de patrones en las extensiones

Carácter o carácteres	Patrón
X	Dígitos del 0 al 9.
Z	Dígitos del 1 al 9.
N	Dígitos del 2 al 9.
[]	Uno de los dígitos especificados entre los corchetes.
.	Cualquier carácter o dígito.
!	Carácter de desambigüación. Indica que el procesamiento debe ser detenido si ha sido encontrado un patrón adecuado.

Por ejemplo, si queremos establecer un patrón para números fijos nacionales podemos escribir *_[789]XXXXXXXX* (7, 8 o 9 y ocho dígitos más), o para números móviles *_6XXXXXXXX* (6 y ocho dígitos más).

- ***Contextos.*** Agrupan diversas extensiones para la aplicación de determinadas acciones a todo el contexto como conjunto. Los contextos son muy importantes dentro de la configuración del *dialplan*, al permitir crear diferentes entres lógicos para el procesamiento de las llamadas dotando al *dialplan* de escalabilidad y flexibilidad. Cuando un usuario es definido en *sip.conf* se le asigna un contexto.

- ***Prioridades y etiquetas.*** Para cada acción a realizar dentro de un contexto tenemos asociada una prioridad de ejecución. La prioridad *1* siempre es necesaria, mientras que el resto de prioridades pueden ser asignadas de forma automática incrementalmente escribiendo *n*, evitando tener que escribir la prioridad para cada extensión.

Las etiquetas son utilizadas para marcar e identificar una determinada prioridad de una extensión para poder hacer referencia a ella mediante la aplicación *Goto*, por ejemplo así:

```
…
exten => 1234,n,Goto(salto)
…
exten => 1234,n(salto),Noop(¡Prioridad con etiqueta salto!)
…
```

La operación *Noop* imprime el mensaje pasado como argumento por pantalla.

- ***Aplicaciones y funciones.*** Son ejecutadas al mismo tiempo que las instrucciones o acciones del *dialplan* van siendo procesadas (como podemos ver en la sintaxis de las extensiones). Al marcar una extensión se ejecuta, por tanto, la aplicación asociada a la prioridad correspondiente (por ejemplo, *Dial* para realizar una marcación teniendo que especificar como argumentos la tecnología –*SIP*- y el usuario: *Dial(SIP/eugenio)*). Las aplicaciones son módulos que realizan alguna acción en algún canal, mientas que las funciones realizan otro tipo de operaciones que no afectan a los canales.

A partir de aquí es posible añadir al *dialplan* más servicios para aumentar su complejidad y funcionalidad: **buzones de voz, macros, *IVRs*** (menús de voz con los que los usuarios pueden interactuar), **salas de conferencias,** etc. Veremos la aplicación de estos conceptos más detenidamente en la creación del *dialplan* a aplicar en las pruebas de rendimiento sobre servidores *Asterisk* que realizamos en el capítulo siguiente.

CAPITULO 5

DISEÑO, IMPLEMENTACION Y PRUEBA DE UNA INFRAESTRUCTURA VIRTUAL DE TELEFONIA IP

En este capítulo vamos a **abordar desde un punto de vista práctico la creación de una infraestructura virtual de consolidación de servidores**, concretamente de servidores de telefonía IP con la centralita software *Asterisk*. Hacemos uso de Xen como plataforma para la virtualización completa, como fue justificado en el capítulo *2. Virtualización de plataforma*.

Ya conocemos en qué consiste virtualizar y cómo hacerlo nos puede proporcionar oportunidades excelentes en la administración de servidores. Es hora de adentrarnos en las **fases de creación y desarrollo de una infraestructura virtual trabajando en un caso práctico real: planificación (análisis y diseño), migración física a virtual (P2V), administración/gestión/monitorización de la arquitectura y automatización**. Una vez finalizado el proceso de consolidación de servidores de telefonía IP **realizamos diversas pruebas de carga con el objetivo de estimar el rendimiento de un servidor *Asterisk* virtual en contraposición con el rendimiento de un servidor *Asterisk* real** (instalado en un equipo físico en lugar de en un dominio invitado). Además esto lo haremos con dos servidores virtuales diferentes: uno ubicado en el propio sistema Xen anfitrión y otro ubicado en un sistema de ficheros compartidos en red.

1 Introducción

A continuación vamos a presentar mediante el desarrollo de un caso práctico real cuáles son las **etapas necesarias para la implantación de la consolidación de servidores mediante técnicas de virtualización**. Como vamos a ver, los pasos en sí son bastante generales y sirven de modelo para cualquier empresa que quiera realizar un proyecto de este tipo, **similares a otros proyectos ubicados dentro de la administración de sistemas informáticos**.

Como siempre, el primer paso a llevar a cabo es la **planificación de la consolidación**. Dentro de la planificación encontramos el análisis y diseño de la virtualización que queremos implementar, por lo que es un paso que cobra una importancia especial: del éxito de esta primera etapa dependerá en gran medida el éxito de las etapas siguientes y por tanto la consecución de los objetivos por los cuales desarrollamos el proyecto. Una vez analizado y diseñado el proceso completo, pasamos a la **migración física a virtual**, en la que las máquinas virtuales se crean siguiendo como referencia las especificaciones obtenidas anteriormente para los equipos físicos disponibles actualmente en la empresa o que existirían en caso de que la infraestructura final a implantar no fuera virtual. Después, queda **administrar y monitorizar la infraestructura virtual** de manera eficiente para conseguir ofrecer el mayor rendimiento posible y que mantenerla sea un proceso fácil, limpio y centralizado; si se considera necesario, finalmente podemos implementar la **automatizar algunos de los procesos de gestión y mantenimiento**. Veamos con mayor detenimiento cada una de las etapas al mismo tiempo que desarrollamos la consolidación de servidores de telefonía IP *Asterisk* con Xen.

2 Planificación

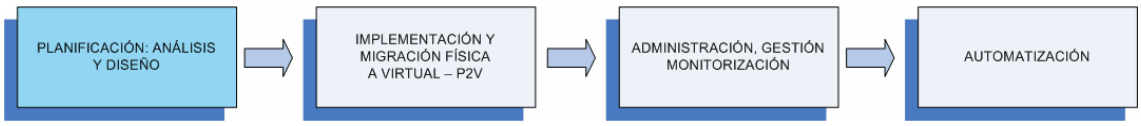


Figura 5.1. Consolidación de servidores: Planificación: Análisis y Diseño.

En esta primera etapa del proceso de implantación de un entorno total o parcialmente virtual se incluye la planificación del mismo mediante el **análisis y diseño de la infraestructura**. **Es importante diferenciar bien en este fase cuando trabajamos partiendo desde una infraestructura informática físicamente instalada y funcionando en una empresa** y deseamos realizar su traslación a virtual (parcial o totalmente, según debemos determinar después tras el análisis) **o cuando debemos virtualizar desde cero**. Lo habitual es que ocurra como en el primer caso.

2.1 ANALISIS

La fase de análisis por tanto consiste en el estudio en profundidad de la estructura y estado actual del *CPD* o *data center*, es decir, la elaboración de un detallado informe o inventario que describa los servidores, incluyendo normalmente la siguiente información de forma detallada:

- Qué servidores hay actualmente disponibles en la empresa.
- Qué servicios ofrece cada uno de los servidores existentes.

- Qué sistemas operativos están ejecutando.
- Qué aplicaciones y servicios están siendo ejecutadas en el servidor, cuáles son críticas para su funcionamiento.
- Qué configuración se está aplicando en el servidor.
- Qué comunicaciones hay establecidas y son usadas.
- Qué protocolos de seguridad está presentes en la actualidad.
- De qué recursos dispone y son indispensables para llevar a cabo su cometido: interfaces y conexiones de red, memoria, procesamiento, almacenamiento, etc.
- Uno de los datos más importantes a obtener en el informe es el uso actual que el servidor hace de los recursos que tiene asignados.
- Además debe ser incluido cualquier otro dato o información que sea considerada de importancia o fundamental para la mejor comprensión del estado actual de la infraestructura informática de la empresa y sea determinante de cara al desarrollo del proyecto.

Existen herramientas que pueden ser de gran utilidad en la planificación de este tipo de proyectos, herramientas que permiten recopilar todo tipo de información sobre el uso de recursos por parte de las aplicaciones antes de realizar la consolidación, y además, haciendo uso de esa información, pueden simular cómo sería su rendimiento o comportamiento en una máquina virtual. Un ejemplo es *HP Capacity Advisor*. **Todo esto nos permite llegar a la conclusión sobre qué aplicaciones/servicios/servidores son mejores candidatos para ser virtualizados y elegir con razonablemente qué tipo de solución de virtualización conviene aplicar**. No debemos olvidar que existen aplicaciones que son especialmente sensibles a la virtualización en su rendimiento, o incluso que incluyen algún tipo de virtualización implementada internamente, por lo que su ejecución puede verse perjudicada.

Si nuestro trabajo consiste en el desarrollo de consolidación de servidores sin disponer de una infraestructura informática previa en producción entonces la fase de diseño es modificada para abordar algunos de los aspectos anteriores de forma más extensa, aunque no su importancia de cara al proceso global.

Una vez realizado el análisis pertinente de cara al desarrollo de nuestro proyecto de consolidación de servidores de telefonía IP sabemos que disponemos de un total de cuatro servidores: **tres servidores HP Proliant DL120 G5 y un servidor ML350 G5**, con las siguientes características recogidas en la tabla 5.1.

Tabla 5-1. Servidores disponibles para la realización del PFC

Modelo	Procesador	Memoria	Virtualiz. HW	Disco	Red
HP Proliant DL120 G5	Intel(R) Xeon(R) CPU E3110 @ 3.00GHz (Coreduo)	4 Gb SDRAM DDR2 PC2-6400	Sí (Intel VT - vmx)	1 disco duro SATA 250 Gb	NetXtreme BCM5722 Gigabit Ethernet PCI Express 1GB/s

					64 bits
HP Proliant ML350 G5	Intel(R) Xeon(R) CPU E5320 @ 1.86GHz (Quadcore)	2 Gb FB-DIMM DDR2 FB-DIMM Synchronous 667 MHz	Sí (Intel VT - vmx)	2 discos duros SAS de 132Gb en RAID1 gestionados por una controladora Smart Array E200i	NetXtreme II BCM5708 Gigabit Ethernet 1GB/s 64 bits

Ninguno de los cuatro servidores dispone de sistema operativo instalado; consecuentemente tampoco se encuentran ejecutando ningún servicio en la actualidad, ni aplicaciones, ni existe una configuración existente para ellos. Como ya hemos dicho en anteriores capítulos **vamos a instalar una distribución GNU/Linux en los servidores, concretamente en los servidores HP Proliant DL120 G5 Debian 5 Lenny para 64 bits y en el servidor ML350 G5 Fedora Core 11, también para 64 bits. Los servidores van a ser usados única y exclusivamente para el desarrollo del proyecto** por lo que no hay ningún problema en cuanto a compatibilidad de los servicios de telefonía IP con otros que debieran estar disponibles.

Los recursos hardware disponibles son más que suficientes para los propósitos del proyecto. Aunque es de gran importancia disponer de suficientes recursos de procesador y memoria (*Asterisk* demanda mucho más procesamiento que memoria, mientras que la virtualización requiere cuanto más mejor de ambos), también los dispositivos de red y disco poseen unas características excelentes para el desarrollo de nuestro proyecto; son más que suficientes proporcionando un ancho de banda y una capacidad que nos permite despreocuparnos de los posibles problemas que pudieran ocasionar. Además, **todos los servidores disponen de virtualización asistida por el hardware del procesador, en este caso la tecnología Intel VT (flag *vmx* en el procesador)**, lo que nos permite la creación y prueba de máquinas virtuales *hardware* (que no precisan un sistema operativo modificable).

Las comunicaciones entre los diferentes servidores están garantizadas al disponer de un *switch Gigabit* que los interconecta. Todos los servidores se encuentran ubicados juntos en un mismo *rack*, el cual dispone de un sistema de alimentación ininterrumpida (*SAI*) que garantiza el suministro eléctrico. Aunque la seguridad no es uno de los objetivos principales, se asegura también la correcta comunicación entre los diferentes dispositivos y el acceso remoto para el trabajo con los servidores mediante *SSH*; para ello se realiza la configuración propia del cortafuegos *iptables*.

Recordamos para finalizar los **requisitos indispensables a la hora de desarrollar consolidación de servidores haciendo uso del software Xen y de configurar servidores de telefonía IP con *Asterisk***, aunque ya fueran presentados en el capítulo segundo:

- **Se va a hacer uso de la versión 1.4.28 de *Asterisk*** *Asterisk* puede ser instalado y configurado sin problemas en las últimas versiones de las distribuciones GNU/Linux a instalar en los servidores: Fedora Core 11 y Debian 5 Lenny.

- **La instalación básica de *Asterisk* no necesita parches o software adicional. Aún así es recomendable instalar el paquete *asterisk-addons***.

- ***Asterisk* necesita más recursos de procesamiento que de memoria para casos con alta carga de trabajo**.

- Tanto las máquinas virtuales como *Asterisk* serán sometidos a pruebas de estrés, por lo que el servidor que actúe como sistema anfitrión de la infraestructura virtual **requiere gran capacidad de procesamiento y memoria, y no tanto de almacenamiento en los servidores anfitriones**. Esto es debido a que vamos a trabajar a lo sumo con dos dominios trabajando en los escenarios del capítulo siguiente, en el que implementamos alta disponibilidad de servicios en dominios y migración en caliente de los mismos entre diferentes servidores anfitriones.

- **Los escenarios a desarrollar no precisan hardware adicional o *drivers* especiales en este proyecto, salvo los propios sistemas anfitriones y equipos que generan la carga en las pruebas de rendimiento**.

- Es importante mantener un **buen equilibrio entre coste y rendimiento**. Es por ello por ejemplo que el almacenamiento compartido es implementado mediante sistemas de ficheros compartidos en red (NFS) por uno de los servidores y no adquiriendo dispositivos adicionales, como podría tratarse de una unidad *NAS*.

- **Xen es lo suficientemente potente como para ser instalado y configurado en entornos empresariales, soportando y ofreciendo funcionalidades avanzadas** como gestión y monitorización de anfitriones remotos o *migración de máquinas virtuales*. Los requisitos software que presenta pueden ser abordados sin problema alguno; el más importante de ellos es que en la actualidad Xen disponer de un mayor soporte en el sistema operativo Debian 5 que en Fedora Core 11.

2.2 DISEÑO

Hecho el análisis, **la fase de diseño consiste en la selección adecuada del hardware para los servidores que actuarán como anfitriones/máquinas físicas y la caracterización al completo de las máquinas virtuales a implementar y que ofrecerán los servicios que hemos decidido mejor posicionados para ser virtualizados, o que nos han encomendado virtualizar**.

Además, se tendrá en cuenta en esta fase si deseamos que algunas máquinas permanezcan en el *data center* como se encontraban anteriormente, si se considera que no es necesario virtualizarlas. **Es especialmente importante diseñar en qué servidores físicos serán albergadas las máquinas virtuales, si se soportarán mecanismos de migración de unos anfitriones a otros en caso de fallos o mantenimiento, si se ubicarán en almacenamientos compartidos como *NFS, NAS, DRBD,* etc**. En la caracterización de las máquinas virtuales entra la definición del número de CPUs virtuales, la cantidad de memoria asignada, los discos de los que disponen, las interfaces de red, qué sistema operativo correrá sobre ellas, su comportamiento ante reinicios o apagados… en definitiva la creación y configuración tanto de sus ficheros de disco como de su archivo de configuración *cfg*. Como hemos visto en capítulos anteriores esto lo podemos hacer siguiendo diferentes metodologías: editando directamente los ficheros, mediante asistentes en modo texto, herramientas gráficas para monitorización y creación de dominios, etc.

Dado que los servidores HP Proliant DL120 G5 disponen de un mayor potencial en cuanto a procesador y memoria uno de ellos se va a utilizar para el desarrollo de la infraestructura virtual, quedando los otros dos para la introducción de carga en las pruebas de rendimiento. **El servidor HP Proliant ML350 G5 lo usamos igualmente en nuestra arquitectura de prueba** como veremos en sucesivos apartados, y en el último de los esquemas implementados **también como dispositivo de almacenamiento compartido para los**

dominios al poseer de discos duros SAS, de mayor velocidad de transferencia. En cuanto al sistema operativo que deben ejecutar los servidores el único requisito es que el servidor que lleve a cabo la virtualización disponga de Debian 5 Lenny, ya que como hemos comentado en el apartado anterior dispone de un mejor soporte para Xen y las utilidades de monitorización y administración de la infraestructura virtual. En la tabla 5.2 queda de forma resumida cómo debe ser configurado el sistema operativo y software en cada uno de los servidores de los que disponemos.

Tabla 5-2. Configuración de los servidores

Rol	Modelo	Sistema operativo	Aplicaciones y Servicios
Servidor anfitrión de la infraestructura virtual	HP Proliant DL120 G5	Debian GNU/Linux 5 Lenny	Xen como virtualizador de plataforma. En el dominio administrativo **dom0**: Herramienta para la obtención de información del sistema *SAR*, utilidades de monitorización y administración de la infraestructura virtual: *xentop*, *XenMan*, *virt-manager*. En el dominio invitado **domU**: *Asterisk*, *SAR*. Acceso por *SSH* en todos los dominios.
Equipo cliente para la generación de la carga de trabajo en las pruebas de rendimiento	HP Proliant DL120 G5	Debian GNU/Linux 5 Lenny	Software de prueba de servidores de telefonía IP *SIPp*. Herramienta para la obtención de información del sistema *SAR*. Acceso por *SSH*.
Equipo servidor para la generación de la carga de trabajo en las pruebas de rendimiento	HP Proliant DL120 G5	Debian GNU/Linux 5 Lenny	Software de prueba de servidores de telefonía IP *SIPp*. Herramienta para la obtención de información del sistema *SAR*. Acceso por *SSH*.
Servidor de apoyo para las pruebas de rendimiento y como sistema de almacenamiento compartido para dominios	HP Proliant ML350 G5	Fedora Core 11	*Asterisk*. Sistema de ficheros en red *NFS*. Acceso por *SSH*.

Aunque no será hasta el capitulo siguiente cuando sea implementado, es bueno tener presente que **el servidor anfitrión Xen debe soportar la migración de dominios**, tanto *en parada* como *en caliente*. Para ello es necesario que se encuentre habilitado **en el cuarto servidor el servicio *NFS*** para la compartición de los ficheros de disco y configuración de los dominios en red; además, **replicaremos entonces la configuración del servidor anfitrión Xen en cualquier otro de los servidores** para realizar tal migración. Lógicamente, los dominios que deben tener movilidad entre los diferentes anfitriones son los servidores *Asterisk* virtuales.

Aplicaciones y servicios adicionales serán instalados y configurados para implementar mecanismos de alta disponibilidad entre dominios Xen que forman *clústeres* virtuales, en concreto *Heartbeat*. Esto también es estudiado en el capítulo siguiente.

Veamos brevemente en el siguiente apartado cómo la infraestructura debe ser creada y configurada, instalando y caracterizando Xen, los dominios invitados, y *Asterisk*.

3 Implementación y Migración física a virtual (P2V)

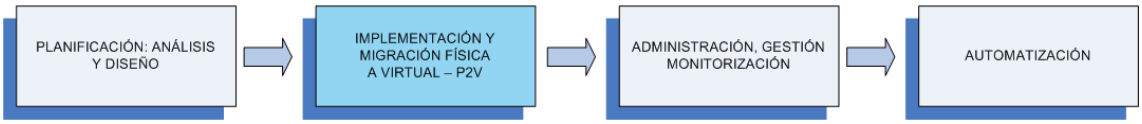


Figura 5.2. Consolidación de servidores: Migración física a virtual – P2V.

Una vez completadas las fases anteriores para el análisis y diseño del entorno virtualizado que queremos crear, finalmente estamos en disposición de crear los dominios y la infraestructura virtual al completo, y llevar a cabo la migración de las aplicaciones y servicios necesarios de las máquinas físicas a las máquinas virtuales (o su instalación y configuración en las máquinas virtuales si es que no partimos de un sistema previo).

Esta fase puede ser llevada a cabo manualmente o bien haciendo uso de herramientas diseñadas para tal efecto. Básicamente, consiste en la instalación y configuración de las soluciones de virtualización en los distintos servidores anfitriones del data center, para pasar después a la creación de las máquina virtuales con las aplicaciones y entornos de los que disponíamos anteriormente en máquinas físicas replicados. En este caso, y si existen muchas máquinas virtuales de características similares, su creación y suministro pueden ser automatizados simplificando bastante el proceso. Es muy importante todo el diseño realizado en la fase anterior para saber, por ejemplo, dónde se ubicará cada máquina virtual y la movilidad que tendrá dentro de los equipos de data center.

En nuestro proyecto partimos desde cero para la creación de la infraestructura virtual. Por lo tanto, **en esta fase llevamos a cabo el grueso de la instalación y configuración de cada uno de los servidores, lo que incluye la puesta en marcha de los sistemas operativos y software citados en el apartado anterior para cada uno de ellos:**

- **Servidor *de apoyo***:
  - Instalamos Fedora Core 11.
  - Instalamos *SSH*.
  - Instalamos *Asterisk* sin configuración alguna.
  - Instalamos y configuramos el servicio de ficheros compartidos en red *NFS*.
- **Equipo cliente**:
  - Instalamos Debian 5 Lenny.
  - Instalamos y configuramos *SIPp* para que desempeñe el rol *cliente* en las pruebas de rendimiento.
  - Instalamos el paquete *sysstat* que incluye la utilidad *SAR*, necesaria para la recogida de estadísticas respecto al consumo de los recursos del sistema.

- **Equipo servidor**:
  - Instalamos Debian 5 Lenny.
  - Instalamos y configuramos *SIPp* para que desempeñe el rol *servidor* en las pruebas de rendimiento.
  - Instalamos el paquete *sysstat* que incluye la utilidad *SAR*.
- **Servidor anfitrión**:
  - Instalamos Debian 5 Lenny.
  - Instalamos Xen como plataforma de virtualización. La configuración de Xen se realiza de forma básica garantizando que el dominio *dom0* disponga al menos de 196Mb de memoria RAM y acceso por completo al procesador; la red virtual y las interfaces de red virtuales para los dominios son configuradas en modo *bridge*.
  - Definimos y creamos el servidor virtual *Asterisk* de forma que disponga de acceso total al procesador del servidor anfitrión y una cantidad de memoria RAM grande, al menos de 2Gb.
  - Realizamos la instalación de *Asterisk* en el dominio, configurándolo adecuadamente para el procesado de las peticiones de llamada mediante un escenario de prueba.
  - Instalamos el paquete *sysstat* tanto en el dominio administrativo *dom0* como en el dominio *domU Asterisk*, ya que es necesario monitorizar el consumo de recursos en todos los dominios Xen.

Parte del proceso de implementación aquí resumido ha sido explicado en capítulos anteriores, mientras que la configuración adicional a realizar es detallada y explicada posteriormente a medida que se presenten y lleven a cabo las pruebas de rendimiento del servidor *Asterisk* en el apartado *6. Pruebas de rendimiento*, ya que en éstas vamos a usar hasta tres arquitecturas diferentes en nuestra infraestructura: servidor *Asterisk* real, servidor *Asterisk* virtual, y servidor *Asterisk* virtual ubicado en un sistema de ficheros en red. Prestaremos especial atención a cómo configurar Xen y *Asterisk*, y sobre al proceso de creación y puesta en marcha de los servidores virtuales (ficheros de disco y archivo de configuración).

4 Administración, Gestión y Monitorización

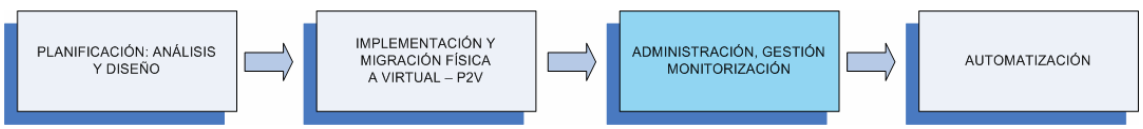


Figura 5.3. Consolidación de servidores: Administración, Gestión y Monitorización.

Una vez que todo el entorno virtual se encuentra bien configurado y en completa operatividad, **es necesario el uso de herramientas que nos permitan administrar toda la infraestructura creada**. **Mecanismos de creación de nuevas máquinas virtuales, su suministro, procedimientos de monitorización del estado tanto de las máquinas virtuales como de los anfitriones, de los servicios ofrecidos, etc**.

Es muy importante tener planificado por ejemplo cómo debemos actuar si en la monitorización son detectados fallos en alguna máquina, de lo contrario los beneficios obtenidos del uso de la virtualización podrían verse disminuidos: políticas de mantenimiento, seguridad, migración de máquinas virtuales, almacenamiento de imágenes de máquinas, etc. Existen multitud de aplicaciones que nos permiten llevar a cabo todas estas tareas con gran efectividad y de manera centralizada.

En el dominio administrativo *dom0* de nuestro servidor anfitrión instalamos y hacemos uso de las siguientes herramientas que nos ayudan a monitorizar, gestionar y administrar nuestra infraestructura virtual:

- Herramientas básicas incluidas en Xen: ***xm top*, comando *xm list***.
- ***Virt-install* y *Xen-Tools*** para la automatización del proceso de creación de nuevos dominios.
- Herramientas gráficas: ***XenMan y Virt-Manager***.

Como vemos, la totalidad de estas herramientas fue presentada y documentada en el capítulo *3. Xen*.

5 Automatización

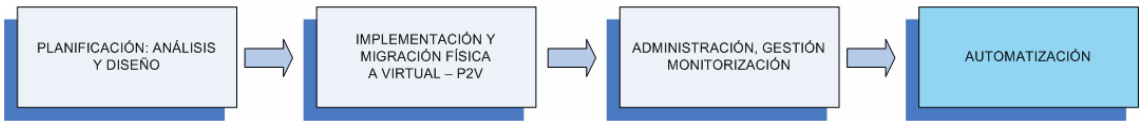


Figura 5.4. Consolidación de servidores: Automatización.

Finalmente, y posiblemente dentro de la fase anterior de gestión y monitorización, queda la **puesta en marcha de procesos automatizados que operen sobre las máquinas virtuales** (como si se tratase de procesos) **y que dote a todo nuestro sistema de un gran dinamismo**, y así exprimir al máximo las características de la virtualización. Así, se puede **establecer el comportamiento de distintas actividades que operan sobre las máquinas virtuales** (como la creación, reposición, migración, reinicio) así como **controlar en tiempo real diversas métricas definidas sobre las máquinas** que constituyan demandas del negocio (como tiempos de respuesta, disponibilidad, rendimiento…). Casi todas las soluciones de virtualización disponen de herramientas que nos permiten implementar todo esto, aunque también hay aplicaciones que actúan de manera independiente.

En nuestro proyecto vamos a habilitar un directorio en nuestro sistema de ficheros compartido en red como repositorio para imágenes de dominios que consideremos disponen de una configuración estándar. De esta forma, siempre que queramos poner en marcha un nuevo dominio que comparte configuración con alguna de las imágenes los ficheros de disco y configuración de éstas pueden ser usados como plantilla reduciendo enormemente el tiempo de

aprovisionamiento del nuevo dominio. Además, **podemos crear directorios en los que realizar copias de seguridad periódicas de las imágenes de nuestros dominios con servicios críticos**, de cara a llevar a cabo una recuperación rápida y efectiva en los casos en los que detectemos cualquier fallo de gravedad en el estado del dominio o en los que debamos realizar operaciones de mantenimiento por cualquier otro motivo.

Otras cuestiones que también constan de gran importancia y que no siempre suelen ser tratadas (la virtualización se suele asociar solamente a tecnología) son cuestiones **relacionadas con el aspecto humano y las consecuencias laborales que puede plantear la implantación de consolidación de servidores mediante virtualización**. También debe dedicarse un gran esfuerzo a las cuestiones culturales, humanas, y relativas a procesos y hábitos de los trabajadores. Así, de igual forma a cómo nos planteamos el análisis y diseño del entorno virtual que queremos implantar, también debemos **pensar y dedicar tiempo a recapacitar sobre cómo los cambios que queremos llevar a cabo pueden afectar a los empleados involucrados, si habrá cambios en sus responsabilidades, si es necesaria una mayor formación para ellos o reubicarlos, etc**. Todas estas cuestiones y aspectos también influyen en el éxito final de todo el proceso, por lo que hay que controlarlos al máximo en la medida de lo posible.

No debemos olvidar que **el uso de la virtualización implica un cambio más que significativo en los procesos de negocio, en la manera de proveer los servicios**.

6 Pruebas de rendimiento

Una vez que hemos completado el proceso de desarrollo de nuestra infraestructura virtual de telefonía IP y conocemos en profundidad las características y configuración de los componentes *hardware* y *software* que forman parte de ella es momento de **estudiar el rendimiento que ésta puede ofrecer y así determinar si el servicio puede ser ofrecido con garantías, en comparación con lo que sucede cuando disponemos de una infraestructura de telefonía IP sin virtualización**.

El rendimiento de nuestro servidor virtual *Asterisk* lo vamos a medir haciendo uso de *SIPp*, *benchmark* que hace uso del protocolo *SIP* y **con el que podemos determinar el número de llamadas simultáneas que puede llegar a soportar**. Las pruebas están diseñadas para que obtengamos este valor en **dos escenarios diferentes**: con cliente y servidor *SIPp* (que crea y recibe la carga de trabajo, respectivamente) usando *códecs* diferentes para las llamadas (***trascoding***) y usando el mismo *códec* (**sin *trascoding***). Aplicamos los dos escenarios de prueba en **tres esquemas distintos** para la posterior extracción de conclusiones: **utilizando un servidor *Asterisk* real, un servidor *Asterisk* virtual, y finalmente ubicamos el servidor *Asterisk* virtual en un sistema de ficheros en red (NFS)**.

En primer lugar hacemos una introducción a la utilidad *SIPp*. Después, vemos cómo realizar la configuración de los dos escenarios (con *trascoding* y sin *trascoding*) de manera genérica, ya que su aplicación es la misma en los tres esquemas diferentes a estudiar. Finalmente ejecutamos las pruebas para cada uno de los esquemas y presentamos los resultados obtenidos para extraer conclusiones al respecto.

6.1 SIPP

SIPp (http://sipp.sourceforge.net) es una herramienta *libre* para la evaluación o prueba de comunicaciones que hacen uso del protocolo *SIP (Session Initiation Protocol)*, como

sabemos el encargado del establecimiento de las sesiones en las llamadas con telefonía IP, trabajando bajo el paradigma cliente/servidor.

Figura 5.5. Página web de SIPp: http://sipp.sourceforge.net.

El funcionamiento de *SIPp* consiste en el establecimiento de escenarios de prueba mediante su definición en ficheros *xml* tanto en cliente como en servidor de manera que queda definido el flujo de las llamadas, pudiendo generar una gran cantidad de tráfico *SIP*. *SIPp* incluye muchos escenarios ejemplo, aunque también tenemos la posibilidad de incluir nuestro propios escenarios, incluso incluir tráfico *RTP* con audio personalizado según nuestras preferencias.

Por lo tanto, como podemos ver en sus características *SIPp* es una excelente herramienta para realizar pruebas de rendimiento y carga en entornos de telefonía IP reales; permite que conozcamos de forma aproximada el número total de llamadas simultáneas que puede llegar a soportar un servidor *Asterisk*.

6.1.1 Instalación

SIPp se instalado de la misma forma que otros paquetes GNU/Linux. En todas las pruebas de rendimiento recogidas en las páginas siguientes *SIPp* es instalado en máquinas Linux, por lo que a continuación sólo se presenta la instalación para este tipo de sistemas operativos; aún así, igualmente se encuentran disponibles en la web de la utilidad *http://sipp.sourceforge.net/* ejecutables para otros sistemas operativos.

Como vamos a instalar *SIPp* tomando la última versión del código fuente, en primer lugar debemos instalar las dependencias requeridas. Entre ellas se encuentran el compilador *gcc*, el paquete con utilidades de compilación *build-essential (make)*, y diversas librerías de *c*. Para instalarlas en la distribución Debian GNU/Linux (para otras distribuciones es de forma similar) ejecutamos:

```
$ apt-get install gcc-4.1 libncurses5-dev libssl-dev zlib1g-dev
libnewt-dev libusb-dev build-essential libpcap-dev
```

Con las dependencias instaladas descargamos la última versión *unstable* de *SIPp* (en tiempo de redacción de esta memoria el paquete *sipp.2009-07-29.tar.gz*) desde la web del proyecto con el comando *wget*:

```
$ wget http://sipp.sourceforge.net/snapshots/sipp.2009-07-29.tar.gz
```

Descargado el paquete lo descomprimimos primero con el comando *gunzip* y después con *tar*:

```
$ gunzip sipp.2009-07-29.tar.gz
$ tar -xvf sipp.2009-07-29.tar
```

Ingresamos entonces en el directorio creado en la descompresión del paquete, *sipp.svn*:

```
$ cd sipp.svn/
```

Finalmente disponemos de varias opciones de compilación e instalación del programa, dependiendo de las características que queramos que estén presentes en cuanto a soporte y tipología de la prueba de carga a realizar:

- Autenticación *SIP* y soporte PCAPplay (flujo real de tráfico RTP).

```
$ make pcapplay_ossl
```

- Autenticación *SIP* sin soporte PCAPplay.

```
$ make ossl
```

- Sin autenticación y sin soporte PCAPplay.

```
$ make
```

- Sin autenticación y con soporte PCAPplay.

```
$ make pcapplay
```

En todas las pruebas presentadas *SIPp* ha sido instalado con la última de las opciones, sin la necesidad de autenticación *SIP* y con soporte PCAPplay, ya que deseamos que se genere tráfico *RTP* real en las llamadas. Para ello, como vemos en la configuración realizada del cliente *SIPp*, situamos un fichero de audio en el directorio *pcap* bajo el directorio de instalación. Para iniciar *SIPp* basta con ejecutar el siguiente comando en el directorio de instalación *sipp.svn*, al que se deben añadir diversos parámetros que veremos al realizar las pruebas de rendimiento:

```
$ ./sipp
```

6.2 CONFIGURACION

En este apartado vamos a ver **cómo es configurado el escenario para la prueba de rendimiento del servidor *Asterisk*, válido para los tres esquemas: servidor *Asterisk* real, virtual y alojado en un sistema de ficheros en red**. En primer lugar vamos a ver cómo necesitamos configurar el escenario para que se asemeje lo máximo posible a uno real; después atendemos a configurar los tres componentes principales de la prueba: servidor *Asterisk*, servidor y cliente *SIPp*. **Hay que resaltar que la configuración del servidor *Asterisk* sigue siendo la misma para cualquiera de los tres esquemas implementados**, independientemente de que sea un servidor real, virtual, o virtual remoto. Es decir, **la virtualización no introduce configuración *extra* en estos casos: cuando configuramos un servidor virtual lo hacemos como si de máquina física real se tratara, a todos los efectos**.

6.2.1 Configuración del escenario de prueba

El objetivo de la prueba es fundamentalmente establecer un escenario de uso de telefonía IP lo más real posible en el que obtener mediante el software de prueba *SIPp* el número máximo de llamadas simultáneas posibles en el servidor *Asterisk*. Esto lo haremos con tres servidores *Asterisk* diferentes: uno real, otro virtual, y finalmente otro virtual alojado en un servidor de ficheros en red y no en el sistema Xen local que lo inicia.

De manera general el escenario será como se puede ver en la figura 5.1.

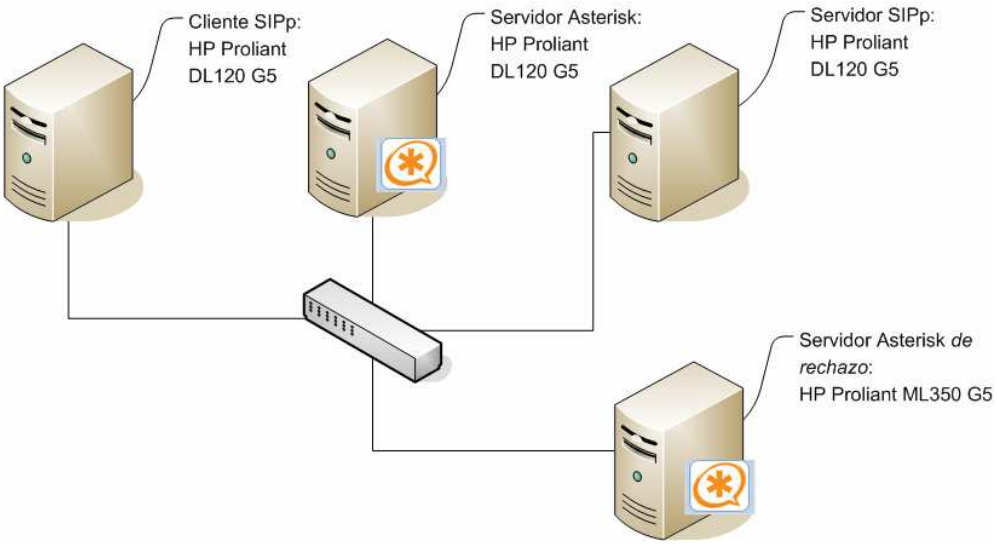


Figura 5.6. Arquitectura general del escenario para las pruebas de rendimiento.

Sobre esta arquitectura se irán realizando diversas modificaciones para adaptarla a las diferentes características de cada uno de los tres esquemas a probar. **El cliente *SIPp* irá enviando peticiones *INVITE* de inicio de llamada al servidor *Asterisk*, el cual generará un valor aleatorio entre uno y cinco para que el entorno sea lo más real posible**, ya que dependiendo de este valor:

- **Envía la petición al servidor *SIPp*** para así establecer la comunicación.
- **Envía la petición al servidor *Asterisk de rechazo***, que se encuentra ejecutando *Asterisk* pero sin ninguna configuración, por lo que rechaza cualquier petición de comunicación que recibe. La razón de la inclusión de este servidor es para la simulación de realización de llamadas a números externos.
- **Envía la petición a una dirección de red que no existe**, aunque dentro del rango de la red.
- **Envía la petición a una dirección de red que no se encuentra en el rango de la red**.
- **Envía la petición a la dirección de red de la puerta de enlace**.

Como vemos, gracias a la generación de este valor aleatorio se simulan diferentes tipos de llamadas que podrían tener lugar en un entorno de producción real. En definitiva, **todas las llamadas terminan alcanzando el servidor *SIPp*, aunque no todas lo hagan en el primer intento**, como vamos a ver detenidamente a la hora de configurar el escenario en el servidor *Asterisk*, en el siguiente apartado.

6.2.2 Configuración del servidor *Asterisk*

Con el servidor de centralitas VoIP *Asterisk* instalado correctamente en el equipo/dominio debemos realizar diferentes cambios para que éste soporte y procese correctamente la carga de trabajo generada por los equipos *SIPp* en la prueba de rendimiento. Primero hay que realizar algunos cambios en el código fuente de *Asterisk*. En el fichero */channels/chan_sip.c* editamos la línea:

```
MAX_RETRANS 1
```

También cambiamos la siguiente línea en el fichero main/autoservice.c:

```
MAX_AUTOMONS 2048
```

Tras cambiar estos valores en el código debemos volver a compilar *Asterisk* para que tengan efecto. Para ello, ejecutamos los siguientes comandos en el directorio obtenido al descomprimir *Asterisk (asterisk-1.4.28)*:

```
$ make clean
$ ./configure
$ make menuselect
$ make
$ make install
$ make config
$ make samples
```

A continuación es necesario realizar la **instalación y carga de los *códecs* que son usados en las comunicaciones** a establecer en las pruebas de rendimiento: **g711 (alaw) y g729**. Ambos *códecs* deben estar presentes en el directorio */usr/lib/asterisk/modules* para poder ser usados por *Asterisk*. Lo comprobamos viendo el contenido del directorio:

```
/usr/lib/asterisk/modules$ ls
app_adsiprog.so         app_hasnewvoicemail.so  app_sendtext.so
chan_agent.so   format_jpeg.so          func_rand.so
app_alarmreceiver.so    app_ices.so             app_setcallerid.so
chan_iax2.so    format_mp3.so           func_realtime.so
app_amd.so              app_image.so            app_setcdruserfield.so
chan_local.so   format_pcm.so           func_sha1.so
app_authenticate.so     app_lookupblacklist.so  app_settransfercapability.so
chan_mgcp.so    format_sln.so           func_strings.so
app_cdr.so              app_lookupcidname.so    app_sms.so
chan_ooh323.so  format_vox.so           func_timeout.so
app_chanisavail.so      app_macro.so            app_softhangup.so
chan_oss.so     format_wav_gsm.so       func_uri.so
app_channelredirect.so  app_milliwatt.so        app_speech_utils.so
chan_phone.so   format_wav.so           pbx_ael.so
app_chanspy.so          app_mixmonitor.so       app_stack.so
chan_sip.so     func_audiohookinherit.so  pbx_config.so
app_controlplayback.so  app_morsecode.so        app_system.so
chan_skinny.so  func_base64.so          pbx_dundi.so
app_db.so               app_mp3.so              app_talkdetect.so
codec_adpcm.so  func_callerid.so        pbx_loopback.so
app_dial.so             app_nbscat.so           app_test.so
codec_alaw.so   func_cdr.so             pbx_realtime.so
app_dictate.so          app_parkandannounce.so  app_transfer.so
codec_a_mu.so   func_channel.so         pbx_spool.so
```

```
app_directed_pickup.so  app_playback.so         app_url.so
codec_g726.so   func_cut.so             res_adsi.so
app_directory.so        app_privacy.so          app_userevent.so
codec_gsm.so    func_db.so              res_agi.so
app_disa.so             app_queue.so            app_verbose.so
codec_lpc10.so  func_enum.so            res_clioriginate.so
app_dumpchan.so         app_random.so           app_voicemail.so
codec_ulaw.so   func_env.so             res_convert.so
app_echo.so             app_readfile.so         app_waitforring.so
format_g723.so  func_global.so          res_crypto.so
app_exec.so             app_read.so             app_waitforsilence.so
format_g726.so  func_groupcount.so      res_features.so
app_externalivr.so      app_realtime.so         app_while.so
format_g729.so  func_language.so        res_indications.so
app_festival.so         app_record.so           app_zapateller.so
format_gsm.so   func_logic.so           res_monitor.so
app_followme.so         app_saycountpl.so       cdr_csv.so
format_h263.so  func_math.so            res_musiconhold.so
app_forkcdr.so          app_sayunixtime.so      cdr_custom.so
format_h264.so  func_md5.so             res_smdi.so
app_getcpeid.so         app_senddtmf.so         cdr_manager.so
format_ilbc.so  func_moh.so             res_speech.so
```

Como se puede ver en la ejecución anterior del comando *ls*, el *códec codec_alaw.so* se encuentra disponible, no así el *códec g729*, por lo que debemos descargarlo y copiarlo en este mismo directorio. Descargamos una versión *libre* del *códec* desde la web *http://asterisk.hosting.lv/bin/*, eligiendo correctamente el fichero a descargar en función de nuestra versión de *Asterisk* y la arquitectura de nuestro servidor:

```
$ wget http://asterisk.hosting.lv/bin/codec_g729-ast14-gcc4-glibc-
x86_64-core2.so
```

Descargado el *códec*, lo copiamos en el directorio */usr/lib/asterisk/modules* con el nombre *codec_g729.so* para que se encuentre accesible:

```
$ cp codec_g729-ast14-gcc4-glibc-x86_64-core2.so
/usr/lib/asterisk/modules/codec_g729.so
```

Finalmente debemos especificar la carga de los dos *códecs* mediante las siguientes sentencias en el fichero */etc/asterisk/modules.conf* cuya finalidad es la carga de módulos:

```
load => codec_alaw.so
load => codec_g729.so
```

Reiniciamos ahora *Asterisk* ejecutando el *script* correspondiente:

```
$ /etc/init.d/asterisk restart
```

Accedemos a la consola de *Asterisk* con el comando *asterisk –rvvv* y comprobamos que efectivamente los dos *códecs* que vamos a usar se encuentran accesibles, ejecutando la instrucción *core show translations* que muestra una tabla con los tiempos que toma la conversión (en milisegundos) de un segundo de datos entre diferentes formatos (*trascoding*):

```
$ asterisk -rvvv
Asterisk 1.4.28, Copyright (C) 1999 - 2009 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=========================================================================
  == Parsing '/etc/asterisk/asterisk.conf': Found
  == Parsing '/etc/asterisk/extconfig.conf': Found
Connected to Asterisk 1.4.28 currently running on deb1 (pid = 2978)
Verbosity was 0 and is now 3
deb1*CLI> core show translation
         Translation times between formats (in milliseconds) for one second of data
          Source Format (Rows) Destination Format (Columns)
```

```
          g723 gsm ulaw alaw g726aal2 adpcm slin lpc10 g729 speex ilbc g726 g722
     g723    -   -    -    -        -     -    -     -    -     -    -    -    -
      gsm    -   -    2    2        2     2    1     2    3     -    -    2    -
     ulaw    -   2    -    1        2     2    1     2    3     -    -    2    -
     alaw    -   2    1    -        2     2    1     2    3     -    -    2    -
 g726aal2    -   2    2    2        -     2    1     2    3     -    -    2    -
    adpcm    -   2    2    2        2     -    1     2    3     -    -    2    -
     slin    -   1    1    1        1     1    -     1    2     -    -    1    -
    lpc10    -   2    2    2        2     2    1     -    3     -    -    2    -
     g729    -   2    2    2        2     2    1     2    -     -    -    2    -
    speex    -   -    -    -        -     -    -     -    -     -    -    -    -
     ilbc    -   -    -    -        -     -    -     -    -     -    -    -    -
     g726    -   2    2    2        2     2    1     2    3     -    -    -    -
     g722    -   -    -    -        -     -    -     -    -     -    -    -    -
```

Es fácil comprobar que ambos formatos (*alaw y g729*) aparecen representados en la tabla y por tanto han sido instalados con éxito y se encuentran accesibles para *Asterisk.* Además, debemos permitir su uso modificando el fichero de configuración del servicio */etc/asterisk/sip.conf* cuyo contenido es el que se detalla a continuación, aceptando llamadas sin requerir autenticación alguna y procesando todo el tráfico *RTP* en el propio *Asterisk*:

```
[general]
context=PruebaSIPp ;Contexto por defecto en el entran las llamadas
allowguest=yes ;Acepta llamadas invitadas
canreinvite=no ;El trafico RTP pasa siempre por Asterisk
trustrpid=yes ;Confiar en una cabecera Remote-Party-ID
sendrpid=yes ;Enviar la cabecera Remote-Party-ID
disallow=all ;Deshabilita el uso de todos los codecs
allow=alaw ;Permite el uso del codec alaw
allow=gsm ;Permite el uso del codec gsm
allow=g729 ;Permite el uso del codec g729
```

Como podemos ver también se establece el contexto *PruebaSIPp* como contexto por defecto. Este contexto para el establecimiento y procesamiento de las llamadas de la prueba es configurado en el fichero *extensions.conf*. Lo hacemos de la siguiente forma:

```
[PruebaSIPp]
exten => _XXXX.,1,Set(ALEATORIO=$[${RAND(1,5)} * 100])
exten => _XXXX.,n,Goto(${ALEATORIO})

exten => _XXXX.,100,Dial(SIP/${EXTEN}@150.214.153.229) ;SIPp Server

exten => _XXXX.,200,Dial(SIP/${EXTEN}@150.214.153.97) ;Reject
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.229) ;SIPp Server

exten => _XXXX.,300,Dial(SIP/${EXTEN}@150.214.153.231) ;No Device
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.97) ;Reject
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.229) ;SIPp Server

exten => _XXXX.,400,Dial(SIP/${EXTEN}@1.1.1.1) ;Busy o No Route
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.97) ;Reject
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.231) ;No Device
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.229) ;SIPp Server

exten => _XXXX.,500,Dial(SIP/${EXTEN}@150.214.153.1) ;No VoIP
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.97) ;Reject
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.231) ;No Device
exten => _XXXX.,n,Dial(SIP/${EXTEN}@1.1.1.1) ;Busy o No Route
exten => _XXXX.,n,Dial(SIP/${EXTEN}@150.214.153.229) ;SIPp Server
```

El contexto se establece de esta manera que podamos realizar una simulación con un escenario lo más real posible:

- En la primera línea del contexto se genera un valor aleatorio entre 1 y 5 que determina cómo será procesada la llamada, cuya marcación debe ser de al menos cuatro dígitos.

- En la segunda línea, en función del valor generado en la línea anterior, se incluye una sentencia que redirige el flujo de procesamiento a una prioridad determinada (100, 200, 300, 400, 500).

- Dependiendo de la prioridad, la petición es gestionada como sigue:

- 100: la petición es redirigida directamente al servidor *SIPp* y se establece la comunicación.

- 200: la petición *SIP* se envía al servidor *Asterisk de apoyo* (*Reject, de rechazo*), el cual la rechaza al no disponer de ninguna configuración para la marcación. Tras el consumo del *timeout* correspondiente, la petición alcanza el servidor *SIPp*, como en el caso de prioridad 100.

- 300: el primer destino es una dirección en la que no hay ningún dispositivo (*No Device*). El *timeout* para la aceptación de la petición se agota y ésta pasa a los mismos destinos que en el caso de prioridad 200.

- 400: en esta oportunidad el primer destino de la petición es una dirección de red inexistente, a la que denominamos *No Route o Busy*.

- 500: en el peor de los casos, el dispositivo que recibe en primer lugar la petición *SIP* no es un dispositivo *VoIP* (es un router). Igualmente el *timeout* se consume totalmente y la petición pasa a ser enviada sucesivamente a las direcciones *No Router*, *No Device*, *Reject*, y finalmente al servidor *SIPp*, con el que sí es posible establecer la comunicación.

Las direcciones de red escritas para los diferentes destinos de las peticiones son siempre las mismas para los tres esquemas que sometemos a prueba:

- Servidor *SIPp*: 150.214.153.229

- Servidor *Asterisk de rechazo*: 150.214.153.97

- Destino *No Device*: 150.214.153.1

- Destino *No Route*: 1.1.1.1

Con toda la configuración necesaria realizada solo resta aumentar el número de archivos que el sistema operativo permite abrir por defecto con el comando *ulimit* y reiniciar *Asterisk* para que tome este nuevo valor y su configuración:

```
$ ulimit -n 20480
```

```
$ /etc/init.d/asterisk restart
```

En el servidor *Asterisk* también debemos ejecutar alguna herramienta que nos permita monitorizar el estado de los recursos de memoria y procesador, ya que como hemos dicho anteriormente en su consumo podemos apreciar con claridad cómo va reaccionando *Asterisk* a la carga de trabajo generada en la prueba. Para ello instalamos el paquete *sysstat* que incluye tres importantes utilidades en la monitorización del rendimiento en un sistema GNU/Linux: *sar*, *iostat y mpstat*.

```
$ apt-get install sysstat
```

6.2.3 Configuración del servidor *SIPp*

En el equipo servidor *SIPp*, una vez que tenemos instalado *SIPp* es sencilla la configuración que debemos realizar para poder llevar a cabo las pruebas. En primer lugar y para evitar problemas, aumentamos el número de archivos que el sistema operativo permite abrir por defecto de la misma forma como lo hemos hecho en el servidor *Asterisk*:

```
$ ulimit -n 20480
```

Después debemos copiar los dos ficheros con formato *xml* que definen los dos escenarios de prueba que vamos a ejecutar, con traducción de formato en las comunicaciones o *trascoding* (*serverG729.xml*) y sin ella (*serverG711.xml*), en el directorio de instalación de *SIPp*, */camino/al/directorio/sipp.svn*:

```
$ cp serverG711.xml /camino/a/sipp.svn/

$ cp serverG729.xml /camino/a/sipp.svn/
```

Veamos de forma breve el contenido de estos dos ficheros *xml*. El primero, *serverG729.xml*, es el que vamos a utilizar en el servidor *SIPp* para las pruebas en las que hay que realizar *trascoding*, con el cliente usando el *códec G711 (alaw)* y el servidor *G729*.

serverG729.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- 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.,
-->
<!-- 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
-->
```

```
<!--
-->
<!--                 Sipp default 'uas' scenario.
-->
<!--
-->

<scenario name="Basic UAS responder">
  <!-- By adding rrs="true" (Record Route Sets), the route sets
-->
  <!-- are saved and used for following messages sent. Useful to
test   -->
  <!-- against stateful SIP proxies/B2BUAs.
-->
  <recv request="INVITE" crlf="true">
  </recv>

  <!-- The '[last_*]' keyword is replaced automatically by the
-->
  <!-- specified header if it was present in the last message
received  -->
  <!-- (except if it was a retransmission). If the header was not
-->
  <!-- present or if no message has been received, the '[last_*]'
-->
  <!-- keyword is discarded, and all bytes until the end of the line
-->
  <!-- are also discarded.
-->
  <!--
-->
  <!-- If the specified header was present several times in the
-->
  <!-- message, all occurences are concatenated (CRLF seperated)
-->
  <!-- to be used in place of the '[last_*]' keyword.
-->

  <send>
    <![CDATA[

      SIP/2.0 180 Ringing
      [last_Via:]
      [last_From:]
      [last_To:];tag=[pid]SIPpTag01[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Length: 0

    ]]>
  </send>

  <send retrans="500">
    <![CDATA[

      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:];tag=[pid]SIPpTag01[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Type: application/sdp
      Content-Length: [len]

      v=0
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[media_ip_type] [media_ip]
```

```
      t=0 0
      m=audio [media_port] RTP/AVP 18
      a=rtpmap:18 G729/8000

    ]]>
  </send>

  <recv request="ACK"
        optional="true"
        rtd="true"
        crlf="true">
  </recv>

  <recv request="BYE">
  </recv>

  <send>
    <![CDATA[

      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Length: 0

    ]]>
  </send>

  <!-- Keep the call open for a while in case the 200 is lost to be
-->
  <!-- able to retransmit it if we receive the BYE again.
-->
  <pause milliseconds="4000"/>


  <!-- definition of the response time repartition table (unit is
ms)   -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150,
200"/>

  <!-- definition of the call length repartition table (unit is ms)
-->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000,
10000"/>

</scenario>
```

El segundo de estos ficheros para la configuración de escenario en el servidor, *serverG711.xml*, es el que define el escenario para las pruebas que realizamos con cliente y servidor compartiendo *códec*: *G711 (alaw)*. Su contenido se presenta a continuación.

serverG711.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- 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.,
-->
<!-- 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
-->
<!--
-->
<!--                 Sipp default 'uas' scenario.
-->
<!--
-->

<scenario name="Basic UAS responder">
  <!-- By adding rrs="true" (Record Route Sets), the route sets
-->
  <!-- are saved and used for following messages sent. Useful to
test   -->
  <!-- against stateful SIP proxies/B2BUAs.
-->
  <recv request="INVITE" crlf="true">
  </recv>

  <!-- The '[last_*]' keyword is replaced automatically by the
-->
  <!-- specified header if it was present in the last message
received  -->
  <!-- (except if it was a retransmission). If the header was not
-->
  <!-- present or if no message has been received, the '[last_*]'
-->
  <!-- keyword is discarded, and all bytes until the end of the line
-->
  <!-- are also discarded.
-->
  <!--
-->
  <!-- If the specified header was present several times in the
-->
  <!-- message, all occurences are concatenated (CRLF seperated)
-->
  <!-- to be used in place of the '[last_*]' keyword.
-->

  <send>
    <![CDATA[

      SIP/2.0 180 Ringing
      [last_Via:]
      [last_From:]
      [last_To:];tag=[pid]SIPpTag01[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
```

```
      Content-Length: 0

    ]]>
  </send>

  <send retrans="500">
    <![CDATA[

      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:];tag=[pid]SIPpTag01[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Type: application/sdp
      Content-Length: [len]

      v=0
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[media_ip_type] [media_ip]
      t=0 0
      m=audio [media_port] RTP/AVP 8
      a=rtpmap:8 PCMA/8000
]]>
  </send>

  <recv request="ACK"
        optional="true"
        rtd="true"
        crlf="true">
  </recv>

  <recv request="BYE">
  </recv>

  <send>
    <![CDATA[

      SIP/2.0 200 OK
      [last_Via:]
      [last_From:]
      [last_To:]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Length: 0

    ]]>
  </send>

  <!-- Keep the call open for a while in case the 200 is lost to be
-->
  <!-- able to retransmit it if we receive the BYE again.
-->
  <pause milliseconds="4000"/>


  <!-- definition of the response time repartition table (unit is
ms)   -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150,
200"/>

  <!-- definition of the call length repartition table (unit is ms)
-->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000,
10000"/>
```

```
</scenario>
```

Con los ficheros de definición de los escenarios que vamos a usar en las pruebas de rendimiento copiados en el directorio *sipp.svn*, terminamos de configurar el servidor *SIPp* con la instalación del paquete *sysstat*, ya que también tenemos que monitorizar el consumo experimentado de memoria y procesador en él:

```
$ apt-get install sysstat
```

6.2.4 Configuración del cliente *SIPp*

La configuración del equipo cliente para las pruebas de rendimiento es prácticamente igual a la del servidor, salvo que en este caso disponemos tan sólo de un fichero de configuración de escenario para ambas pruebas (con y sin *trascoding*) ya que el cliente *SIPp* siempre usa el *códec alaw (G711)*. Este fichero, *clientG711.xml*, debe estar copiado en el directorio de instalación de SIPp /camino/a/sipp.svn:

```
$ cp clientG711.xml /camino/a/sipp.svn/
```

A continuación es mostrado el contenido de esta configuración.

***clientG711.xml**:*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- 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.,
-->
<!-- 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
-->
<!--
-->
<!--                 Sipp 'uac' scenario with pcap (rtp) play
-->
```

```
<!--
-->

<scenario name="UAC with media">
  <!-- In client mode (sipp placing calls), the Call-ID MUST be
-->
  <!-- generated by sipp. To do so, use [call_id] keyword.
-->
  <send retrans="500" start_rtd="1">
    <![CDATA[

      INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport]
[local_ip]:[local_port];branch=[branch]
      From: sipp
<sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 1 INVITE
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]

      v=0
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[local_ip_type] [local_ip]
      t=0 0
      m=audio [auto_media_port] RTP/AVP 8 101
      a=rtpmap:8 PCMA/8000
      a=rtpmap:101 telephone-event/8000
      a=fmtp:101 0-11,16

    ]]>
  </send>

  <recv response="100" optional="true" rtd="1" start_rtd="2">
  </recv>

  <recv response="180" optional="true" rtd="2">
  </recv>

  <!-- By adding rrs="true" (Record Route Sets), the route sets
-->
  <!-- are saved and used for following messages sent. Useful to
test   -->
  <!-- against stateful SIP proxies/B2BUAs.
-->
  <recv response="200" crlf="true">
  </recv>

  <!-- Packet lost can be simulated in any send/recv message by
-->
  <!-- by adding the 'lost = "10"'. Value can be [1-100] percent.
-->
  <send>
    <![CDATA[

      ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport]
[local_ip]:[local_port];branch=[branch]
      From: sipp
<sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
      To: sut
<sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
```

```
      CSeq: 1 ACK
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Length: 0

    ]]>
  </send>

  <!-- Play a pre-recorded PCAP file (RTP stream)
-->
  <nop>
    <action>
      <exec play_pcap_audio="pcap/call_g711a.pcap"/>
    </action>
  </nop>

  <!-- Pause 3 minutes, which is approximately the duration of the
-->
  <!-- PCAP file
-->
  <pause milliseconds="180000"/>

  <!-- Play an out of band DTMF '1'
-->
  <!--<nop>
    <action>
      <exec play_pcap_audio="pcap/dtmf_2833_1.pcap"/>
    </action>
  </nop>

  <pause milliseconds="1000"/>
  -->
  <!-- The 'crlf' option inserts a blank line in the statistics
report. -->
  <send retrans="500">
    <![CDATA[

      BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport]
[local_ip]:[local_port];branch=[branch]
      From: sipp
<sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
      To: sut
<sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      CSeq: 2 BYE
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Length: 0

    ]]>
  </send>

  <recv response="200" crlf="true">
  </recv>

  <!-- definition of the response time repartition table (unit is
ms)   -->
  <ResponseTimeRepartition value="50, 100, 200, 500, 1100, 2100,
3100, 4100, 5100, 6100, 10000"/>

  <!-- definition of the call length repartition table (unit is ms)
-->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000,
10000"/>

</scenario>
```

Otra diferencias en la configuración del cliente *SIPp* y que podemos apreciar en las líneas destacadas en negrita del fichero anterior son la inclusión de audio para simular comunicaciones reales que los clientes van a enviar al servidor *Asterisk*, que a su vez lo reenvía al servidor *SIPp*, y un tiempo de espera igual a un segundo antes de continuar con la ejecución de la llamada. Este audio en formato *pcap*, que dispone de una duración de tres minutos, debe ser copiado en el directorio */pcap* bajo el directorio de instalación de *SIPp /camino/a/sipp.svn*:

```
$ cp call_g711a.pcap /camino/a/sipp.svn/pcap/
```

Para terminar con la configuración del cliente *SIPp* aumentamos el límite de ficheros que permite abrir el sistema operativo:

```
$ ulimit -n 20480
```

E instalamos el paquete *sysstat* ya que también vamos a recoger estadísticas sobre el consumo de recursos en el sistema cliente que genera las llamadas de la prueba de rendimiento:

```
$ apt-get install sysstat
```

6.3 EJECUCION DE LAS PRUEBAS

En los siguientes apartados vamos a ver tras la configuración de servidor *Asterisk*, cliente y servidor *SIPp*, qué comandos debemos ejecutar en cada uno de ellos para el desarrollo de las pruebas, tanto en el primer caso incluyendo traducción de *códecs* como en el segundo que no lo hace. La configuración es la misma para los tres esquemas que queremos probar, con la única diferencia de la dirección de red del servidor *Asterisk*.

6.3.1 Prueba con *trascoding*

En esta primera prueba como sabemos cliente y servidor *SIPp* usan *códecs* diferentes para las comunicaciones por lo que una traducción entre *códecs* es necesaria en el servidor *Asterisk*, lo que lógicamente vamos a comprobar conlleva una mayor carga computacional para el procesamiento de las llamadas. El escenario queda configurado en el cliente con el fichero *clientG711.xml* y en el servidor con *serverG729.xml*. Debemos realizar la prueba siempre iniciando la ejecución en este orden: servidor *Asterisk*, servidor *SIPp*, cliente *SIPp*.

Ejecución del servidor *Asterisk*

En primer lugar y antes de iniciar la ejecución de la prueba en el servidor *Asterisk* debemos aumentar el número de ficheros que permite abrir el sistema operativo e iniciar el servicio:

```
$ ulimit –n 20480

$ /etc/init.d/asterisk start
```

Después tan sólo iniciamos la herramienta de monitorización *sar*, que pertenece al paquete *sysstat* antes instalado. *Sar (System Activity Reporter)* permite tomar todo tipo de muestras de valores sobre el consumo de recursos como memoria, procesador, interfaces de red, etc. Para ello pasamos al comando *sar* como parámetros el fichero de salida que almacena la información (con el flag *-o*), el tiempo de muestreo (por defecto en segundos), y el número de muestras:

```
$ sar -o ./fichero.sar frecuencia total >/dev/null 2>&1 &
```

Así, tomamos por ejemplo 500 muestras, una cada segundo:

```
$ sar -o ./ast.sar 1 500 >/dev/null 2>&1 &
```

Cuando la ejecución de *sar* finaliza es posible extraer y filtrar la información del fichero de salida con el propio comando. Siempre extraemos la información en dos ficheros *.txt* para su correcta visualización y realización de gráficos sin problemas; uno con los datos relativos al procesador y otro con los datos relativos a memoria (añadiendo el flag *-r*):

```
$ sar -f ast.sar > ast_sar_proc.txt

$ sar -f ast.sar -r > ast_sar_mem.txt
```

Ejecución del servidor *SIPp*

En el servidor *SIPp*, que dispondrá siempre de la dirección de red 150.214.153.229, debemos ejecutar tanto el software *SIPp* como la utilidad *sar*, ya que también debemos monitorizar su estado durante la ejecución de la prueba. Antes de iniciar *SIPp* como servidor aumentamos el número de ficheros que pueden ser abiertos por el sistema operativo y ejecutamos *sar* de igual forma que en el servidor *Asterisk*:

```
$ ulimit –n 20480

$ sar -o ./ser.sar 1 500 >/dev/null 2>&1 &
```

Para iniciar el servidor *SIPp* ejecutamos el comando *sipp* en el directorio de instalación de *SIPp*, */camino/a/sipp.svn/*, pasándole algunos parámetros para indicar el tipo de simulación y el escenario que vamos a usar en ella:

```
$ ./sipp -sf serverG729.xml -nd -i 150.214.153.229 -mi
150.214.153.229 -rtp_echo
```

Veamos el significado de cada uno de los parámetros y valores introducidos:

- *-sf*. Nos permite cargar el escenario definido en el fichero facilitado a continuación, en este caso para la prueba con *trascoding serverG729.xml*.

- *-nd*. Permite configurar el comportamiento de *SIPp* con las siguientes características:

  - Cuando se produce una retransmisión porque el *timeout* ha sido consumido la llamada es abortada.

  - Si el servidor recibe inesperadamente un mensaje de tipo *BYE* o *CANCEL*, éste responde con un mensaje *200 OK* y la llamada es abortada.

  - Si el servidor recibe inesperadamente un mensaje que no sea de tipo *BYE* o *CANCEL*, éste responde con un mensaje *CANCEL* finalizando o abortando así la llamada, respectivamente.

- *-i*. Indicamos de esta forma que el servidor va a recibir tráfico *SIP* en la dirección 150.214.153.229.

- *-mi*. El servidor también va a recibir en la misma dirección anterior tráfico *RTP*.

- *-rtp_echo*. Utilizamos este parámetro para que el servidor replique la comunicación del cliente, simulando una llamada real en la que intervienen llamante y llamado.

De esta forma, el servidor queda a la espera de recibir llamadas por parte del cliente *SIPp*, en el estado que podemos observar en la figura 5.7.

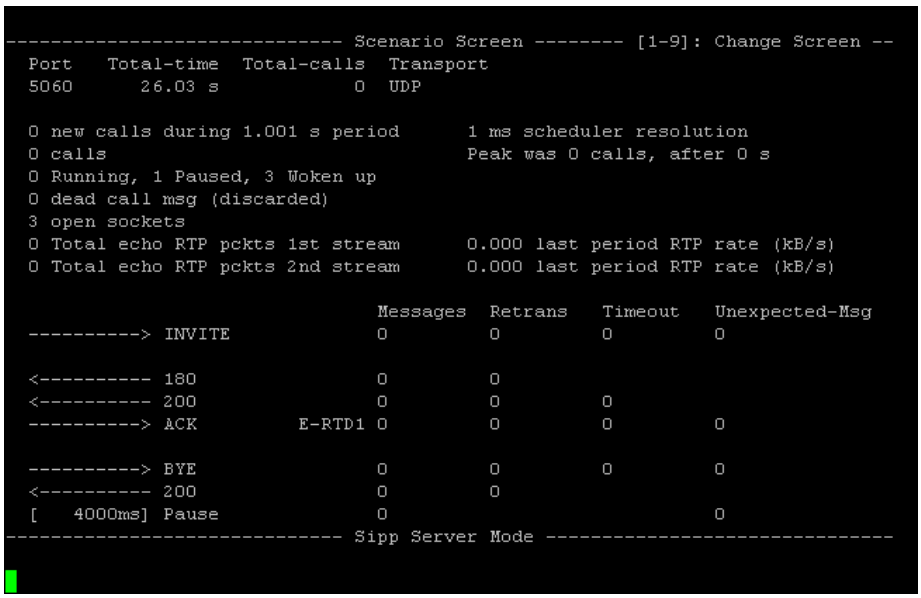


Figura 5.7. Servidor SIPp en ejecución.

Ejecución del cliente *SIPp*

El cliente *SIPp*, al igual que el servidor, siempre va a estar también ubicado en el mismo equipo durante todas las pruebas realizadas en los tres esquemas implementados, con dirección de red 150.214.153.228. Como hemos comentado anteriormente siempre utiliza el *códec alaw* para las llamadas de las pruebas, quedando el escenario de las mismas definido en el fichero *clienteG711.xml*. Antes de iniciar las pruebas de rendimiento aumentamos como en el resto de equipos el número de ficheros que permite abrir el sistema operativo e iniciamos la utilidad de monitorización de los recursos del sistema *sar*:

```
$ ulimit –n 20480

$ sar -o ./cli.sar 1 500 >/dev/null 2>&1 &
```

Para la ejecución de *SIPp* nos situamos en el directorio de instalación */camino/a/sipp.svn* y ejecutamos el comando *sipp* como sigue, facilitando parámetros que nos permiten configurar la generación de peticiones en la prueba:

```
$ ./sipp -sf clientG711.xml -nd -m 500 -r 3 -rp 1s -l 340 -s
9876543210 remote_host 150.214.153.227
```

En las líneas anteriores se recoge un ejemplo de ejecución de prueba. Veamos qué significa cada uno de estos parámetros y cómo puede ser configurada la prueba a nuestro gusto dependiendo de la carga de trabajo que queramos aplicar al servidor *Asterisk*:

- *-sf*. Comentado ya anteriormente, permite indicar el escenario a usar para la prueba. Como fue presentado, se trata del fichero *clientG711.xml*.

- *-nd*. Establece el comportamiento de *SIPp* ante determinadas situaciones, de la misma forma que fue indicado para la ejecución del comando para el servidor *SIPp*.

- *-m*. Permite establecer el número total de llamadas que el cliente *SIPp* va a realizar.

- *-r y -rp*. Mediante el uso de estos dos parámetros configuramos la frecuencia de envío de las llamadas por parte del cliente. En el ejemplo presentado, tres llamadas por segundo.

- *-l*. Permite establecer el número máximo de llamadas que el cliente *SIPp* va a mantener simultáneamente.

- *-s*. Permite escribir la extensión a la que llamar durante las pruebas.

- *remote_host*. Mediante esta opción especificamos el *host remoto* al que el cliente *SIPp* va a enviar las peticiones *SIP*, es decir, la dirección de red del servidor *Asterisk* cuya capacidad en número de llamadas simultáneas queremos medir. En el caso del primer esquema, con servidor *Asterisk* real, la dirección es 150.214.153.227. En los casos del segundo y tercer esquema, es la dirección que se configura en el fichero de configuración de la máquina virtual sobre la que se ha instalado *Asterisk*.

El estudio y selección de los valores de todos estos parámetros presentados es muy importante a la hora de ejecutar y evaluar las pruebas de rendimiento realizadas. Lo recomendable, y así es como se ha trabajado durante la ejecución de las mismas, es comenzar con un número total y de llamadas simultáneas (parámetros *–m* y *-l*) bajo para ir aumentándolo progresivamente al ver la evolución del consumo de recursos por parte del servidor *Asterisk*, cliente y servidores *SIPp*. Por ejemplo, trabajando con estos parámetros, también se observó que cuanto mayor es la diferencia entre los dos valores mayor es el estrés al que se somete al servidor *Asterisk*, ya que debe permanecer durante más tiempo con el mayor número de llamadas simultáneas generadas por el cliente *SIPp*.

6.3.2 Prueba sin *trascoding*

La ejecución de pruebas sin realizar *trascoding* difiere de la anterior solamente en el escenario usado por parte del servidor *SIPp*, que en este caso es el recogido en el fichero *serverG711.xml*. Todos los comandos ejecutados por tanto son idénticos a los citados en el apartado anterior, con la salvedad de este cambio en la ejecución del servidor *SIPp*.

Ejecución del servidor *SIPp*

La ejecución del servidor *SIPp* es similar a la anteriormente vista, cambiando el fichero de configuración del escenario de la prueba para que haga uso del *códec G711*, al igual que el cliente *SIPp*. Esto lo hacemos con el parámetro *–sf* del comando *sipp* en el directorio de instalación */camino/a/sipp.svn*:

```
$ ./sipp -sf serverG711.xml -nd -i 150.214.153.229 -mi
150.214.153.229 -rtp_echo
```

6.4 ESQUEMA 1: SERVIDOR REAL

En este primer esquema diseñado **no se incluye ningún tipo de virtualización**, sino que **el servidor *Asterisk* se ejecuta como lo hace normalmente en un servidor físico**. La puesta en marcha y prueba de este escenario es importante ya que permite extraer conclusiones comparando el rendimiento presentado por un servidor *Asterisk* instalado en una máquina física y el obtenido al ejecutarlo dentro de una máquina virtual, como vemos en el *Escenario 2: Servidor virtual*. Procedemos entonces a presentar su arquitectura y los resultados obtenidos tanto en la prueba de rendimiento incluyendo *trascoding* como la que no lo incluye.

6.4.1 Arquitectura

La arquitectura que presenta este esquema es sencilla como vamos a ver a continuación. **Incluye cuatro servidores: un servidor *Asterisk***, cuyo rendimiento queremos *testear*, **dos servidores para la ejecución del software de prueba *SIPp*** (*cliente y servidor*), y finalmente **un servidor *Asterisk* adicional requerido por el escenario de prueba *SIPp***, también con el servicio *Asterisk* ejecutándose pero sin configuración alguna pues debe rechazar todas las llamadas que reciba.

El objetivo de la prueba recordamos es llegar a **conocer aproximadamente la carga máxima soportada por el servidor *Asterisk*, representada por el número de llamadas simultáneas que puede procesar**. En la figura *5.8 Arquitectura del Esquema 1: Servidor real* podemos ver los diferentes servidores que intervienen en la prueba:

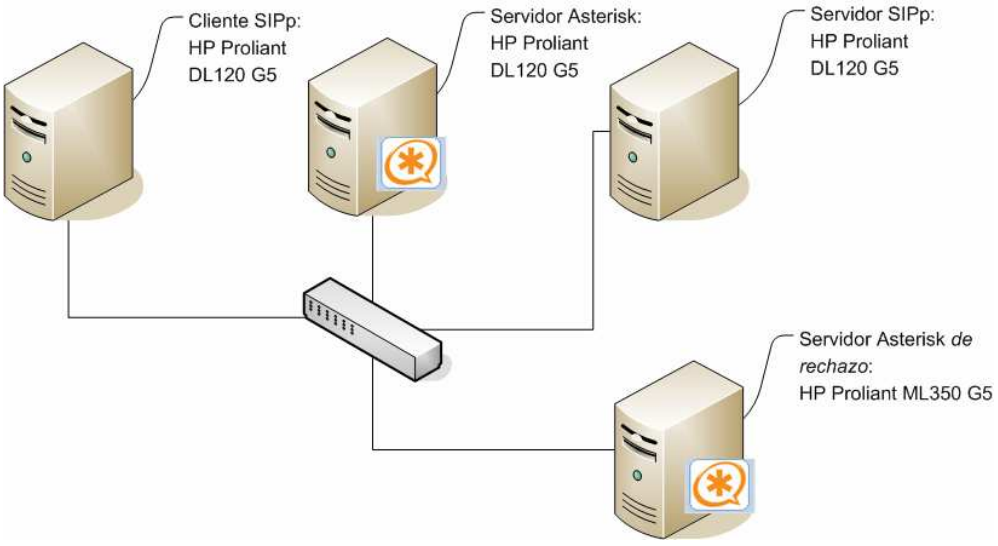


Figura 5.8. Arquitectural del Esquema 1: Servidor real.

Detallamos un poco más cada uno de los componentes:

- **Servidor *Asterisk (deb1)*.** Se trata de un servidor **HP Proliant DL120 G5** sobre el que únicamente se está ejecutando el servicio *Asterisk*, que se encarga del establecimiento,

mantenimiento y finalización de las llamadas. El servidor dispone de un procesador **Intel(R) Xeon(R) CPU E3110 @ 3.00GHz** y de **4Gb de memoria** RAM.

- **Servidor *SIPp (deb2).*** Al igual que el servidor *Asterisk* es un **HP Proliant DL120 G5**. Ejecutará el servidor *SIPp*, encargado de recibir las llamadas creadas por el cliente *SIPp*.

- **Cliente *SIPp (deb3).*** También es un servidor **HP Proliant DL120 G5**, aunque en este caso será el cliente *SIPp* el que se ejecute en el, creando las llamadas que componen la carga de trabajo.

- **Servidor *Asterisk de apoyo (fed1).*** Este último servidor es necesario incorporarlo en el escenario de prueba de *SIPp* para que éste se asemeje lo máximo posible a un escenario real. Como veremos más adelante en la configuración del servidor *SIPp*, no todas las llamadas son procesadas por el servidor *Asterisk* sino que existe la posibilidad de que sean enviadas a otros destinos, como este servidor *Asterisk de apoyo* que *por defecto* rechaza toda llamada. Para ello *Asterisk* no tiene configuración alguna. Es el único servidor diferente: **HP Proliant ML350 G5**, con **2Gb de memoria** RAM y procesador **Intel(R) Xeon(R) CPU E5320 @ 1.86GHz**.

Las pruebas de rendimiento haciendo uso del software *SIPp* requieren de una gran cantidad de recursos *hardware*, tanto de memoria como de procesamiento en los servidores con rol de cliente o servidor *SIPp*, debido a la gran carga de trabajo con la que deben lidiar, tanto de mensajes *SIP* como mensajes *RTP*. De lo contrario es más que probable que lleguen a saturarse antes que el propio servidor *Asterisk*. También hay que tener en cuenta el consumo de ancho de banda de las conexiones de red; los 4 servidores disponen de interfaces de red *Gigabit Ethernet* y se encuentran interconectados mediante un *switch Gigabit*, por lo que no hay problema alguno en este aspecto debido a que estos adaptadores de red sólo podrían representar un cuello de botella tras saturarse los equipos en memoria y procesamiento, ya que la cantidad de información que son capaces de procesar es superior. Por lo tanto, al mismo tiempo que ejecutamos el software de prueba *SIPp* monitorizamos el estado de los recursos de memoria y procesador en el servidor *Asterisk*, servidor *SIPp* y cliente *SIPp* mediante la utilidad *sar* del paquete *sysstat*, para recogida de estadísticas sobre el rendimiento de los componentes de un equipo.

6.4.2 Resultados con *trascoding*

Como hemos comentado con anterioridad el procedimiento consistió en ejecutar una serie de pruebas haciendo ligeras variaciones en los parámetros que configuran la carga de la prueba en la generación de las llamadas en el cliente *SIPp*, comenzando por un número reducido tanto de llamadas totales como de llamadas simultáneas.

Después de realizar en total 5 ejecuciones, se eligió la cuarta de ellas pues aunque aparecían pequeños picos de saturación en el procesador del servidor *Asterisk* el total de las llamadas fue completado con éxito. En lo que respecta a memoria el consumo registrado no fue de importancia (y así ocurre siempre, en todas las pruebas realizadas para todos los esquemas).

En la figura 5.9 podemos apreciar el consumo de procesador por parte del servidor *Asterisk* durante la prueba elegida. A partir de que el máximo de 335 llamadas simultáneas se estableciera en el cliente comenzaron a aparecer pequeños picos de saturación, sin que aparezcan retransmisiones ni mensajes inesperados como hemos dicho. Después, al finalizar las llamadas, siguieron creándose las restantes hasta completar un total de 500, pero ya no se mantuvo el máximo número de llamadas simultáneas por lo que el consumo de procesador fue

progresivamente recuperándose. En cuanto a los otros dos equipos principales de la prueba, el cliente *SIPp* ha registrado un consumo cercano al 90% de CPU, mientras que en el servidor *SIPp* ha sido mucho menor (10% aproximadamente).

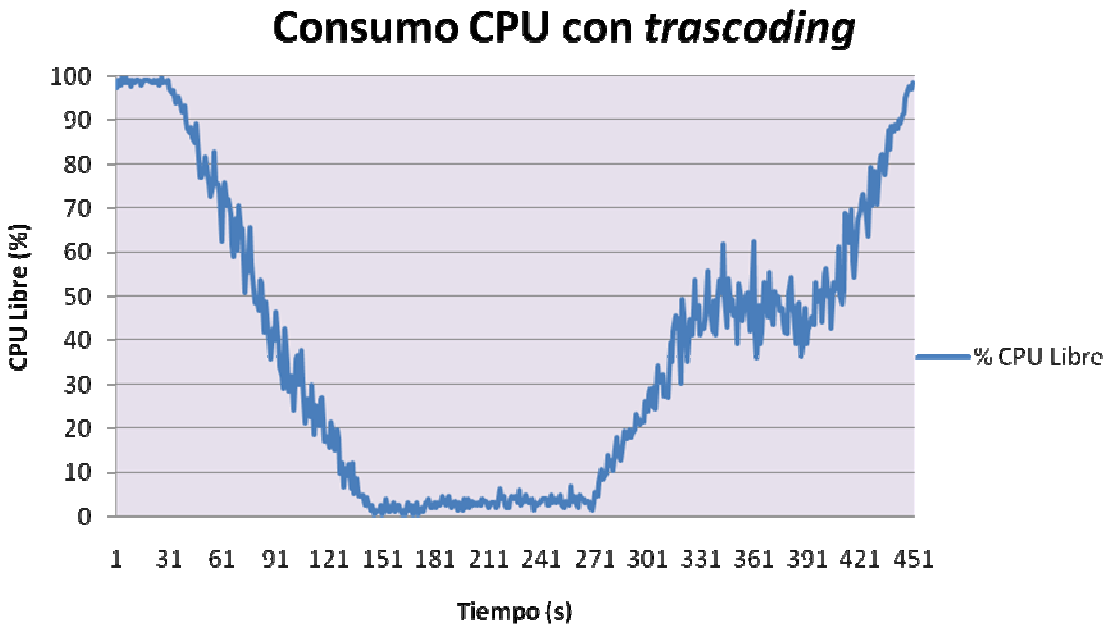


Figura 5.9. Consumo de CPU en la prueba con trascoding para el servidor Asterisk real.

Así, ni servidor ni cliente *SIPp* tuvieron que realizar retransmisión alguna de mensajes, ni recibieron mensajes inesperados (esto es lo que suele ocurrir cuando la saturación del servidor llega a un punto en el que le hace perder su estabilidad en el procesamiento de las peticiones) como se puede observar en las figuras 5.10 y 5.11, que reflejan el estado final del software al concluir la simulación.

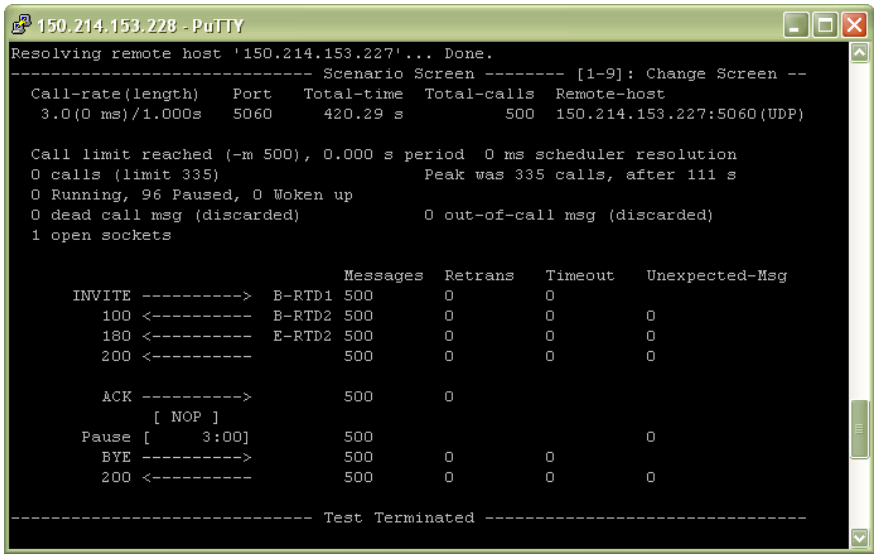


Figura 5.10. Estado final de SIPp en el cliente para el escenario con trascoding.

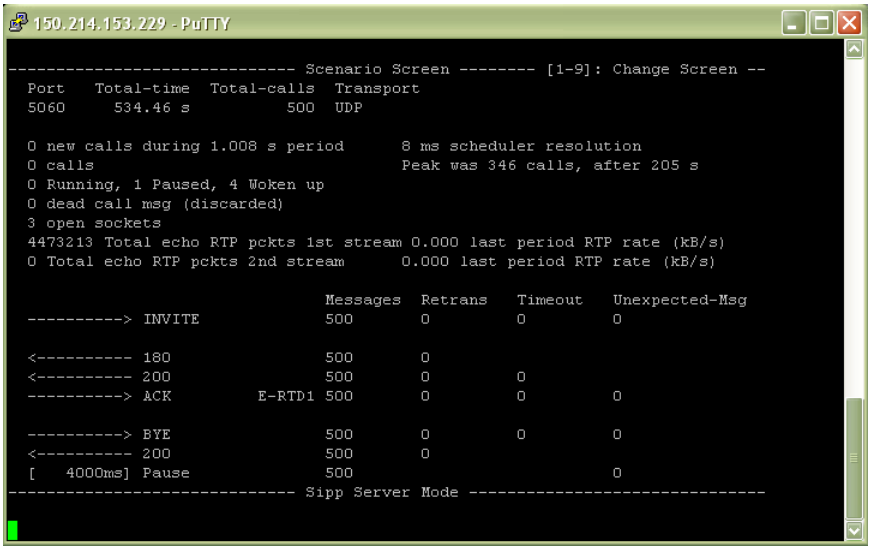


Figura 5.11. Estado final de SIPp en el servidor para el escenario con trascoding.

Por tanto, realizada la prueba, podemos concluir que **el número máximo de llamadas simultáneas que soporta aproximadamente el servidor *Asterisk* real (pico observado en el servidor *SIPp*) es de aproximadamente 346 llamadas a los 205 segundos del comienzo de la prueba**. El cliente *SIPp* fue configurado para la generación de 335 llamadas simultáneas como máximo y 500 llamadas totales, con una frecuencia de 3 llamadas generadas por segundo.

6.4.3 Resultados sin *trascoding*

En esta segunda prueba de rendimiento en la que los dos equipos, cliente y servidor *SIPp*, hacen uso del mismo *códec G711* para las comunicaciones hemos llegado a realizar una serie de hasta 7 pruebas, teniendo en la última **el cliente *SIPp* saturado por completo tras el establecimiento de las 345 llamadas simultaneas**. Como se puede apreciar en la figura 5.5 **comienzan a aparecer retransmisiones de mensajes *SIP* en el cliente**, **provocando que no se completen con éxito las 500 llamadas totales**, sino que solamente lo hacen 486 (fallando 14).

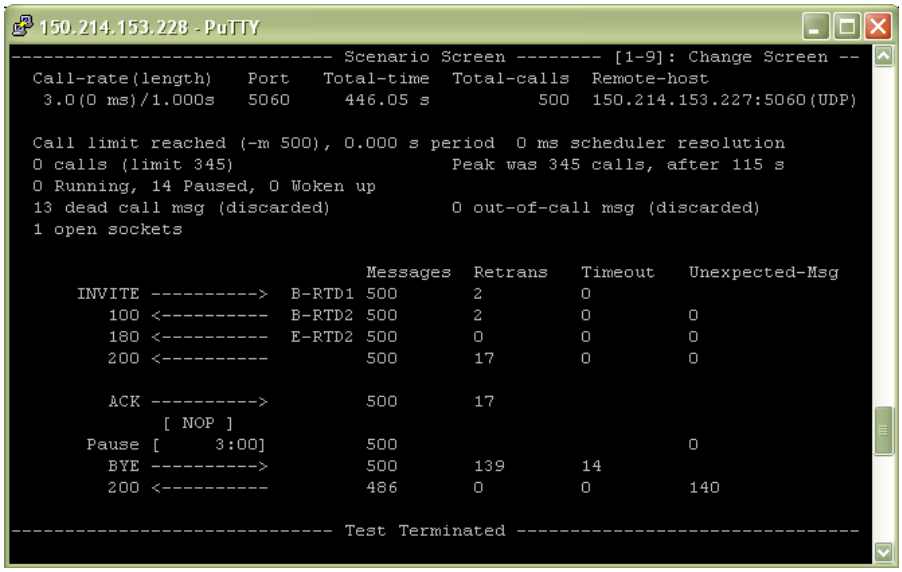


Figura 5.12. Estado final de SIPp en el cliente para el escenario sin trascoding-I.

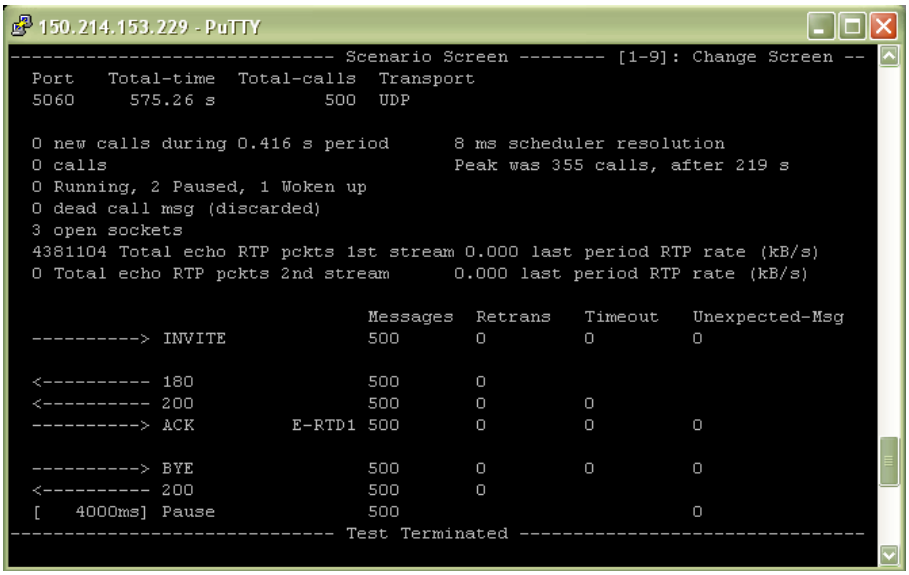


Figura 5.13. Estado final de SIPp en el servidor para el escenario sin trascoding-I.

El cliente *SIPp* se satura antes que el servidor *Asterisk* al liberar a éste de la traducción entre dos *códecs* diferentes, lo que en la prueba anterior vista suponía una gran carga de trabajo *extra* en el procesado de las peticiones y llamadas de la prueba. De hecho, el servidor *Asterisk* llega a presentar una carga como máximo de 55% de CPU, aproximadamente (véase la figura 5.14).

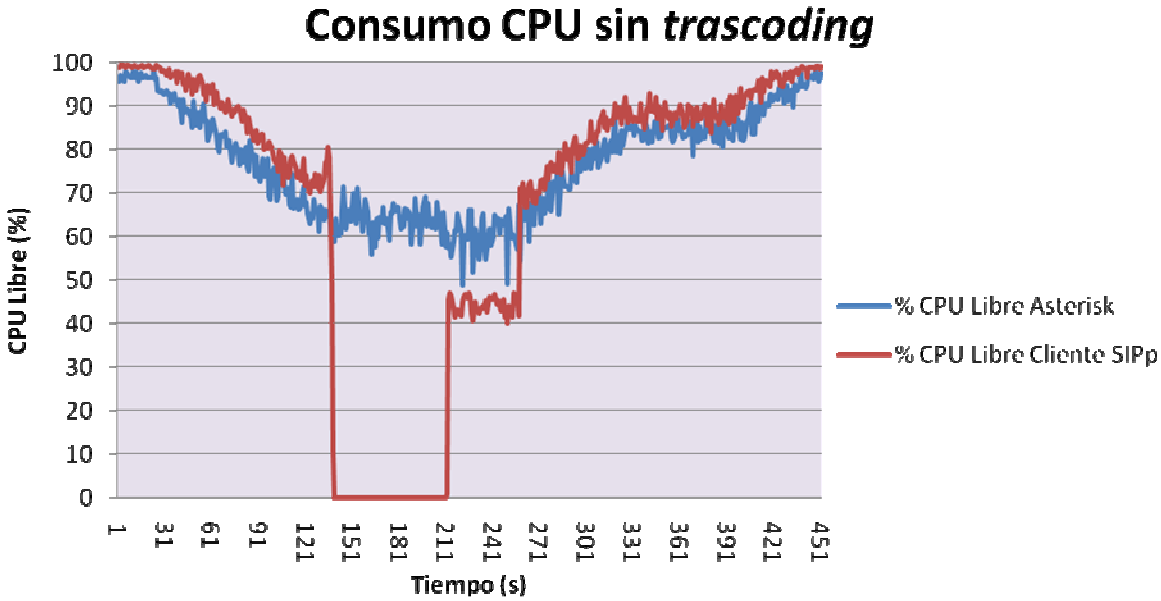


Figura 5.14. Consumo de CPU en la prueba sin trascoding para el servidor Asterisk real-I.

Llegado este punto podemos bien incluir más clientes para intentar llegar a saturar el servidor *Asterisk* con un mayor número de llamadas simultáneas y totales, **o bien**

realizar una estimación del total de llamadas simultáneas que puede llegar a soportar al saber basándonos en una prueba anterior (ver figuras 5.15 a 5.17) en la que se han podido completar todas las llamadas con éxito. En esta prueba elegida para la estimación se mantuvo un total de 346 llamadas simultáneas como pico en el servidor *SIPp* representando una carga total del 30% de CPU aproximadamente en el servidor *Asterisk*. Elegimos realizar una estimación obteniendo así que **el número de llamadas simultáneas que el servidor *Asterisk* puede soportar sin realizar *trascoding* es en torno a las 1150 llamadas**.

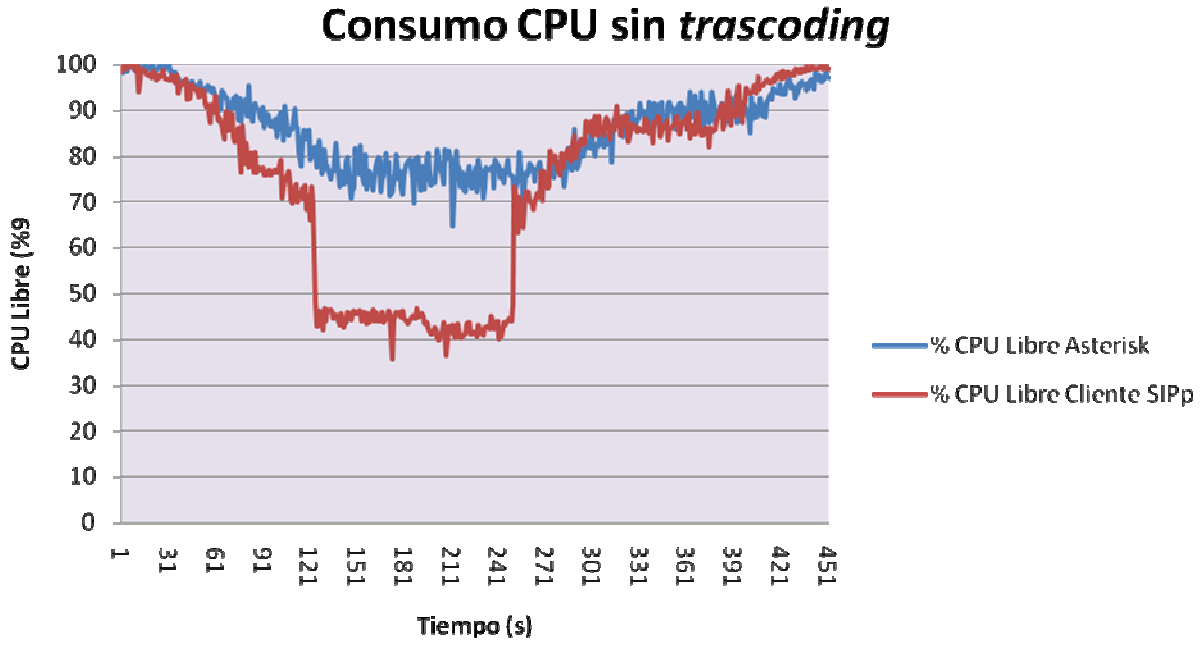


Figura 5.15. Consumo de CPU en la prueba sin trascoding para el servidor Asterisk real-II.

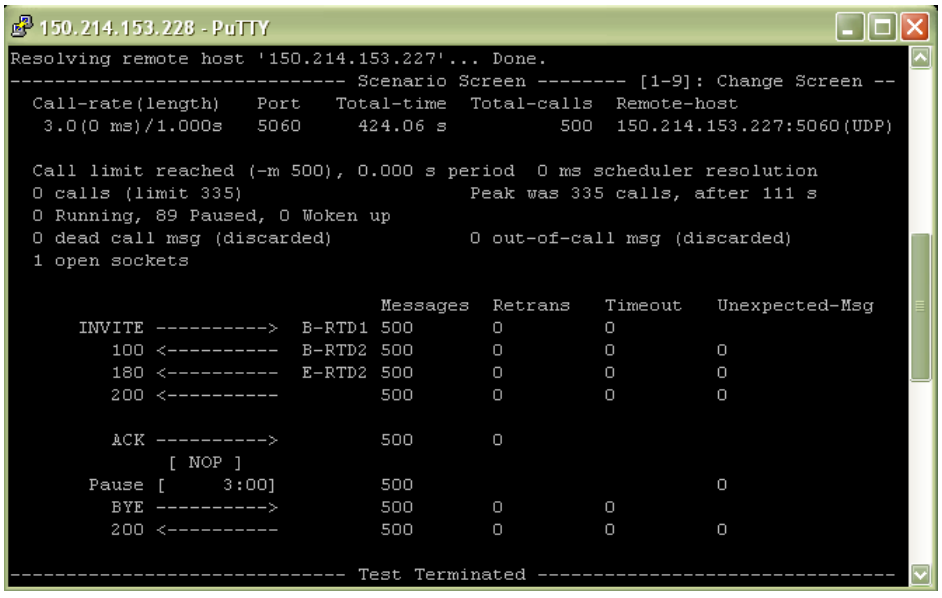


Figura 5.16. Estado final de SIPp en el cliente para el escenario sin trascoding-II.

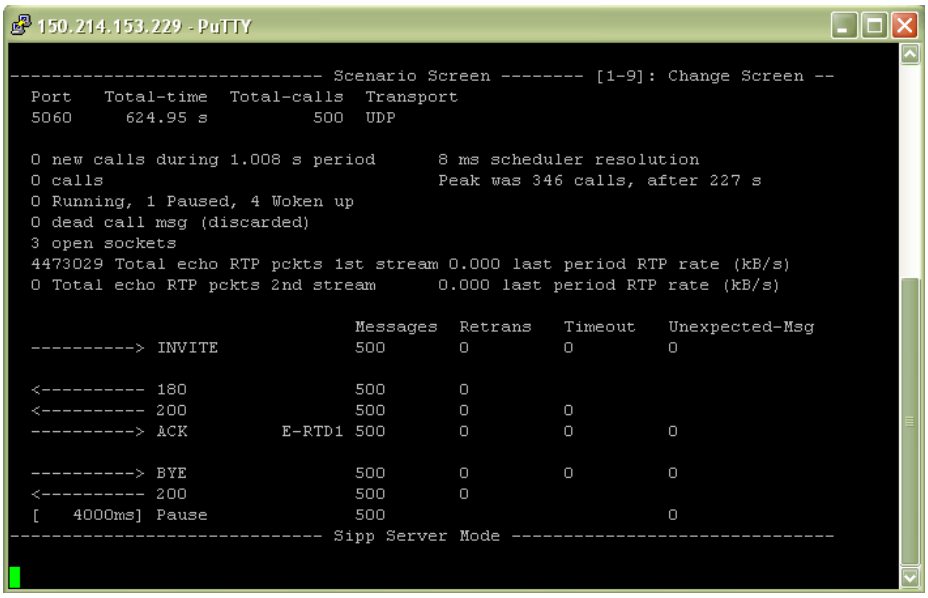


Figura 5.17. Estado final de SIPp en el cliente para el escenario sin trascoding-II.

6.5 ESQUEMA 2: SERVIDOR VIRTUAL

Ahora, damos un paso adelante en nuestras pruebas de rendimiento e integramos el servidor *Asterisk* en un dominio Xen en lugar de en un servidor físico. Por tanto ahora sí **introducimos virtualización, por lo que hay que prestar atención a la configuración del dominio que acoge al servicio para que disponga de los recursos suficientes y los dispositivos necesarios, sobre todo en cuanto a memoria y procesamiento**. Solamente hay un dominio en nuestra infraestructura virtual, motivo por el cual le son asignados recursos más que suficientes y que su rendimiento y resultados puedan ser comparados de igual a igual con los obtenidos para el *Asterisk* en el servidor real. Veamos la arquitectura detallada para este segundo esquema a continuación.

6.5.1 Arquitectura

Como en el esquema anterior, en este segundo esquema la arquitectura **incluye cuatro servidores** (véase la figura 5.18)**: dos servidores para la ejecución del software de prueba *SIPp*** (*cliente y servidor*)**, un servidor *Asterisk* adicional requerido por el escenario de prueba *SIPp* que rechaza todas las llamadas**, y como novedad **un servidor anfitrión con Xen instalado como plataforma para la virtualización del servidor *Asterisk***.

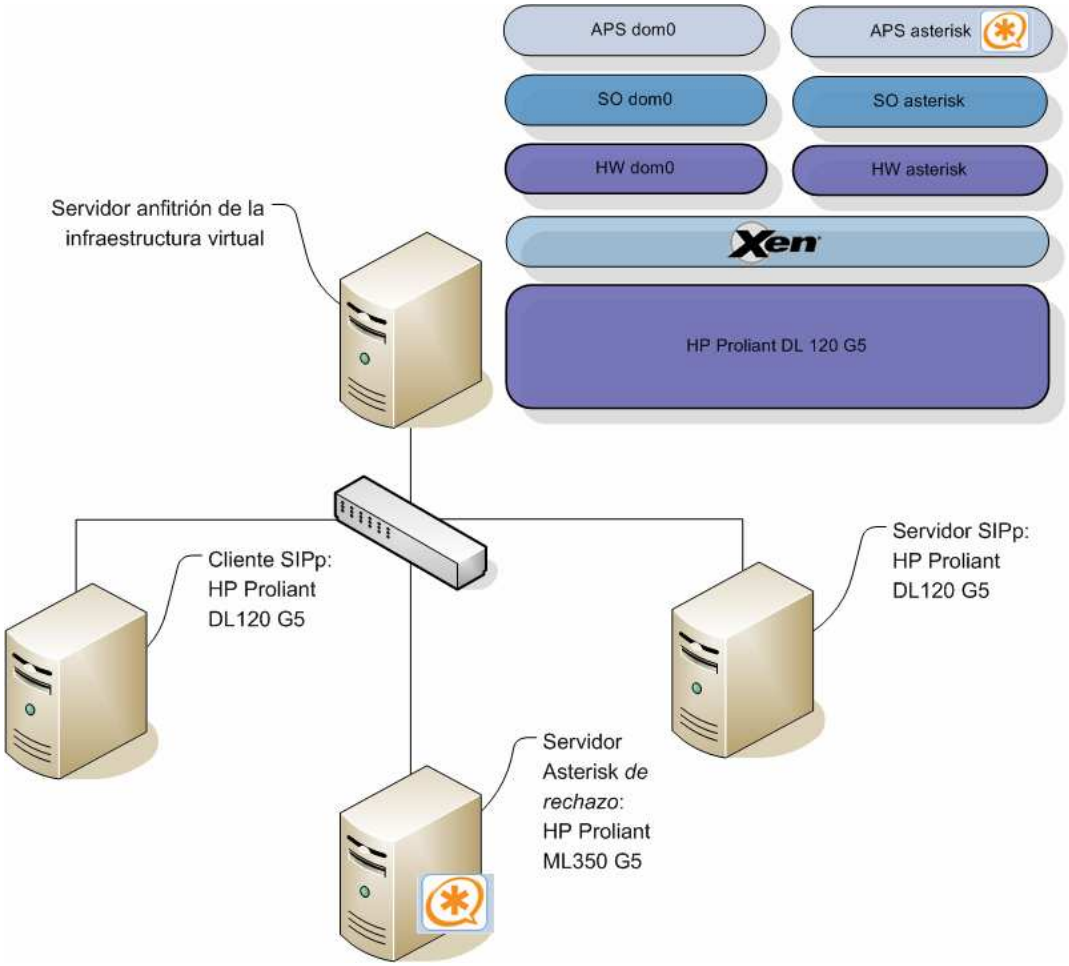


Figura 5.18. Arquitectura del Esquema 2: Servidor virtual.

Veamos en detalle la configuración de los cuatro servidores que participan en las pruebas para este segundo esquema:

- **Servidor *SIPp.*** Ejecuta el servidor *SIPp* para la recepción de las llamadas en la prueba. **HP Proliant DL120 G5** (Intel(R) Xeon(R) CPU E3110 @ 3.00GHz y 4Gb de memoria RAM).

- **Cliente *SIPp.*** También es un servidor **HP Proliant DL120 G5**; se encarga de la generación de las llamadas que componen la carga de trabajo en la prueba.

- **Servidor *Asterisk de apoyo*.** Como sabemos este servidor *Asterisk* no tiene configuración alguna y por tanto rechaza cualquier llamada recibida. Se trata en este caso de un **HP Proliant ML350 G5** (2Gb de memoria RAM y procesador Intel(R) Xeon(R) CPU E5320 @ 1.86GHz).

- **Servidor *Xen*.** Se trata de un servidor **HP Proliant DL120 G5** sobre el que hemos instalado y configurado Xen como plataforma de virtualización. Sobre él hemos creado un dominio sobre el que vamos a ejecutar el servicio *Asterisk*. El servidor dispone como recursos principales de cara a las pruebas de rendimiento de un procesador **Intel(R) Xeon(R) CPU E3110 @ 3.00GHz** y de **4Gb de memoria** RAM.

El dominio *Asterisk* tiene acceso a los dos núcleos del procesador para que pueda trabajar en igualdad de condiciones que el servidor real del esquema anterior (aunque ahora, como es lógico con la penalización de la virtualización). **En cuanto a memoria, con 2Gb de los 4 disponibles es más que suficiente**, ya que no supone un problema de cara a las pruebas al presentarse el consumo de memoria siempre bajo. A continuación podemos ver el fichero de configuración *.cfg* para el dominio:

```
kernel = "/boot/vmlinuz-2.6.26-2-xen-amd64"
ramdisk = "/boot/initrd.img-2.6.26-2-xen-amd64"

memory = 2048

name = "asterisk"

disk =
['file:/media/imagenes/asterisk_lenny/imagen,sda1,w','file:/m
edia/imagenes/asterisk_lenny/swap,sda2,w']

root = "/dev/sda1 ro"

dhcp = "no"

vif = ['mac=1A:1B:1C:2A:2B:2C']
netmask = '255.255.255.0'
gateway = '150.214.153.1'
ip = '150.214.153.233'
broadcast = '150.214.153.255'

on_poweroff = 'destroy'
on_reboot = 'restart'
on_crash = 'restart'

vcpus = 2

extra = 'console=hvc0 xencons=tty'
```

En él podemos observar la configuración de los recursos de procesamiento y memoria comentados. Además, entre otras opciones importantes configuradas, podemos ver que hace uso de un ***kernel Xen* para 64 bits** (dispone de Debian GNU/Linux Lenny como sistema operativo), que tiene dos ficheros de disco (*imagen* y *swap*) y que **la interfaz de red ha sido configurada de forma estática al tratarse de un servidor**, con la dirección de red 150.214.153.233.

También debemos tener en mente la **configuración realizada para el dominio administrativo *dom0*, más en concreto para el demonio *xend* mediante la modificación del fichero */etc/xen/xend-config.sxp***. Las siguientes cuatro líneas son las que se han configurado en este fichero:

```
(network-script network-bridge)
(vif-script vif-bridge)
(dom0-min-mem 196)
(dom0-cpus 0)
```

De esta forma se ha establecido que la configuración de red de la infraestructura virtual y la creación de las interfaces de red virtuales para todos los dominios sean en modo *bridge*, es decir, mediante la creación de un puente y como

si de conexiones de red directas al medio físico se tratara. **Los dos últimos parámetros son muy importantes ya que configuran el acceso a los recursos de memoria y procesador por parte del dominio *dom0***, que como sabemos es que el inicia, administra, monitoriza y destruye el resto de dominio. **Garantizamos un mínimo de memoria para *dom0* de 196Mb y permitimos su acceso a todos los núcleos del procesador**. Recordamos que Xen se encuentra instalado en un **HP Proliant DL120 G5** con procesador Intel(R) Xeon(R) CPU E3110 @ 3.00GHz y de 4Gb de memoria RAM.

Creamos el nuevo dominio con el comando de la interfaz administrativa *xm create* y ejecutamos *xm list* para ver claramente cómo queda configurado el acceso a los recursos de memoria y procesador en nuestra infraestructura virtual, sobre la que vamos a realizar las pruebas de rendimiento con y sin *trascoding*:

```
$ xm create asterisk.cfg
Using config file "/etc/xen/asterisk.cfg".
Started domain Asterisk

$ xm list
Name                ID       Mem     VCPUs      State   Time(s)
Domain-0             0       196         2     r-----      22.9
asterisk             5      2048         2     -b----       1.6
```

Además, si ejecutamos la herramienta de monitorización *xm top* podemos apreciar en tiempo real el consumo que realizan de estos recursos, incluyendo también otros como interfaces virtuales de red y dispositivos virtuales de bloque, como podemos ver en la figura 5.19.

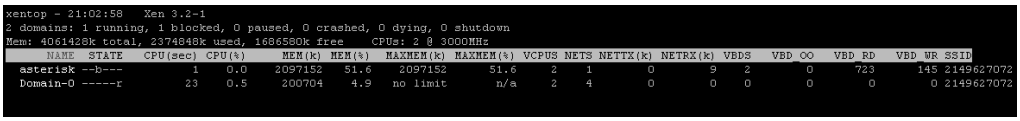


Figura 5.19. Monitorización de la infraestructura virtual del Esquema 2 con xm top.

6.5.2 Resultados con *trascoding*

Tras realizar diversas pruebas de rendimiento con diferentes configuraciones para la calibración de los parámetros que la conforman **elegimos una como válida en la que aunque existe saturación durante varias marcas de tiempo en el servidor virtual *Asterisk*, la totalidad de las llamadas es completada con éxito**. En la figura 5.20 podemos ver en detalle **la evolución del consumo de CPU durante la duración total de la prueba para el dominio *Asterisk* (*domU*) y el dominio *dom0*, además del porcentaje de CPU dedicado en cada momento a instrucciones para virtualización (*% steal*)**. Como vemos, **el máximo consumo de CPU por parte de instrucciones de virtualización coincide efectivamente con la saturación experimentada en el dominio *domU***. Notar que el consumo mostrado por cada dominio y que ha sido registrado por *sar* es *local*, es decir, ***sar* no sabe que se trata de dominios virtuales y los porcentajes ofrecidos son respecto al total del recurso asignado y disponible en cada momento para el dominio**.

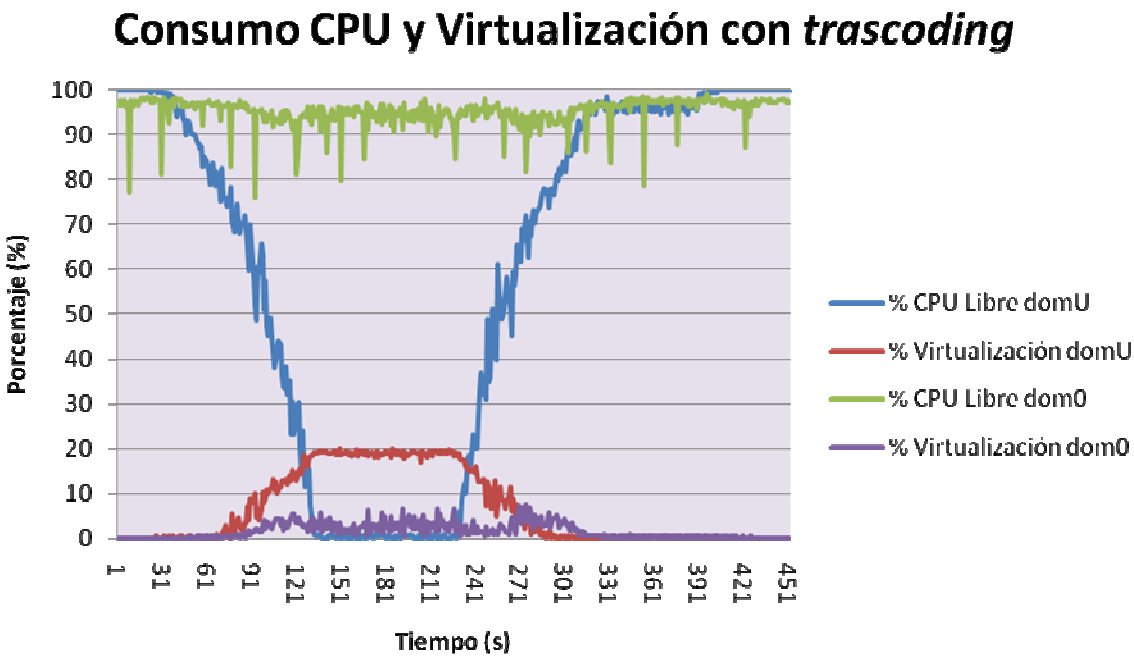


Figura 5.20. Consumo de CPU y Virtualización en la prueba con trascoding para el dominio Asterisk (domU) y el dominio administrativo dom0.

Estos instantes de saturación que aparecen en el servidor virtual *Asterisk* provocan que haya un reducido número de retransmisiones de mensajes *SIP* tanto en el cliente como en el servidor *SIPp*, como podemos observar en las figuras 5.21 y 5.22. **Aún así**, como ya hemos dicho anteriormente, **la prueba pudo completar las 275 llamadas totales con éxito**.

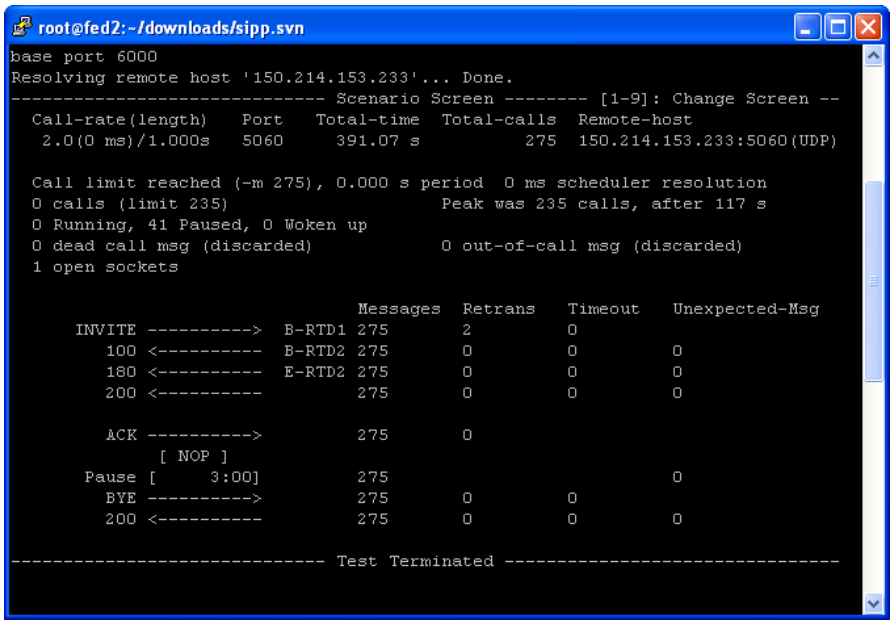


Figura 5.21. Estado final de SIPp en el cliente para el escenario con trascoding.

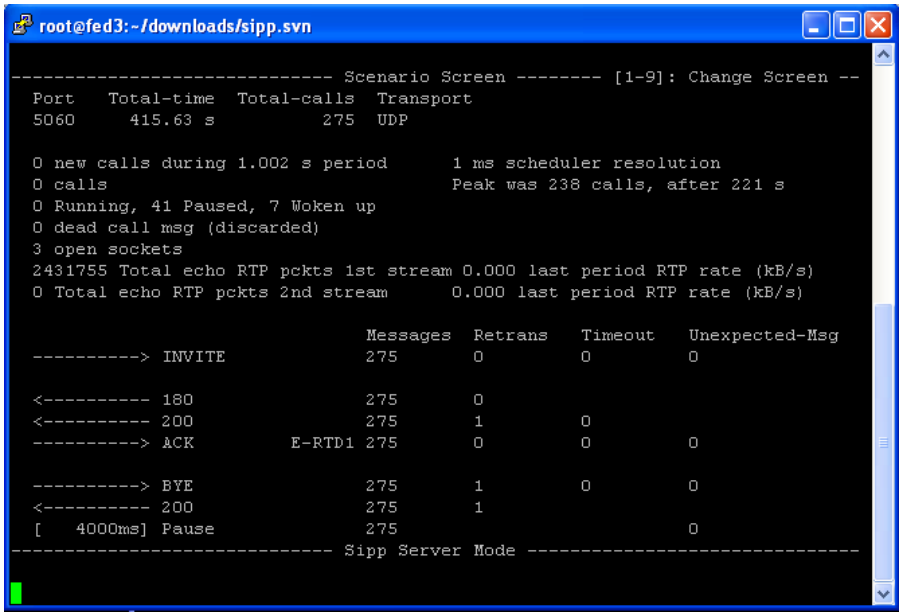


Figura 5.22. Estado final de SIPp en el servidor para el escenario con trascoding.

Como podemos ver en la figura *5.22. Estado final de SIPp en el servidor para el escenario con trascoding* **el pico de llamadas simultáneas alcanzado en el servidor *SIPp* fue de 238 llamadas a los 221 segundos del comienzo de la prueba**. En la figura 5.23 podemos ver una imagen tomada en el instante en el que se alcanzó en el servidor *SIPp* el máximo de llamadas simultáneas especificado en el cliente, siendo monitorizado el pico con la utilidad *xm top*.

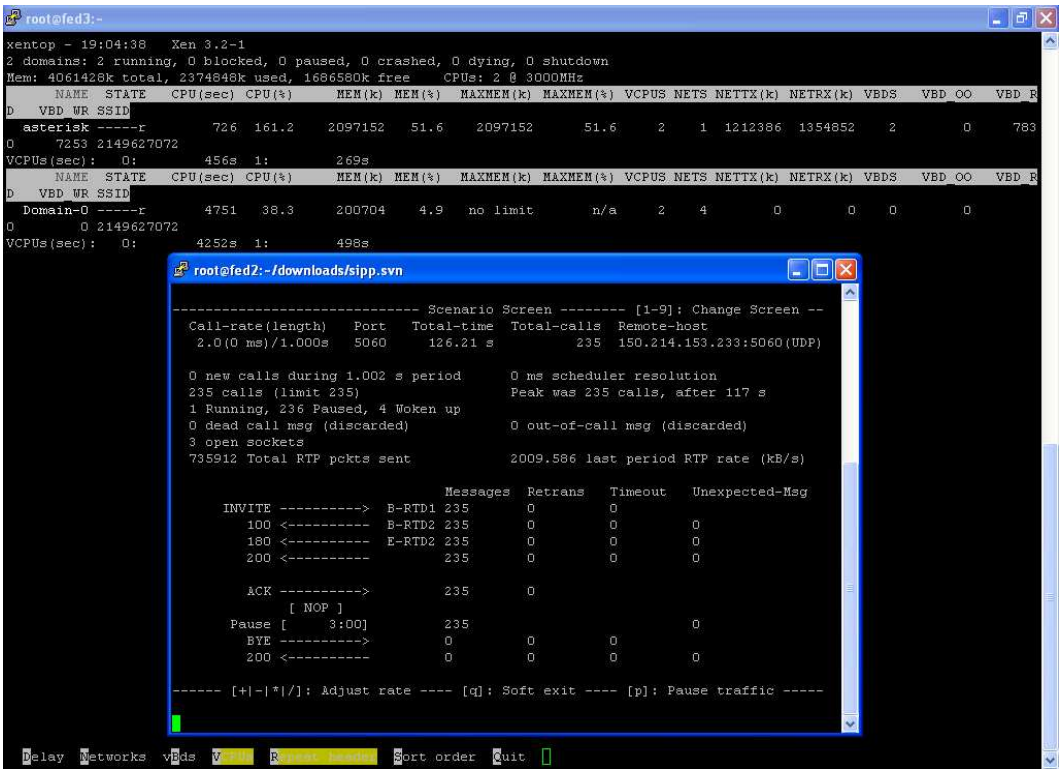


Figura 5.23. Pico de llamadas simultáneas alcanzado mientras la infraestructura virtual es monitorizada con xm top.

Por tanto podemos concluir que **el número máximo de llamadas simultáneas que soporta aproximadamente el servidor virtual *Asterisk* es de 238**. La prueba fue configurada en el cliente *SIPp* para mantener como máximo 235 llamadas simultáneas y crear 275 llamadas totales, con una frecuencia de 2 llamadas por segundo.

6.5.3 Resultados sin *trascoding*

A continuación mostramos los resultados obtenidos al realizar la prueba sin *trascoding* para el esquema 2 con servidor *Asterisk* virtual. Tras la realización de una serie de varias pruebas en la que fuimos **aumentando la carga de trabajo progresivamente llegamos al punto en el que el cliente *SIPp* prácticamente presenta saturación tras el establecimiento de 306 llamadas simultáneas.**

En la figura 5.24 podemos ver como **al mismo tiempo que el cliente *SIPp* prácticamente se encuentra saturado, el dominio *Asterisk* apenas presenta una carga como máximo del 20% de CPU**. Esto es debido, como dijimos en el esquema anterior, a que ahora el servidor *Asterisk* queda liberado de realizar *trascoding* para cada una de las llamadas que establece, por lo que puede soportar así la gestión de un mayor número de llamadas simultáneas. Esto no ocurre en el cliente *SIPp*, en el que al aumentar considerablemente el número de llamadas simultáneas admisible porque el servidor *Asterisk* lo permite llega a saturarse. Además, el gráfico es acompañado con los porcentajes de consumo de CPU por parte de instrucciones de virtualización en el servidor *Asterisk* (*domU*) y *dom0* así como el porcentaje de CPU libre para *dom0*.

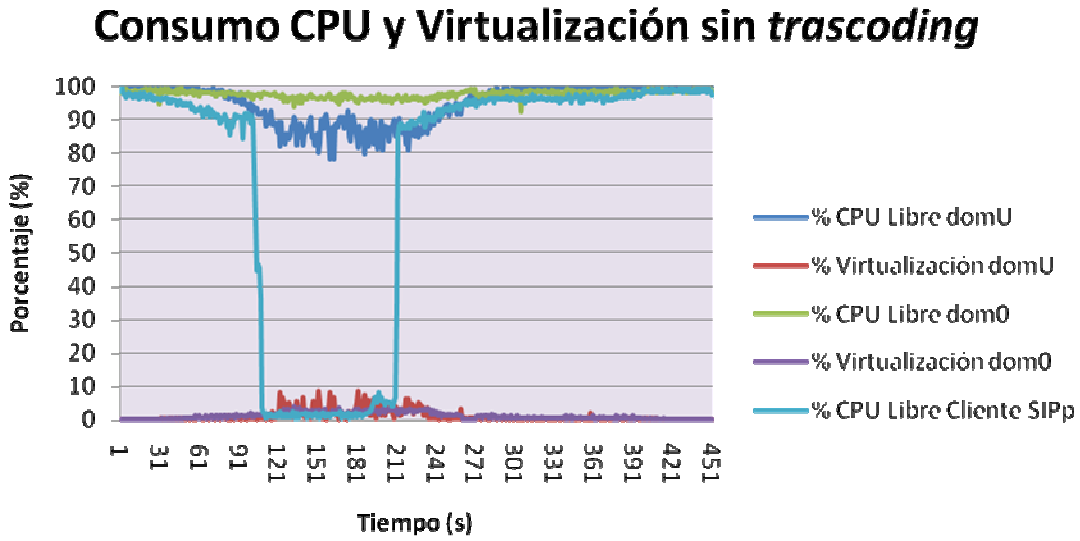


Figura 5.24. Consumo de CPU y Virtualización en la prueba sin trascoding para el dominio Asterisk (domU) y el dominio administrativo dom0.

La saturación no impidió que se completara un total de 375 llamadas correctamente, incluso sin aparecer mensajes *SIP* retransmitidos ni inesperados. El resultado de la ejecución de la prueba tanto en cliente como en servidor *SIPp* lo podemos apreciar en las figuras 5.25 y 5.26.

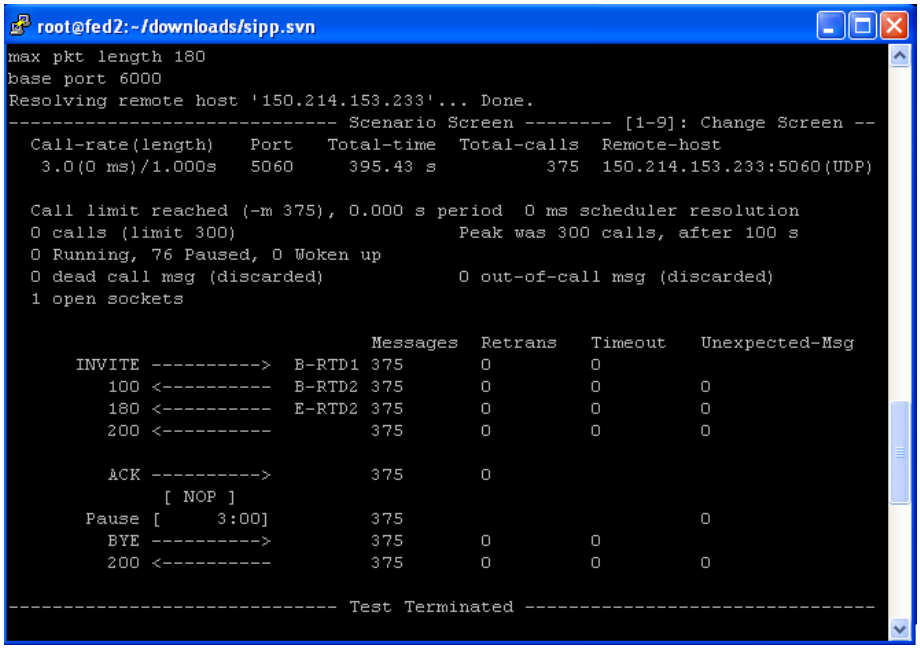


Figura 5.25. Estado final de SIPp en el cliente para el escenario sin trascoding.

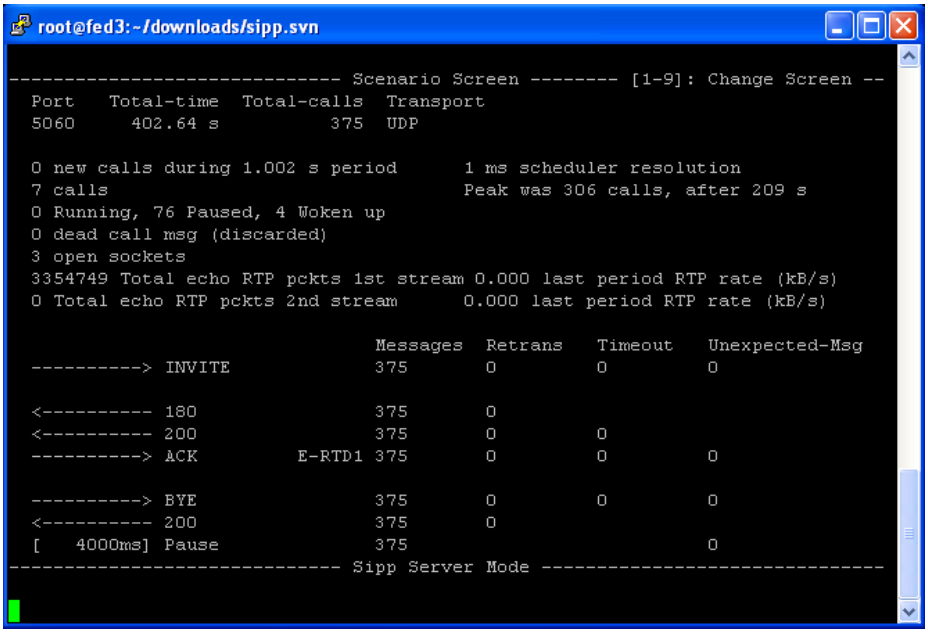


Figura 5.26. Estado final de SIPp en el servidor para el escenario sin trascoding.

Durante esta prueba se registró en el servidor *SIPp* un pico de un total de 306 llamadas simultáneas. En la figura 5.27 podemos ver el momento en el que se alcanza el total de llamadas simultáneas establecido en el cliente *SIPp* monitorizado mediante *xm top*.

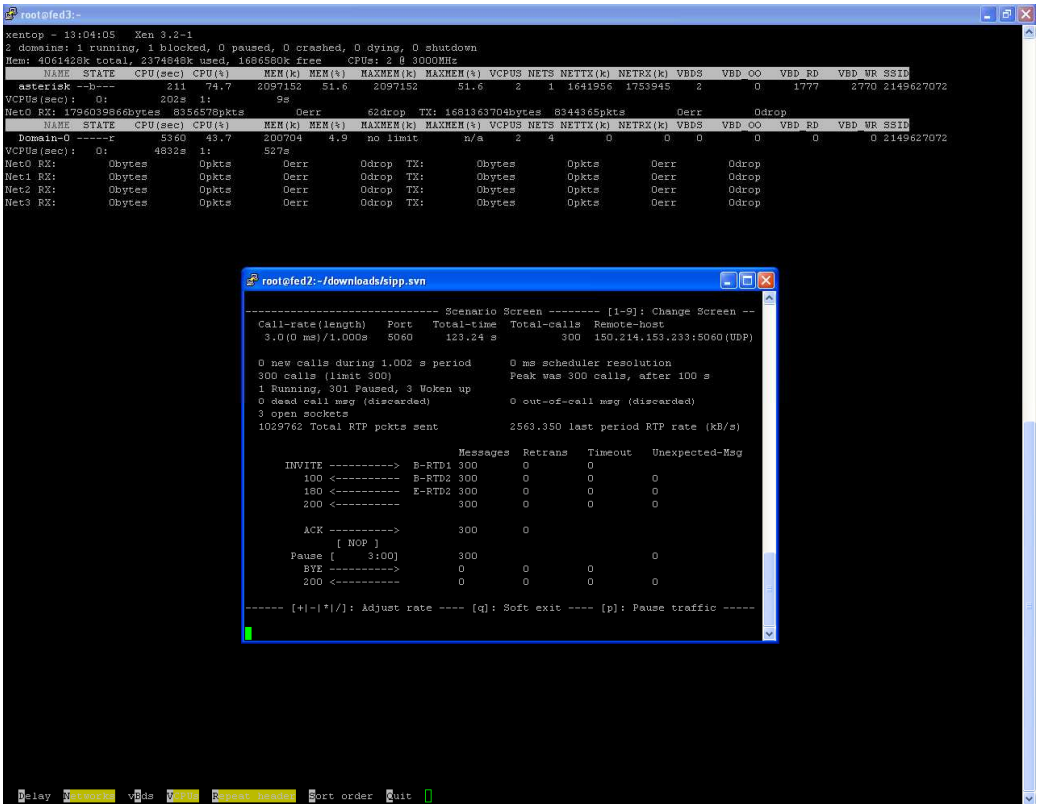


Figura 5.27. Pico de llamadas simultáneas alcanzado mientras la infraestructura virtual es monitorizada con xm top.

Como en el esquema anterior, **optamos a continuación por realizar una estimación del número de llamadas simultáneas que podría soportar el servidor virtual *Asterisk*.** En este caso no basta con tener en cuenta el porcentaje máximo de utilización en el servidor virtual *Asterisk* (20% de CPU) registrado con *SAR*, sino que al mediar virtualización debemos de tener en cuenta también la evolución del consumo en el dominio *dom0* (en la prueba anterior con *trascoding* esto no ocurre ya que el servidor virtual *Asterisk* llega a saturarse por completo y no es necesario realizar ninguna estimación para obtener los resultados), que es como máximo de 6-7% aproximadamente en algunos picos. Así, si con un consumo máximo de aproximadamente del 27% en el *host Xen* se alcanzó un total de 306 llamadas, tenemos que **el número de llamadas simultáneas que el servidor *Asterisk* virtual puede soportar sin realizar *trascoding* es en torno a las 1133llamadas de forma aproximada**.

6.6 ESQUEMA 3: SERVIDOR VIRTUAL ALOJADO EN UN SISTEMA DE FICHEROS EN RED

En este tercer y último esquema para las pruebas de rendimiento partimos del esquema anterior con virtualización del servidor *Asterisk*, con la diferencia que **en esta ocasión el alojamiento del dominio *Asterisk* pasa de ser local a encontrarse en un sistema de ficheros en red (NFS) en otro servidor.** Para realizar la configuración de este nuevo esquema debemos editar la configuración del dominio *Asterisk* e instalar y configurar correctamente el servicio NFS (Network File System) en otro servidor, como vamos a ver en el siguiente apartado.

6.6.1 Arquitectura

Como en el esquema anterior la arquitectura de este tercero **incluye cuatro servidores** (véase la figura 5.28)**: dos servidores para la ejecución del software de prueba *SIPp*** (*cliente y servidor*), **un servidor *Asterisk* adicional requerido por el escenario de prueba *SIPp* que rechaza todas las llamadas y en el que configuramos NFS para la compartición en red de dominios Xen**, **y el servidor anfitrión con Xen instalado como plataforma para la virtualización del servidor *Asterisk***.

La elección del servidor *Asterisk* adicional (necesario para la configuración del escenario de las pruebas y que éste sea lo más cercano a un escenario real) **como servidor NFS se debe a que es el que presenta menor carga de trabajo en las pruebas**. **Además el servidor HP Proliant ML350 G5 dispone de un sistema RAID 1 con discos duros SAS, cuya velocidad de transferencia es superior a la de los discos SATA del HP Proliant DL120 G5**. **Las características que presenta son inmejorables para el establecimiento de un sistema de ficheros en red que debe trabajar con ficheros de gran tamaño, como es el caso de las imágenes de disco del dominio *Asterisk***.

El poder compartir los dominios Xen en un sistema de ficheros en red implica que nuevas posibilidades se abran a la hora de administrar nuestra infraestructura virtual, ya que los ficheros de disco y configuración de éstos pueden ser utilizados por diferentes servidores Xen anfitriones. Ello, como veremos en el capítulo siguiente, nos permite dotar a nuestra infraestructura virtual de características fundamentales en la actualidad como alta disponibilidad, alto rendimiento… y sienta las bases para la migración de dominios entre diferentes servidores anfitriones.

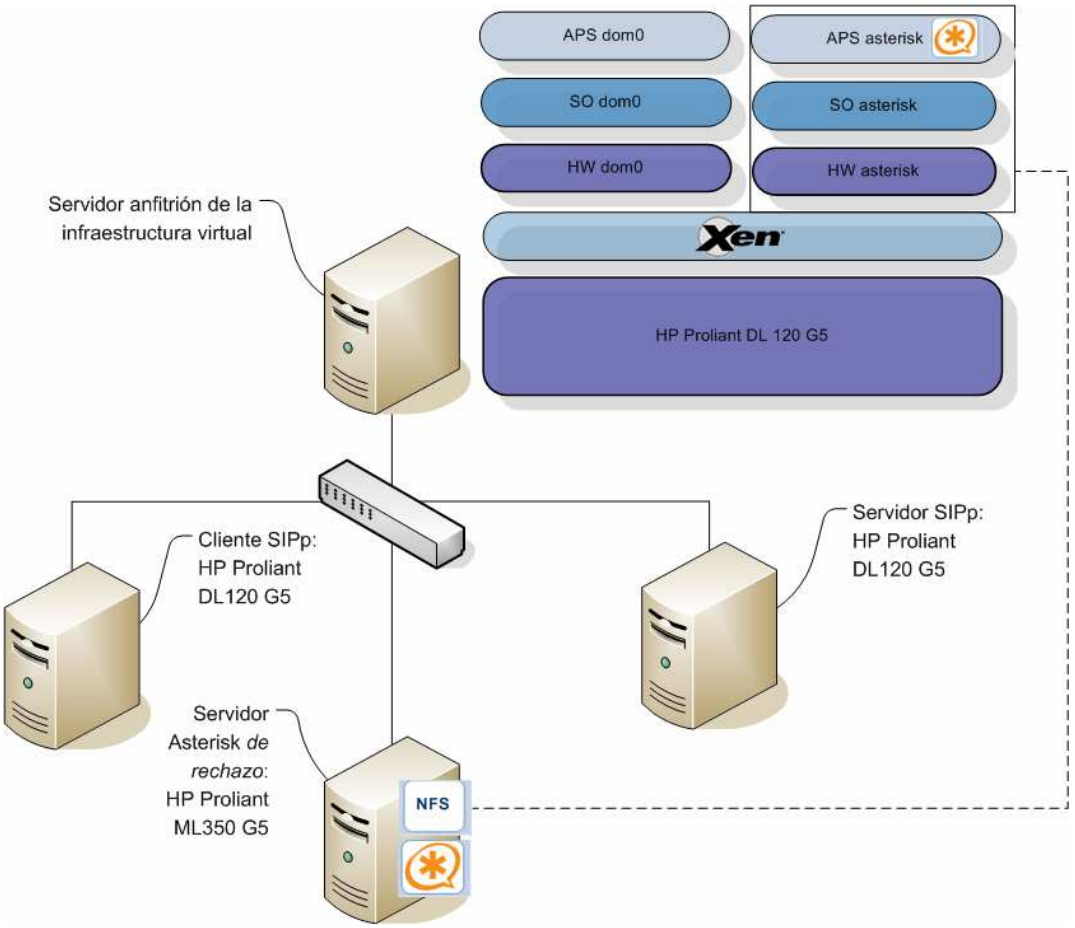


Figura 5.28. Arquitectura del Esquema 3: Servidor virtual alojado en un sistema de ficheros en red.

Veamos, como hicimos con los dos esquemas anteriores, los detalles de los cuatro servidores que intervienen en las pruebas de este esquema:

- **Servidor *SIPp.*** Ejecuta el servidor *SIPp* para la recepción de las llamadas en la prueba. **HP Proliant DL120 G5** (Intel(R) Xeon(R) CPU E3110 @ 3.00GHz y 4Gb de memoria RAM).

- **Cliente *SIPp.*** También es un servidor **HP Proliant DL120 G5**; se encarga de la generación de las llamadas que componen la carga de trabajo en la prueba.

- **Servidor *Asterisk de apoyo* y NFS.** Este servidor *Asterisk* no tiene configuración alguna y por tanto rechaza cualquier llamada recibida. Es un **HP Proliant ML350 G5** que dispone de 2Gb de memoria RAM, procesador Intel(R) Xeon(R) CPU E5320 @ 1.86GHz, y un sistema de almacenamiento en RAID 1 con discos duros SAS.

  Veamos cómo **configurar este equipo como servidor NFS**. En primer lugar ingresamos en el sistema **como usuario privilegiado *root*** ya que es necesario para realizar todas las operaciones:

```
$ su root
Contraseña:
```

A continuación **creamos un directorio, por ejemplo llamado *dominios*, el cual vamos a compartir en red**, y que es usado para contener los ficheros de los dominios Xen que queremos que se compartan:

```
$ mkdir dominios

$ ls
bin   dev       etc   lib    lost+found  mnt  proc  sbin
srv  tmp  var
boot  dominios  home  lib64  media       opt  root  selinux
sys  usr
```

Con el directorio creado **configuramos el fichero */etc/exports* para compartirlo**. Escribimos la ruta completa al directorio, la dirección de red del equipo (o si quisiéramos varios equipos varias direcciones de red) que tiene acceso permitido al directorio, en este caso el servidor con el *hipervisor Xen* instalado, y los permisos para éste acceso (*rw* para lectura y escritura):

```
/dominios     150.214.153.227(rw)
```

Finalmente, terminamos el acceso al directorio compartido y por tanto NFS extendiendo **cualquier privilegio para el usuario que accede de forma remota al directorio compartido en el sistema de ficheros**:

```
$ chmod 777 /dominios -R
```

Sólo queda **iniciar el servicio mediante el comando *service*** de la siguiente forma:

```
$ service nfs start
Inicio de los servicios NFS:                               [  OK  ]
Iniciando cuotas NFS:                                      [  OK  ]
Inicialización del demonio NFS:                            [  OK  ]
Inicialización de NFS mountd:                              [  OK  ]
```

A partir de este momento el directorio */dominios* puede ser montado por el servidor *Xen* con dirección de red 150.214.153.227 y acceder a él como si de un directorio en el sistema de ficheros local se tratara. Para esta prueba disponemos en él del directorio */dominios/asterisk* en el cual tenemos el fichero de configuración del dominio *Asterisk* (*asterisk.cfg*, que vemos en el siguiente punto) y los ficheros de disco *imagen y swap*.

- **Servidor *Xen*.** Como en el esquema anterior, se trata de un servidor **HP Proliant DL120 G5** (Intel(R) Xeon(R) CPU E3110 @ 3.00GHz y de 4Gb de memoria RAM) sobre el que hemos instalado y configurado Xen como plataforma de virtualización y sobre el que hemos creado e iniciado un dominio sobre el que vamos a ejecutar el servicio *Asterisk*.

  El dominio *Asterisk* sigue con acceso a los dos núcleos del procesador y con una memoria RAM de 2Gb. A continuación podemos ver el fichero de configuración *.cfg* para el dominio, igual al anterior, con la única diferencia de la localización de los ficheros de disco (*imagen* y *swap*), haciendo referencia al directorio compartido en el equipo 150.214.153.97 (servidor *Asterisk de apoyo*). El fichero de configuración se ubica en el directorio compartido como sabemos:

```
kernel = "/boot/vmlinuz-2.6.26-2-xen-amd64"
ramdisk = "/boot/initrd.img-2.6.26-2-xen-amd64"

memory = 2048

name = "asterisk"

disk = ['file:
150.214.153.97:/dominios/asterisk/imagen,sda1,w','file:
150.214.153.97:/dominios/asterisk/swap,sda2,w']

root = "/dev/sda1 ro"

dhcp = "no"
vif = ['mac=1A:1B:1C:2A:2B:2C']
netmask = '255.255.255.0'
gateway = '150.214.153.1'
ip = '150.214.153.233'
broadcast = '150.214.153.255'

on_poweroff = 'destroy'
on_reboot = 'restart'
on_crash = 'restart'

vcpus = 2

extra = 'console=hvc0 xencons=tty'
```

Antes de proceder a iniciar el dominio *Asterisk* debemos **comprobar que el acceso al directorio compartido se realiza de forma correcta**. Para conseguirlo ejecutamos el comando ***showmount***, que nos permite para un equipo listar el contenido del fichero */etc/exports*:

```
$ showmount -e 150.214.153.97
Export list for 150.214.153.97:
/dominios 150.214.153.227
```

El directorio se está compartiendo correctamente para el servidor Xen. Ahora **creamos un directorio en el que montar el directorio compartido**, y a través del cual accedemos al fichero de configuración del dominio:

```
$ mkdir /mnt/dominios_en_fed4
```

A continuación montamos el directorio compartido en el nuevo directorio y comprobamos su contenido:

```
$ mount 150.214.153.97:/dominios /mnt/dominios_en_fed4/


$ ls -la /mnt/dominios_en_fed4/asterisk/
total 2228240
drwxrwxrwx 2 root root       4096 mar  2 18:27 .
drwxrwxrwx 3 root root       4096 mar  2 18:13 ..
-rwxrwxrwx 1 root root        545 mar  2 18:19
asterisk_lenny.cfg
-rwxrwxrwx 1 root root 2147483648 mar  2 18:26 imagen
```

```
-rwxrwxrwx 1 root root  134217728 mar  2 18:27 swap
```

Una buena práctica antes de iniciar el nuevo dominio es confirmar que dispone de acceso a sus recursos sin problema alguno para lo cual podemos utilizar el comando *xm dry-run* sobre su fichero de configuración (véase la figura 5.29).

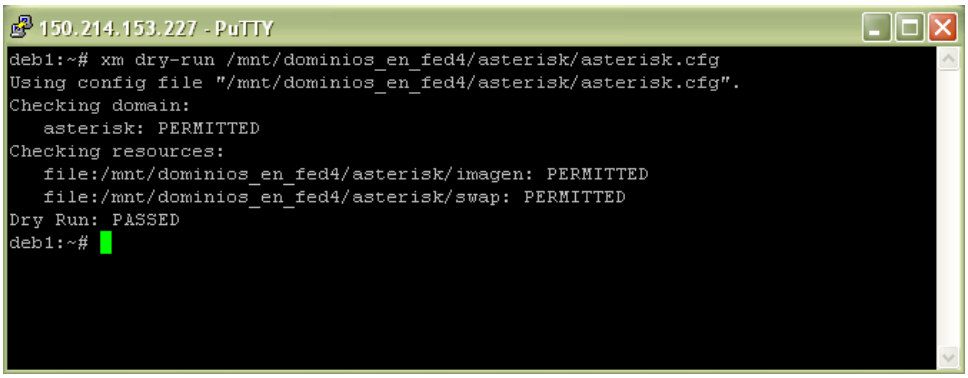


Figura 5.29. Comprobando que el dominio Asterisk accede correctamente a sus recursos mediante el domando xm dry-run.

Ya podemos crear el nuevo dominio con el comando de la interfaz administrativa *xm create* especificando la localización en el directorio compartido del nuevo fichero de configuración y ejecutamos *xm list* para así comprobar que se ha hecho correctamente y con los recursos asignados como esperamos:

```
$ xm create -f
/mnt/dominios_en_fed4/asterisk/asterisk_lenny.cfg
Using config file
"/mnt/dominios_en_fed4/asterisk/asterisk_lenny.cfg".
Started domain asterisk

$ xm list
Name                        ID   Mem VCPUs      State   Time(s)
Domain-0                     0   512     2     r-----   6619.9
asterisk                     6  2048     2     -b----      1.9
```

Como se puede ver en la salida anterior del comando *xm list* hemos aumentado además la memoria accesible como mínimo para el dominio *dom0*, aunque ello no influye de ninguna manera en la ejecución de las pruebas ya que apenas se consume un bajo porcentaje de memoria. En la figura 5.30 podemos ver **la salida de la utilidad de monitorización *xm top* tras iniciar este nuevo dominio *Asterisk***:

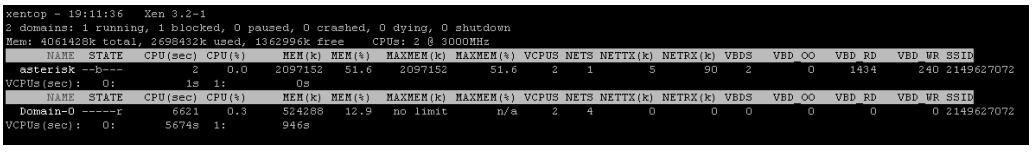


Figura 5.30. Monitorización de la infraestructura virtual del Esquema 3 con xm top.

6.6.2 Resultados con *trascoding*

Presentamos los resultados obtenidos para la prueba con *trascoding* para el tercer esquema de la misma forma como lo hemos hecho hasta ahora. De todas las pruebas realizadas

elegimos la mejor, en la que **aun existiendo cierta saturación en el servidor *Asterisk* la totalidad de las llamadas son completadas con éxito**. En la figura 5.31 podemos ver gráficamente la evolución del consumo de CPU y porcentaje de instrucciones de virtualización para el dominio *Asterisk (domU)* y el dominio administrativo *dom0*. Una vez más, **los mayores porcentajes de instrucciones de virtualización coinciden con la saturación del servidor**.

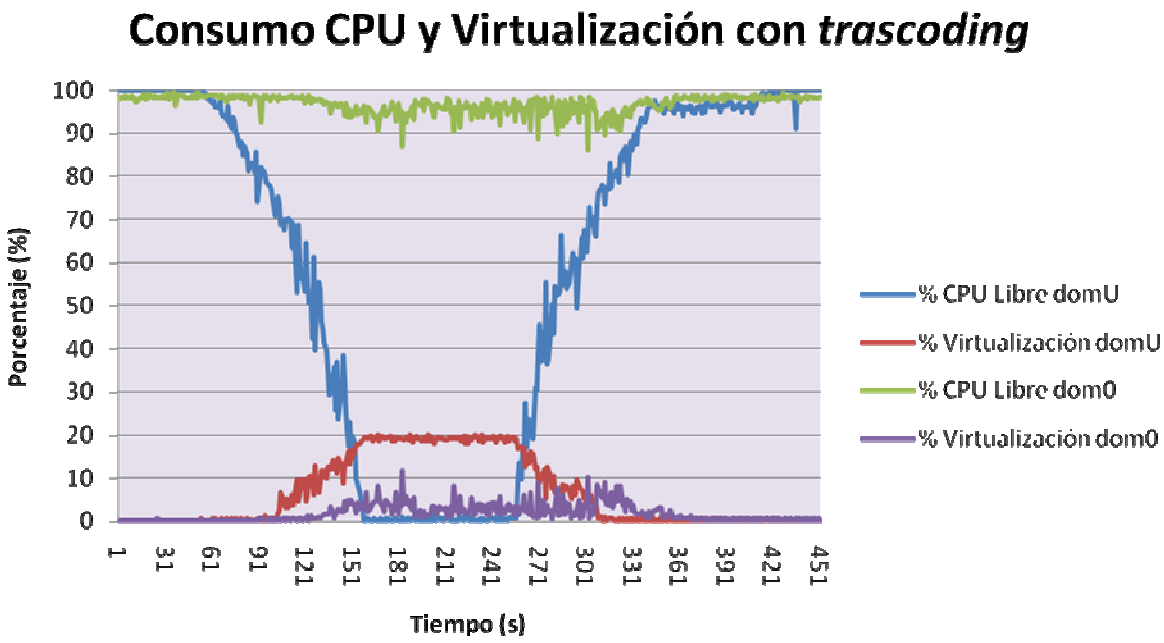


Figura 5.31. Consumo de CPU y Virtualización en la prueba con trascoding para el dominio Asterisk (domU) y el dominio administrativo dom0.

En esta ocasión el cliente *SIPp* fue configurado para realizar un total de 275 llamadas con un máximo de 235 llamadas simultáneas y una frecuencia de 2 llamadas generadas por segundo. En las figuras 5.32 y 5.33 podemos ver el estado final de ejecución del cliente *SIPp* y el servidor *SIPp*, que como vemos **apenas muestran unas pocas retransmisiones de mensajes *SIP* en el cliente *SIPp***.

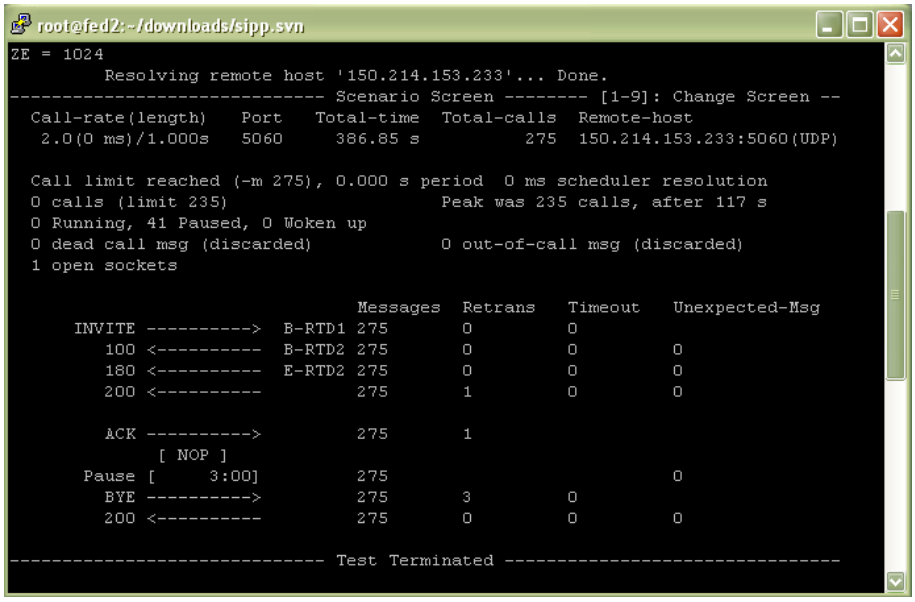


Figura 5.32. Estado final de SIPp en el cliente para el escenario con trascoding.

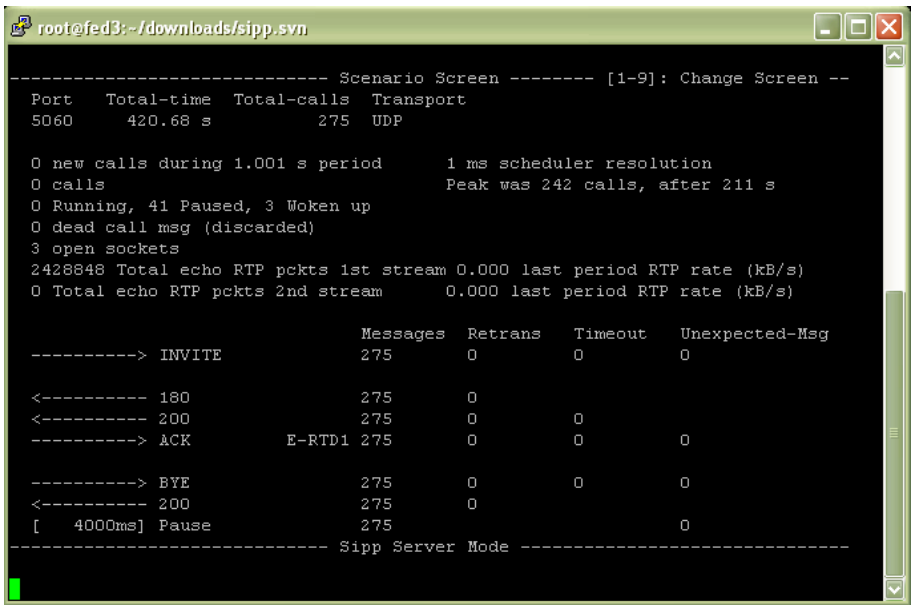


Figura 5.33. Estado final de SIPp en el servidor para el escenario con trascoding.

El pico de llamadas simultáneas registradas por el servidor *SIPp* fue de 242 llamadas a los 211 segundos transcurridos del inicio de la prueba de rendimiento. El momento en el que se alcanzó el total de llamadas simultáneas establecido en cliente fue monitorizado con la utilidad *xm top* como muestra la figura 5.34.

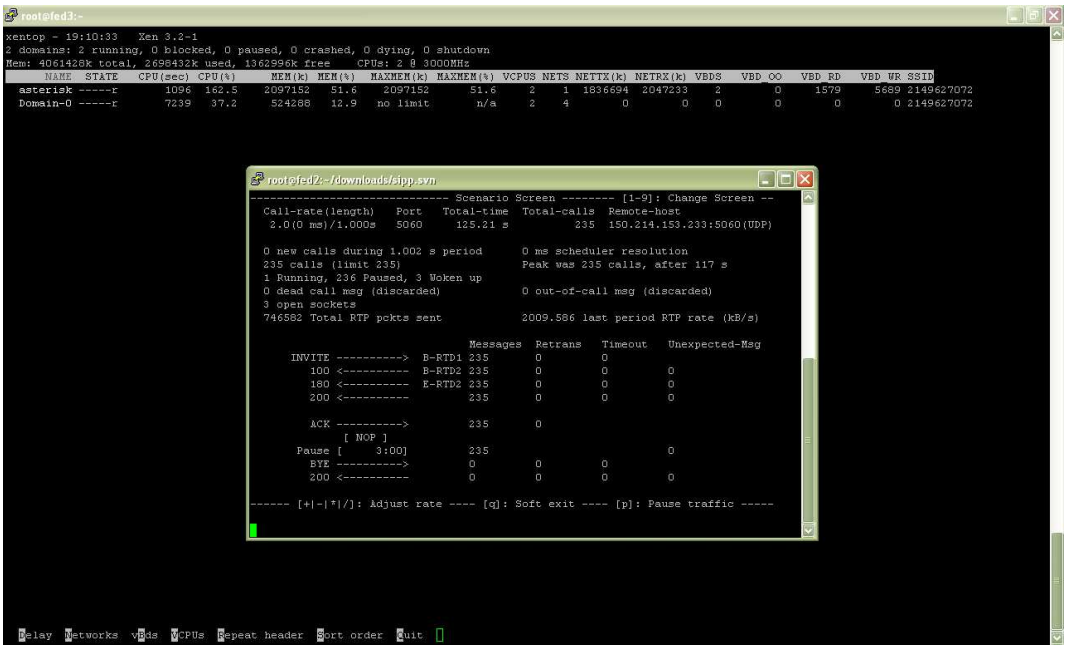


Figura 5.34. Pico de llamadas simultáneas alcanzado mientras la infraestructura virtual es monitorizada con xm top.

Finalmente añadir que el cliente *SIPp* se ha registrado una carga máxima del 80% de CPU mientras que en el servidor *SIPp* estuvo alrededor del 10%.

6.6.3 Resultados sin *trascoding*

Como en los esquemas anteriores volvemos a mostrar para contraste con la prueba anterior la prueba en la que se realiza *trascoding* de las llamadas por parte del servidor *Asterisk*. Como ocurriera en las anteriores pruebas de rendimiento sin *trascoding* para los dos primeros esquemas, **el cliente *SIPp* es saturado en procesamiento mucho antes de que pueda hacerlo el servidor *Asterisk*, que muestra como máximo aproximadamente un 20% de consumo de CPU; lo hace prácticamente tras el establecimiento de las 308 llamadas simultáneas registradas por el servidor *SIPp*, aunque las 375 llamadas totales se completan sin problemas**.

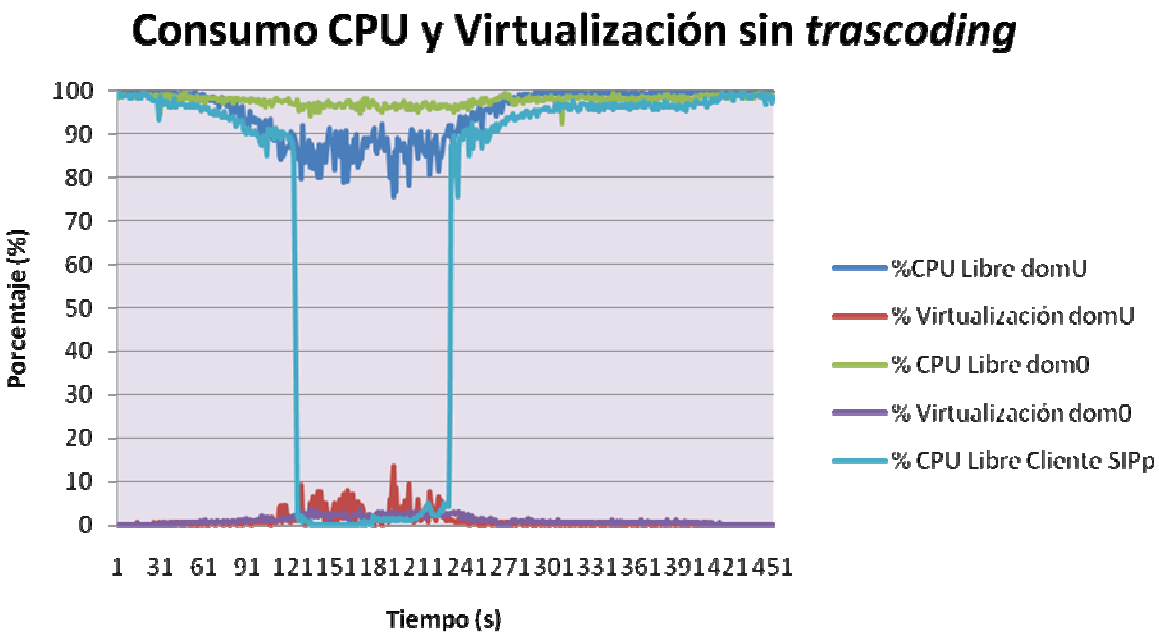


Figura 5.35. Consumo de CPU y Virtualización en la prueba sin trascoding para el dominio Asterisk (domU) y el dominio administrativo dom0.

En las figuras 5.36 y 5.37 podemos ver como la prueba se ha realizado sin que aparezca ninguna retransmisión ni ningún mensaje inesperado, obteniendo un pico de 308 llamadas simultáneas en el servidor *SIPp*.

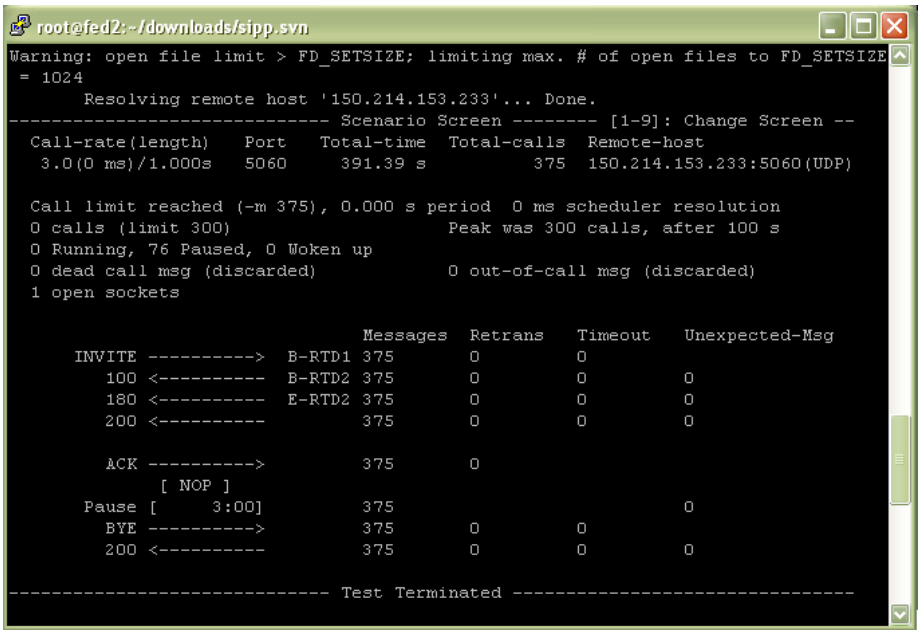


Figura 5.36. Estado final de SIPp en el cliente para el escenario sin trascoding.

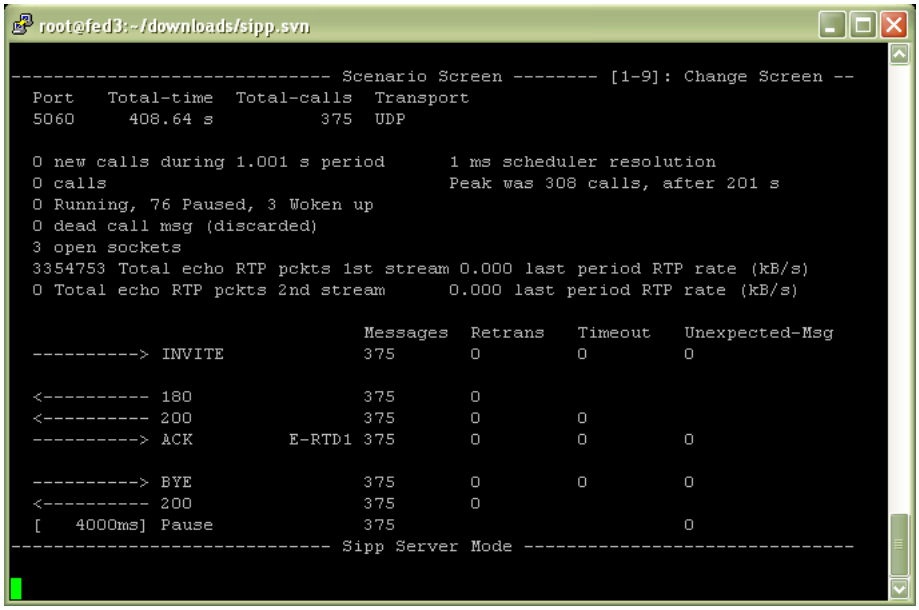


Figura 5.37. Estado final de SIPp en el servidor para el escenario sin trascoding.

Para saber aproximadamente el número de llamadas simultáneas que soportaría el servidor *Asterisk* ubicado en un sistema de ficheros en red **realizamos una estimación basándonos en la información obtenida en la prueba presentada: 308 llamadas con una carga de aproximadamente del 22% en el servidor *Asterisk* virtual y un 5-6% en el dominio *dom0*. Por lo tanto concluimos que el servidor soportaría alrededor de 1140 llamas simultáneas.**

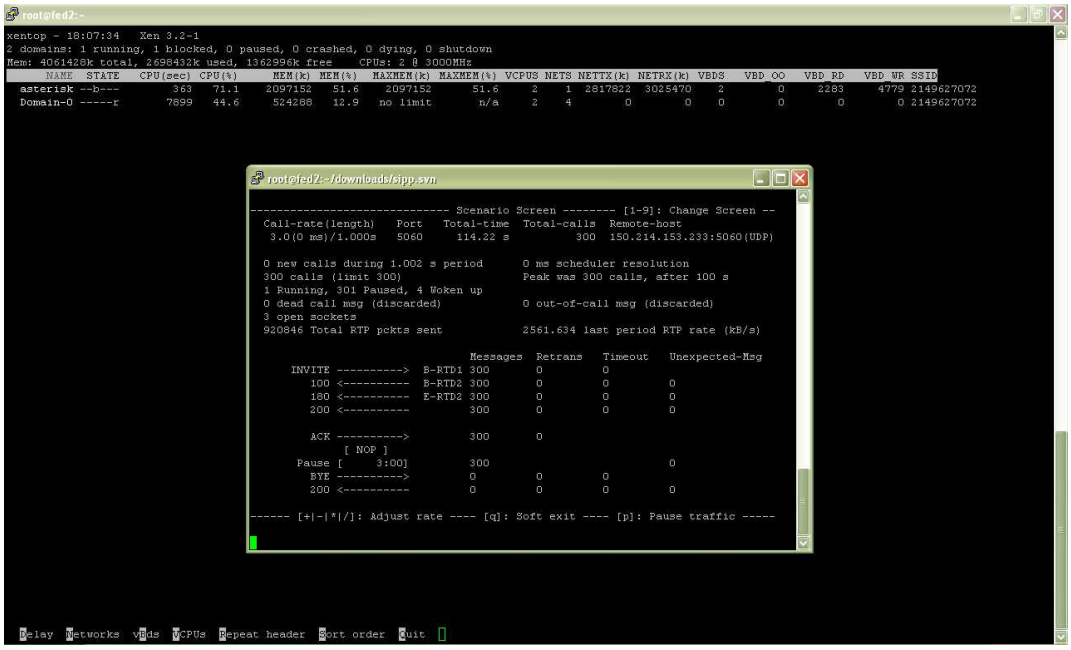


Figura 5.38. Pico de llamadas simultáneas alcanzado mientras la infraestructura virtual es monitorizada con xm top.

6.7 COMPARATIVA Y EXTRACCION DE CONCLUSIONES

En éste último apartado resumimos los resultados observados durante la serie de pruebas de rendimiento realizadas en los tres esquemas anteriores para el servidor *Asterisk*, ya sea real, virtual, o virtual alojado en un sistema de ficheros compartido en red. Como vamos a ver a continuación, **a pesar de la ligera disminución en el rendimiento que implica el introducir virtualización en un servidor, los resultados han sido excelentes y las ventajas que por otra parte ello supone son determinantes**, sobre todo desde el punto de vista administrativo de una infraestructura de servidores.

En la figura 5.39 podemos ver gráficamente los resultados obtenidos para el **número total de llamadas simultáneas que soporta el servidor *Asterisk* configurado en cada esquema para las pruebas realizadas con y sin *trascoding***. Los resultados en las pruebas con *trascoding* se obtuvieron tras saturar el servidor *Asterisk*, mientas que en las pruebas sin *trascoding* hubo que realizar una estimación pues el cliente *SIPp* se saturaba mucho antes de que pudiera hacerlo el servidor *Asterisk*.

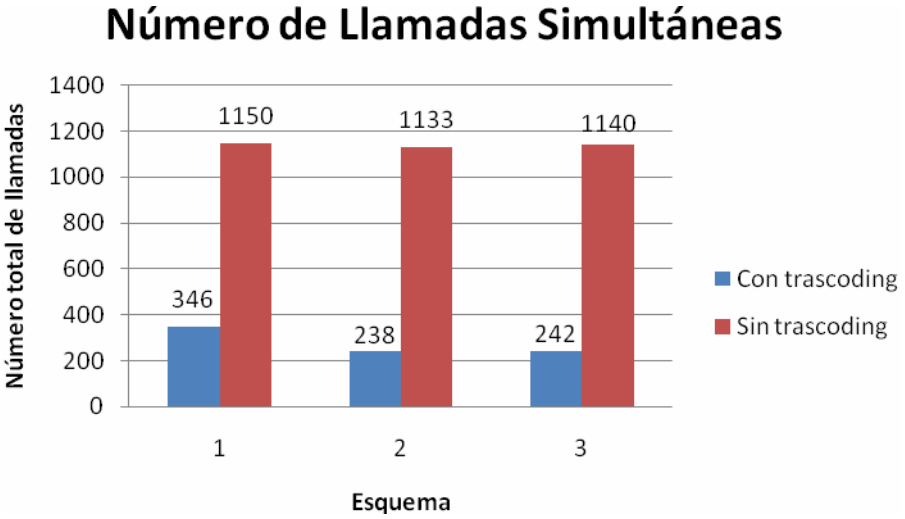


Figura 5.39. Comparativa de número de llamadas simultáneas máximo para cada esquema con y sin trascoding.

Es sencillo ver cómo **la diferencia entre el número de llamadas simultáneas máximo obtenido con y sin *trascoding* es bastante grande para los tres esquemas, debido como sabemos a la carga adicional que introduce el propio proceso de traducción entre los diferentes *códecs* que usan cliente y servidor *SIPp***.

Servidor real Vs Servidor virtual

Es completamente normal la **pérdida de rendimiento observada en los resultados al pasar de un servidor *Asterisk* real a uno virtual**. El servicio ya no es ejecutado directamente sobre el sistema operativo, sino que hemos introducido una capa intermedia de virtualización Xen que interviene en las instrucciones generadas por el servicio y le permite el acceso a los recursos hardware del sistema anfitrión. En este caso, al consumo de CPU y memoria generado por el dominio *Asterisk* se suma el originado en el dominio administrativo *dom0*, que lo monitoriza, y el porcentaje de instrucciones procesadas por el *hipervisor Xen* y que aparecen en las estadísticas ofrecidas por *SAR* como *% steal*: porcentaje de tiempo consumido en esperas

involuntarias en la CPU virtual o CPU mientas que el *hipervisor* sirve a otros procesadores virtuales.

Servidor virtual local Vs Servidor virtual remoto

Aunque las diferencias observadas entre los resultados obtenidos con una y otra configuración no son muy grandes, sí ha llamado la atención que éstos sean **ligeramente mejores para el caso en el que el servidor virtual está disponible de forma remota a través de almacenamiento compartido**. Esto es normal sin analizamos con detenimiento los recursos disponibles en nuestra configuración en cuanto a red y almacenamiento:

- **Los recursos de red no suponen ninguna limitación** para la realización de las pruebas, son más que suficientes. Los diferentes equipos se encuentran interconectados a través un *switch Gigabit* y disponen de interfaces de red Gigabit Ethernet 1GB/s 64 bits.

- **El almacenamiento compartido se encuentra localizado en un disco duro SAS** (HP Proliant ML350 G5), frente al disco duro SATA local del que dispone el servidor Xen anfitrión (HP Proliant DL120 G5). Los discos duros SAS disponen de tiempos de acceso muy rápidos, lecturas o escrituras aleatorias, y por tanto velocidades de transferencia considerablemente superiores.

Ambas cuestiones han permitido que aún encontrándose el dominio *Asterisk* virtual localizado en otro equipo, el número de llamadas soportadas se encuentre ligeramente por encima que el obtenido para el dominio *Asterisk* virtual local, aunque todavía menor que el experimentado con un servidor *Asterisk* real debido al *overhead* introducido por la virtualización. En el capítulo siguiente podemos ver algunas técnicas que nos permiten mejorar el rendimiento obtenido al aplicar virtualización; por ejemplo dotando a los servicios ejecutados en los dominios de alta disponibilidad o permitiendo la *migración en caliente* de dominios entre diferentes *hosts Xen* anfitriones.

CAPITULO 6

ALTA DISPONIBILIDAD EN CLUSTERES VIRTUALES

En este último capítulo del proyecto vamos a profundizar en el concepto de *clúster virtual* y vamos a ver como algunas técnicas aplicables en los casos reales pueden ser implementadas también en este tipo de infraestructuras de igual forma. Más concretamente dotamos a nuestra infraestructura virtual de una de las características y requisitos imprescindibles hoy en día a la hora de ofrecer servicios: alta disponibilidad. Cada día que pasa resulta habitual encontrarnos con servicios que requerimos o que debemos proporcionar con una gran exigencia: rapidez, eficiencia, simplicidad, las 24 horas al día, los 7 días de la semana, todo el año. Es entonces cuando consideramos que ese determinado servicio (base de datos, página web, centralita VoIP, almacenamiento) debe estar continuamente disponible, lo que implica que debamos aplicar determinadas técnicas tanto *hardware* como *software* para cumplir este objetivo.

Como podemos imaginar existe una gran variedad de soluciones de ambos tipos. Sin embargo, una de ellas destaca sobre el resto debido al gran potencial que presenta al tratarse de *software libre*. Hablamos de *Heartbeat*, que será introducido en la primera parte de este capítulo. Instalando y configurando *Heartbeat* en nuestro *clúster virtual* nos va a permitir que si el servicio *Asterisk* que se está ejecutando en un dominio cae, se arranque de forma automática en otro dominio virtual del *clúster*. Como hemos visto en el contenido desarrollado hasta el momento mediante virtualización también es posible proporcionar alta disponibilidad para nuestros datos y servicios, haciendo uso de mecanismos como la automatización de procesos de aprovisionamiento, copias de seguridad, *backups* de dominios… o, quizás el más importante de todos, la migración de dominios tanto *en parada* como *en caliente*. Esta última modalidad permite que un dominio sea *migrado* desde un servidor Xen anfitrión a otro sin que su estado sea bloqueado. En este capítulo profundizamos en todos estos conceptos relacionados con alta disponibilidad y virtualización al mismo tiempo que presentamos ejemplos prácticos que demuestran su gran utilidad.

1 Introducción a la Alta Disponibilidad

La **alta disponibilidad**, ya sea **de servicios o de datos**, puede ser alcanzada tanto haciendo uso de soluciones *software* como de soluciones *hardware*. La variedad que existe en ambos grupos es muy grande, pudiendo elegir siempre una solución que encaje perfectamente según nuestras necesidades. Veamos brevemente de forma introductoria en qué consiste cada una de ellas.

Las **soluciones *software*** nos permiten alcanzar alta disponibilidad instalando y configurando determinadas herramientas y aplicaciones que han sido diseñadas para tal efecto.

En lo que respecta a **alta disponibilidad en servicios** son usadas sobre en todo en servicios de ejecución crítica, como por ejemplo un servidor de bases de datos, una web de una tienda *online*, una centralita de telefonía IP, o el Directorio Activo. Lo normal es que con el *software* de alta disponibilidad no sea suficiente y haya que hacer uso de herramientas o complementos adicionales para su configuración, tales como dirección IP, reglas de cortafuegos, dependencias, políticas de seguridad, copia y recuperación, etc. Entre las herramientas más importantes dentro de esta categoría en sistemas operativos GNU/Linux podemos citar *Heartbeat, HA-OSCAR, KeepAlived, Red Hat Cluster Suite, The High Availability Linux Project, LifeKeeper* o *Veritas Cluster Server.*

De una gran importancia son también las **soluciones *software* para alta disponibilidad de datos**, casi siempre requeridos por los servicios críticos de nuestro servidor. Fundamentalmente consiste en la replicación de los datos, disponiendo de ellos en diversas localizaciones, permitiendo su efectiva recuperación en caso de fallos desastrosos o cuando el nodo en el que normalmente se encuentran accesibles no se está disponible. Lo habitual es que las herramientas que ofrecen alta disponibilidad/replicación de datos trabajen de dos formas diferentes: **replicación de bloque de datos y de bases de datos**. En el primer tipo la información en los sistemas de ficheros es replicada *bloque a bloque*; algunos ejemplos son *DRBD* (ya analizado en mayor detalle en capítulos anteriores) o *rsync*. En cuanto a la replicación de bases de datos la forma de trabajar consiste en la replicación de una base de datos en varias bases de datos, localizadas remotamente, llevando a cabo la replicación *instrucción a instrucción*. Algunas utilidades que siguen este modelo de replicación en GNU/Linux son *MySQL Replication*, *Slony-I*, o las propias de *Oracle*. Otros importantes proyectos de alta disponibilidad para datos son *FreeNAS, NAS Lite-2,* u *Openfiler*. Con los dos primeros por ejemplo usamos un sistema operativo actuando como *NAS (Network-Attached Storage)*, logrando que un equipo pueda desempeñar funcionalidades de servidor *NAS* soportando una amplia gama de protocolos para compartición de almacenamiento en red y replicación: *cifs* (*samba*), *FTP, NFS, rsync, RAID software*… y otros.

También mediante *hardware* es posible alcanzar alta disponibilidad tanto para datos como para servicios. Este tipo de soluciones depende en gran medida del tipo de datos o servicios a los que queremos dotar de esta característica, y normalmente suelen usarse para apoyar las soluciones de tipo *software* que estemos aplicando, para mantener o superar el nivel de seguridad y disponibilidad de las mismas. En la mayoría de los casos entendemos como solución *hardware* para alta disponibilidad la replicación de recursos *hardware*, como memoria o almacenamiento, o la puesta en marcha de dispositivos *hardware* especializados. Por ejemplo, para la **alta disponibilidad de servicios** de telefonía IP podemos instalar un balanceador de primarios de telefonía *E1/T1* (por ejemplo, *Redfone*), permitiendo que aunque cayera uno de los nodos del *clúster* de telefonía IP sea posible establecer llamadas a través de la red de telefonía tradicional. En cuanto a **alta disponibilidad de datos**, soluciones como *NAS, RAID, SAN,*… son ampliamente conocidas y usadas.

Como podemos ver, dependiendo del proyecto en el que trabajamos disponemos de una gran variedad de soluciones tanto *software* como *hardware* que además es posible usar de forma

conjunta integrándolas. Las hay propietarias, *libres* o incluso gratuitas. En el siguiente apartado la que es en la actualidad la solución *open source* sobre GNU/Linux más usada en los que a alta disponibilidad de servicios críticos se refiere (como es nuestro caso, el de las centralitas de telefonía IP con *Asterisk*). *Heartbeat* no sólo es *open source* sino que además puede ser descargada y usada de forma gratuita, obteniendo un gran soporte por parte de su activa comunidad de desarrolladores.

2 Alta disponibilidad de servicios críticos con Heartbeat

Heartbeat (http://www.linux-ha.org/wiki/Heartbeat) es una solución ***software* de alta disponibilidad para servicios críticos** que permite que éstos se encuentren disponibles en ejecución de forma ininterrumpida a pesar de los fallos o problemas que puedan surgir durante su funcionamiento. Mediante su uso es posible obtener alta disponibilidad de servicios como hemos dicho, además de **mejorar sustancialmente su rendimiento, todo ello con un coste de implementación y administrativo reducido**. *Heartbeat* trabaja sobre un *clúster* compuesto por nodos, los cuales son los encargados de ejecutar el servicio dependiendo de la configuración realizada en *Heartbeat* y de las circunstancias, que normalmente suelen determinar la disponibilidad de un nodo. De esta forma un servicio puede ser migrado de forma rápida y transparente de un nodo del *clúster* a otro por el motivo que fuese (por ejemplo ante un fallo en la alimentación del nodo o por motivos administrativos), logrando que éste solamente deje de estar operativo durante el tiempo en el que *Heartbeat* detecta el fallo y lleva a cabo la migración.

Por ejemplo, en el caso del proyecto en el que trabajamos podemos desarrollar un *clúster de alta disponibilidad de telefonía IP* formado por dos nodos, cada uno de ellos con *Asterisk* instalado. Sobre él es posible establecer una configuración en la que un nodo, llamado *activo*, posee y ejecuta el recurso *Asterisk* mientras que el otro nodo se mantiene a la espera de posibles fallos (**configuración Activo/Pasivo**). Datos adicionales, como la base de datos que contiene información sobre los usuarios, mensajes de voz, ficheros de registro… pueden ser almacenados en otro nodo a modo de almacenamiento compartido (véase la figura 6-1).

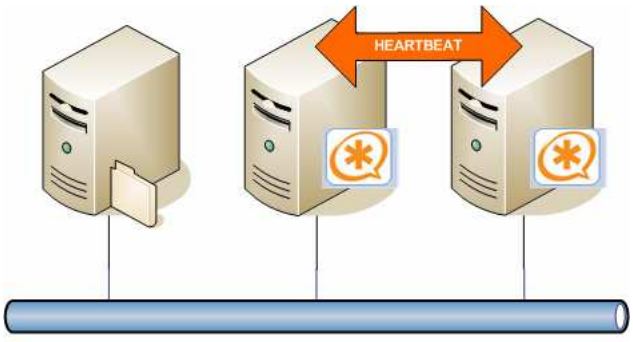


Figura 6.1. Clúster de alta disponibilidad de telefonía IP con Asterisk y Heartbeat.

Mediante *Heartbeat* los nodos que pueden ejecutar el servicio *Asterisk* se comunican mediante mensajes *ICMP* (*Internet Control Message Protocol*, por ejemplo *ping*) de forma que uno puede monitorizar el estado del otro (y viceversa). Además, cada nodo monitoriza también en qué estado se encuentran los recursos que actualmente se encuentra ejecutando, para la detección de posibles irregularidades y fallos. El nodo activo se encuentra por tanto monitorizando el estado de *Asterisk* y el nodo pasivo, mientras que el nodo pasivo monitoriza solamente al nodo activo.

Situémonos en el caso en el que un fallo es detectado en el nodo activo, ya sea un fallo *hardware* o un error en la ejecución de los recursos que está ejecutando (*Asterisk*) o cualquier otro *software* que sea imprescindible para prestar los servicios. Por ejemplo, imaginemos que el servicio *Asterisk* es saturado y por tanto no puede operar de forma correcta. En este momento el nodo activo (más concretamente *Heartbeat*) detecta el fallo en la ejecución de *Asterisk* y comprueba la disponibilidad del nodo pasivo para traspasar la ejecución del servicio. De esta forma el recurso es migrado del primer nodo al segundo, que pasa a ser consecuentemente el nuevo nodo activo. Ante situaciones de fallo similares el sistema de alta disponibilidad actúa de forma análoga, consiguiendo que **cuando se produzcan fallos en la ejecución del servicio y aunque éste deje de estar disponible durante unos instantes, el tiempo que transcurre hasta ejecutar el servicio de nuevo en otro nodo sea mínimo, normalmente inapreciable por los usuarios**. Por ejemplo, en el caso de que el nodo pasivo detecte que falla la monitorización del nodo activo (ya sea porque la fuente de alimentación no funciona de la forma adecuada y por tanto fue apagado o cualquier otro motivo), éste considera que el nodo activo no se encuentra en disposición de ejecutar *Asterisk* y se inicia el proceso para encargarse de ello, pasando de nuevo el nodo pasivo a ser el activo y logrando que el servicio no deje de estar disponible a los usuarios. **Independientemente de la recuperación ante desastres que permiten *Heartbeat*, los recursos también pueden ser migrados por *Heartbeat* de forma manual**, ya que ocasionalmente es necesario realizar actividades de administración y mantenimiento (actualizaciones, instalaciones, reparación y sustitución de componentes, etc.) de los nodos que ejecutan los servicios críticos, aunque no haya sido detectado ningún error.

Lo habitual es utilizar además del recurso principal (en nuestro caso *Asterisk*) una **dirección de red IP *virtual***. La dirección IP *virtual* permite que el servicio se encuentre siempre accesible a través de la misma dirección de red independientemente del nodo del *clúster* en el que se encuentre en ejecución; de lo contrario ello supondría un grave problema. Así, sea cual sea el nodo que ejecuta *Asterisk* éste requiere al mismo tiempo tener configurada la interfaz de red con la IP *virtual*; ésta es considerada también un recurso por *Heartbeat* en su configuración y es monitorizada y migrada de igual forma que *Asterisk*.

Como vemos hemos logrado recuperarnos ante un fallo en la ejecución de nuestros servicios de forma limpia y rápida; es el momento de **estudiar el porqué del error en la operativa del nodo activo y solucionar los problemas que han surgido**. Una vez que llevamos a cabo estas actividades **el nodo (anteriormente activo) pasa a encontrarse de nuevo disponible para prestar sus servicios al *clúster***. Ahora, dependiendo de la configuración que realicemos en *Heartbeat* **los servicios que ahora se encuentran en otro nodo pueden volver al nodo original o bien permanecer en el que actualmente es activo**. Cada una de las dos opciones posee ventajas y desventajas como vemos a continuación.

Si hacemos que los recursos vuelvan de nuevo a su ubicación original una vez que el nodo que causó el fallo se recupera debemos otra vez detener la ejecución de los servicios para su migración. Sin embargo, esto permite que la carga de trabajo del nodo que *tomó* los recursos tras el fallo se vea considerablemente reducida; esto es especialmente importante cuando éste nodo normalmente ejecuta otros servicios diferentes a los que fueron migrados.

2.1 ARQUITECTURA HEARTBEAT

Históricamente con el nombre *Heartbeat* hacíamos referencia a un conjunto de herramientas y utilidades de alta disponibilidad con una arquitectura estructurada en capas, formado fundamentalmente por el componente *Heartbeat* en la capa de mensajes o comunicación, un administrador de recursos locales (*Local Resource Manager* o *LRM*) y un administrador de recursos del *clúster (Cluster Resource Manager o CRM)* en la capa de asignación de recursos, y un agente de recursos (*Resource Agent*) en la capa de recursos, entre otros componentes.

A partir de la versión 2.1.4 esta generalización ya no es válida, quedando el nombre ***Hearbeat* para denominar exclusivamente la capa de comunicación o mensajes entre los nodos que forman el *clúster***. El resto de componentes son ahora proyectos independientes, igualmente necesarios para establecer un *clúster de alta disponibilidad*, como vemos en el apartado siguiente con la instalación de *Heartbeat 3*, referenciados en la actualidad grupalmente como *Linux-HA (Linux High Availability*, http://www.linux-ha.org*)*.

La arquitectura presentada en la actualidad por *Linux-HA* dispone de una gran flexibilidad debido a la modularidad comentada, algo muy positivo de cara al desarrollo que experimente el proyecto en los próximos años.

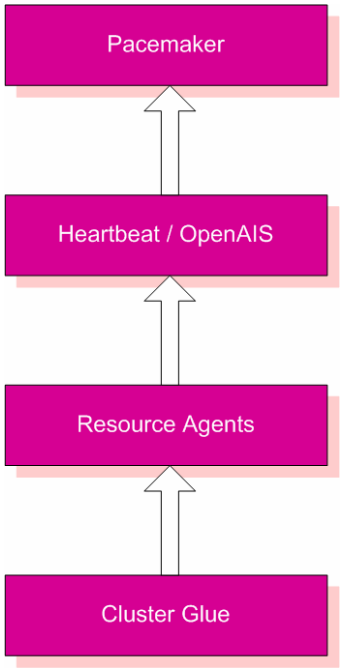


Figura 6.2. Arquitectura del proyecto Linux-HA.

En la figura anterior 6.2 podemos apreciar esta nueva arquitectura, cuyos elementos vemos a continuación:

- **El módulo *CRM* -gestor del los recursos del *clúster*- ha sido renombrado a *Pacemaker*** (http://clusterlabs.org/), encargado de arrancar los recursos o servicios, detenerlos, configurarlos, monitorizarlos, establecer la configuración global del *clúster*, además de otros muchos aspectos que presentaremos en el apartado *2.3 Configuración de Heartbeat 3*. Con él es posible configurar de forma directa (antes había que hacerlo editando ficheros *XML*) el *clúster*. *Pacemaker* funciona como parte central del proyecto *Linux-HA* por encima de la capa de comunicación, que puede hacer uso de *Heartbeat* u *OpenAIS*.

- ***Heartbeat*** es ahora solamente la capa que comunica los nodos del *clúster*, permitiendo el intercambio de mensajes para determinar la presencia o la ausencia de pares de procesos entre los nodos. Esta funcionalidad también puede ser soportada por otro proyecto denominado *OpenAIS*, como sabemos. Para alcanzar el mayor rendimiento posible *Heartbeat* debe ser usado en combinación con *Pacemaker*, encargado de llevar a cabo las tareas de arranque y parada de los servicios a los que queremos dotar de alta disponibilidad.

- ***Resource Agents*** son los agentes de recurso, un extenso conjunto de *scripts* para la gestión y acceso a los diferentes recursos y servicios que son ofrecidos y mantenidos por el *clúster*.

- **El componente *STONITH* ha sido renombrado como *ClusterGlue*** (http://www.linux-ha.org/wiki/Cluster_Glue), un elemento imprescindible en la arquitectura de alta disponibilidad de *Linux-HA* ya que permite la interoperabilidad entre sus diferentes elementos.

Una vez conocida cómo es la arquitectura del proyecto de alta disponibilidad *Linux-HA* y los diferentes elementos que la componen nos centramos por completo en la capa de mensajes/comunicación *Heartbeat* en su versión 3. Veremos por tanto a continuación cómo podemos instalar *Heartbeat 3*, y cómo podemos configurar nuestro *clúster* de alta disponibilidad con *Heartbeat* de una forma sencilla gracias a *Pacemaker*.

2.2 INSTALACION DE HEARTBEAT 3

A continuación vamos a ver cómo es el proceso de instalación de una infraestructura de alta disponibilidad *Linux-HA* con *Heartbeat 3* y el resto de componentes necesarios que han sido comentados en el apartado anterior a partir de su código fuente, disponible en la web del proyecto http://www.linux-ha.org/. **La instalación y configuración de cada uno de los componentes debe ser realizada en todos los nodos del *clúster*.**

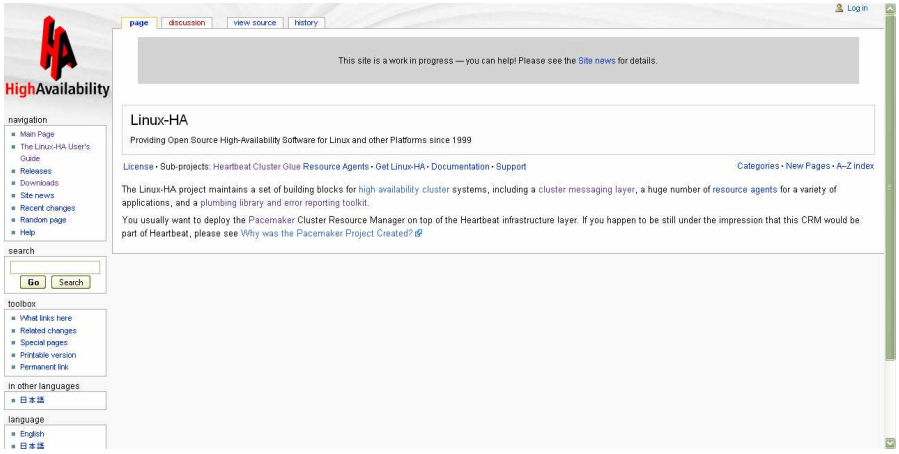


Figura 6.3. Web oficial del proyecto Linux-HA: http://www.linux-ha.org/.

Antes de comenzar con la instalación propiamente dicha debemos cumplir ciertas dependencias *software*, sobre todo librerías que son imprescindibles para que *Linux-HA* y sus componentes puedan operar de forma correcta. Lo hacemos instalándolas desde repositorios mediante el comando *apt-get install*, ya que usamos la distribución Debian 5 Lenny (para el resto de distribuciones Linux podemos proceder de forma análoga):

```
# apt-get install xsltproc docbook-xml docbook-xsl

# apt-get install build-essential libglib2.0-dev pkg-config
```

Una vez resultas estas dependencias descargamos e instalamos la última versión estable para cada uno de los componentes del proyecto *Linux-HA* por este orden: *ClusterGlue, ResourceAgents, Heartbeat, Pacemaker*.

2.2.1 Instalación de *ClusterGlue 1.0.5*

ClusterGlue es un elemento importante ya que gracias a él el resto de componentes de *Linux-HA* pueden trabajar de forma conjunta. Además de las dependencias instaladas anteriormente debemos instalar las siguientes para *ClusterGlue*:

```
# apt-get install autoconf libtool libglib2.0-dev flex bison
libxml2-dev libbz2-dev uuid-dev automake
```

Descargamos la última versión estable de *ClusterGlue* (en el momento de redacción de esta memoria es la 1.0.5) desde los repositorios *mercurial* de *Linux-HA* mediante el comando *wget*:

```
# wget http://hg.linux-ha.org/glue/archive/glue-1.0.5.tar.bz2
```

Con el paquete descargado, lo descomprimimos con el uso del comando *tar* e ingresamos en el directorio creado en la descompresión:

```
# tar xjvf glue-1.0.5.tar.bz2

# cd Reusable-Cluster-Components-glue-1.0.5/
```

Después, en el directorio *Reusable-Cluster-Components-glue-1.0.5* ejecutamos los *scripts* de preparación de la instalación *autogen.sh* y *configure*:

```
Reusable-Cluster-Components-glue-1.0.5# ./autogen.sh

Reusable-Cluster-Components-glue-1.0.5# ./configure
```

Finalmente y como es habitual, compilamos el código fuente y realizamos la instalación ejecutando *make* y *make install*, respectivamente:

```
Reusable-Cluster-Components-glue-1.0.5# make

Reusable-Cluster-Components-glue-1.0.5# make install
```

Con *ClusterGlue* instalado avanzamos un nivel en la arquitectura *Linux-HA* instalando *ResourceAgents.*

2.2.2 Instalación de *ResourceAgents 1.0.3*

Para la instalación de los agentes de recurso procedemos de la misma forma que para la instalación de *ClusterGlue*. En primer lugar descargamos el paquete con las fuentes de la versión 1.0.3 (última versión estable) y los descomprimimos:

```
# wget http://hg.linux-ha.org/agents/archive/agents-1.0.3.tar.bz2
```

```
# tar xjvf agents-1.0.3.tar.bz2
```

Después cambiamos al directorio obtenido en la descompresión para ejecutar los *scripts autogen.sh* y *configure*, y compilar y realizar la instalación mediante los comandos *make* y *make install*:

```
# cd Cluster-Resource-Agents-agents-1.0.3/

Cluster-Resource-Agents-agents-1.0.3# ./autogen.sh

Cluster-Resource-Agents-agents-1.0.3# ./configure

Cluster-Resource-Agents-agents-1.0.3# make

Cluster-Resource-Agents-agents-1.0.3# make install
```

A continuación, una vez que también hemos instalado ya *ResourceAgents*, ya podemos instalar *Heartbeat 3*. Además, vamos a realizar unos pequeños cambios en el sistema para que éste pueda trabajar correctamente.

2.2.3 Instalación de *Heartbeat 3*

Vamos a instalar la última versión estable de *Heartbeat* en el momento de redacción de este capítulo, 3.0.3. Antes debemos cumplir algunas dependencias de utilidades y librerías necesarias para configurar y llevar a cabo el proceso de instalación; lo hacemos utilizando el comando *apt-get install* como hasta ahora:

```
# apt-get install libbz2-dev perl-base libperl-dev libltdl3-dev
libxml2-dev libnet1-dev libglib2.0-dev gawk zlib1g zlib1g-dev
```

Cuando hemos instalado las dependencias podemos descargar el paquete *STABLE-3.0.3.tar.bz2* desde los repositorios *mercurial* oficiales de *Linux-HA* usando el comando *wget*:

```
# wget http://hg.linux-ha.org/heartbeat-STABLE_3_0/archive/STABLE-
3.0.3.tar.bz2
```

Descomprimimos entonces el paquete descargado usando *tar* e ingresamos en el directorio creado durante la descompresión, que contiene el código fuente de *Heartbeat 3*:

```
# tar xjvf STABLE-3.0.3.tar.bz2

# cd Heartbeat-3-0-STABLE-3.0.3/
```

En primer lugar y antes de nada, ejecutamos el comando *bootstrap* para prepararnos de cara al proceso de instalación:

```
Heartbeat-3-0-STABLE-3.0.3# ./bootstrap
```

Ahora sí podemos empezar a configurar y realizar la instalación. No lo vamos a hacer utilizando los comandos clásicos *configure*, *make* y *make install*, sino que ejecutamos como es recomendable el *script ConfigureMe* con las opciones *configure*, *make* y *make install*,

respectivamente. Además, a cada una de las ejecuciones añadimos la opción *–enable-fatal-warnings=no* para evitar que avisos impidan que culminemos adecuadamente la instalación:

```
Heartbeat-3-0-STABLE-3.0.3# ./ConfigureMe configure --enable-fatal-
warnings=no

Heartbeat-3-0-STABLE-3.0.3# ./ConfigureMe make --enable-fatal-
warnings=no

Heartbeat-3-0-STABLE-3.0.3# ./ConfigureMe install --enable-fatal-
warnings=no
```

En este punto ya disponemos de *Heartbeat 3* instalado, aunque todavía debemos llevar a cabo algunas acciones para adecuar el sistema para que *Heartbeat* trabaje. Creamos primero un grupo de usuarios denominado *haclient* y un usuario *hacluster* dentro de este grupo:

```
# addgroup haclient

# adduser hacluster –gid 1000
```

El usuario es añadido al grupo en el momento de su creación especificando el identificador de grupo *gid* que podemos obtener del fichero */etc/group*. Después, hacemos tanto a *hacluster* como a *haclient* propietarios de los ficheros de *Heartbeat* con el comando *chown*, que permite cambiar el propietario de un fichero/directorio:

```
# chown –R hacluster:haclient /var/run/heartbeat

# chown –R hacluster:haclient /var/lib/heartbeat

# chown –R hacluster:haclient /usr/lib/heartbeat

# chown –R hacluster:haclient /usr/var/run/heartbeat
```

Finalmente es necesario antes de poner en marcha el servicio *Heartbeat* copiar los siguientes ficheros desde el subdirectorio *doc* del directorio principal de las fuentes descargadas en el directorio */etc/ha.d/*. Posteriormente cuando configuramos *Heartbeat* comentamos en qué consiste cada uno de estos ficheros:

```
Heartbeat-3-0-STABLE-3.0.3# cp doc/authkeys /etc/ha.d/

Heartbeat-3-0-STABLE-3.0.3# cp doc/ha.cf /etc/ha.d/

Heartbeat-3-0-STABLE-3.0.3# cp doc/haresources /etc/ha.d/
```

Es necesario también cambiar los permisos de acceso al fichero *authkeys* (el primero de los copiados en el código anterior) para que solo los propietarios tengan acceso de lectura y escritura al mismo:

```
Heartbeat-3-0-STABLE-3.0.3# chmod 600 /etc/ha.d/authkeys
```

Hasta aquí la instalación de *Heartbeat*. El siguiente paso que debemos dar es su configuración, algo más complejo como podemos ver en el apartado siguiente de este capítulo. Nos preparamos para su configuración estableciendo en cada nodo del *clúster* un nombre único por el que identificarlo. Esto lo podemos hacer con el comando *hostname*:

```
# hostname <nombre>
```

Si queremos comprobar cuál es el nombre del nodo podemos ejecutar *hostname* sin opciones y el comando *uname* con la opción *–n*:

```
# uname -n
```

La salida de estos dos comandos es el nombre a través del cual haremos referencia a los nodos del *clúster* en la configuración de *Heartbeat*. Para terminar, debemos editar el fichero */etc/hosts* en cada nodo del *clúster* para que pueda comunicarse con el resto de nodos, incluyendo una línea por nodo de la siguiente forma:

```
<direccion_ip> <nombre>
```

De esta forma cada nodo identifica al resto mediante el par dirección IP – nombre, siendo el nombre el establecido con el comando *hostname*.

Comentar que aunque la instalación de *Heartbeat* sea realizada con éxito es posible que aparezcan algunos problemas en su ejecución. La mayoría de ellos se pueden subsanar copiando los ficheros *ha.cf* y *authkeys* del directorio */etc/ha.d/* al directorio */usr/etc/ha.d/*, y el fichero *shellfuncs* desde el directorio descomprimido de *ClusterGlue* al directorio */etc/ha.d/*.

```
cp /downloads/Cluster-Resource-Agents-agents-
1.0.2/heartbeat/shellfuncs /etc/ha.d/shellfuncs

cp /etc/ha.d/ha.cf /usr/etc/ha.d/

cp /etc/ha.d/authkeys /usr/etc/ha.d
```

2.2.4 Instalación de *Pacemaker 1.0.8*

Acabamos la instalación de nuestra infraestructura con *Pacemaker*, en su última versión estable 1.0.8. Instalar *Pacemaker* es fundamental ya que sin él configurar el *clúster* y los recursos resulta mucho más complejo. Además de las dependencias instaladas anteriormente de forma global instalación también la librería *libxslt-dev*:

```
# apt-get install libxslt-dev
```

Descargamos en primer lugar las fuentes desde los repositorios *mercurial* de *Cluster Labs*, web oficial del proyecto *Pacemaker* (http://www.clusterlabs.org/):

```
# wget http://hg.clusterlabs.org/pacemaker/stable-
1.0/archive/Pacemaker-1.0.8.tar.bz2
```

Ahora que disponemos del paquete *Pacemaker-1.0.8.tar.bz2* con las fuentes de la utilidad, lo descomprimíos con *tar* y cambiamos al directorio generado:

```
# tar xjvf Pacemaker-1.0.8.tar.bz2

# cd Pacemaker-1-0-Pacemaker-1.0.8/
```

Procedemos a continuación de forma similar a como lo hemos hecho para los otros componentes de *Linux-HA*. Ejecutamos los *scripts autogen-sh* y *configure*, éste último con la opción *–with-heartbeat* para especificar que *Heartbeat* es el que actúa como capa de comunicación/mensajes entre los nodos del *clúster* (recordemos que es posible también utilizar *OpenAIS*):

```
Pacemaker-1-0-Pacemaker-1.0.8# ./autogen.sh

Pacemaker-1-0-Pacemaker-1.0.8# ./configure –-with-heartbeat
```

Ejecutamos *make* y *make install* para finalizar compilando e instalando *Pacemaker:*

```
Pacemaker-1-0-Pacemaker-1.0.8# make

Pacemaker-1-0-Pacemaker-1.0.8# make install
```

Con *Pacemaker* instalado ya disponemos de nuestra solución de alta disponibilidad *Linux-HA* completamente instalada y lista para ser configurada según nuestras necesidades. En el siguiente apartado vemos cómo podemos hacerlo de forma general editando los ficheros de configuración de *Heartbeat* y mediante el administrador *Pacemaker*, uno de los puntos fuertes de esta última versión disponible.

2.3 CONFIGURACION DE HEARTBEAT 3

En este apartado vamos a estudiar cómo configurar *Heartbeat 3* y nuestro *clúster* de alta disponibilidad de una forma sencilla y práctica. Para ello editamos algunos ficheros de configuración para el componente *Heartbeat*, para caracterizar las comunicaciones y la seguridad entre los nodos, y presentamos y usamos la interfaz administrativa para el *clúster* y sus recursos *Pacemaker*. Más adelante en este mismo documento vemos un ejemplo concreto de cómo llevar a cabo esta configuración en un ***clúster virtual de alta disponibilidad,*** uniendo Xen y *Linux-HA*.

Dividimos la configuración de nuestro *clúster* de alta disponibilidad en varias etapas, distinguiendo además de si la configuración debe ser realizada en todos los nodos del *clúster* o no.

2.3.1 Configuración en todos los nodos del *clúster*

A continuación vemos paso a paso la configuración que debemos realizar en cada uno de los nodos que forman parte del *clúster*. Como podemos ver, algunos de estos cambios se realizan de la misma forma en cada nodo, mientras que otros simplemente debemos adaptarlos en función del nodo que estamos configurando.

Nombre del nodo y fichero */etc/hosts*

Cada uno de los nodos debe disponer de un nombre único que lo identifica unívocamente en el *clúster*. Como se ha comentado en el apartado anterior, podemos establecer el nombre del nodo haciendo uso del comando ***hostname***. Si no facilitamos ningún nombre como parámetro al comando obtenemos el nombre actual para el nodo.

Además, es fundamental que en cada uno de los nodos del *clúster* editemos el fichero ***/etc/hosts*** como también hemos indicado en el apartado anterior. Incluimos una línea por cada uno de los nodos del *clúster*, especificando dirección de red y nombre.

Fichero */etc/ha.d/authkeys*

En el fichero */etc/ha.d/authkeys* vamos a configurar las claves de las que hacen uso los nodos del *clúster* para autenticarse, habiendo tres esquemas diferentes posibles para ser usados: *crc*, *md5* o *sha1*. Es decir, **en este fichero configuramos los aspectos de seguridad del *clúster***. Usaremos uno u otro modo de autenticación dependiendo de las exigencias de seguridad y rendimiento que establezcamos:

- El **método *crc*** es recomendable cuando los nodos del *clúster* se comunican a través de una red segura. *Crc* apenas consume recursos por lo que su aplicación también es recomendable cuando necesitamos disponer de la máxima cantidad de recursos posible.

- La **autenticación mediante *sha1*** es justamente lo contrario, es el método que requiere una mayor cantidad de recursos de CPU en su aplicación al mismo tiempo que aporta una gran seguridad.

- Como **solución intermedia** a los dos métodos anteriores existe ***md5***.

De forma general el fichero muestra el siguiente contenido:

```
auth <identificador>
<identificador> <metodo_autenticacion> [<clave_autenticacion>]
```

Fichero */etc/ha.d/ha.cf*

La configuración principal del *clúster* es la que incluimos en el fichero *ha.cf*, ubicado en el directorio */etc/ha.d/*. El fichero contiene todas las características configurables para el *clúster*, las cuales pueden ser habilitadas o deshabilitadas en función de la configuración requerida; para ello basta con descomentar o comentar la línea correspondiente, respectivamente. De manera general podemos decir que algunos de los aspectos configurables son los ficheros que almacenan información sobre la ejecución de los servicios como *log*, los parámetros que rigen la monitorización de recursos y nodos, puertos e interfaces de escucha para *Heartbeat* o los nodos que conforman el *clúster*.

Veamos cuáles son las opciones configurables más destacables:

- ***debugfile***. Permite especificar la ubicación del fichero que almacena los mensajes generados por *Heartbeat* en modo *depuración*.

- ***logfile*.** Al igual que la opción anterior, nos permite generar el resto de mensajes generados como *log*.

- ***logfacility*.** En él indicamos la facilidad para el uso de *syslog()* o *logger*.

- ***keepalive*.** Es el tiempo (por defecto en segundos, aunque es posible especificarlo en milisegundos escribiendo *ms*) que transcurre entre la emisión de dos *pings* hacia cada nodo del *clúster* para su monitorización.

- ***deadtime.*** Permite especificar el tiempo en segundos que deben transcurrir para asumir que un nodo no encuentra disponible o que está *muerto.* Si establecemos este parámetro demasiado bajo podemos experimentar un conocido problema denominado *splitbrain* o *partición del clúster*, en el que varios nodos o grupos de nodos pueden *tomar* la ejecución de un recurso al mismo tiempo al creer que el resto no se encuentran disponibles.

- ***warntime.*** En la opción *warntime* especificamos el tiempo en segundos que transcurre antes de indicar la advertencia de último intento de *ping* en la monitorización. Si es habitual alcanzar el último intento de *ping* entonces quiere decir que nos encontramos próximos a considerar el nodo como *muerto*.

- ***initdead.*** Es el tiempo en segundos que debe transcurrir de manera obligatoria antes de la puesta en marcha de *Heartbeat*. Si por el motivo que fuese un nodo no logra iniciarse antes de que transcurra este tiempo es considerado como un nodo *muerto*.

- ***udpport.*** Permite especificar el número de puerto que utilizar *Hearbeat* para las comunicaciones de tipo *unicast* y *broadcast*. Este puerto es el mismo tanto para el envío de *pings* de monitorización al resto de nodos como para su recepción.

- ***bcast.*** Interfaz de red que usa *Heartbeat* para el envío de los *pings*. Lógicamente, si nodos diferentes disponen de interfaces de red diferentes debemos tenerlo en cuenta a la hora de establecer el valor de esta opción.

- ***ucast.*** Su uso es similar a la opción anterior, con la diferencia de que es usada cuando disponemos solamente de comunicaciones entre pares de nodos en el *clúster*. Además de la interfaz de red para la comunicación escribimos la dirección de red del otro nodo, por lo que este parámetro es diferente en cada nodo.

- ***autofailback.*** Esta opción sirve para especificar el comportamiento de *Heartbeat* cuando un fallo se produjo en el nodo activo que ejecutaba los servicios, éstos fueron migrados a otro nodo y después se solventaron los problemas en el nodo primario. Hay dos opciones posibles: que una vez que el nodo que falló se recupera los recursos vuelvan a él, o que permanezcan en el nuevo nodo activo. Para la primera opción escribimos el valor *true* mientras que para la segunda escribimos *false*.

- ***node.*** Con *node* especificamos los nodos que forman parte del *clúster*. Para cada nodo escribimos el nombre que obtenemos al ejecutar en él el comando *hostname* o *uname –n*, de ahí la importancia de su configuración como hemos visto anteriormente.

- ***crm.*** Con el uso de la opción *crm* permitimos o no el uso del administrador de recursos del *clúster CRM* (*Cluster Resource Manager*). Escribimos *yes* para permitirlo o *no* para no hacerlo. Como ya hemos dicho en varias ocasiones, *Pacemaker* es el *CRM* que estudiamos en este capítulo y que aplicamos para la administrar la funcionalidad y los recursos de nuestro *clúster*.

Como podemos imaginar existen otras muchas opciones disponibles para su configuración en este fichero; aunque las presentadas son las más básicas con su uso podemos garantizar el correcto funcionamiento de nuestro sistema *Linux-HA*.

2.3.2 Configuración en uno de los nodos del *clúster*

A continuación seguimos configurando el *clúster* desde uno de los nodos del mismo, ya que como vamos a ver los cambios que aplicamos se replican y distribuyen al resto de nodos realizada la configuración vista hasta el momento. Es momento de comenzar a definir la funcionalidad de nuestro *clúster*: qué servicios se van a ser ejecutados y en qué nodos, cómo van a ser monitorizados tanto nodos como servicios o recursos, qué tipo de comunicaciones se establecen entre los nodos, cuando se produce la migración de los servicios o grupos de servicios, etc. Para configurar todos estos aspectos usamos la aplicación *Pacemaker*.

Introducción a la configuración del *CRM/Pacemaker*

Como ya hemos citado en algunas ocasiones *Pacemaker* es el *administrador de recursos del clúster* (*Cluster Resource Manager o CRM*) que vamos a usar de manera conjunta con *Heartbeat 3* (como capa de mensajes/comunicación) en nuestro sistema de alta disponibilidad *Linux-HA*. Con *Pacemaker* gestionamos toda la funcionalidad del *clúster*: nodos, recursos, ficheros de configuración, propiedades globales del *clúster*, *agentes de recursos*, etc. Todos estos aspectos quedan recogidos en la configuración del *clúster* en forma de fichero *xml*, por lo que uno de los objetivos principales de *Pacemaker* es facilitar la manipulación de estos ficheros por parte del administrador.

A continuación presentamos y analizamos algunos de los elementos más importantes del fichero *cib.xml*, antes de realizar su configuración en el *clúster*. Es de importancia que el administrador del *clúster* conozca bien en qué consisten, cuál es su finalidad o qué puede obtener cuando se incluyen o aplican. El fichero *cib.xml* (*Cluster Information Base*), ubicado en el directorio */usr/var/lib/heartbeat/crm/* es el fichero principal que recoge toda la funcionalidad del *clúster* y cómo ésta se encuentra configurada y debe ser explotada. Además de editar el contenido de *cib.xml* con *Pacemaker* (*crm*) es posible hacerlo de forma manual si así lo deseamos; la sintaxis utilizada en los comandos *crm* es prácticamente igual a la notación *xml* utilizada en el fichero para representar los diferentes elementos.

El fichero *cib.xml*

Analicemos la estructura de este fichero tan importante y qué tipo de elementos puede contener en sus líneas para definir el comportamiento del *clúster*. El archivo *cib.xml* contiene dos secciones bien diferenciadas: **una sección *estática* <*configuration*> que recoge la configuración del *clúster* y otra *dinámica* <*status*> que contiene su estado en todo momento**:

```
<cib>
   <configuration>
         <crm_config/>
         <nodes/>
         <resources/>
         <constraints/>
   </configuration>
   <status/>
</cib>
```

La **sección <*status*>** (dinámica) es mantenida en todo momento por el *CRM* mostrando información sobre el estado actual del *clúster:* qué nodos se encuentran *online* y cuáles *offline*, en qué nodos se encuentran ejecutándose los recursos, cuáles fueron los últimos fallos

registrados por *Heartbeat*, etc. Ya que es actualizada por el *CRM* no es recomendable manipular manualmente esta sección y sí en cambio hacer uso de las herramientas disponibles para ello.

Como podemos ver en el código anterior la **sección <*configuration*>** contiene a su vez otras cuatro sub-secciones. En cada una de estas sub-secciones es donde se incluyen los diferentes elementos que componen y determinan la configuración del *clúster*: propiedades generales para el *clúster*, definición y configuración de nodos, definición y configuración de recursos y restricciones o relaciones entre los recursos. Veamos en los siguientes puntos en qué consiste cada una de ellas:

- ***Configuración del CRM*.** La sección <*crm_config*> contiene los distintos atributos y propiedades que determinan cómo se comporta el *clúster* como entidad grupal. Esta configuración era por lo general más difícil de establecer en versiones anteriores de *Heartbeat*, pero en la actual gracias a *Pacemaker* es mucho más sencillo ya que éste permite que se gestionen de forma automática los identificadores de las propiedades o las etiquetas *xml* que debemos utilizar.

  Entre las propiedades básicas para la configuración de nuestro *clúster* encontramos las siguientes:

  - ***default-action-timeout.*** Es el tiempo máximo o *timeout* fijado para que un *agente de recurso* realice una operación determinada (*start, stop, status, monitor*) sobre un recurso. Si definimos un *timeout* particular para la operación en el recurso, es éste último el *timeout* que se aplica y no el establecido en *action-idle-timeout*.

  - ***symmetric-cluster.*** Permite indicar si el *clúster* es simétrico (los recursos pueden ser ejecutados en cualquier nodo, *true*) o no (*false*), en cuyo caso debemos especificar en cuáles es posible.

  - ***stonith-enabled.*** Mediante esta propiedad podemos habilitar o no el que un dispositivo adicional al *clúster* (*dispositivo stonith*) pueda apagar un reiniciar un nodo que ha experimentado dificultades (esta operación es conocida como *node fencing*). Es necesario su uso en ocasiones, ya que es posible que aunque un nodo haya sido identificado como *fallido* no haya liberado los recursos que estaba ejecutando.

  - ***stonith-action.*** Permite especificar la acción a llevar a cabo por el *dispositivo stonith*, visto en la propiedad anterior. Tenemos dos posibilidades: *off* para apagar el nodo *fallido* y *reboot* para reiniciarlo.

  - ***cluster-infrastructure.*** Permite especificar el tipo de infraestructura para el *clúster*, pudiendo elegir entre *heartbeat* u *openais* dependiendo de la utilidad empleada para realizar las funciones propias en la capa de comunicación y mensajes de la solución de alta disponibilidad.

  - ***no-quorum-policy.*** Permite definir qué hacer cuando el *clúster* no dispone de mecanismo *quorum*, el encargado de resolver el problema citado anteriormente llamado *SplitBrain*. La solución planteada por *quorum* es la de no seleccionar más de una partición en el *clúster* para la ejecución de los recursos cuando fallen las comunicaciones.

  - ***default-resource-stickiness.*** Permite establecer la *adherencia por defecto* de los recursos para ser ejecutados en el nodo actual o por el contrario ser migrados a

otro nodo que ofrece unas condiciones mejores. Aunque admite valores negativos, lo normal es establecer un valor mayor o igual a cero, teniendo que cuanto mayor sea el valor mayor es la *adherencia* para permanecer en el nodo actual. Si establecemos como valor *INFINITY*, el recurso nunca es migrado a otro nodo. Es útil sobre todo cuando establecemos sistemas de puntuaciones para la ejecución de los recursos en determinados nodos, en los que el valor escrito en *default-resource-stickiness* o *resource-stickiness* para el recurso es añadido a su puntuación particular para ser ejecutado en un nodo.

- ***default-resource-failure-stickiness.*** De forma similar a *default-resource-stickiness*, en este caso establecemos un valor que es usado como penalización en la puntuación del recurso para ser ejecutado en un determinado nodo. Análogamente, aunque admite valores positivos lo normal es utilizar valores menores o iguales a cero, teniendo que si escribimos *–INFINITY* el recurso es migrado a otro nodo siempre que un fallo sea detectado.

- ***is-managed-default.*** Mediante esta propiedad es posible habilitar la administración manual del *clúster*, lo que incluye operaciones tales como la manipulación del estado de un recurso –iniciarlos, pararlos-, el nodo en el que se encuentra ejecutándose, etc.

- ***Nodos y grupos de nodos.*** En esta segunda parte de la sección <*configuration*>, <*nodes*>, son incluidos los nodos que forman el *clúster*. Para identificar cada nodo de forma única dentro del *clúster* disponemos de su nombre de equipo (el que establecemos con el comando *hostname*) y un identificador alfanumérico.

  Para hacer más fácil la configuración del *clúster* cuando el número de nodos comienza a no ser lo manejable que deseamos *Heartbeat* permite **agrupar varios nodos en *grupos de nodos***. Así, de igual forma que es posible establecer restricciones o relaciones de *orden, colocación* o *localización* entre diferentes nodos lo es entre *grupos de nodos*. Por ejemplo, podemos establecer para un determinado recurso que sólo sea posible ejecutarlo en un *grupo de nodos* determinado.

- ***Recursos y grupos de recursos.*** En la sección <*resources*> definimos los recursos del *clúster*, es decir, los elementos a los que queremos dotar de alta disponibilidad. En esta definición incluimos servicios, aplicaciones, herramientas, configuraciones, elementos *hardware*… Por ejemplo, un recurso puede ser una dirección IP, el servicio de centralitas de VoIP *Asterisk*, un servidor web, un disco duro, una tarjeta de red, etc.

  Un recurso debe poder soportar cuatro operaciones diferentes: ***start, stop, status* y *monitor***. Es decir, para que un elemento pueda actuar como recurso en la infraestructura de alta disponibilidad administrada por *Heartbeat* debe poder ser iniciado, detenido, y debe ser posible comprobar su estado en cualquier instante de tiempo (*status:* iniciado o detenido) y el estado de su ejecución (*monitor*). Para administrar los recursos *Heartbeat* hace uso de unos *scripts* que contienen información sobre cómo llevar a cabo estas operaciones: los ***agentes de recursos***. Los *agentes de recursos* son creados siguiendo diferentes estándares, por ejemplo *lsb* u *ocf*. Es obligatorio que para poder ofrecer alta disponibilidad de un recurso dispongamos de un *agente de recurso* para operar sobre él; *Heartbeat* proporciona una gran cantidad de *agentes de recursos* para los recursos más habituales, pero para otros debemos implementarlos nosotros mismos como usuarios o cambiar los existentes (por ejemplo, para dotar de alta disponibilidad al servicio *Asterisk* debemos editar el *script asterisk*

instalado por defecto con la aplicación ya que éste no incluye las operaciones *status* y *monitor*).

En muchas ocasiones, cuando el número de recursos en nuestro sistema de alta disponibilidad crece, es necesario tratar varios recursos de la misma forma, aplicar un gran número de restricciones o relaciones entre los recursos, etc. Es por ello que *Heartbeat* permite el establecimiento de ***grupos de recursos***, siendo éstos manipulados como si de un solo recurso se tratara, facilitando enormemente las tareas de configuración y administración de la alta disponibilidad del *clúster*. **Los recursos que forman parte de un *grupo de recursos*:**

- **Siempre son ejecutados en el mismo nodo**, es decir, existe una *restricción de colocación* entre los recursos de un mismo grupo de forma que cada recurso siempre es ejecutado en el mismo nodo que el resto.

- **Siempre son puestos en marcha y detenidos en el mismo orden**, es decir, existe una *restricción de orden* entre los recursos de un mismo grupo que los obliga a ello.

Como los *grupos de recursos* son tratados como si fueran recursos individuales también es posible definir **restricciones o relaciones entre *grupos de recursos*** de la misma forma.

- ***Restricciones.*** Como se ha citado en varias ocasiones en las últimas páginas es posible establecer ciertas ***restricciones* o *relaciones* entre nodos o *grupos de nodos* y entre recursos o *grupos de recursos*** con el objetivo de definir de la forma más concisa posible su comportamiento. Aplicar restricciones se hace necesario cuando existen dependencias entre recursos. Por ejemplo, el recurso *Asterisk* requiere ser ejecutado en el mismo nodo que el recurso *IPaddr (IP virtual)*, ya que de lo contrario el *Asterisk* estaría en funcionamiento pero no accesible para el establecimiento de llamadas.

  En *Heartbeat* existen tres tipos de *restricciones* o *relaciones* entre recursos:

  - ***Restricciones de localización.*** Permiten especificar para un recurso en qué nodos es posible su ejecución. Para ello son utilizadas reglas de preferencia para la ejecución del recurso en un nodo y se evalúan determinadas expresiones como condiciones para ver si la regla se cumple o no. Para referenciar el recurso utilizados su identificador único. La prioridad de ejecución del recurso en el nodo es recogida en el atributo *score*. Si al evaluar la condición la regla se cumple, la puntuación del recurso es modificada teniendo en cuenta *score*.

  - ***Restricciones de colocación.*** Permiten indicar qué recursos pueden ser ejecutados en un mismo nodo y qué recursos no pueden hacerlo. En este tipo de restricciones dos recursos son citados junto a una puntuación o *score* asociada que determina si pueden ser ejecutados en el mismo nodo o no. Lo habitual es escribir como puntuación *INFINITY* si deseamos que puedan hacerlo o –*INFINITY* en caso contrario. El orden en que son facilitados los dos recursos es determinante (el primer recurso es denominado *recurso 'from'* y el segundo recurso o *'to'*), ya que no es lo mismo imponer que un *recurso A* sea ejecutado siempre en el mismo nodo que el *recurso B* que al contrario.

  - ***Restricciones de orden.*** Son utilizadas para especificar el orden en que las operaciones *start* y *stop* que debe realizar un *agente de recurso* tienen que llevarse a cabo. Esto es necesario, por ejemplo, cuando ponemos en marcha un sitio web que hace uso de una base de datos: el servicio de base de datos debe

estar iniciado previamente. Es posible especificar que la operación se realice *antes* o *después* para el primer recurso en relación al segundo.

Configuración del fichero *cib.xml* con *Pacemaker*

Con configuración *cib (Cluster Information Base)* hacemos referencia a la configuración que realizamos en el fichero *cib.xml*. Este fichero contiene toda la configuración funcional del *clúster*, especificada en notación *xml*. En versiones anteriores de *Heartbeat* teníamos que editar este fichero directamente, con las complicaciones que ello conlleva. En cambio, en la versión actual disponemos de *Pacemaker*, y más concretamente de su utilidad *crm*, para su edición y configuración.

Antes de comenzar a utilizar *crm* debemos iniciar el servicio *Heartbeat* en cada uno de los nodos del *clúster*, para lo que ejecutamos el *script* ubicado en */etc/init.d/* con el parámetro *start*:

```
# /etc/init.d/heartbeat start
Starting High-Availability services: Done.
```

Con *Heartbeat* ejecutándose basta con escribir el comando *crm* en una terminal para acceder a la consola de esta herramienta como podemos ver a continuación:

```
# crm
crm(live)#
```

A partir de aquí se abre un abanico de diversos menús y posibilidades para la configuración del *clúster* estructurados en niveles administrativos. Si pulsamos dos veces la tecla tabulador o escribimos *help* en el primer nivel podemos observar los diferentes menús de configuración y comandos disponibles para comenzar, como se aprecia en la figura 6-4.

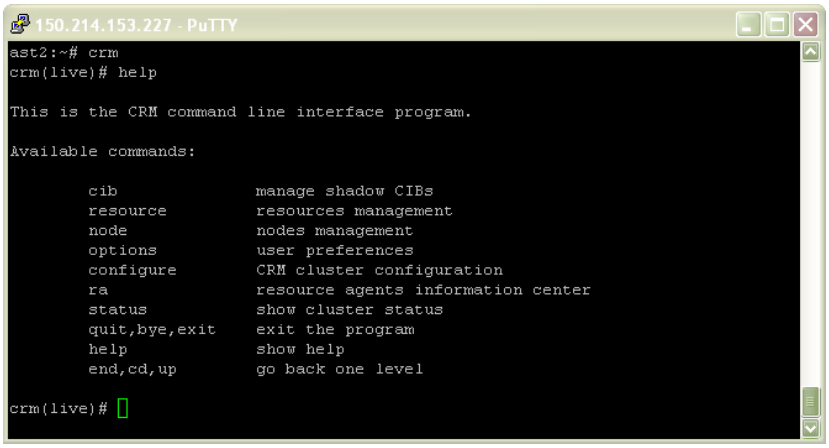


Figura 6.4. Diferentes menús disponibles al comenzar a trabajar con crm.

Veamos de forma breve en qué consiste cada una de estas posibilidades:

- ***cib.*** Eligiendo *cib* estamos en disposición de editar y configurar el fichero *cib.xml*. *crm* guarda en diferentes ficheros denominados *cib shadow* las distintas configuraciones que vayamos realizando. Por ejemplo, podemos crear un *cib shadow* para la configuración del *clúster* con alta disponibilidad para el servicio *Asterisk*, y otro *cib shadow* para la configuración con alta disponibilidad para el servicio web *Apache*; es decir, podemos

mantener un fichero por cada servicio que ofrece el *clúster* y aplicarlo cuando lo creamos conveniente.

Desde *cib* podemos crear nuevos ficheros *shadow*, editar los existentes o eliminarlos, elegir el que queremos usar, distribuir los cambios realizados al resto de nodos del *clúster*, etc. Si escribimos *help* después de ingresar en *cib* podemos ver las distintas posibilidades que tenemos, como muestra la figura 6.5: ***new, delete, reset, commit, use, diff, list, import, cibstatus, quit, bye, exit, help, end, cd* y *up***.

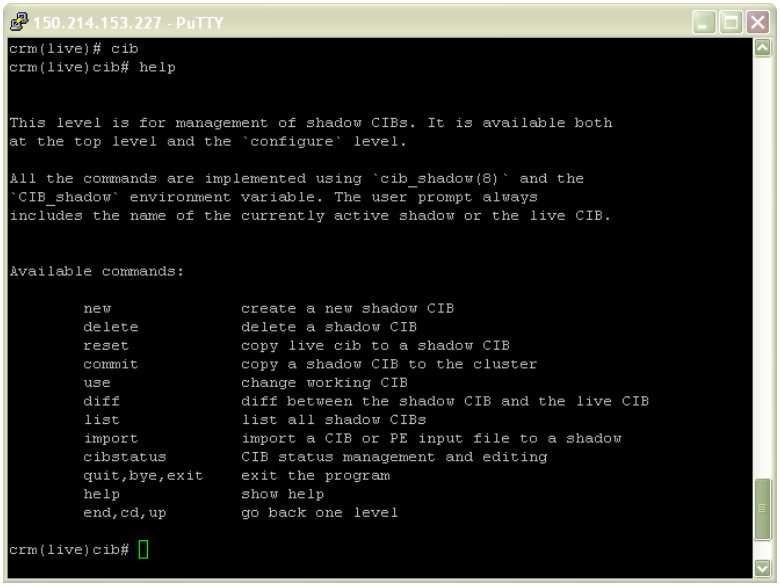


Figura 6.5. Comandos posibles a ejecutar en el nivel cib.

- ***resource.*** Escribiendo *resource* accedemos a la administración de los recursos del *clúster*. Podemos conocer el estado de los recursos, listarlos, cambiar su estado (iniciarlos, detenerlos, reiniciarlos…), migrarlos a otro nodo del *clúster*, editar los parámetros de los mismos así como sus *metadatos*, ver la cuenta de fallos asociada, etc.

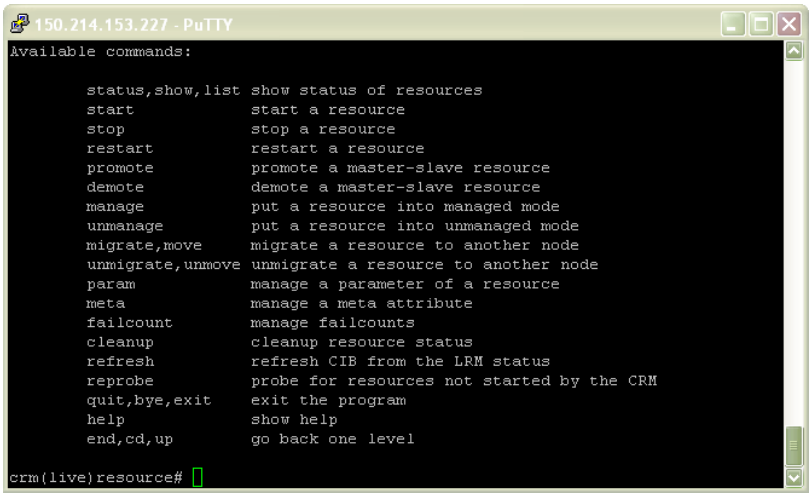


Figura 6.6. Comandos posibles a ejecutar en el nivel resource.

- ***node.*** *Node* permite el acceso a la administración de los nodos que forman parte del *clúster*. Disponemos de comandos para mostrar o listar los nodos del *clúster*, cambiar el estado de un nodo (*online/offline*), apagarlo, eliminarlo de la configuración… también podemos conocer el estado de sus atributos y editarlos si lo consideramos necesario.

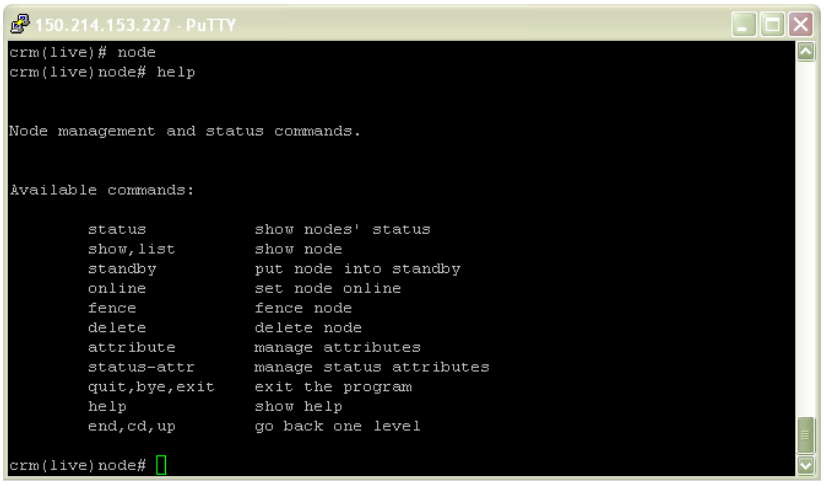


Figura 6.7. Comandos posibles a ejecutar en el nivel node.

- ***options.*** Desde aquí podemos editar las preferencias relativas al usuario para la interfaz de la consola. Por ejemplo, es posible cambiar el usuario para el *clúster*, qué editor abrir cuando editamos directamente ficheros, o el esquema de colores utilizado para la salida de *crm*.

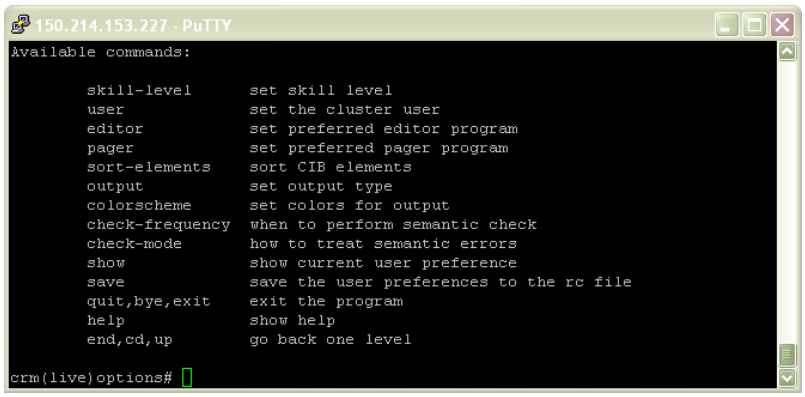


Figura 6.8. Comandos posibles a ejecutar en el nivel node.

- ***configure.*** A través de *configure* desarrollamos la configuración que contienen los ficheros *cib shadow*. El número de comandos disponibles bajo *configure* es mucho mayor que en los casos anteriores, ya que la configuración posible a realizar también es mayor. Es muy importante este menú ya que a través de él podemos realizar las operaciones más relevantes, como la administración de nodos y recursos en el *clúster*, la administración de grupos de recursos, de restricciones de localización/orden para los recursos, o propiedades generales del *clúster*, por ejemplo.

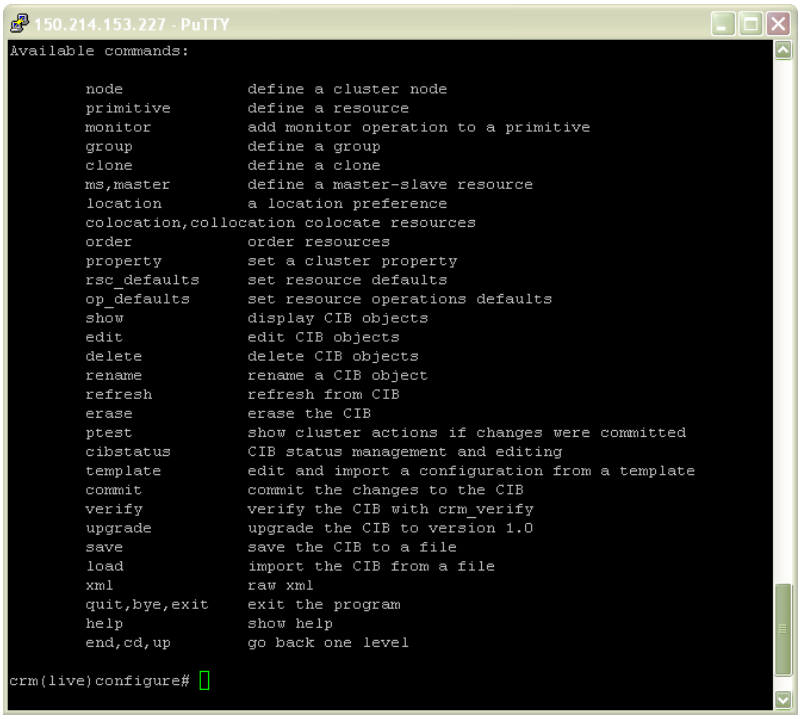


Figura 6.9. Comandos posibles a ejecutar en el nivel cib.

Asimismo podemos realizar muchas más operaciones necesarias e interesantes, como la manipulación *en vivo* del fichero *CIB*, su publicación y distribución en el resto de nodos del *clúster*… Dos de los comandos más importantes son *show* y *commit*. Si ejecutamos el comando *show* podemos ver la configuración actual del *clúster* (véase la figura 6.10); ejecutando *commit* actualizaremos la configuración *CIB* en todos los nodos.

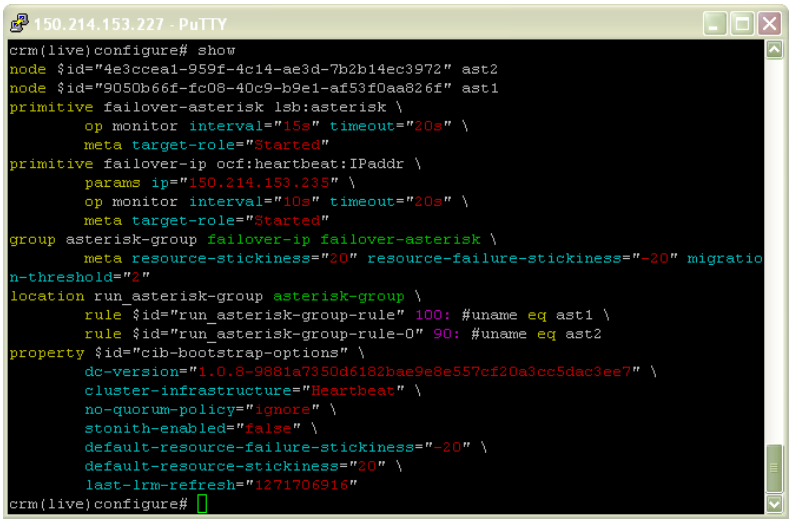


Figura 6.10. Ejecución del comando show en el nivel configure.

- ***ra.*** En el nivel *ra* encontramos el *centro de información sobre los agentes de recurso.* Los *agentes de recurso* permiten controlar los recursos del *clúster* y realizar diversas operaciones sobre ellos (como iniciarlos, detenerlos, etc.) pudiendo seguir para ello diferentes estándares (*lsb, ocf,*…).

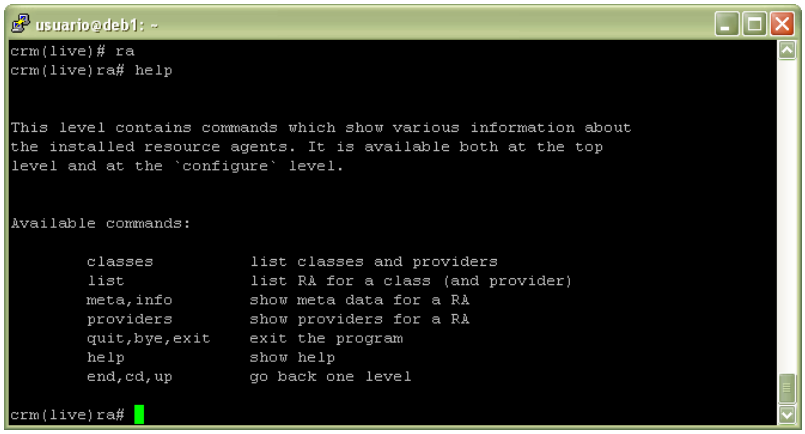


Figura 6.11. Comandos posibles a ejecutar en el nivel ra.

Como se puede apreciar en la figura 6.11, es posible por ejemplo listar las clases y proveedores de *agentes de recursos*, listar los *agentes de recursos* para una determinada clase o proveedor o mostrar los *metadatos* sobre un *agente de recurso.*

- ***status.*** Tecleando *status* accedemos al nivel para conocer el estado del *clúster.* Muestra si los nodos se encuentran actualmente *online*, qué servicios se encuentran ejecutándose en qué nodos, cuáles fueron los últimos fallos detectados, etc (véase la figura 6.7).

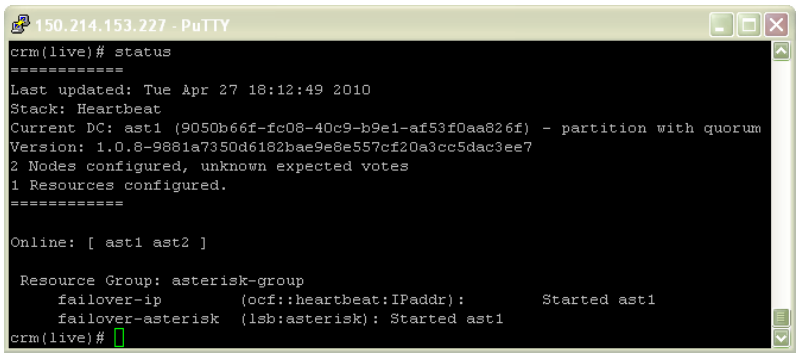


Figura 6.12. Ejecución del comando/nivel status.

- ***quit, bye, exit.*** Se trata de los tres comandos disponibles para abandonar *crm*.

- ***help.*** Ejecutando *help* obtenemos ayuda en cualquier momento, no sólo en el primer nivel de la aplicación, sobre los comandos y opciones disponibles.

- ***end, cd, up.*** Mediante estos tres comandos podemos navegar entre los diferentes niveles administrativos en los que se encuentra dividido *crm*, ya que permiten regresar al nivel inmediatamente superior.

Conocemos en mayor profundidad cómo realizar una configuración de nuestro sistema de alta disponibilidad con *Pacemaker* y *crm* en el siguiente apartado, en el que desarrollamos la implementación de un *clúster virtual de alta disponibilidad* para el servicio *Asterisk.* Además de las utilidades de configuración presentadas *Heartbeat* dispone de otras muchas que pueden ser utilizadas en función de nuestras necesidades, como por ejemplo *Softdog*, *Stonith* o *pingd*, que

permite determinar si un nodo ha perdido la conectividad cuando su interfaz de red cae o no es alcanzable y migra los recursos que estuviera ejecutando.

Hemos podido comprobar que las posibilidades presentadas por *Pacemaker*, *crm* y *Heartbeat* son muy grandes. Es de absoluta recomendación visitar la web de *Cluster Labs* (http://www.clusterlabs.org/) y revisar la documentación facilitada para conocer mejor cómo configurar nuestro *clúster*; hay determinadas cuestiones, como por ejemplo los atributos configurables para un nodo o un recurso, que es recomendable consultar en la documentación facilitada: la guía *Cluster from scratch* (*clúster* desde cero) *wiki*, ejemplos, guías *howto*, referencias, etc.

3 Alta disponibilidad en clústeres virtuales con Heartbeat 3

Una vez que hemos presentado en las páginas anteriores en qué consiste teóricamente un *clúster de alta disponibilidad* y cómo es posible instalar y configuración la solución más importante en este campo vamos a aplicarlo de forma práctica en la construcción de un ***clúster virtual de alta disponibilidad* para el servicio de telefonía IP *Asterisk*.**

Como fue introducido en el capítulo *2. Virtualización de Plataforma* **podemos relacionar de forma inmediata las técnicas de *clustering* y la virtualización** mediante la creación de ***clústeres virtuales***, en los que los *dominios invitados* pueden actuar como nodos del *clúster*. Los ***nodos virtuales*** pueden por tanto encontrarse ubicados en un mismo equipo físico si comparten el sistema anfitrión o bien en anfitriones distintos si así lo consideráramos necesario. De esta forma podemos ver cómo es posible integrar estas dos tecnologías y conseguir **exprimir al máximo las ventajas de una y otra, especialmente las que puede aportar la virtualización**: podemos ahorrar espacio y consumo por parte de los nodos del *clúster*, aprovechamos en mayor grado el *hardware* de los servidores, y los costes administrativos se reducen, permitiendo debido a la nueva naturaleza de los nodos del *clúster* mejorar los procesos de recuperación ante desastres, puesta en marcha, copias de seguridad, etc. Esta mejoría la podemos apreciar más aún si cabe si implantamos un ***clúster virtual* de tipo *Beowulf*, en el que todos los nodos son idénticos.** En tal caso tan sólo debemos configurar una máquina virtual y replicarla tantas veces como nodos queramos tener disponibles.

Mediante el establecimiento de *clústeres virtuales* potenciamos dos de las características fundamentales que deben estar presentes en un *clúster*: **escalabilidad y robustez.** Es indudable la gran escalabilidad de las máquinas virtuales frente a equipos físicos, resultando mucho más económico, aumentando la seguridad y flexibilidad de nuestras configuraciones. Respecto a estas características debemos puntualizar un detalle: si disponemos nuestro *clúster virtual* en un único servidor físico tanto escalabilidad como robustez disminuyen considerablemente, ya que la depende fundamentalmente de los recursos disponibles en él y si cae el servidor físico por un fallo hardware, las máquinas virtuales que aloja caerían también. Por este motivo **es totalmente recomendable que un *clúster virtual* no sólo disponga de un servidor físico como anfitrión**.

En la figura 6.13 podemos apreciar la arquitectura presentada por el *clúster virtual de alta disponibilidad* que desarrollamos y configuramos en los apartados siguientes. Como podemos ver en la imagen y de acuerdo a todo el trabajo realizado hasta el momento usamos *Xen* como *virtualizador de plataforma* en la consolidación de servidores de telefonía IP *Asterisk*, y *Heartbeat-Linux-HA* para proporcionar alta disponibilidad a los nodos virtuales.

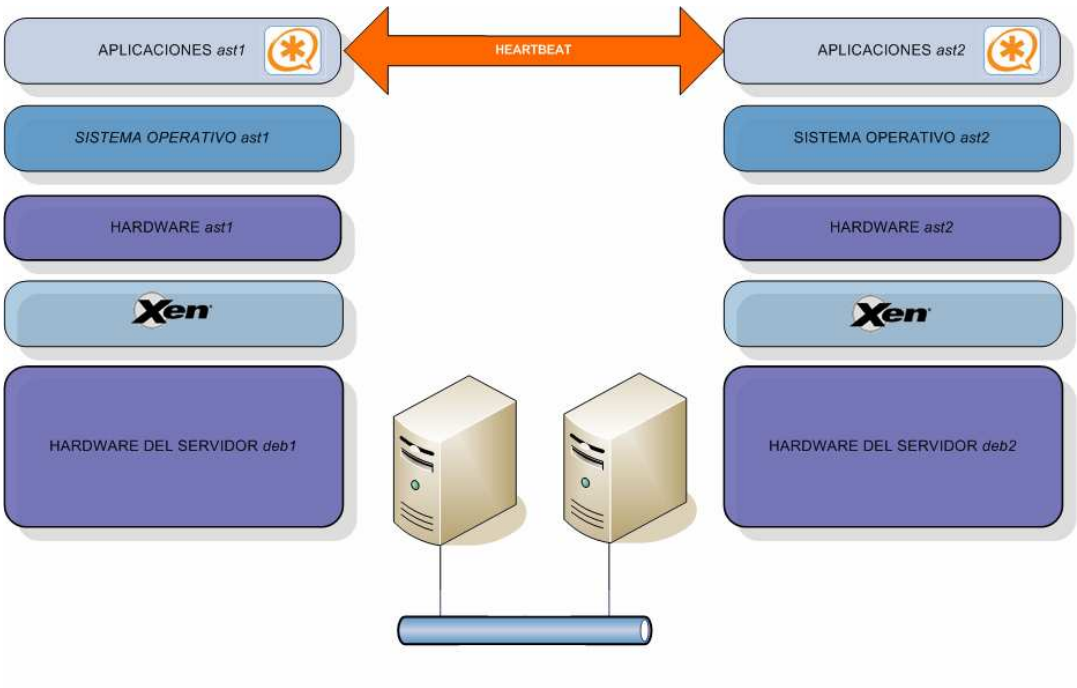


Figura 6.13. Clúster virtual de alta disponibilidad usando Xen, Asterisk y Heartbeat.

Disponemos de dos anfitriones diferentes para crear una infraestructura lo más real y eficiente posible; aunque en ella sólo incluimos un dominio por anfitrión en condiciones reales éstos pueden convivir con otros que sean configurados con otros propósitos. Los equipos usados como anfitriones son exactamente idénticos en hardware y sistema operativo: ambos son servidores **HP Proliant DL120 G5** con procesador **Intel(R) Xeon(R) CPU E3110 @ 3.00GHz** y de **4Gb de memoria** RAM. Como sistema operativo disponen de Debian 5 Lenny. Veamos a continuación los pasos seguidos para la creación del *clúster virtual:* configuración de los servidores anfitriones con *Xen*, puesta en marcha de los *nodos virtuales* con *Asterisk* como servicio crítico en ejecución y finalmente la configuración de alta disponibilidad gracias al uso de *Heartbeat 3.*

3.1 CONFIGURACION DE LOS SERVIDORES ANFITRIONES

Un *clúster virtual* no puede ser creado sin configurar previamente los servidores anfitriones que van a soportar la creación y mantenimiento de los dominios invitados o *nodos virtuales.* Contamos para nuestro caso práctico de dos servidores con las características recogidas en la tabla 6.1.

Tabla 6-1. Servidores anfitriones que aloja el *clúster virtual*

Modelo	Procesador	Memoria	Virtualiz. HW	Disco	Red
HP Proliant DL120 G5	Intel(R) Xeon(R) CPU E3110 @ 3.00GHz (Coreduo)	4 Gb SDRAM DDR2 PC2-6400	Sí (Intel VT - vmx)	1 disco duro SATA 250 Gb	NetXtreme BCM5722 Gigabit Ethernet PCI Express 1GB/s 64 bits

Ambos servidores disponen del mismo *hardware* y el mismo sistema operativo instalado, Debian 5 Lenny como hemos citado en el apartado anterior. Partiendo de este punto,

Xen ha sido instalado en ambos servidores (*deb1* y *deb2*) como *virtualizador de plataforma*, siguiendo los procedimientos y configuraciones explicadas en detalle en los capítulos 3 y 4, *Xen* y *Desarrollo, Implementación y Prueba de una Infraestructura Virtual.*

3.2 PUESTA EN MARCHA DE LOS *NODOS VIRTUALES*

Para la creación y puesta en marcha de los *nodos virtuales* de nuestro *clúster* creamos una máquina virtual tipo en uno de los anfitriones la cual replicamos en el segundo anfitrión. De esta forma los tiempos de aprovisionamiento y disposición de los *nodos virtuales* pueden verse considerablemente reducidos, quedando pendiente la configuración particular para cada nodo, como por ejemplo sus interfaces de red o nombre de *host*.

Las imágenes de disco de las máquinas virtuales han sido creadas siguiendo la metodología vista en los capítulos 3 y 4, teniendo como sistema operativo instalado Debian 5 Lenny. A continuación podemos ver el contenido de los ficheros de configuración de ambos dominios (*ast1* y *ast2*, los dos *nodos virtuales*), usados por los anfitriones para el arranque de los mismos.

Fichero de configuración para el dominio/*nodo virtual ast1*:

```
kernel = "/boot/vmlinuz-2.6.26-2-xen-amd64"
ramdisk = "/boot/initrd.img-2.6.26-2-xen-amd64"
memory = 1024
name = "ast1"
disk =
['file:/media/imagenes/ast1_hb/imagen,sda1,w','file:/media/imagenes/
ast1_hb/swap,sda2,w']
root = "/dev/sda1 ro"
dhcp = "no"
vif = ['mac=1A:1B:1C:2A:2B:2C']
netmask = '255.255.255.0'
gateway = '150.214.153.1'
ip = '150.214.153.233'
broadcast = '150.214.153.255'
on_poweroff = 'destroy'
on_reboot = 'restart'
on_crash = 'restart'
cpus = "0"
vcpus = 1
extra = 'console=hvc0 xencons=tty1'
vnc=1
sdl=0
```

Fichero de configuración para el dominio/*nodo virtual ast2*:

```
kernel = "/boot/vmlinuz-2.6.26-2-xen-amd64"
ramdisk = "/boot/initrd.img-2.6.26-2-xen-amd64"
memory = 1024
name = "ast2"
disk =
['file:/media/imagenes/ast2_hb/imagen,sda1,w','file:/media/imagenes/
ast2_hb/swap,sda2,w']
root = "/dev/sda1 ro"
dhcp = "no"
vif = ['mac=2A:2B:2C:3A:3B:3C']
netmask = '255.255.255.0'
gateway = '150.214.153.1'
ip = '150.214.153.234'
```

```
broadcast = '150.214.153.255'
on_poweroff = 'destroy'
on_reboot = 'restart'
on_crash = 'restart'
cpus = "1"
vcpus = 1
extra = 'console=hvc0 xencons=tty1'
vnc=1
sdl=0
```

Es fácil observar que ambos dominios son similares, con las diferencias en su configuración relativas al nombre de dominio, ficheros imagen de disco y *swap*, y configuración de la interfaz de red; es importante resaltar que el nodo *ast1* dispone de la dirección IP 150.214.153.233 y el nodo *ast2* 150.214.153.234. Ambos nodos hacen uso de un solo procesador de su equipo anfitrión.

Además, en ambos nodos instalamos *Asterisk* de acuerdo a los pasos presentados en el capítulo 4, *Introducción a la telefonía IP*. No es necesario realizar configuración alguna de este servicio pues las pruebas a realizar en este capítulo tienen como objetivo observar la correcta migración de los servicios de un nodo a otro del *clúster* cuando algún fallo es detectado, y no si *Asterisk* se encuentra configurado de forma eficiente o conocer el número de llamadas simultáneas que es capaz de soportar, como se hizo en el capítulo anterior.

Finalmente instalamos también paso por paso *ClusterGlue*, *ResourceAgents*, *Heartbeat* y *Pacemaker* en cada uno de los dominios.

3.3 CONFIGURACION DE ALTA DISPONIBILIDAD

Alcanzado este punto disponemos de dos servidores anfitriones con un dominio invitado ejecutándose en cada uno. Es hora de configurar nuestro sistema de alta disponibilidad como ha sido presentado anteriormente tomando como nodos de nuestro *clúster* los dos sistemas invitados *ast1* y *ast2*. Como podemos ver, una vez que disponemos de nuestros *nodos virtuales* ejecutándose, las diferencias entre configurar un *clúster virtual* y uno *normal* son inexistentes.

3.3.1 Configuración en todos los nodos del *clúster virtual*

Sigamos los pasos establecidos en el punto anterior para comenzar a configurar nuestro *clúster virtual*. Primero realizamos la configuración replicada en cada uno de los *nodos virtuales* del *clúster*, nos preparamos para dar soporte de alta disponibilidad. Después, configuramos en un solo nodo el fichero *cib.xml*, que define cómo es el comportamiento de nuestro *clúster*, distribuyéndose automáticamente esta configuración en el resto de nodos.

Nombre del nodo y fichero */etc/hosts*

Asignamos a los dos nodos su nombre usando el comando *hostname*. Comprobamos que lo hemos hecho correctamente con el propio comando sin parámetros o mediante el comando *uname* con el *flag -n*:

Nodo virtual ast1:

```
ast1:/# hostname ast1

ast1:/# hostname
```

```
ast1

ast1:/# uname -n
ast1
```

Nodo virtual ast2:

```
ast2:/# hostname
ast2

ast2:/# uname -n
ast2
```

Una vez que disponemos de cada nodo con su correspondiente nombre de *host* correctamente asignado editamos el fichero */etc/hosts* e incluimos una línea por cada nodo. De esta forma conseguimos que los dos nodos puedan comunicarse usando para ello el nombre de *host* en lugar de su dirección de red, requisito necesario presentado por *Heartbeat*:

```
150.214.153.233 ast1

150.214.153.234 ast2
```

Fichero */etc/ha.d/authkeys*

A continuación comenzamos con la configuración de *Heartbeat* editando el fichero */etc/ha.d/authkeys*. Mediante este fichero la seguridad del *clúster* es configurada, ya que establecemos cómo vamos a realizar la autenticación entre nodos. Como los servidores anfitriones se encuentran directamente interconectados a través de un *switch Gigabit Ethernet* no son necesarias grandes medidas de seguridad por lo que es suficiente con aplicar el primero de los métodos vistos, *crc*:

```
auth 1
1 crc
```

Fichero */etc/ha.d/ha.cf*

Realizamos ahora la configuración de los aspectos principales de la operatividad de *Heartbeat* en el fichero *ha.cf*. Este fichero es muy importante ya que permite indicar a *Heartbeat* por ejemplo cada cuánto tiempo realizar la monitorización de los recursos y nodos o qué puerto e interfaz utilizar para sus comunicaciones.

A continuación podemos ver un el **contenido del fichero *ha.cf*** configurado de forma idéntica en los dos *nodos virtuales* (se han destacado las opciones habilitadas en negrita)**:**

```
#
#   There are lots of options in this file.  All you have to have is
a set
#   of nodes listed {"node ...} one of {serial, bcast, mcast, or
ucast},
#   and a value for "auto_failback".
#
#   ATTENTION: As the configuration file is read line by line,
#              THE ORDER OF DIRECTIVE MATTERS!
#
```

```
#   In particular, make sure that the udpport, serial baud rate
#   etc. are set before the heartbeat media are defined!
#   debug and log file directives go into effect when they
#   are encountered.
#
#   All will be fine if you keep them ordered as in this example.
#
#
#       Note on logging:
#       If all of debugfile, logfile and logfacility are not
defined,
#       logging is the same as use_logd yes. In other case, they are
#       respectively effective. if detering the logging to syslog,
#       logfacility must be "none".
#
#   File to write debug messages to
debugfile /var/log/ha-debug
#
#
#   File to write other messages to
#
logfile   /var/log/ha-log
#
#
#   Facility to use for syslog()/logger
#
logfacility    local0
#
#
#   A note on specifying "how long" times below...
#
#   The default time unit is seconds
#         10 means ten seconds
#
#   You can also specify them in milliseconds
#         1500ms means 1.5 seconds
#
#
#   keepalive: how long between heartbeats?
#
keepalive 2
#
#   deadtime: how long-to-declare-host-dead?
#
#         If you set this too low you will get the problematic
#         split-brain (or cluster partition) problem.
#         See the FAQ for how to use warntime to tune deadtime.
#
deadtime 30
#
#   warntime: how long before issuing "late heartbeat" warning?
#   See the FAQ for how to use warntime to tune deadtime.
#
warntime 10
#
#
#   Very first dead time (initdead)
#
#   On some machines/OSes, etc. the network takes a while to come up
#   and start working right after you've been rebooted.  As a result
#   we have a separate dead time for when things first come up.
#   It should be at least twice the normal dead time.
#
initdead 30
#
#
#   What UDP port to use for bcast/ucast communication?
#
udpport   694
```

```
#
#   Baud rate for serial ports...
#
#baud    19200
#
#   serial      serialportname ...
#serial  /dev/ttyS0  # Linux
#serial  /dev/cuaa0  # FreeBSD
#serial /dev/cuad0      # FreeBSD 6.x
#serial  /dev/cua/a  # Solaris
#
#
#   What interfaces to broadcast heartbeats over?
#
bcast     eth0        # Linux
#bcast    eth1 eth2   # Linux
#bcast    le0         # Solaris
#bcast    le1 le2           # Solaris
#
#   Set up a multicast heartbeat medium
#   mcast [dev] [mcast group] [port] [ttl] [loop]
#
#   [dev]       device to send/rcv heartbeats on
#   [mcast group]     multicast group to join (class D multicast
address
#               224.0.0.0 - 239.255.255.255)
#   [port]            udp port to sendto/rcvfrom (set this value to
the
#               same value as "udpport" above)
#   [ttl]       the ttl value for outbound heartbeats.  this effects
#               how far the multicast packet will propagate.  (0-255)
#               Must be greater than zero.
#   [loop]            toggles loopback for outbound multicast
heartbeats.
#               if enabled, an outbound packet will be looped back
and
#               received by the interface it was sent on. (0 or 1)
#               Set this value to zero.
#
#
#mcast eth0 225.0.0.1 694 1 0
#
#   Set up a unicast / udp heartbeat medium
#   ucast [dev] [peer-ip-addr]
#
#   [dev]       device to send/rcv heartbeats on
#   [peer-ip-addr]    IP address of peer to send packets to
#
#ucast eth0 192.168.1.2
#
#
#   About boolean values...
#
#   Any of the following case-insensitive values will work for true:
#         true, on, yes, y, 1
#   Any of the following case-insensitive values will work for
false:
#         false, off, no, n, 0
#
#
#
#   auto_failback:  determines whether a resource will
#   automatically fail back to its "primary" node, or remain
#   on whatever node is serving it until that node fails, or
#   an administrator intervenes.
#
#   The possible values for auto_failback are:
#         on    - enable automatic failbacks
#         off   - disable automatic failbacks
#         legacy      - enable automatic failbacks in systems
```

```
#              where all nodes do not yet support
#              the auto_failback option.
#
#   auto_failback "on" and "off" are backwards compatible with the
old
#         "nice_failback on" setting.
#
#   See the FAQ for information on how to convert
#         from "legacy" to "on" without a flash cut.
#         (i.e., using a "rolling upgrade" process)
#
#   The default value for auto_failback is "legacy", which
#   will issue a warning at startup.  So, make sure you put
#   an auto_failback directive in your ha.cf file.
#   (note: auto_failback can be any boolean or "legacy")
#
auto_failback off
#
#
#       Basic STONITH support
#       Using this directive assumes that there is one stonith
#       device in the cluster.  Parameters to this device are
#       read from a configuration file. The format of this line is:
#
#         stonith <stonith_type> <configfile>
#
#       NOTE: it is up to you to maintain this file on each node in
the
#       cluster!
#
#stonith baytech /etc/ha.d/conf/stonith.baytech
#
#       STONITH support
#       You can configure multiple stonith devices using this
directive.
#       The format of the line is:
#         stonith_host <hostfrom> <stonith_type> <params...>
#         <hostfrom> is the machine the stonith device is attached
#              to or * to mean it is accessible from any host.
#         <stonith_type> is the type of stonith device (a list of
#              supported drives is in /usr/lib/stonith.)
#         <params...> are driver specific parameters.  To see the
#              format for a particular device, run:
#           stonith -l -t <stonith_type>
#
#
#   Note that if you put your stonith device access information in
#   here, and you make this file publically readable, you're asking
#   for a denial of service attack ;-)
#
#   To get a list of supported stonith devices, run
#         stonith -L
#   For detailed information on which stonith devices are supported
#   and their detailed configuration options, run this command:
#         stonith -h
#
#stonith_host *     baytech 10.0.0.3 mylogin mysecretpassword
#stonith_host ken3  rps10 /dev/ttyS1 kathy 0
#stonith_host kathy rps10 /dev/ttyS1 ken3 0
#
#   Watchdog is the watchdog timer.  If our own heart doesn't beat
for
#   a minute, then our machine will reboot.
#   NOTE: If you are using the software watchdog, you very likely
#   wish to load the module with the parameter "nowayout=0" or
#   compile it without CONFIG_WATCHDOG_NOWAYOUT set. Otherwise even
#   an orderly shutdown of heartbeat will trigger a reboot, which is
#   very likely NOT what you want.
#
```

```
#watchdog /dev/watchdog
#
#   Tell what machines are in the cluster
#   node  nodename ...      -- must match uname -n
node     ast1
node     ast2
#
#   Less common options...
#
#   Treats 10.10.10.254 as a psuedo-cluster-member
#   Used together with ipfail below...
#   note: don't use a cluster node as ping node
#
#ping 10.10.10.254
#
#   Treats 10.10.10.254 and 10.10.10.253 as a psuedo-cluster-member
#       called group1. If either 10.10.10.254 or 10.10.10.253 are up
#       then group1 is up
#   Used together with ipfail below...
#
#ping_group group1 10.10.10.254 10.10.10.253
#
#   HBA ping derective for Fiber Channel
#   Treats fc-card-name as psudo-cluster-member
#   used with ipfail below ...
#
#   You can obtain HBAAPI from http://hbaapi.sourceforge.net.  You
need
#   to get the library specific to your HBA directly from the vender
#   To install HBAAPI stuff, all You need to do is to compile the
common
#   part you obtained from the sourceforge. This will produce
libHBAAPI.so
#   which you need to copy to /usr/lib. You need also copy hbaapi.h
to
#   /usr/include.
#
#   The fc-card-name is the name obtained from the hbaapitest
program
#   that is part of the hbaapi package. Running hbaapitest will
produce
#   a verbose output. One of the first line is similar to:
#       Apapter number 0 is named: qlogic-qla2200-0
#   Here fc-card-name is qlogic-qla2200-0.
#
#hbaping fc-card-name
#
#
#   Processes started and stopped with heartbeat.  Restarted unless
#       they exit with rc=100
#
#respawn userid /path/name/to/run
#respawn hacluster /usr/lib/heartbeat/ipfail
#
#   Access control for client api
#       default is no access
#
#apiauth client-name gid=gidlist uid=uidlist
#apiauth ipfail gid=haclient uid=hacluster

############################
#
#   Unusual options.
#
############################
#
#   hopfudge maximum hop count minus number of nodes in config
#hopfudge 1
#
#   deadping - dead time for ping nodes
```

```
#deadping 30
#
#   hbgenmethod - Heartbeat generation number creation method
#         Normally these are stored on disk and incremented as
needed.
#hbgenmethod time
#
#   realtime - enable/disable realtime execution (high priority,
etc.)
#         defaults to on
#realtime off
#
#   debug - set debug level
#         defaults to zero
#debug 1
#
#   API Authentication - replaces the fifo-permissions-based system
of the past
#
#
#   You can put a uid list and/or a gid list.
#   If you put both, then a process is authorized if it qualifies
under either
#   the uid list, or under the gid list.
#
#   The groupname "default" has special meaning.  If it is
specified, then
#   this will be used for authorizing groupless clients, and any
client groups
#   not otherwise specified.
#
#   There is a subtle exception to this.  "default" will never be
used in the
#   following cases (actual default auth directives noted in
brackets)
#          ipfail    (uid=HA_CCMUSER)
#          ccm             (uid=HA_CCMUSER)
#          ping            (gid=HA_APIGROUP)
#          cl_status (gid=HA_APIGROUP)
#
#   This is done to avoid creating a gaping security hole and
matches the most
#   likely desired configuration.
#
#apiauth ipfail uid=hacluster
#apiauth ccm uid=hacluster
#apiauth cms uid=hacluster
#apiauth ping gid=haclient uid=alanr,root
#apiauth default gid=haclient

#   message format in the wire, it can be classic or netstring,
#   default: classic
#msgfmt  classic/netstring

#   Do we use logging daemon?
#   If logging daemon is used, logfile/debugfile/logfacility in this
file
#   are not meaningful any longer. You should check the config file
for logging
#   daemon (the default is /etc/logd.cf)
#   more infomartion can be fould in the man page.
#   Setting use_logd to "yes" is recommended
#
# use_logd yes/no
#
#   the interval we  reconnect to logging daemon if the previous
connection failed
#   default: 60 seconds
#conn_logd_time 60
```

```
#
#
#   Configure compression module
#   It could be zlib or bz2, depending on whether u have the
corresponding
#   library     in the system.
#compression    bz2
#
#   Confiugre compression threshold
#   This value determines the threshold to compress a message,
#   e.g. if the threshold is 1, then any message with size greater
than 1 KB
#   will be compressed, the default is 2 (KB)
#compression_threshold 2
# Enable CRM (Pacemaker)
crm yes
```

Aunque hemos destacado en negrita las opciones habilitadas y configuradas para el comportamiento de *Heartbeat* quedan resumidas como vemos a continuación:

```
debugfile /var/log/ha-debug
logfile /var/log/ha-log
logfacility local0
keepalive 2
deadtime 30
warntime 10
initdead 30
udpport 694
bcast eth0
auto_failback off
node ast1
node ast2
crm yes
```

De esta forma determinamos que:

- El fichero para *log* de mensajes generados por *Heartbeat* en modo depuración se ubica en */var/log/ha-debug*.
- El fichero para *log* del resto de mensajes generados por *Heartbeat* se ubica en */var/log/ha-log*.
- El tiempo que debe transcurrir entre *ping* y *ping* hacia cada nodo del *clúster* para monitorización *hardware* es de dos segundos (*keepalive*).
- El tiempo que debe transcurrir para considerar un nodo del *clúster* como *muerto* o *no accesible* de 30 segundos (*deadtime*).
- El número de segundos que tienen que transcurrir antes de indiciar *último ping* de monitorización es de 10 (*warntime*).
- Antes de iniciar el servicio *Heartbeat* los *nodos virtuales* deben esperar por espacio de 30 segundos de forma obligatoria (*initdead*).
- El puerto de escucha para las *comunicaciones Heartbeat* es el 694.
- La interfaz de red usada para las *comunicaciones Heartbeat* es *eth0*. En ambos nodos es la misma; si en alguno de los nodos el nombre de la interfaz fuera diferente debemos editar el fichero haciéndolo corresponder con el nombre válido.

- Cuando un fallo es detectado en un nodo y los recursos son migrados a otro válido, una vez que los problemas son solucionados en el nodo original los recursos permanecen en el nuevo nodo (*auto_failback off*).

- El *clúster* se compone de dos nodos: *ast1* y *ast2*.

- Habilitamos el uso del *Cluster Resource Manager* o *crm* (*Pacemakerk)* para la administración de los recursos del *clúster*.

3.3.2 Configuración en uno de los nodos del *clúster virtual*

Una vez concluida la configuración común para los nodos del *clúster virtual* debemos pasar a establecer la configuración de la funcionalidad del *clúster*. Esto lo hacemos en uno sólo de los nodos, ya que como hemos citado anteriormente la configuración realizada mediante *Pacemaker* es distribuida y replicada entre los diferentes nodos una vez que es aceptada y guardada.

En el siguiente apartado vemos por tanto cómo definir los servicios/recursos que van a ser ejecutados en el *clúster virtual*, en qué nodos van a hacerlo, cómo va a ser implementada la monitorización de los mismos, qué comunicaciones van a tener lugar entre los nodos y bajo qué condiciones va a tener lugar la migración de los servicios. De manera resumida, las siguientes directrices nos permiten configurar la funcionalidad que queremos sea gestionada por *Heartbeat 3* y *Pacemaker*, teniendo en mente la arquitectura desarrollada, mostrada en la figura 6.13 *Clúster virtual de alta disponibilidad usando Xen, Asterisk y Heartbeat*:

- En nuestro *clúster virtual* contamos con **dos nodos: *ast1* y *ast2***. La definición y creación de ambos ha sido desarrollada en apartados anteriores. Van a ser los encargados de la ejecución de los servicios con alta disponibilidad.

- Debemos definir el **recurso *Asterisk***. Este recurso consiste en la ejecución del software *Asterisk* para el establecimiento de la centralita VoIP en el clúster. Debemos dotar a este recurso de alta disponibilidad, siendo ejecutado solamente en uno de los *nodos virtuales* del *clúster*.

- Se hace necesario disponer también de un **recurso del tipo *IP virtual***. Las *IPs virtuales* permiten identificar al servicio que es ofrecido a través de ellas (en nuestro caso *Asterisk*), sea cual sea su ubicación. Por ejemplo, si un nodo que está ejecutando *Asterisk* falla y no disponemos de una *IP virtual* migrar el servicio a otro nodo no sirve de nada ya que los clientes siguen intentando acceder al servicio a través de la dirección IP del nodo que ha fallado; es entonces cuando aparece la necesidad de utilizar una dirección *IP virtual*, tratada como un recurso más en el *clúster* y que es migrado igualmente entre los nodos, permitiendo que los servicios se encuentren siempre accesibles en la misma dirección.

- Otro requisito imprescindible es que **el recurso *Asterisk* y el recurso *IP virtual* deben conformar un grupo de recursos**. El recurso *Asterisk* precisa que la dirección *IP virtual* esté disponible para su funcionamiento, por lo que ambos recursos deben ejecutarse siempre en el mismo nodo. En definitiva, el grupo de recursos es tratada como un único recurso. Es el grupo de recursos el lugar en el que debemos definir las condiciones mediante las cuales los recursos son migrados de un nodo a otro.

- Del apartado anterior podemos concluir que es fundamental definir las **restricciones de localización, colocación y orden** para los recursos del *clúster virtual*. Es decir:

  - Los recursos (o más bien, el grupo de recursos) tiene preferencia para ejecutarse en el primero de los nodos del *clúster*, denominado *ast1*.

  - Los dos recursos deben ser ejecutados siempre en el mismo nodo; esto, como ya hemos dicho en el punto anterior, implica la creación del grupo de recursos.

  - Finalmente, los recursos del grupo de recursos deben ser iniciados y detenidos siempre en el mismo orden: primero la *IP virtual*, después *Asterisk*.

- También es necesario como vamos a ver en el siguiente apartado la definición de algunas variables globales para el *clúster virtual*, por ejemplo, para especificar el tipo de infraestructura para el *clúster*, las políticas *quorum* o la *pegajosidad* por defecto para los recursos (valores que permiten calcular la *adherencia* de un recurso para determinados nodos).

A continuación pasamos a detallar cómo podemos lograr establecer toda esta configuración utilizando *Pacemaker* sobre nuestro *clúster virtual* con *Heartbeat*.

Configuración del fichero *cib.xml* con *Pacemaker*

Antes de comenzar a trabajar es requisito previo **iniciar el servicio *Heartbeat*** en cada uno de los nodos del *clúster virtual* (*ast1* y *ast1*) de la siguiente forma, ejecutando el *script* disponible tras su instalación en el directorio */etc/init.d*:

```
ast1:~# /etc/init.d/heartbeat start
Starting High-Availability services: Done.

ast2:~# /etc/init.d/heartbeat start
Starting High-Availability services: Done.
```

Ahora que *Heartbeat* se encuentra en ejecución podemos iniciar la utilidad *CRM (Cluster Resource Manager)* de *Pacemaker*, para configurar todos los detalles de la funcionalidad para el *clúster virtual de alta disponibilidad*. Recordamos que a partir de ahora sólo debemos trabajar en uno de los nodos del clúster, ya que la configuración realizada es replicada en el otro nodo de forma automática. **Accedemos al intérprete *crm*** ejecutando el siguiente comando:

```
ast1:/# crm
crm(live)#
```

Como primer paso debemos crear el **fichero *cib shadow*** en el que vamos a almacenar la configuración del *clúster*. Esto lo hacemos ejecutando el comando *new* en el nivel *cib* y escribiendo el nombre que queremos dar al fichero, por ejemplo *HA_Asterisk*:

```
crm(live)# cib new HA_Asterisk
```

```
INFO: HA_Asterisk shadow CIB created
```

Con el fichero *cib shadow* creado debemos indicar a *crm* que queremos hacer uso de él de aquí en adelante, ejecutando el comando *use* también del nivel *cib* para elegirlo:

```
crm(HA_Asterisk)# cib use HA_Asterisk
```

Sentadas las bases para la configuración a realizar, comenzamos creando a continuación los recursos individuales sobre los que va a operar *Heartbeat* (dirección *IP virtual* y *Asterisk*). Ahora debemos bajar un nivel y operar en *configure*, para lo cual operamos así:

```
crm(HA_Asterisk)# configure
```

Con *crm* los recursos son credos mediante el comando ***primitive***. *Primitive* acepta como parámetros el nombre para el recurso, el agente de recurso que lo administra, los parámetros a su vez que éste precisa así como las diferentes operaciones que debe soportar. En la parte final además podemos definir el valor para algunas de las características del recurso, como por ejemplo su estado inicial. Para definir el **recurso *IP virtual*** lo hacemos como sigue:

```
crm(HA_Asterisk)configure# primitive failover-ip
ocf:heartbeat:IPaddr params ip=”150.215.153.235” op monitor
interval=”10s” meta target-role=”Started”
```

Donde:

- Damos el nombre *failover-ip* al recurso.
- Indicamos que el agente del recurso fue creado siguiendo el estándar *ocf:heartbeat*, y se llama *IPaddr* (en concreto este agente de recurso es proporcionado por el propio *Heartbeat* listo para poder ser usado en la configuración del *clúster* si lo estimamos conveniente).
- Como parámetro al agente del recurso añadimos la dirección de red que queremos establecer como *IP virtual:* 150.214.153.235. Recordamos que para los dominios *ast1* y *ast2* las direcciones de red configuradas fueron 150.214.153.233 y 150.214.153.234, respectivamente.
- Recordamos que un recurso debe poder soportar cuatro operaciones diferentes: ***start, stop, status*** **y** ***monitor***. Si no especificamos nada para ellas, *Heartbeat* toma los valores por defecto de los que dispone. En nuestro caso hemos considerado necesario concretar el intervalo de tiempo entre la ejecución de dos operaciones *monitor* en 10 segundos.
- Al final, mediante la etiqueta *meta* hemos indicado a *Hearbeat* que el estado inicial para el recurso debe ser *Started* (iniciado). Esto es, cada vez que el *clúster* es puesto en marcha el recurso es iniciado.

De forma similar vamos a definir el **recurso para el servicio *Asterisk***, con la importante salvedad de que *Heartbeat* no dispone de un *script* que actúe como *agente de recurso* para *Asterisk* de manera predeterminada. Podemos usar como agente de recurso el *script asterisk* de inicio y parada del servicio instalado con *Asterisk* y que se encuentra en el directorio */etc/init.d/*; en cambio, si observamos el contenido de este fichero, podemos ver que no contiene

una definición para las operaciones *monitor* y *status*, necesarias para que *Heartbeat* pueda monitorizar el estado del recurso. Así, finalmente hemos hecho uso de un *script* especial para *Asterisk* con las operaciones necesarias definidas, el cual hemos copiado en el directorio */etc/init.d/* para sustituir al existente (previamente creamos una copia de seguridad del mismo):

```
ast1:/etc/init.d# mv asterisk asterisk.old

ast1:~# cp /root/asterisk /etc/init.d/asterisk
```

Para poder hacer uso de este *script* y las funciones *monitor* y *status* que contiene debemos crear un grupo de usuarios *asterisk* e instalar la utilidad para el rastreado de redes y servicios de red *nmap*:

```
# groupadd asterisk

# apt-get install nmap
```

También es necesario eliminar todas las referencias para iniciar el servicio *Asterisk* de forma automática cada vez que el sistema es puesto en marcha (ejecutando el *script asterisk*), ya que a partir de ahora va a ser *Heartbeat* el encargado de iniciar y detener este servicio. Para ello ejecutamos el siguiente comando:

```
ast1:~# update-rc.d -f asterisk remove
```

La definición del recurso *failover-asterisk* queda por tanto de la siguiente forma:

```
crm(HA_Asterisk)configure# primitive failover-asterisk lsb:asterisk
op monitor interval="15s" meta target-role="Started"
```

A diferencia de los parámetros vistos para el recurso *IP virtual*, ahora el agente del recurso es del tipo *lsb* y la operación *monitor* es llevada a cabo cada 15 segundos, es decir, *Heartbeat* comprueba cada 15 segundos que *Asterisk* está siendo ejecutado con normalidad: comprueba primera si existe el proceso; después comprueba si el puerto *SIP* 5060 se encuentra a la escucha; para terminar se asegura de que el *módulo SIP* está cargado.

Con los dos recursos individuales creados, procedemos después a **crear el grupo de recursos al que van a pertenecer**. Para crear un grupo de recursos utilizamos la cláusula ***group***, también en el nivel *configure* como *primitive*. Escribimos después de *group* el nombre que queremos asignar al grupo así como los nombres de los recursos que pertenecen a él, en el **orden en el que queremos que sean iniciados/detenidos** –así creamos la **restricción de orden** en la que la *IP virtual* debe estar disponible antes de que el servicio *Asterisk* sea iniciado-:

```
crm(HA_Asterisk)configure# group asterisk-group failover-ip
failover-asterisk
```

Creando el grupo de recursos *asterisk-group* también hemos establecido la **restricción de colocación** necesaria para nuestro escenario: **los recursos *IP virtual* y *asterisk* deben ejecutarse siempre en el mismo nodo**. Antes de proseguir configurando la monitorización del grupo de recursos por parte de *Heartbeat* vamos a establecer los valores para dos de las **propiedades globales del *clúster***: ***no-quorum-policy y stonith-enabled***. Con la primera podemos establecer la política *quorum* para el *clúster*, es decir, cómo debe actuar en el caso en que el problema *SplitBrain* sea detectado: *ignore* (ignorar). Mediante la segunda habilitamos/deshabilitamos el que un dispositivo adicional al *clúster* (*dispositivo stonith*) pueda apagar un reiniciar un nodo que ha experimentado dificultades; como no es el objetivo del proyecto establecer una configuración excesivamente compleja deshabilitamos esta propiedad

(con el valor *false*). Para modificar los valores de las propiedades del *clúster* usamos la palabra clave ***property***, también en el nivel *configure*:

```
crm(HA_Asterisk)configure# property no-quorum-policy=”ignore”
crm(HA_Asterisk)configure# property stonith-enabled=”false”
```

Durante la configuración del comportamiento del *clúster* mediante *crm* siempre tenemos la posibilidad de verificar que lo estamos haciendo bien y que no existen errores en los procedimientos y sintaxis. Para ello, podemos escribir ***verify***:

```
crm(HA_Asterisk)configure# verify
```

Si la ejecución de *verify* no devuelve nada, ningún problema ha sido detectado. También es bastante útil cada cierto tiempo observar de un vistazo el contenido del fichero *cib shadow* que estamos creando ejecutando ***show***:

```
crm(HA_Asterisk)configure# show
```

Por ejemplo, si ejecutamos *show* en este punto de la configuración realizada para el nuestro *clúster virtual*:

```
crm(HA_Asterisk)configure# show
node $id="4e3ccea1-959f-4c14-ae3d-7b2b14ec3972" ast2
node $id="9050b66f-fc08-40c9-b9e1-af53f0aa826f" ast1
primitive failover-asterisk lsb:asterisk \
        op monitor interval="15s" \
        meta target-role="Started"
primitive failover-ip ocf:heartbeat:IPaddr \
        params ip="150.214.153.235" \
        op monitor interval="10s" \
        meta target-role="Started"
group asterisk-group failover-ip failover-asterisk
property $id="cib-bootstrap-options" \
        dc-version="1.0.8-9881a7350d6182bae9e8e557cf20a3cc5dac3ee7" \
        cluster-infrastructure="Heartbeat" \
        no-quorum-policy="ignore" \
        stonith-enabled="false"
crm(HA_Asterisk)configure#
```

Figura 6.14. Clúster virtual de alta disponibilidad usando Xen, Asterisk y Heartbeat.

Podemos apreciar en la Figura 6.14 como además han sido incluidos en la configuración del fichero *cib shadow* los **nodos *virtuales*** que forman parte del *clúster*. Éstos han sido tomados de forma automática por *Pacemaker* al ser descritos en el **fichero *ha.cf***. Debemos continuar entonces con la configuración del *clúster* ya que aún no hemos especificado ninguna restricción de localización ni las condiciones bajo las cuales va a tener lugar la migración de los servicios de un nodo a otro en caso de fallo.

Añadimos ahora la **restricción de localización** para el grupo de recursos *asterisk-group*. Continuando en el nivel *configure*, usamos la palabra clave ***location*** utilizando la sintaxis siguiente:

```
location <id> <rsc> {node_pref|rules}

node_pref :: <score>: <node>

rules ::
```

```
rule [id_spec] [$role=<role>] <score>: <expression>
[rule [id_spec] [$role=<role>] <score>: <expression> ...]
```

Como podemos ver en primer lugar debemos escribir un identificador (nombre que queremos dar a la restricción), por ejemplo *run_asterisk_group*. Después especificamos el nombre del recurso para que queremos establecer la restricción, en este caso el nombre del grupo de recursos (*asterisk-group*). Para finalizar debemos escribir el nombre del nodo preferido para la ejecución del recurso junto con una puntuación o detallar una regla para determinar esta preferencia (las reglas son creadas con una puntuación y una expresión a evaluar). Si la preferencia es resuelta favorablemente entonces la puntuación es incrementada en el valor indicado.

Siguiendo esta notación, la restricción de localización para el grupo de recursos *asterisk-group* la definimos de la siguiente forma:

```
crm(HA_Asterisk)configure# location run_asterisk-group asterisk-
group rule 100: #uname eq ast1 rule 90: #uname eq ast2
```

Vamos a completar para terminar un poco más los detalles para los recursos y grupo de recursos, así como otras propiedades globales para el *clúster*. En este último aspecto, establecemos los valores para las **propiedades *default-resource-stickiness* y *default-resource-failure-stickiness***, con las que podemos establecer la *adherencia por defecto* de los recursos para ser ejecutados en el nodo actual o por el contrario ser migrados a otro nodo que ofrece unas condiciones mejores. Asignamos una puntuación positiva en *default-resource-stickiness* para premiar la correcta ejecución del grupo en un nodo y una puntuación negativa en *default-resource-failure-stickiness* para penalizar la ejecución del grupo en un nodo si han sido detectado fallos. Esto lo hacemos de nuevo utilizando *property*:

```
crm(HA_Asterisk)configure# property default-resource-stickiness=20

crm(HA_Asterisk)configure# property default-resource-failure-
stickiness=-20
```

De todos modos, y aunque estos valores son *por defecto* para cualquier recurso definido en el fichero para el *clúster*, añadimos estas propiedades con sus valores respectivos de forma local en el grupo de recursos *asterisk-group*. Para ello salimos del nivel *configure* ejecutando ***end***. Antes, es necesario que salvemos los cambios realizados hasta ahora en la configuración por lo que ejecutamos el comando ***commit***:

```
crm(HA_Asterisk)configure# commit
crm(HA_Asterisk)configure# end
```

Es necesario también comunicar los cambios realizados cuando guardamos al resto de nodos del *clúster*. Esto lo logramos con la ejecución del comando ***commit*** también, pero en el nivel ***cib*** y escribiendo además el nombre del fichero *cib shadow* que queremos distribuir:

```
crm(HA_Asterisk)# cib commit HA_Asterisk
INFO: commited 'HA_Asterisk' shadow CIB to the cluster
```

Finalmente establecemos los valores para las propiedades ***resource-stickiness* y *resource-failure-stickiness*** para el grupo de recursos cambiando al nivel ***resource*** y haciendo uso del comando ***meta*** de la siguiente forma:

```
crm(HA_Asterisk)resource# meta asterisk-group set resource-
stickiness 20

crm(HA_Asterisk)resource# meta asterisk-group set resource-failure-
stickiness -20
```

Como último paso debemos establecer los mecanismos por los cuales *Heartbeat* va a **determinar si es necesario migrar los servicios del nodo *activo* al otro o no**. Para ello vamos a concretar la definición de las operaciones de monitorización de cada uno de los recursos individuales que forman el grupo *asterisk-group* (*failover-ip* y *failover-asterisk*) y hacer uso de la propiedad ***migration-threshold***.

Tanto para *failover-ip* como para *failover-asterisk* vamos a establecer el valor de la propiedad ***timeout*** para la operación ***monitor***, ya que permite fijar el *timeout* o tiempo máximo permisible para la ejecución de la operación. Así *Heartbeat*, una vez que transcurre este tiempo sin haber culminado la operación correctamente, determina que algo debe ir mal en la ejecución/disponibilidad del recurso. Los *timeouts* para las operaciones deben ser al menos de la misma duración que las recomendaciones disponibles en *meta-data* (si establecemos un valor no recomendado somos avisados), aunque eso no quiere decir que para un recurso determinado sea suficiente. Un error común en la configuración de este tipo de *clústeres* es establecer valores demasiado pequeños para los *timeouts* de las operaciones.

La forma más cómoda de cambiar la configuración de la operación *monitor* una vez que ésta ya sido definida anteriormente es la de editar el fichero *cib shadow* directamente de forma manual. Para ello vamos al nivel ***configure*** y ejecutamos ***edit***. Entonces el fichero es abierto con el editor seleccionado por defecto (puede ser cambiado a través del nivel ***options***) y escribimos ***timeout="20s"*** para las operaciones *monitor* en los dos recursos que tenemos en nuestro *clúster*, como podemos observar en la Figura 6.15.

```
crm(HA_Asterisk)configure# edit
```

```
node $id="4e3ccea1-959f-4c14-ae3d-7b2b14ec3972" ast2
node $id="9050b66f-fc08-40c9-b9e1-af53f0aa826f" ast1
primitive failover-asterisk lsb:asterisk \
        op monitor interval="15s" timeout="20s" \
        meta target-role="Started"
primitive failover-ip ocf:heartbeat:IPaddr \
        params ip="150.214.153.235" \
        op monitor interval="10s" timeout="20s" \
        meta target-role="Started"
group asterisk-group failover-ip failover-asterisk \
        meta resource-stickiness="20" resource-failure-stickiness="-20" migratio
n-threshold="2"
location run_asterisk-group asterisk-group \
        rule $id="run_asterisk-group-rule" 100: #uname eq ast1 \
        rule $id="run_asterisk-group-rule-0" 90: #uname eq ast2
property $id="cib-bootstrap-options" \
        dc-version="1.0.8-9881a7350d6182bae9e8e557cf20a3cc5dac3ee7" \
        cluster-infrastructure="Heartbeat" \
        no-quorum-policy="ignore" \
        stonith-enabled="false" \
        default-resource-failure-stickiness="-20" \
        default-resource-stickiness="20" \
        last-lrm-refresh="1271706916"
"/tmp/tmpoyKDMm" [noeol] 22L, 976C
```

Figura 6.15. Edición del fichero cib shadow manualmente.

Con los *timeouts* establecidos para la monitorización de los dos recursos individualmente, vamos a cambiar la propiedad ***migration-threshold*** para el grupo de recursos. Esta propiedad permite fijar el número de fallos o errores que pueden tener lugar para el recurso o grupo de recursos en el que se aplica para el nodo *actual*. Una vez alcanzado este límite, el

recurso o grupo de recursos es migrado a otro nodo. La utilidad de esta propiedad es grande, ya que nos permite implementar un mecanismo para la migración de los servicios de una forma muy sencilla y eficiente. Establecemos este umbral por ejemplo en 1, es decir, con tan solo detectar un fallo para el grupo de recursos *asterisk-group* éste es migrado a otro nodo para continuar con su ejecución:

```
crm(HA_Asterisk)resource# meta asterisk-group set migration-
threshold 1
```

Junto a la propiedad *migration-threshold* es necesario introducir y conocer dos procedimientos que nos permiten en todo momento comprobar el número de fallos registrados por *Heartbeat* para un recurso o grupo de recursos y *limpiar* o *resetear* esta cuenta. Si queremos conocer el valor de esta cuenta debemos hacer uso del comando ***failcount*** en el nivel ***resource***, ya que su valor es referente a un recurso/grupo de recursos concreto y para un determinado nodo. Este comando es utilizado siguiendo la sintaxis:

```
failcount <rsc> set <node> <value>
failcount <rsc> delete <node>
failcount <rsc> show <node>
```

Es decir, escribimos *failcount* seguido del nombre del recurso/grupo de recursos, la operación a realizar sobre la cuenta (establecer su valor, eliminarla o mostrarla), el identificar del nodo y, en el caso de haber elegido *set* como operación, el valor a establecer. Por tanto, por ejemplo para conocer el número de fallos experimentados por el recurso *failover-asterisk* en el nodo *ast1* ejecutamos:

```
crm(live)# resource

crm(live)resource# failcount failover-asterisk show ast1

scope=status  name=fail-count-failover-asterisk value=0
```

Podemos ver que *value=0*, por tanto aún no ha sido registrado fallo alguno. Es tarea del administrador encontrar la causa del error que tuvo lugar y *resetear* el contador de fallo para el recurso en el nodo. Para *limpiar* el valor de la cuenta de fallos hacemos uso de esta misma sintaxis y la operación ***delete***, como podemos ver a continuación:

```
crm(live)resource# failcount failover-asterisk delete ast1
```

Otra de las utilidades que es interesante ejecutar para **conocer el estado del *clúster*, los nodos que lo componen, los recursos monitorizados, monitorizar el estado de la migración de los servicios, el valor de la cuenta de fallos y errores registrados, etc.,…** es **crm_mon**. Para ejecutar esta herramienta escribimos en la terminal ***crm_mon***, con el parámetro ***-1f*** para que el estado del *clúster* sea mostrado por pantalla justo al terminar la consulta:

```
# crm_mon -1f
```

En la Figura 6.16 podemos ver un ejemplo de ejecución de *crm_mon*.

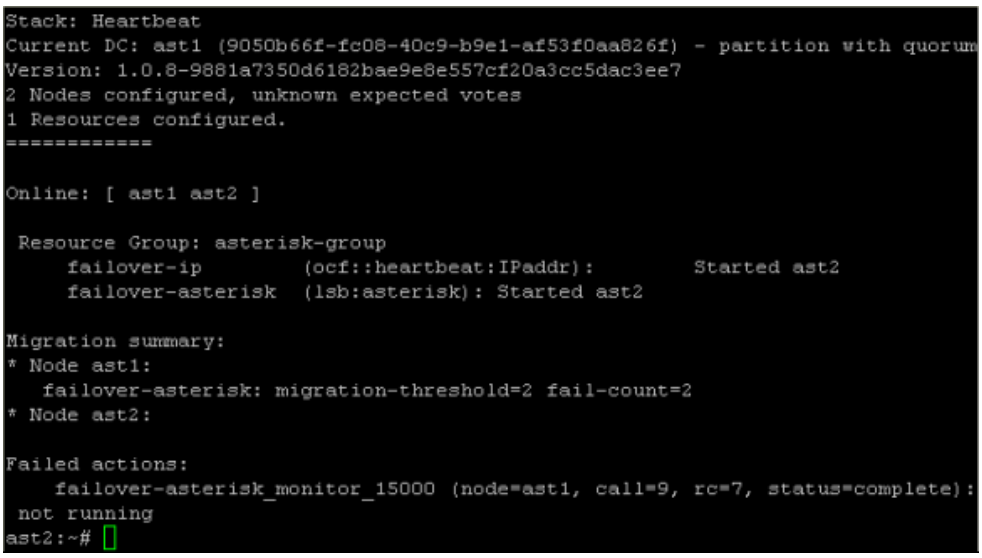


Figura 6.16. Salida mostrada con la ejecución de crm_mon.

Entre la información mostrada podemos ver:

- **Información general** como la arquitectura usada en el *clúster (Heartbeat)*, la versión del mismo, el número de nodos y recursos configurados así como el nodo que actúa como *DC (Designated Coordinator* o nodo principal *designado)* actual.

- Estado de los **nodos**: qué nodos se encuentran *online* y cuáles *offline*.

- **Recursos y grupos de recursos**: nombre, tipo, y en qué nodo se encuentran iniciados.

- Resumen de **migración**: para cada nodo y recurso muestra el umbral establecido para llevar a cabo la migración y el valor actual de la cuenta de fallos.

- **Acciones fallidas**: muestra la descripción del último fallo registrado.

Hasta aquí la configuración del *clúster virtual de alta disponibilidad para Asterisk*. Si cambiamos al nivel ***configure*** y ejecutamos nuevamente ***show*** podemos ver el contenido del fichero *cib shadow* generado con todos los pasos realizados (véase la Figura 6.17).

```
crm(live)configure# show
node $id="4e3ccea1-959f-4c14-ae3d-7b2b14ec3972" ast2
node $id="9050b66f-fc08-40c9-b9e1-af53f0aa826f" ast1
primitive failover-asterisk lsb:asterisk \
        op monitor interval="15s" timeout="20s" \
        meta target-role="Started"
primitive failover-ip ocf:heartbeat:IPaddr \
        params ip="150.214.153.235" \
        op monitor interval="10s" timeout="20s" \
        meta target-role="Started"
group asterisk-group failover-ip failover-asterisk \
        meta resource-stickiness="20" resource-failure-stickiness="-20" migratic
n-threshold="1"
location run_asterisk-group asterisk-group \
        rule $id="run_asterisk-group-rule" 100: #uname eq ast1 \
        rule $id="run_asterisk-group-rule-0" 90: #uname eq ast2
property $id="cib-bootstrap-options" \
        dc-version="1.0.8-9881a7350d6182bae9e8e557cf20a3cc5dac3ee7" \
        cluster-infrastructure="Heartbeat" \
        no-quorum-policy="ignore" \
        stonith-enabled="false" \
        default-resource-failure-stickiness="-20" \
        default-resource-stickiness="20" \
        last-lrm-refresh="1271706916"
crm(live)configure#
```

Figura 6.17. Configuración realizada.

Abandonamos el nivel *configure* guardando los cambios realizados en la configuración:

```
crm(HA_Asterisk)configure# commit
crm(HA_Asterisk)configure# end
```

Como último paso comunicamos los cambios en el fichero *cib shadow* al otro nodo del *clúster*, ejecutando el comando *commit* también pero en el nivel *cib*:

```
crm(HA_Asterisk)# cib commit HA_Asterisk
INFO: commited 'HA_Asterisk' shadow CIB to the cluster
```

Para salir de *crm* tenemos varias posibilidades, por ejemplo podemos escribir ***bye***:

```
crm(HA_Asterisk)# bye
bye
ast1:~#
```

A partir de ahora basta con **iniciar *Heartbeat* en todos los nodos del *clúster virtual* para poner en marcha nuestro sistema de alta disponibilidad para *Asterisk* ya configurado.** En el siguiente apartado vamos a ver cómo la configuración realizada ha sido puesta a prueba mediante diversos procedimientos provocando fallos en la ejecución de los servicios y permitiendo que éstos migren de un nodo a otro de forma transparente.

3.3.3 Puesta en marcha del nodo activo: dominio *ast1*

Para el correcto funcionamiento de la configuración realizada en el punto anterior fue necesario añadir un alias para la interfaz de red del nodo *activo* (*ast1*) del *clúster virtual* asignándole la dirección de red de la *IP virtual*. De esta forma garantizamos que pueda ser iniciado por primera vez el grupo de recursos de forma satisfactoria en el nodo *ast1*. Editamos por tanto el fichero */etc/network/interfaces* cuyo contenido final es el siguiente:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 150.214.153.233
netmask 255.255.255.0
gateway 150.214.153.1

auto eth0:0
iface eth0:0 inet static
address 150.214.153.235
netmask 255.255.255.0
gateway 150.214.153.1
```

3.4 PRUEBA DEL *CLUSTER VIRTUAL* DE ALTA DISPONIBILIDAD

Para probar la configuración establecida para el *clúster virtual de alta disponibilidad* hemos considerado a bien analizar el comportamiento del mismo antes tres sucesos diferentes que provocan la migración del grupo de recursos *asterisk-group* de un *nodo virtual* del clúster a otro: **apagado del nodo *activo*, caída del servicio *Asterisk* y saturación del servidor *Asterisk***

3.4.1 Apagado del nodo *activo*

Con el apagado del nodo *activo* en el que se encuentran actualmente el grupo de recursos en ejecución podemos simular algunas situaciones reales como la pérdida de suministro eléctrico en el servidor anfitrión en el que tenemos alojado el *nodo virtual* o un apagado del sistema forzado por motivos fundados en operaciones de mantenimiento (para instalación de software adicional en el dominio o anfitrión, actualizaciones, reparar o sustituir algún elemento hardware del anfitrión, etc.).

Veamos paso a paso como ha sido realizada esta prueba.

1. **Puesta en marcha de los dominios o *nodos virtuales*.** Ejecutamos en cada uno de los sistemas anfitriones la instrucción *xm create* para iniciar el nodo que corresponda: en el primer anfitrión (*deb1*) iniciamos *ast1* y en el segundo (*deb2*) *ast2*. Añadimos el flag *–c* para acceder directamente a su consola:

   ```
   deb1:/# xm create –c ast1_hb.cfg
   deb2:/# xm create –c ast2_hb.cfg
   ```

2. **Iniciamos *Heartbeat* en los dos *nodos virtuales*.** Lo hacemos prácticamente al mismo tiempo para evitar problemas en la sincronización inicial de los nodos del *clúster*. Ejecutamos el *script* de inicio tanto en *ast1* como en *ast2*:

   ```
   deb1:/# /etc/init.d/heartbeat start
   deb2:/# /etc/init.d/heartbeat start
   ```

3. Transcurrido el tiempo de inicialización del grupo de recursos en el nodo *ast1*, establecido en 30 segundos con el parámetro *initdead* en la configuración de *Heartbeat*, podemos **comprobar que efectivamente los recursos *IP virtual* y *Asterisk* han sido iniciados de forma correcta en el nodo predeterminado como principal *ast1***.

   En el caso de la dirección *IP virtual* lo hacemos ejecutando el comando ***ifconfig*** y observando que efectivamente el alias *eth0:0* con la dirección 150.214.153.235 ha

sido activado. En cuanto a *Asterisk*¸ es suficiente con conectarnos a la consola del servicio ejecutando ***asterisk –r***, apareciendo el *command line interfaces* de *Asterisk* (*CLI*). En las Figuras 6.18 y 6.19 podemos apreciar ambas comprobaciones: en la parte superior de las capturas disponemos de la monitorización de los *nodos virtuales* en los dos anfitriones (*deb1 y deb2*) en tiempo real mediante *xm top* y en la parte inferior tenemos las consolas de los dos *nodos virtuales*.

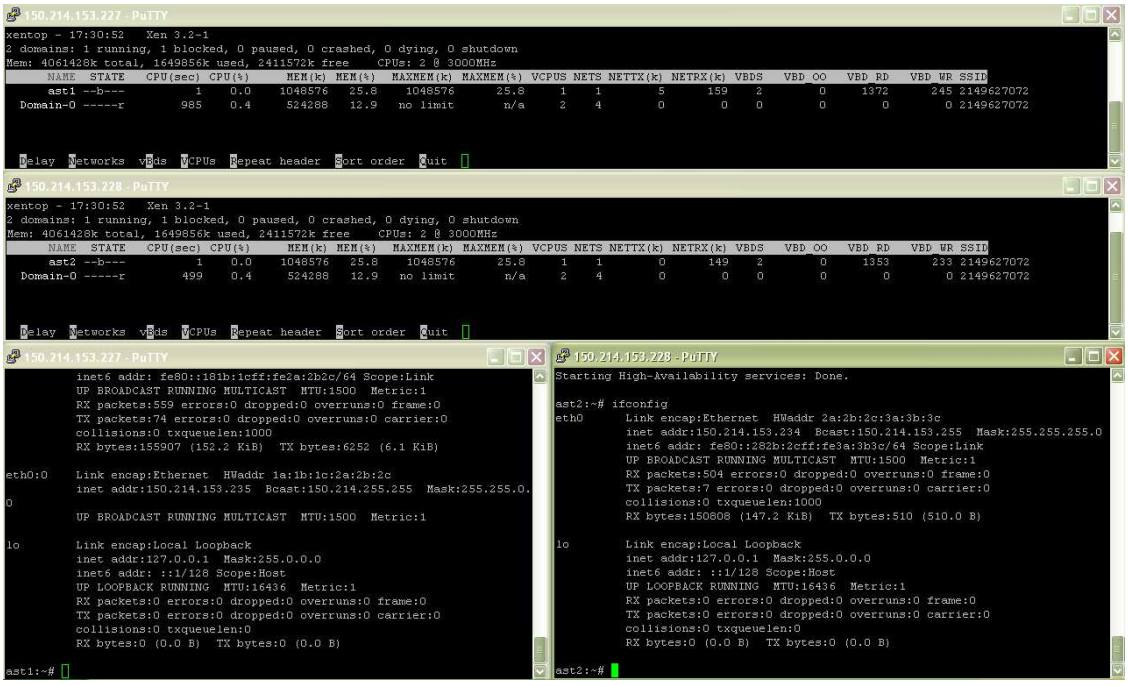


Figura 6.18. Ejecución de ifconfig en ast1 y ast2.

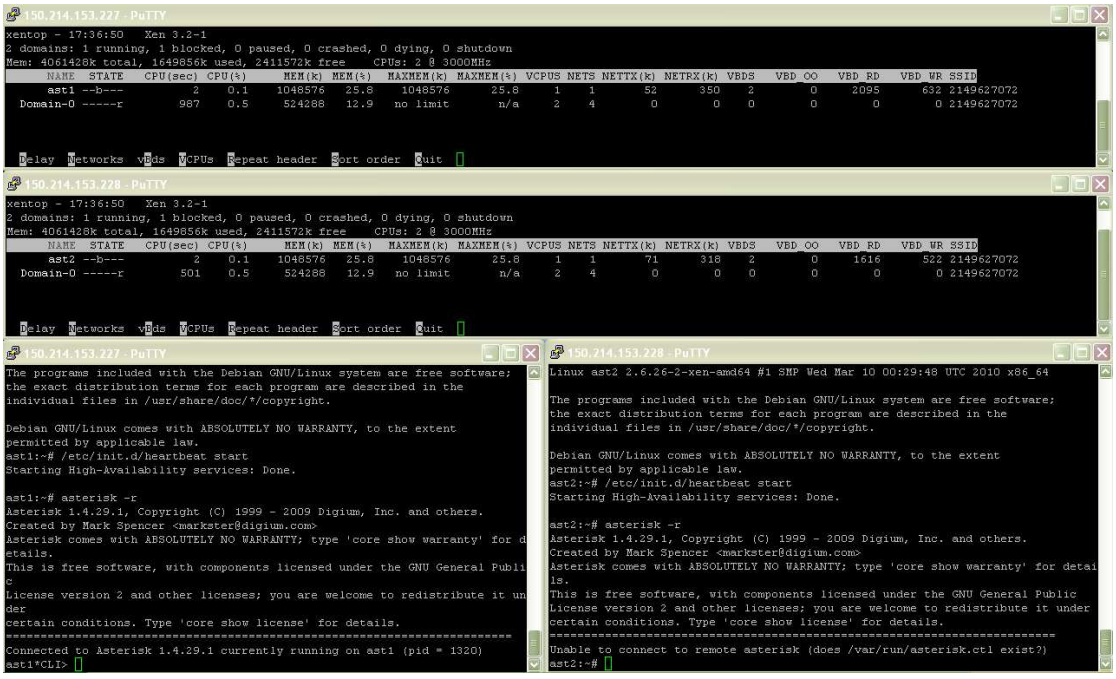


Figura 6.19. Ejecución de asterisk –r en ast1 y ast2.

En el nodo *ast2*, en cambio, disponemos solamente de la dirección de red 150.214.153.234 configurada (no la dirección *IP virtual*) y *Asterisk* no se encuentra accesible (el servicio no ha sido iniciado en este nodo).

También podemos realizar esta comprobación viendo el estado del *clúster* y toda la información relacionada que devuelve la ejecución de la herramienta ***crm_mon*** (véase la Figura 6.20). Ambos recursos, *failover-asterisk y failover-ip,* se encuentran iniciados en el nodo *ast1*; además, ningún fallo ha sido detectado hasta el momento, como cabe esperar ya que acabamos de iniciar el *clúster*.

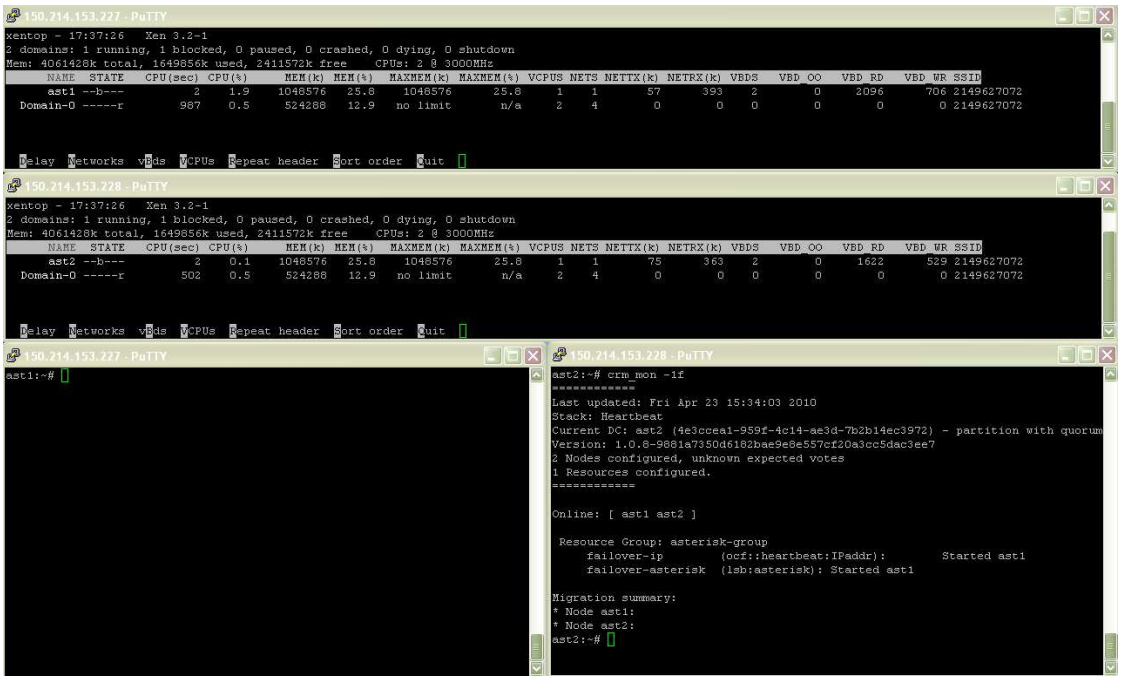


Figura 6.20. Comprobación del estado del clúster con crm_mon.

4. Procedemos con el **apagado del nodo *activo (ast1)* y posterior comprobación de la migración de los servicios**. Para apagarlo simplemente ejecutamos el comando ***poweroff***, simulando así las diferentes situaciones reales comentadas anteriormente. Al ejecutar el apagado del nodo *ast1 Heartbeat* reconoce que éste dejar de estar disponible (fallo en las comunicaciones de reconocimiento entre los dos nodos del *clúster*) y automáticamente inicia el grupo de recursos en el nodo *ast2*.

   Si volvemos a comprobar el estado del *clúster* ejecutando ***crm_mon*** tras el apagado podemos ver cómo ahora el nodo *ast1* es listado como *OFFLINE*, y los recursos *failover-ip* y *failover-asterisk* aparecen iniciados (*Started*) en el nodo *ast2* (ver la Figura 6.21).

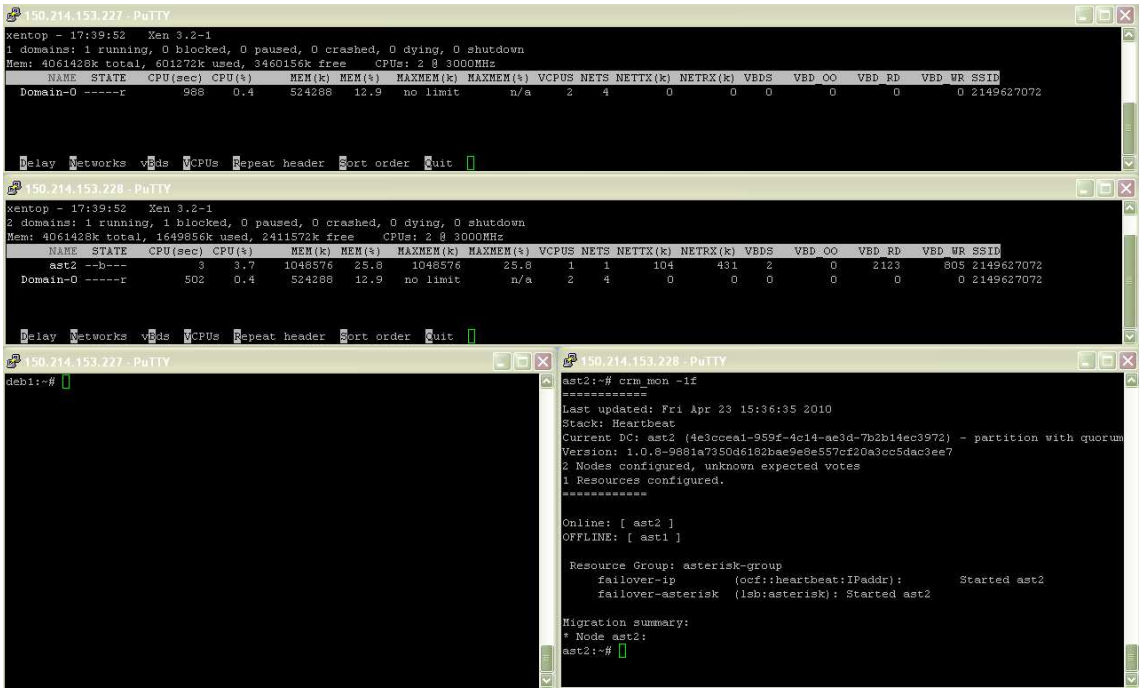


Figura 6.21. Comprobación del estado del clúster con crm_mon tras el apagado de ast1.

Si ejecutamos ahora tanto *ifconfig* como *asterisk –r* en el nodo *ast2* podemos comprobar fácilmente que la dirección de red para la *IP virtual* 150.214.153.235 ha sido *levantada* y que tenemos acceso a la interfaz de línea de comandos para *Asterisk* (éste se encuentra en correcto funcionamiento).

Así hemos podido comprobar que hemos dotado al servicio *Asterisk* en nuestra infraestructura virtual de alta disponibilidad con efectividad. En este caso, por parte de *Heartbeat*, no ha sido necesario tener en consideración el *umbral para migración* y los *timeouts* en las operaciones de monitorización establecidos para los recursos, ya que la detección del apagado del nodo ha sido motivo suficiente para llevar a cabo la inicialización de los servicios en el nodo *ast2* como respuesta. En las dos pruebas siguientes sí vamos a ver cómo intervienen estas características cuando sí es necesario.

3.4.2 Caída del servicio

En esta segunda prueba vamos a provocar la caída del servicio *Asterisk* y observar cómo *Heartbeat* reacciona ante este evento dependiendo del *umbral para migración* establecido para los recursos en los nodos. El inicio de la prueba es exactamente igual que en el caso anterior, poniendo en marcha los dos *nodos virtuales ast1 y ast2* en sus respectivos anfitriones e iniciando *Heartbeat*. De igual manera, el grupo de recursos es iniciado con éxito en el nodo definido como *primario*, *ast1*, como puede ser comprobando la ejecución del comando *ifconfig*, accediendo a la consolta *Asterisk* (con *asterisk -r*) o en el estado del *clúster* con *crm_mon.*

A diferencia de la configuración realizada anteriormente hemos establecido el valor para la propiedad ***migration-threshold*** del grupo de recursos *asterisk-group* en dos en lugar de uno (véase la Figura 6.22), para poder apreciar qué es lo que ocurre cuando tiene lugar un fallo y aún no es necesario iniciar los servicios en otro nodo (recordamos que *migration-threshold* permite indicar el número de fallos que deben registrarse para que un recurso/grupo de recursos sea migrado).

```
primitive failover-asterisk lsb:asterisk \
        op monitor interval="15s" timeout="20s" \
        meta target-role="Started"
primitive failover-ip ocf:heartbeat:IPaddr \
        params ip="150.214.153.235" \
        op monitor interval="10s" timeout="20s" \
        meta target-role="Started"
group asterisk-group failover-ip failover-asterisk \
        meta resource-stickiness="20" resource-failure-stickiness="-20" migrati
on-threshold="2"
location run_asterisk-group asterisk-group \
        rule $id="run_asterisk-group-rule" 100: #uname eq ast1 \
        rule $id="run_asterisk-group-rule-0" 90: #uname eq ast2
property $id="cib-bootstrap-options" \
        dc-version="1.0.8-9881a7350d6182bae9e8e557cf20a3cc5dac3ee7" \
        cluster-infrastructure="Heartbeat" \
        no-quorum-policy="ignore" \
        stonith-enabled="false" \
        default-resource-failure-stickiness="-20" \
        default-resource-stickiness="20" \
        last-lrm-refresh="1271708916"
crm(live)configure#
```

Figura 6.22. Establecemos el valor de migration-threshold a 2.

A continuación, para simular la caída del servicio vamos a pararlo manualmente haciendo uso del *script asterisk* en */etc/init.d/*:

```
deb1:/# /etc/init.d/asterisk stop
```

La caída de *Asterisk* es detectada por *Heartbeat* al consumirse el *timeout* asociado a la operación *monitor* del recurso *failover-asterisk*, aumentando en 1 como cabe esperar el *fail-count* asociado (pasa de ser 0 a 1). Después, como el límite de fallos no ha sido alcanzado, el recurso vuelve a ser iniciado en el mismo nodo. En la Figura 6.23 podemos ver el estado del *clúster* en este momento al ejecutar *crm_mon -1f*.

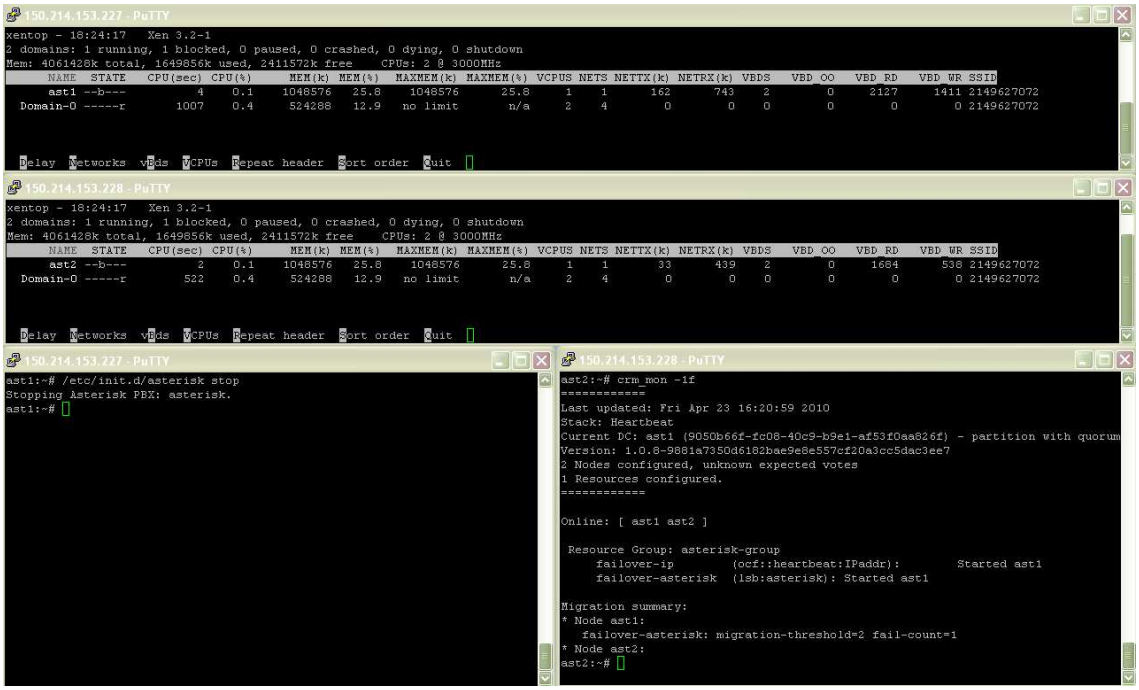


Figura 6.23. Estado del clúster tras un fallo.

Heartbeat ha permitido en este caso la completa recuperación del servicio pero en el mismo nodo en el que se encontraba en ejecución anteriormente debido a que la cuenta de fallos

no ha alcanzado el límite impuesto en la propiedad *migration-threshold* (2). Provocamos otro fallo actuando de la misma forma:

```
deb1:/# /etc/init.d/asterisk stop
```

Ahora, el *timeout* asociado a la operación *monitor* de *failover-asterisk* vuelve a consumirse y vuelve por tanto a aumentar en una unidad la cuenta de fallos, alcanzando el límite de 2. Ante esta situación, los servicios son migrados (iniciados) en el otro *nodo virtual* del *clúster*, como podemos ver en la siguiente Figura, que muestra el estado del *clúster* tras el procesamiento del segundo fallo.

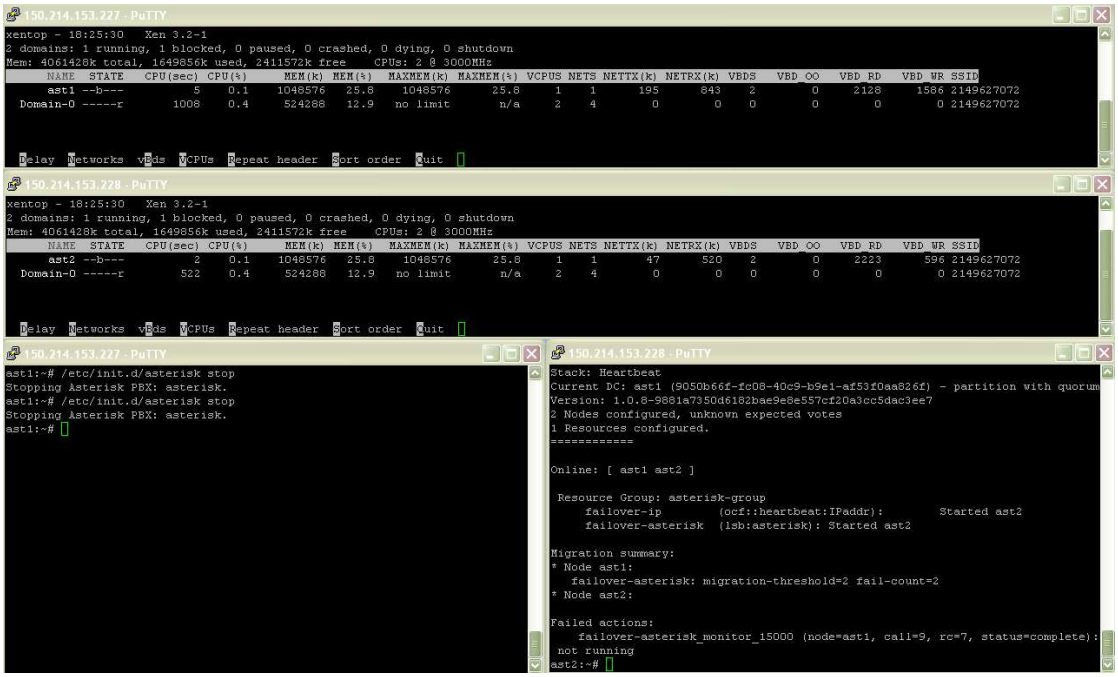


Figura 6.24. Estado del clúster tras el segundo fallo.

Además, podemos comprobar esto mismo ejecutando en *ast2* los comandos *ifconfig* (para ver que la dirección *IP virtual* ha sido *levantada*) y *asterisk –r* (para comprobar que tenemos acceso a la consola de *Asterisk*, que éste ha sido iniciado), como recogen las capturas de las Figuras 6.25 y 6.26.

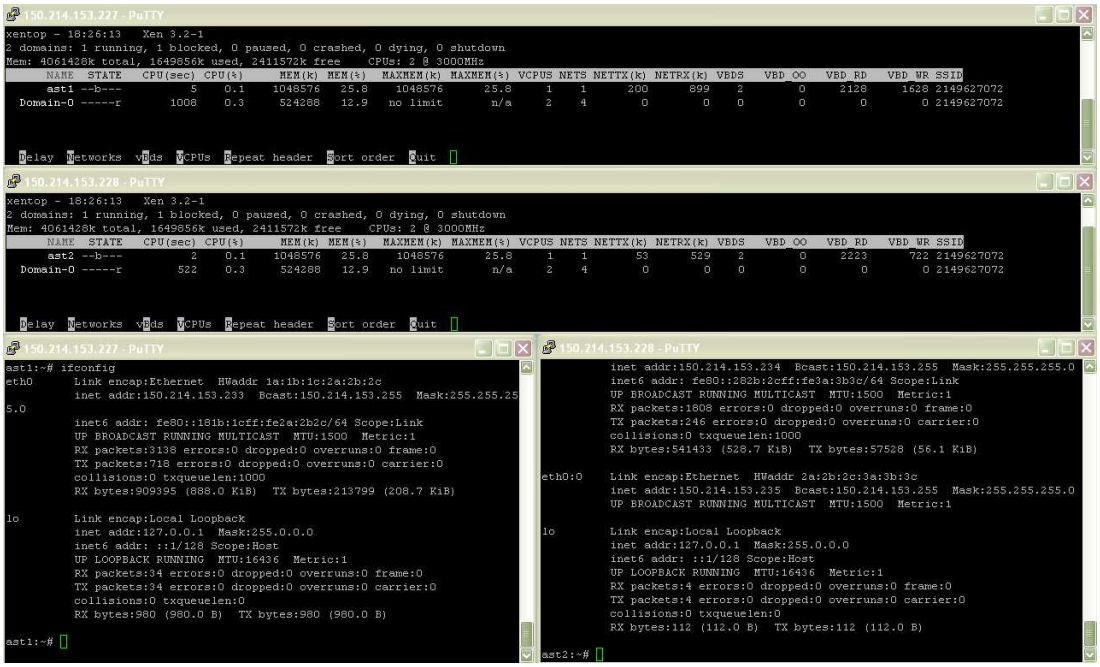


Figura 6.25. Ejecución de ifconfig para comprobar la ejecución del recurso failover-ip.

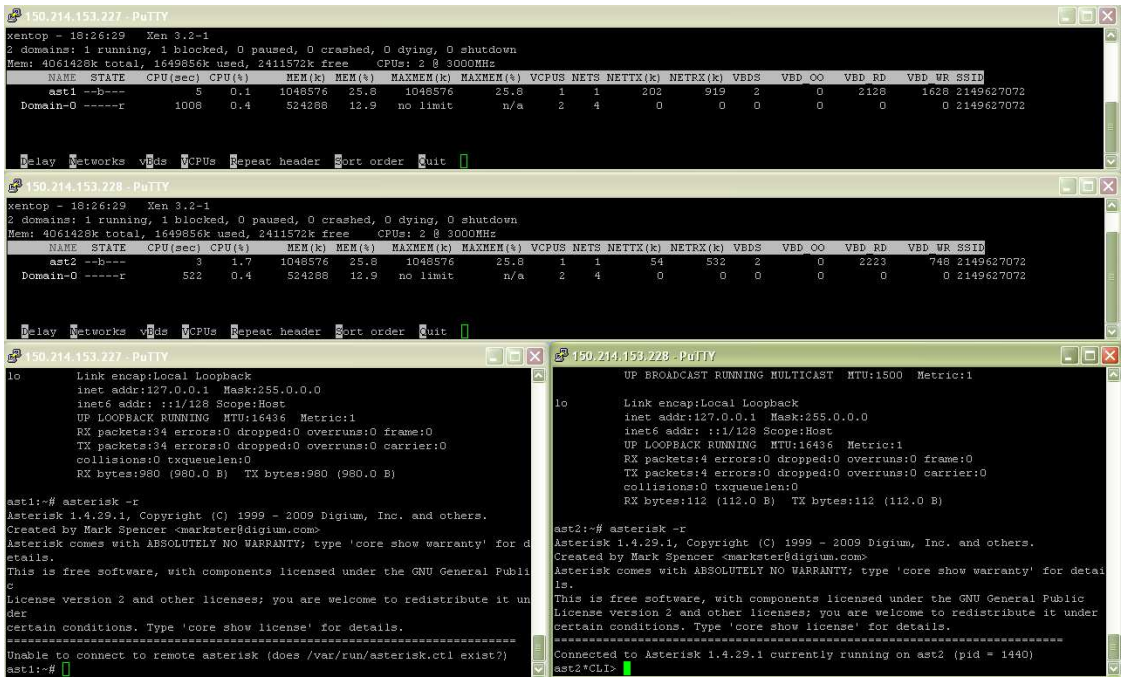


Figura 6.26. Ejecución de asterisk -r para comprobar la ejecución del recurso failover-asterisk.

Como fue citado antes es tarea del administrador ahora reiniciar la cuenta de fallos para el recurso *failover-asterisk* en el *nodo virtual ast1*:

```
crm(live)# resource
crm(live)resource# failcount failover-asterisk delete ast1
```

Nuevamente hemos comprobado que los servicios son iniciados de nuevo con éxito, bien en el mismo nodo en el que se encontraban ejecutándose si no fue alcanzado el límite de

fallos para el curso en ese nodo, o bien en otro nodo en caso contrario. En el siguiente apartado comprobamos esto mismo pero provocando la caída de *Asterisk* mediante saturación.

3.4.3 Saturación del servicio

Para esta tercera prueba la arquitectura empleada ha sido algo diferente, debido a la necesidad de un gran número de servidores para la ejecución de las pruebas de carga. La prueba de carga realizada sigue por completo los procedimientos expuestos en el capítulo anterior, haciendo uso del software *SIPp*. De esta forma, y sólo para esta última prueba, los dos *nodos virtuales* de nuestro *clúster* son integrados en un mismo servidor anfitrión. Además de éste, hacemos uso de un servidor que actúa como *servidor SIPp*, otro como *cliente SIPp*, y finalmente uno como servidor de apoyo para las pruebas tal y como fue descrito en el capítulo anterior. Otra diferencia con las pruebas anteriores es el número de fallos admisibles para llevar a cabo la iniciación de los servicios en el otro nodo; en este caso el valor de la propiedad *migration-threshold* ha sido fijado en 1 para el grupo de recursos *asterisk-group*, lo que significa que al notar el primer error en la ejecución de cualquiera de los recursos del grupo éste es iniciado en el segundo *nodo virtual* del *clúster*.

Los pasos seguidos de todas formas para el seguimiento de la prueba han sido los mismos que los realizados para las dos anterior. En primer lugar, iniciamos los dos *nodos virtuales* al tiempo que accedemos a sus respectivas consolas. Acto seguido iniciamos *Heartbeat* utilizando el *script* disponible en el directorio */etc/init.d/* (antes de ello debemos aumentar el límite de ficheros que el sistema operativo puede abrir con el comando *ulimit –n 1024* ya que de lo contrario las pruebas de carga con *SIPp* no pueden operar durante un tiempo comprensible).

Si transcurrido el tiempo fijado para el inicio de los recursos por parte de *Heartbeat* (*initdead*) ejecutamos los comandos *ifconfig* y *asterisk –r* en el nodo *ast1* podemos ver que efectivamente los recursos *IP virtual* y *asterisk* han sido iniciados de forma satisfactoria (en cambio, en *ast2* no lo han hecho como cabe esperar). En la Figura 6.27 podemos ver cómo se encuentra accesible la interfaz de línea de comandos *CLI* para *Asterisk* en *ast1*.

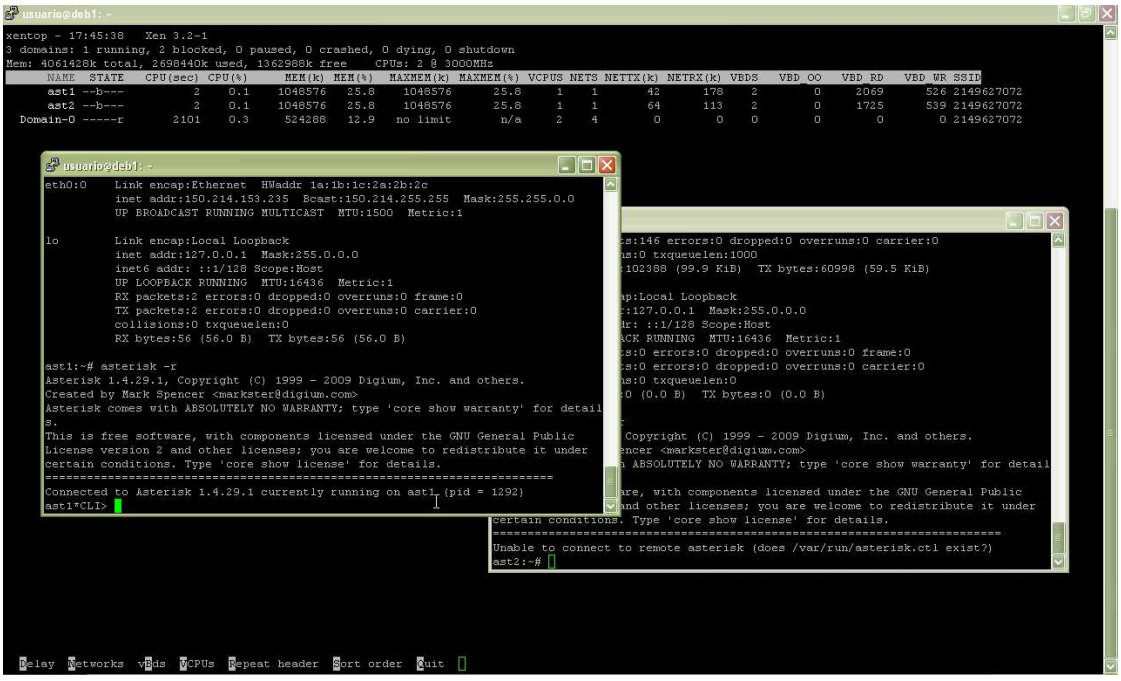


Figura 6.27. Acceso a la interfaz CLI de Asterisk en ast1.

Igualmente hemos comprobado mediante la ejecución de la utilidad *crm_mon* que los recursos se encuentran en funcionamiento en el nodo *ast1*, teniendo ambos *nodos virtuales online* y ningún fallo registrado pues acabamos de poner en funcionamiento el *clúster*.

Alcanzado este punto debemos realizar la **prueba de carga para saturar el servidor *Asterisk* en funcionamiento, esto es, en ejecución en *ast1* con *IP virtual* 150.214.153.235**. El comando ejecutado en el ***servidor SIPp*** de la prueba es el siguiente:

```
# ./sipp –sf serverg729.xml –nd –i 150.214.153.229 –mi
150.214.153.229 –rtp_echo
```

Como vimos en el capítulo anterior, de esta forma en el *servidor SIPp* es cargado el escenario de prueba *serverg729.xml* (procesando el *códec g729*), indicamos que va a recibir tráfico *SIPp* (*-i*) y tráfico *RTP* (*-mi*), además contestará con *echo* a éste último para la simulación de llamadas reales en las que dos interlocutores participan. En cuanto al *cliente SIPp* la ejecución es como sigue:

```
# ./sipp –sf clientg711.xml –nd –m 300 –r 3 –rp 1s –l 265 –s
9876543210 remote_host 150.214.153.235 –i 150.214.153.228
```

Así, el *cliente SIPp* va a generar un total de 265 llamadas incluyendo tráfico *RTP* a un ritmo de 3 llamadas por segundo. El máximo de llamadas simultáneas fijado es 300. Es importante apreciar que la dirección de red especificada para el servidor *Asterisk* es la de la dirección *IP virtual*. Además, también es de importancia ver que el *códec* utilizado por el *cliente SIPp* es *g711*, para forzar así al servidor *Asterisk* a realizar *trascoding* en el procesamiento de las llamadas lo que permite aumentar la carga y por tanto realizar la prueba con mayor rapidez.

Ejecutados los comandos *SIPp* en cliente y servidor, esperamos un tiempo hasta que el servidor *Asterisk* comienza a experimentar saturación, y aparecen mensajes *SIPp* retransmitidos o inesperados, teniendo un consumo de procesador de 100% durante bastantes marcas de tiempo (véase la Figura 6.28).

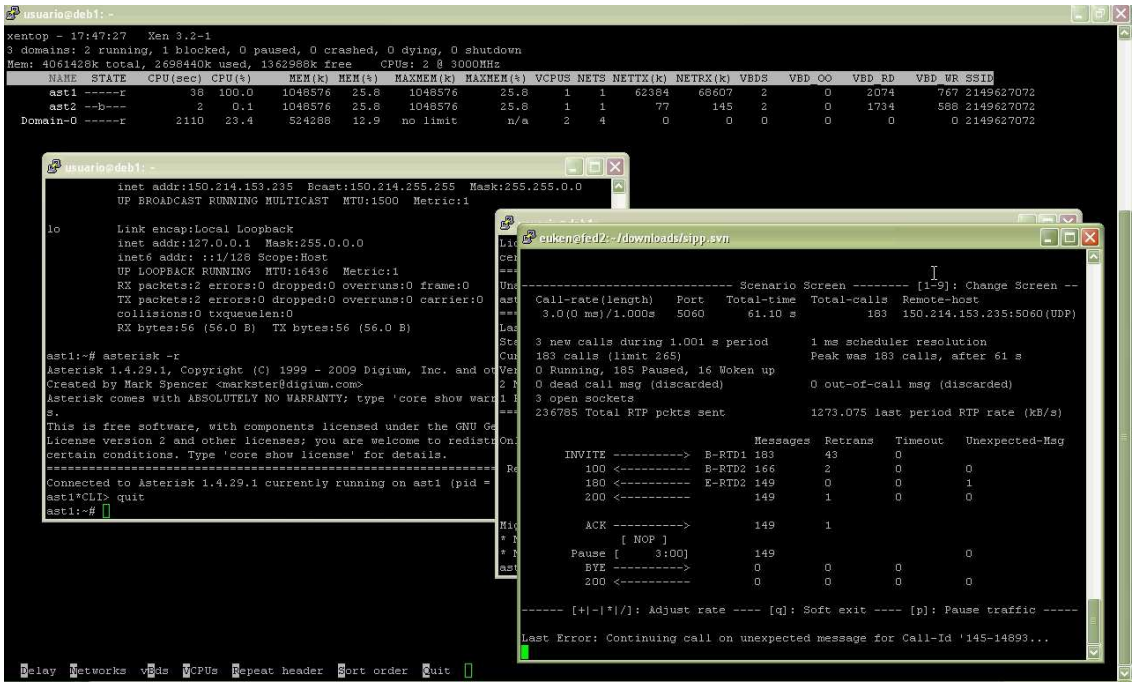


Figura 6.28. Saturación del servidor Asterisk en ast1.

Ante tal saturación en procesamiento *Heartbeat* no obtiene respuesta a la operación *monitor* sobre el recurso *failover-asterisk* en el tiempo máximo fijado para ello como *timeout* (20s), por lo que registra el fallo en la ejecución del mismo para *ast1* y aumenta la cuenta de fallos relacionada. El número de fallos es ahora 1, igual al valor de *migration-threshold* para *asterisk-group* en *ast1*, por lo que *Heartbeat* de forma automática detiene el servicio en *ast1* y lo inicia en *ast2*, además de la dirección *IP virtual* (si no se alcanza el valor de *migration-threshold* el servicio es reiniciado en *ast1*). Así, a partir de ahora las llamadas generadas en la prueba pueden continuar a través del nodo *ast2*. Esto lo podemos comprobar como lo hemos hecho hasta ahora (*ifconfig, asterisk –r, crm_mon -1f*). En la Figura 6.29 podemos observar el estado del *clúster* en este punto al ejecutar *crm_mon -1f*:

```
Stack: Heartbeat
Current DC: ast2 (4e3ccea1-959f-4c14-ae3d-7b2b14ec3972) - partition with quorum
Version: 1.0.8-9881a7350d6182bae9e8e557cf20a3cc5dac3ee7
2 Nodes configured, unknown expected votes
1 Resources configured.
============

Online: [ ast1 ast2 ]

 Resource Group: asterisk-group
     failover-ip        (ocf::heartbeat:IPaddr):        Started ast2
     failover-asterisk  (lsb:asterisk): Started ast2

Migration summary:
* Node ast2:
* Node ast1:
   failover-asterisk: migration-threshold=1 fail-count=1

Failed actions:
    failover-asterisk_monitor_15000 (node=ast1, call=6, rc=-2, status=Timed Out)
: unknown exec error
```

Figura 6.29. Estado del clúster tras iniciar el grupo de recursos en ast2.

Finalmente reiniciamos la cuenta de fallos para el recurso *failover-asterisk* en el *nodo virtual ast1* para que, en caso de fallo en el nodo *ast2*, el grupo de recursos pueda volver a *ast1*. Esto es muy importante ya que si el valor de *migration-threshold* es alcanzado en todos los nodos del *clúster* el grupo de recursos no puede ser ejecutado en ninguno de ellos:

```
crm(live)# resource
crm(live)resource# failcount failover-asterisk delete ast1
```

4 Migración de dominios Xen

En este apartado final vamos a configurar y realizar la migración de dominios Xen entre dos servidores anfitriones diferentes. Lo vamos a hacer tomando como referencia la arquitectura presentada en este mismo capítulo para la creación de un *clúster virtual de alta disponibilidad para Asterisk*, y que podemos recordar en la Figura 6.30. La migración de máquinas virtuales es el **proceso por el cual una máquina virtual y su estado pueden ser realojadas en un servidor físico diferente al que originalmente disponía de ellos.** Esto, dependiendo de la solución de virtualización que estemos usando puede ser llevado a cabo siguiendo diferentes técnicas y usando distintas utilidades (en nuestro caso vamos a ver cómo conseguirlo con Xen).

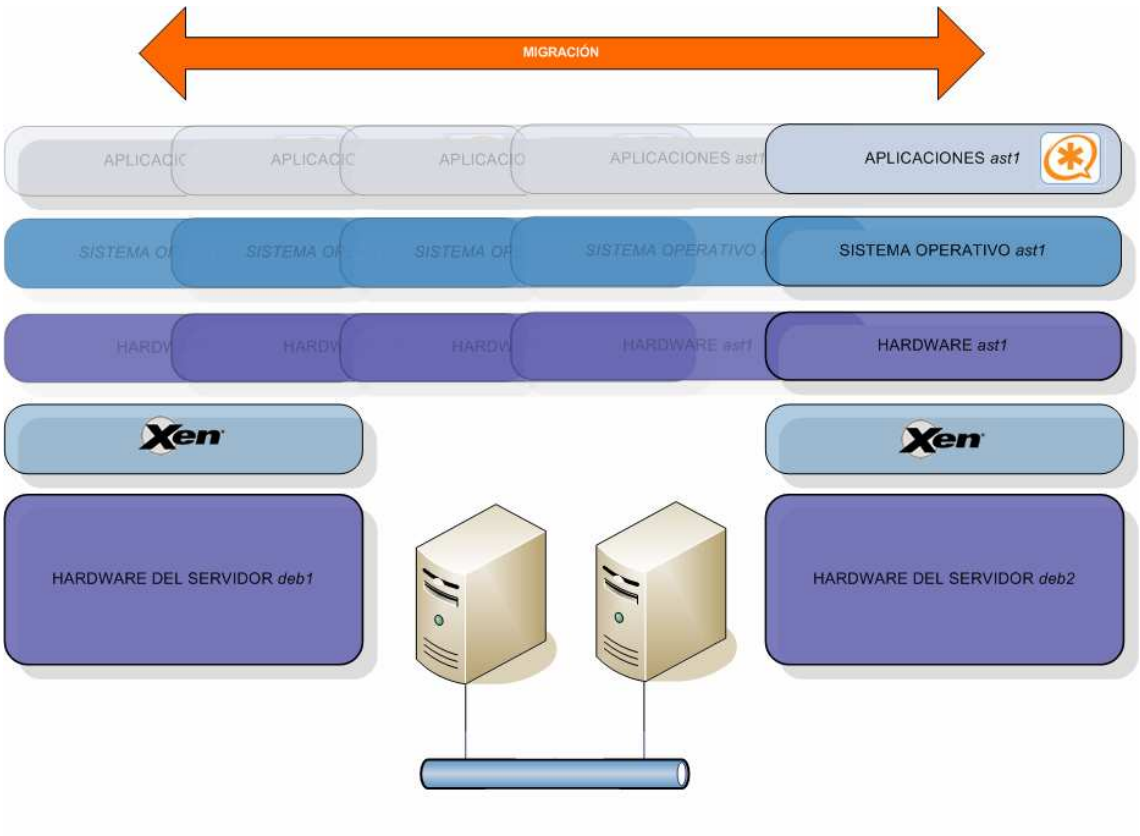


Figura 6.30. Arquitectura usada para la configuración y prueba de la migración de dominios Xen.

Como muestra la Figura anterior disponemos de dos servidores anfitriones (*deb1* y *deb2*), un dominio *ast1* a migrar y un almacenamiento compartido exportado mediante *NFS* por el primero de los anfitriones. En este almacenamiento compartido son alojados los dominios los cuales queremos tener la posibilidad de migrar (sus ficheros imágenes de disco así como los ficheros de configuración *.cfg* de los mismos). Los dos servidores anfitriones se encuentran conectados a través de un *switch Gigabit.*

La migración de máquinas virtuales fue introducida teóricamente en el capítulo *2 Virtualización de plataforma* (concretamente en el apartado *1.1.2. Migración de máquinas virtuales*), por lo que recordamos algunas de las características principales de cara a poder trabajar de forma práctica con dominios Xen. **Mediante su aplicación es posible obtener alta disponibilidad en nuestra infraestructura virtual: mejor aprovisionamiento de máquinas virtuales, balanceo de carga en los servidores anfitriones, recuperación efectiva ante desastres, etc.**

Al aplicar migración una nueva perspectiva sobre nuestra infraestructura informática aparece, pasando de ser totalmente estática a tener que pensar en ella de forma dinámica, por lo que **es posible que aparezcan algunas nuevas cuestiones y problemáticas**: cómo monitorizar las máquinas que pueden migrar, conocer de antemano los recursos disponibles en los servidores anfitriones que pueden alojarlas, cómo conocer en qué ubicación se encuentran… Normalmente, al aplicar migración en infraestructuras sencillas estos posibles problemas quedan absolutamente minimizados.

La migración de máquinas virtuales presenta **requisitos en cuanto a configuración, almacenamiento y red** que es necesario tener en cuenta antes de proceder con su realización, citados en el capítulo 2 de esta memoria. Comprobamos que son cumplidos en nuestra arquitectura planteada:

- **Los servidores físicos origen y destino ejecutan la misma solución de virtualización: Xen**.

- **El servidor destino del dominio dispone de suficientes recursos** de almacenamiento, memoria y procesamiento para su ejecución. No realiza operación alguna hasta que aloja al dominio migrado, por lo que este requisito es cumplido con creces.

- **Los servidores físicos origen y destino tienen la misma arquitectura y extensiones de virtualización: HP Proliant DL120 G5** (como podemos ver en la tabla 6.1 anterior). Evitamos así problemas de compatibilidad entre los juegos de instrucciones usados por librerías y sistemas operativos.

- En algunas configuraciones **es posible la necesidad de utilizar algún tipo de almacenamiento distribuido para los dominios y su configuración**. En nuestro caso vamos a realizar una **configuración simple para el almacenamiento de los dominios**: el servidor Xen anfitrión *origen* exporta un almacenamiento compartido mediante *NFS*, montado por el servidor Xen anfitrión *destino*, y que contiene los ficheros de disco y configuración de los dominios.

- **Es necesario habilitar explícitamente el soporte de este tipo de operaciones en Xen.** Cómo lograrlo es visto en la primera sección de este punto, *4.1 Configuración de Xen*.

Podemos encontrar infinidad de motivos para conocer e implementar técnicas de migración de dominios: aprovisionamiento de dominios de forma rápida y eficiente, para la realización de operaciones de mantenimiento o instalación de actualización/software adicional en uno de los servidores anfitriones, para balanceo de carga entre diferentes servidores anfitriones (siempre que haya recursos disponibles), por eficiencia energética, para recuperación ante desastres y dotar de alta disponibilidad a servidores y servicios, etc.

A continuación vemos cómo es necesario configurar los *hosts Xen* para soportar la migración de dominios; después exploramos tres técnicas diferentes para llevarlo a cabo, a elegir dependiendo de las circunstancias y necesidades para la migración: ***Save & Restore,*** **migración *en parada* y migración *en vivo*.**

4.1 CONFIGURACION DE XEN

La configuración de Xen para el soporte de las operaciones de migración deben ser realizada en todos los servidores anfitriones que van a participar en las mismas; en nuestro caso en ambos *hosts Xen deb1* y *deb2*.

Para permitir que tenga lugar la migración de dominios debemos **habilitar en el fichero de configuración */etc/xen/xend-config.sxp* para el demonio *xend* el servidor de *reubicación* (*relocation server*).** Por defecto, y debido a motivos de seguridad, *xend* no inicia este servidor, y sí el servidor para la administración de *sockets* en el dominio Unix (*xend-unix-server*) ya que es necesario para las comunicaciones de las diferentes interfaces administrativas (entre ellas *xm*) con *xend*.

Antes de realizar cualquier cambio en este fichero **es recomendable hacer una copia de seguridad** del mismo, para su posible recuperación en el supuesto de que experimentemos problemas. Editamos por tanto el fichero */etc/xen/xend-config.sxp* para **incluir la siguiente configuración**:

```
(xend-relocation-server yes)
(xend-relocation-port 8002)
(xend-relocation-address '')
(xend-relocation-hosts-allow '')
```

Explicamos el valor para cada uno de estas variables y opciones en la configuración de *xend*:

- ***xend-relocation-server.*** Permite habilitar el servidor de *reubicación*, requisito indispensable para la lograr la consecución de la migración. Al habilitarlo debemos configurar con el resto de opciones el puerto y dirección de red para la escucha así como los equipos que tienen permitido establecer este tipo de comunicaciones.

- ***xend-relocation-port.*** Puerto que usa *xend* para la interfaz de *reubicación* (lo habitual es que sea el 8002).

- ***xend-relocation-address.*** Dirección en la que *xend* está a la escucha para las conexiones con *sockets de reubicación.* Por defecto es la misma que la especificada en la opción *xend-address*, mientras que la cadena vacía (‘’) permite la escucha en cualquier dirección.

- ***xend-relocation-host-allow.*** Equipos que tienen permitido establecer comunicaciones para la migración de dominios en el puerto y dirección(es) especificadas con las opciones anteriores. Si escribimos cadena vacía (‘’) cualquier equipo tiene acceso siempre que cumpla con el resto de requisitos en la conexión.

Finalmente, una vez guardados los cambios que hemos hecho en el fichero de configuración de *xend* debemos reiniciar el mismo, ejecutando el *script* disponible en */etc/init.d/*:

```
# /etc/init.d/xend restart
```

4.2 CONFIGURACION DEL ALMACENAMIENTO COMPARTIDO

Como hemos comentado anteriormente vamos a configurar un espacio de almacenamiento en el *host Xen deb1* compartido mediante *NFS*; primero debemos configurar *NFS* como servidor y exportar el directorio que queremos compartir y después, en el *host Xen deb2*, debemos realizar la configuración necesaria para que actúe como cliente *NFS*. Si queremos además trabajar con nombres de equipo en lugar de con direcciones IP debemos configurar las correspondencias en el fichero */etc/hosts* para cada uno de los equipos que intervienen.

4.2.1 Configuración en el *host Xen deb1* (servidor *NFS*)

El servicio *NFS* suele encontrarse instalado de forma predeterminada en prácticamente todas las distribuciones GNU/Linux. De todas formas, y si disponemos de Debian 5 Lenny como es nuestro caso para ambos servidores anfitriones, podemos instalarlo de la siguiente forma:

```
# apt-get install nfs-kernel-server
```

Una vez instalado editamos el fichero ***/etc/exports*** que contiene una línea por cada directorio de nuestro sistema de ficheros que queremos exportar y compartir en red, añadiendo información sobre los equipos que tienen permitido el acceso a ese directorio y con qué permisos. Como queremos compartir el directorio */etc/xen* (que contiene todos los ficheros de las imágenes de disco y configuración para los dominios) al *host Xen deb2* (con dirección IP 150.214.153.228) lo hacemos de la siguiente forma:

```
/etc/xen  150.214.153.228(rw)
```

Como podemos ver, le otorgamos permiso de acceso para lectura y escritura. Iniciamos el servicio para que el directorio compartido comience a estar disponible:

```
# /etc/init.d/nfs-kernel-server start
```

4.2.2 Configuración en el *host Xen deb2* (cliente *NFS*)

En el segundo *host Xen* tan solo es necesario *montar* el directorio que está compartiendo el primero y al que tiene acceso al haber configurado el fichero */etc/exports*. Para comprobar esto último podemos ejecutar el comando ***showmount*** especificando la dirección IP de *deb1*:

```
# showmount –e 150.214.153.227
Export list for 150.214.153.227:
/etc/xen 150.214.153.228
```

Para montar el directorio compartido por *deb1 /etc/xen* como si se tratara de un directorio local en *deb2* ejecutamos el comando ***mount***, especificando el directorio destino del montaje:

```
# mount 150.214.153.227:/etc/xen /xen
```

A partir de ahora el hipervisor Xen disponible en *deb2* puede operar sobre los dominios almacenados en el directorio compartido como si fueran locales disponibles en su propio sistema de ficheros.

4.3 SAVE & RESTORE

El primer método que vamos a ver para llevar a cabo la migración de dominios en Xen es el primero que apareció como disponible en prácticamente en todas las soluciones de virtualización y está basado en la toma de imágenes sobre el estado de los dominios: **el estado actual del dominio en ejecución es guardado en un fichero en disco, creando una *imagen* o *snapshot* del mismo, el cual puede ser usado a posteriori en la misma o en cualquier otra localización** (servidor físico anfitrión) **para iniciar de nuevo el estado de ejecución del dominio**.

El tamaño para la imagen creada del estado del dominio es aproximadamente equivalente al tamaño de la memoria que estaba siendo usada en el momento de la toma de la imagen; esto es bastante importante si queremos distribuir la imagen a través de la red ya que el tiempo destinado a ello es mayor cuanto mayor es el tamaño de la imagen.

Para la toma y recuperación de imágenes o *snapshots* del estado de los dominios disponemos en Xen de los comandos ***xm save y xm restore***. Cuando estamos ejecutando un dominio el cual queremos migrar (o simplemente guardar su estado) a otro *host Xen* lo hacemos de la siguiente forma, escribiendo el identificador del dominio cuyo estado queremos salvar y un nombre para el fichero destino de la imagen:

```
# xm save ast1 ast1_23_04_10.chk
```

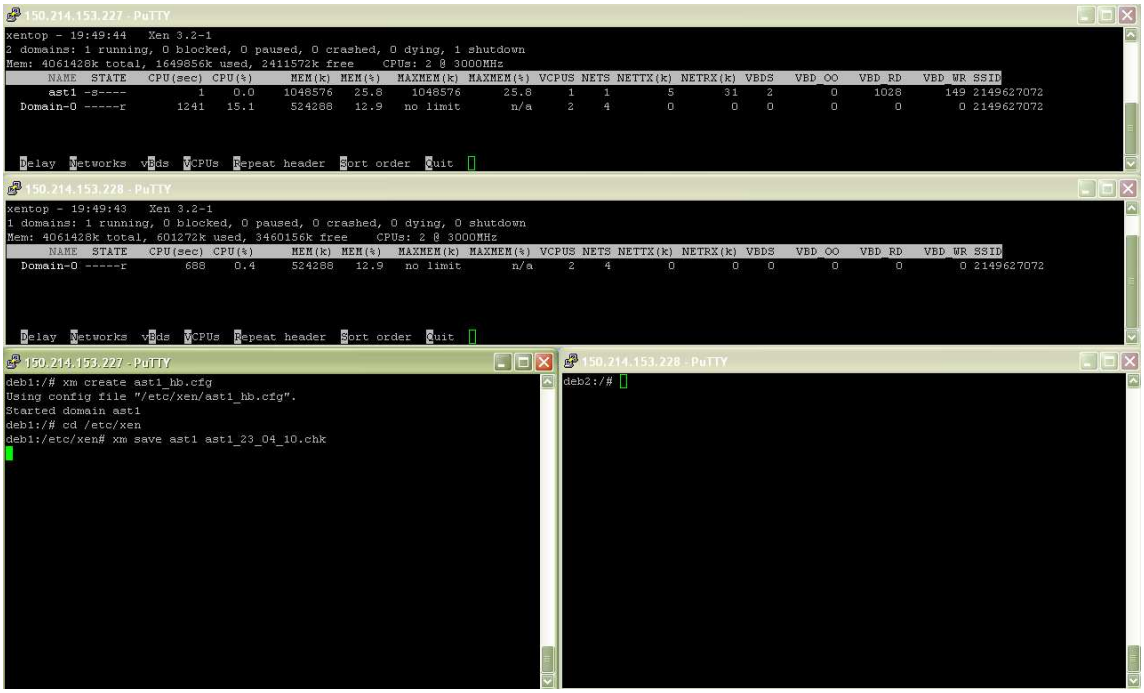


Figura 6.31. Guardando la imagen del dominio ast1 con xm save.

Una vez completada esta operación el dominio deja de estar disponible: **este tipo de migración proporciona alta disponibilidad en menor medida**; para que vuelva a hacerlo debemos restaurar su estado. Ello lo podemos hacer con *xm restore* tras copiar el fichero imagen en el segundo *host Xen deb2* o bien que éste lo tome a través del directorio compartido que ha montado (*/etc/xen* en */xen*). De cualquier forma logramos recuperar e iniciar el estado del dominio en una localización física diferente:

```
# xm restore as1_23_04_10.chk
```

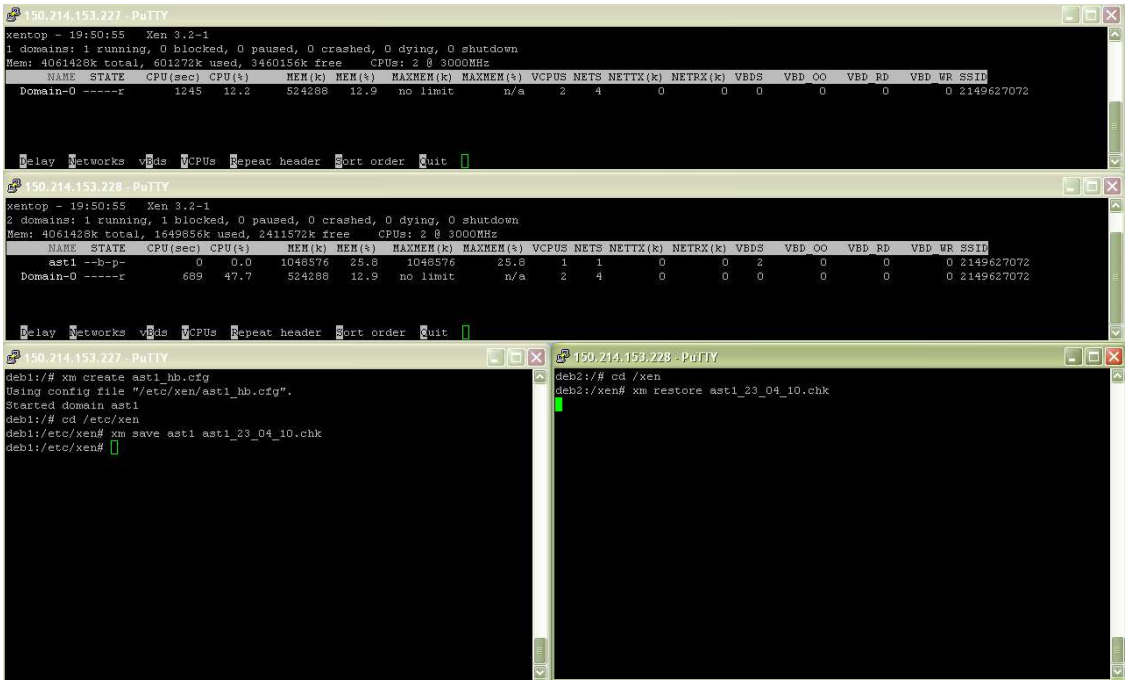


Figura 6.32 Restaurando la imagen del dominio ast1 en una localización distinta.

Cuando usamos este tipo de migración y restauramos una el estado de un dominio hay que tener muy presente el estado previo del dominio migrado y las aplicaciones que se estaban ejecutando y si crea conflictos con otros en el servidor destino, si hay recursos disponibles para su ejecución, su configuración de red, etc. Es recomendable también el cifrado de la imagen del dominio y su protección con algún tipo de clave así como los directorios que almacenan este tipo de ficheros: un ataque en el proceso de migración puede comprometer el estado al completo del dominio migrado y por tanto toda la información con la que este trabaja.

4.4 MIGRACION EN PARADA Y EN CALIENTE O EN VIVO

Como segunda opción para llevar a cabo tareas de migración vamos a hacer uso del comando proporcionado para Xen en la interfaz administrativa *xm (Xen Manager)*: ***xm migrate***. Como fue analizado en el capítulo *3 Xen*, mediante la ejecución de este comando podemos realizar la migración directa de un dominio a otro *host Xen*, siempre que se cumplan los requisitos citados en los puntos anteriores, proporcionando la impresión de ejecución con mínimo tiempo de caída para los servicios. La migración al nivel de un dominio completo significa que el estado de la memoria del mismo sea transferido a otro *host Xen* de una forma consistente y eficiente.

El comando *xm mígrate* admite dos parámetros o *flags* en su ejecución:

- ***-r, --resource.*** Permite fijar la cantidad máxima de megabytes permitidos para la migración del dominio. Mediante su uso es posible garantizar que los enlaces de red no se saturan debido al tráfico de la migración permitiendo que otro tráfico pueda seguir teniendo lugar.

- ***-l, --live.*** Permite usar ***migración en caliente o en vivo.*** Es el tipo de migración que debe ser **aplicada cuando los servicios migrados son de carácter crítico, ya que permite mantener lo máximo posible la alta disponibilidad y minimizar la interrupción de los mismos al no tener que detener el dominio en el proceso de migración.** El objetivo principal por el que es usada es que el usuario final

prácticamente no aprecie ningún efecto en el servicio que esté usando durante la migración y permitir a administradores llevar a cabo actividades de mantenimiento o actualización offline de servidores físicos sin que ello afecte a la disponibilidad del servicio.

Las diferencias entre usar *migración en parada* (sin el *flag -l*) o *migración en caliente* son evidentes. En el primer tipo la operación es realizada de forma análoga a ejecutar las operaciones *xm save* y *xm mígrate* de forma consecutiva: el dominio es parado completamente (estado *shutdown* en el ciclo de vida de un dominio) para cambiar el dominio de anfitrión. Para ejecutar este tipo de migración procedemos de la siguiente forma, indicando el identificador del dominio que queremos migrar y el *host Xen destino*:

```
# xm migrate ast1 deb2
```

En la Figura 6.33 podemos ver un ejemplo para este tipo de *migración en parada*. Durante la ejecución de la misma el dominio migrado aparece en ambos *hosts Xen* (*origen y destino*), aunque en el primero con estado *s-shutdown (apagado)* y en el segundo con *b-p-blocked-paused (bloqueado, sin nada en ejecución-en pausa).* Una vez concluida la migración el dominio migrado *ast1* desaparece del listado de dominios mostrado por *xm top* en el *host Xen origen deb1*.

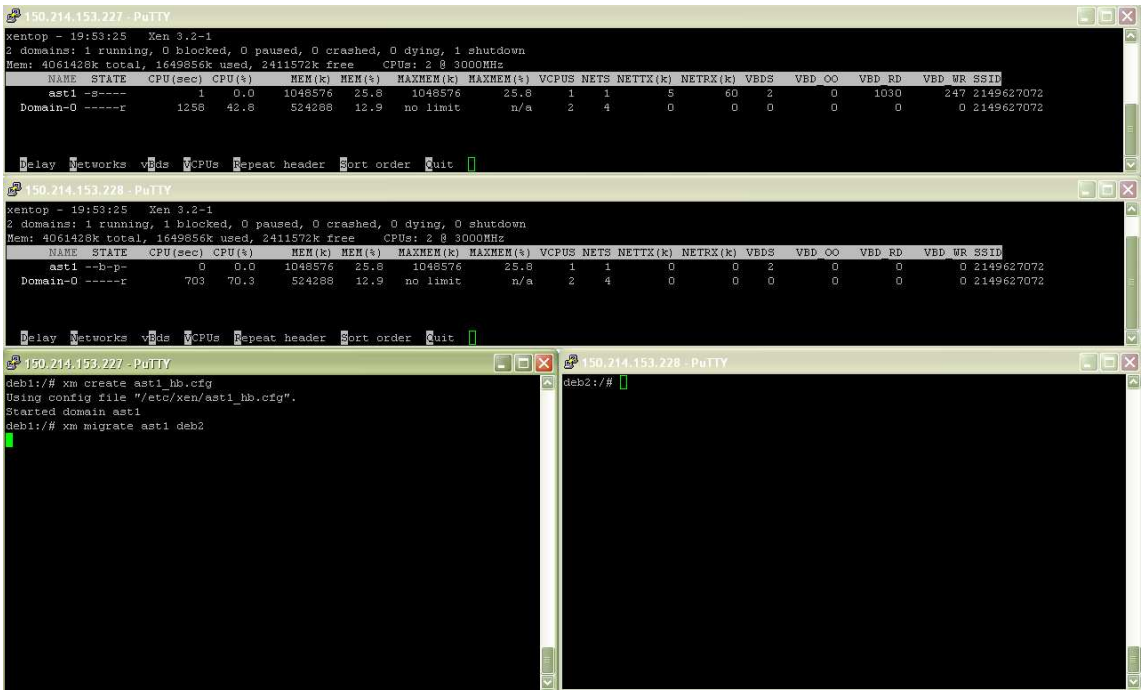


Figura 6.33 Ejecución de una migración en parada.

La ***migración en caliente o en vivo***, en cambio, consta de una serie de pasos diferentes, normalmente dependientes de la implementación realizada en la solución de virtualización. Por lo general **los pasos de los que suele constar son** (aunque de cara al administrador de la infraestructura virtual es en definitiva transparente: la migración queda encapsulada en la ejecución de un comando**): pre-migración, reserva, pre-copia iterativa, parada y copia, finalización y activación**.

En Xen ejecutar este tipo de migración hemos visto que es realmente sencillo, basta con incluir el flag *–l/--live* en la sentencia *xm migrate* anterior:

```
# xm migrate --live ast1 deb2
```

Si nos fijamos en la Figura 6.34 podemos ver las diferencias visibles en los estados por los que pasa el dominio durante el proceso de migración. A diferencia de la *migración en parada* vista antes, el dominio migrado nunca pasa por el estado *s-shutdown* en el *host Xen origen*, sino que queda en estado *b-blocked* (*bloqueado, sin nada en ejecución*). Culminada la migración, el dominio se encuentra disponible en el *host Xen destino*.

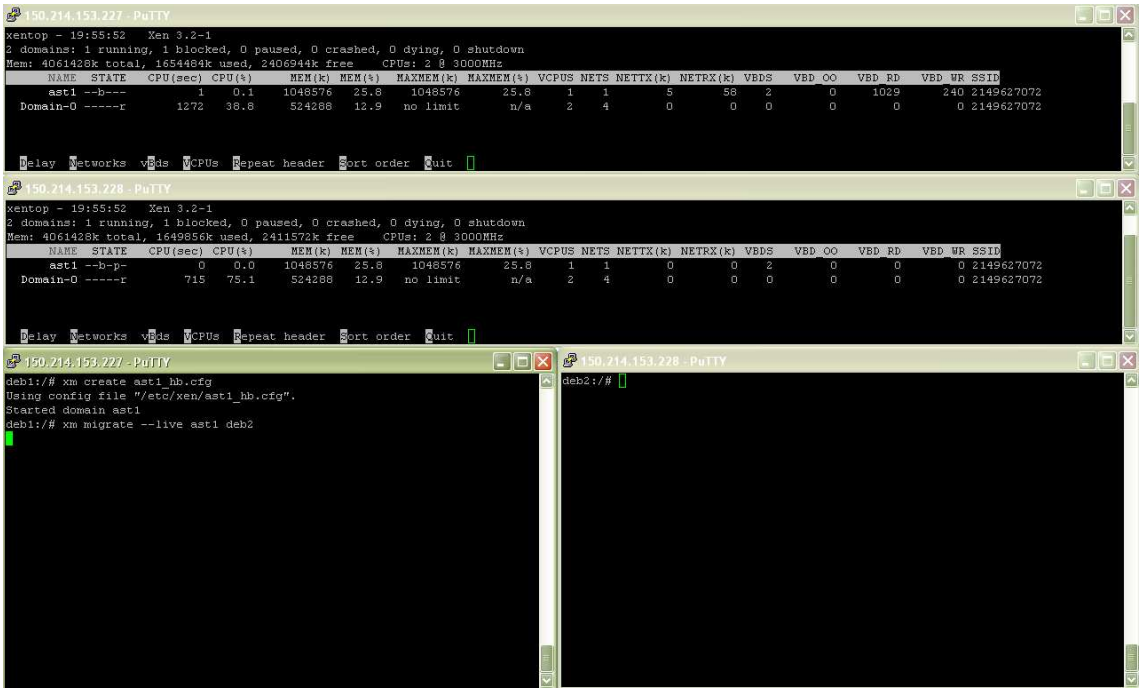


Figura 6.34 Ejecución de una migración en caliente o en vivo.

Los efectos y ventajas de poder aplicar *migración en vivo* son notables: mantenimiento de servidores físicos proactivo, **implementación de soluciones de alta disponibilidad**, **garantía de *SLAs***, balanceo de carga a través de múltiples servidores para optimizar el uso de recursos, mejora y reemplazo de hardware… La *migración en vivo* permite migrar un sistema operativo al completo junto a las aplicaciones y procesos que están siendo ejecutados sobre él como una unidad, resultando en definitiva una **herramienta potente para la administración de *clústeres***.

CONCLUSIONES FINALES

Con el desarrollo de este proyecto hemos podido conocer de primera mano el porqué del éxito de la novedosa aplicación de las tecnologías de virtualización (aunque con aproximadamente cuarenta años de existencia, con mayor o menor relevancia), concretamente de la paravirtualización como solución que permite consolidar servidores de forma eficiente, personificado en la solución libre Xen.

Aunque el trabajo realizado ha sido enfocado únicamente en la consolidación de servidores de telefonía IP con Xen como elemento virtualizador, las conclusiones generales que podemos extraer del mismo son extrapolables a la virtualización de prácticamente cualquier servicio, si es que ésta es posible.

Como hemos visto en la presentación del estado del arte realizado en cuanto a virtualización las características que su aplicación proporciona en la construcción de infraestructuras informáticas empresariales son las necesarias y demandadas para una solución de éxito: flexibilidad, escalabilidad, adaptabilidad y bajo coste fundamente:

- Una infraestructura virtual es enormemente flexible: los servidores físicos pasan a ser máquinas virtuales –entes lógicos- independientes, con configuración hardware y software propia. Esto posibilita su puesta en marcha de forma rápida y fácil mediante la creación de plantillas y su aprovisionamiento, o incluso la migración al completo del sistema entre diferentes servidores físicos anfitriones.

- La escalabilidad presentada es muy alta, siempre que los recursos físicos (memoria, procesamiento y almacenamiento fundamentalmente) lo permitan. Aún así, cuando éstos no son suficientes, adquirir un nuevo disco duro o memoria es más sencillo y económico que comprar un nuevo servidor.

- Adaptabilidad porque las infraestructuras virtuales pueden ser diseñadas para cualquier propósito. Existe multitud de soluciones de virtualización que pueden ser implementadas para encajar perfectamente en nuestras necesidades. Las máquinas virtuales son creadas con una configuración adaptable bajo la responsabilidad del administrador.

- El coste de implantanción y a largo plazo en cuestiones de administración, mantenimiento, escalabilidad, es muy bajo en comparación con el de una infraestructura informática completamente física. Comenzando por la disminución de equipos físicos, la centralización y simplificación de la administración,… si además usamos una solución de virtualización libre y gratuita, el ahorro es mayor, aumentando también la flexibilidad y adaptabilidad de la solución, perfectamente caracterizable en cualquier momento para cumplir nuestras pretensiones.

A parte de éstas, virtualizar puede proporcionar infinidad de ventajas tanto si se realiza *desde cero* (diseñando de forma directa la infraestructura como virtual) como si es utilizada para migrar nuestra infraestructura (parcial o totalmente) de servidores física: administración centralizada y simplificada, obtención de alta disponibilidad y mecanismos de recuperación ante desastres, alto rendimiento y redundancia, reducción de costes en hardware o consumo energético, mejora de las políticas de puesta en marcha, copias de seguridad y restauración, etc. También permite optimizar y controlar el uso de los recursos de memoria, red, almacenamiento, procesamiento,… mejorar la seguridad y aislamiento de servicios y servidores, proporciona mecanimos fiables para prueba y depuración de sistemas operativos, software, configuraciones,… Lógicamente, la presencia de una u otra de estas características depende de varios factores importantes, como la solución de virtualización implementada, los servicios y aplicaciones que configuramos en las máquinas virtuales, o los medios físicos disponibles, por ejemplo; en el caso de virtualización del servicio *Asterisk* sobre Xen los resultados son positivos ya que: se ha logrado obtener un mayor aprovechamiento de los recursos de los servidores integrando varias máquinas virtuales en ellos, hemos conocido herramientas que nos permiten agilizar y simplificar las actividades administrativas de la infraestructura, se han presentado configuraciones y esquemas que incluyen mejoras de seguridad y rendimiento permitiendo llevar a cabo actividades para obtener alta disponibilidad, migración de máquinas virtuales en caliente, etc.

Como acabamos de comentar, la virtualización por tanto permite resolver los problemas más preocupantes presentes en los *centros de procesado de datos* de empresas, universidades y organizaciones en la actualidad como: hardware en los servidores infrautilizado, agotamiento del espacio físico en los *data centers*, demanda de una mejor eficiencia energética, coste de la administración de sistemas y la necesidad de responder a fuertes requisitos de alta disponibilidad y rendimiento en servicios.

Aún así, siempre debemos tener presente que virtualizar puede provocar la aparición de algunos problemas o desventajas en su aplicación dependiendo de las características del proyecto, a los que anticiparnos es de vital importancia en la fase de análisis y diseño de la nueva infraestructura: pérdida de rendimiento, compartición del servidor, soporte hardware, riesgo en la organización al migrar los elementos físicos a virtuales, el anfitrión aparece como único punto de fallo, disponibilidad de recursos suficientes, dependencia tanto del sistema operativo anfitrión como de la solución de virtualización, posible aumento en la complejidad de la administración y/o actividades de *Networking*…. Es muy importante estudiar con detenimiento qué solución es la mejor para ser utilizada en las circunstancias que rodean a nuestro proyecto de virtualización; de todas formas, no es muy preocupante, ya que como hemos comentado el abanico de posibilidades de virtualización es enorme y permite encontrar una solución adecuada para cualquier necesidad. Más concretamente, Xen nos ha permitido solventar algunos de los problemas aparecidos estableciendo más de un servidor anfitrión en la infraestructura, reponiéndonos a la pérdida de rendimiento con soporte para migración y alta disponibilidad, o permitiéndonos usar su interfaz y herramientas para la automatización del proceso de creación, copias de seguridad, y restauración de máquinas virtuales.

En concreto, durante el desarrollo del proyecto, hemos puesto de manifiesto la conveniencia de la implantación de consolidación de servidores de telefonía IP. Las enormes ventajas que presenta el disponer de centralitas de telefonía software con *Asterisk* son evidentes:

bajo coste *software*, flexibilidad, grandes posibilidades y funcionalidades, precios sin competencia, calidad del servicio garantizada, integración con las redes de comunicaciones actuales, soporte multimedia, etc.

En lo que respecta a su implantanción en máquinas virtuales, no se ha detectado ninguna diferencia en el comportamiento de *Asterisk* en relación a su instalación y configuración habituales en equipos físicos. Como han demostrado las pruebas de rendimiento presentadas para hasta tres esquemas diferentes (real, virtual con las máquinas virtuales alojadas localmente y virtual con las máquinas virtuales alojadas en un sistema de ficheros en red), el uso de virtualización en telefonía IP implica una pequeña pérdida de rendimiento, en tanto que el número de llamadas simultáneas máximo soportable por la centralita es menor si fue configurada en una máquina virtual, como es por otra parte lógico debido a la introducción del nuevo nivel de abstracción que supone la virtualización. Comparando entre las dos configuraciones virtuales, el nivel de rendimiento obtenido ha sido prácticamente similar, aunque con valores ligeramente por encima en el caso que contempla máquinas virtuales administradas por los servidores anfitriones con Xen alojadas en un sistema de ficheros en red NFS, por lo que siempre que los recursos de red y almacenamiento lo permitan (en nuestro caso hemos tenido todos los servidores interconectados por red *Gigabit Ethernet*, almacenando las máquinas virtuales en un disco duro SAS) es recomendable su uso, estableciendo la ubicación de las máquinas de forma segura y centralizada en un servidor físico dedicado exclusivamente para ello.

Esta pérdida de rendimiento experimentada en entornos de virtualización ha sido subsanada gracias al establecimiento de mecanismos para la recuperación del servicio *Asterisk* en el caso de que este sea denegado, bien por sobrecarga, fallo hardware en el sistema anfitrión, fallo software en la máquina virtual, o cualquier otro motivo . Como sabemos, los mecanismos aplicados en este caso han sido la creación de clústeres de alta disponibilidad para *Asterisk* y la aplicación de técnicas de migración de dominios Xen, tanto en caliente como en parada, ya que uno u otro método puede ser conveniente dependiendo de las circunstancias. Por ejemplo, si queremos realizar operaciones de mantenimiento en un servidor físico que aloja en ese momento varios dominios con servicios críticos, podemos hacer la migración de los mismos a otro servidor físico en caliente, sin la necesidad de parar por completo la ejecución del dominio. Sin por el contrario esos servicios no son críticos o las condiciones lo permiten, podemos simplemente detener el dominio salvando su imagen, distribuirla a través de la red, y restaurarla finalmente en una ubicación diferente. La evolución de las técnicas de virtualización, además, se encuentra en la línea de mejorar todo lo posible este rendimiento obtenido, sobre todo al usar soporte hardware para las mismas en los procesadores Intel y AMD.

La aplicación de las técnicas de virtualización en clústeres es de vital importancia, aumentando más si cabe las ventajas que podemos obtener por ello. Es más patente en el caso de clústeres tipo *Beowulf*, como los vistos en el proyecto, ya que en ellos los nodos virtuales son idénticos en configuración hardware, por lo que los tiempos para aprovisionamiento y mantenimiento de los mismos son optimizados. Hemos comprobado que al establecer un clúster virtual de alta disponibilidad con *Heartbeat* para *Asterisk* podemos dotar a este servicio de la posibilidad de que, ante cualquier fallo o error que tuviera lugar en él, el servicio puede ser iniciado o restablecido de forma rápida y transparente en cualquiera de los otros nodos del clúster, garantizando así que los clientes pueden seguir usándolo sin prácticamente interrupción. El que esta característica haya sido implementada entre dominios Xen no ha supuesto ninguna desventaja ni diferencia respecto a su aplicación en entornos de clústeres físicos.

No podemos terminar este apartado final sin destacar el papel fundamental que ha jugado en el desarrollo al completo del proyecto que los sistemas operativos utilizados hayan sido GNU/Linux y la solución de virtualización y telefonía IP software libre. Su uso y aplicación ha permitido, como cabía esperar, brindar soluciones y esquemas prácticos de gran flexibilidad, potencial, y robustez.

APENDICE: BIBLIOGRAFIA Y URLs

A continuación es listada la documentación y páginas *Web* que han servido de referencia bibliográfica complementaria durante el desarrollo del proyecto.

Bibliografía

- Francisco José Méndez Cirera. *Diseño e implementación de un sistema VoIP de alta disponibilidad y alto rendimiento*. Universidad de Almería. 2008

- Tim Jones, Consultant Engineer, Emulex. *Virtual Linux, An Overview of virtualization methods, architectures, and implementations*. IBM. 2006

- David Santo Orcero. *Virtualización en GNU/Linux: Entrega 1 – La tecnología de virtualización y GNU/Linux*. Todo Linux, Número 101. Studio Press. 2009

- David Santo Orcero. *Virtualización en GNU/Linux: Entrega 2 – Xen: Instalación y configuración*. Todo Linux, Número 102. Studio Press. 2009

- David Santo Orcero. *Virtualización en GNU/Linux: Entrega 3 – Xen: Administración y Red*. Todo Linux, Número 103. Studio Press. 2009.

- David Santo Orcero. *Virtualización en GNU/Linux: Entrega 4 – QEMU: Instalación y configuración*. Todo Linux, Número 104. Studio Press. 2009

- Bernard Golden, Clark Scheffy. *Understand why virtualization is so important. Virtualization for Dummies.Sun and AMD Special Edition*. Wiley Publishing. 2008. ISBN: 978-0-470-29264-8

- Francisco Javier Zorraquino García, Jefe de Sistemas Informáticos, Ministerio de Economía y Hacienda. *Virtualización: máquina virtual*. Boletic Diciembre 2006 www. astic.es.

- David Chisnall. *The Definitive Guide to the Xen Hypervisor*. Prentice Hall. 2007. ISBN: 978-0-13-234971-0

- William von Hagen. *Professional Xen Virtualization*. Wrox, Wiley Publishing. 2008. ISBN: 978-0-470-13811-3

- Stephen Alan Herrod, VP of Technology VMWare, Presentation. *The future of Virtualization Technology*. 2008

- Prabhajar Chaganti. *Xen Virtualization: A Practical Handbook.* Packt Publishing. 2007. ISBN: 978-1-847192-48-6

- University of Cambridge, UK, XenSource Inc., IBM Corp., Hewlett-Packard Co., Intel Corp., AMD Inc., and others. *Xen v3.0 Users' Manual*. 2002-2005.

- Stephen Spector, Community Support on xen-users mailing list. *Xen Configuration File Options*. 2009

- Jordi Prats Català. *Migración en caliente en Xen usando DRBD*. Linux+. 11/2007

- Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul Christian Limpach, Ian Pratt, Andrew Warfield. University of Cambridge Computer Laboratory, University of Copenhagen – Department of Computer Science. *Live Migration of Virtual Machines*

- Jon Oberheide, Evan Cooke, FarnamJahanian. Electrical Engineering and Computer Science Department, University of Michigan. *Empirical Exploitation of Live Virtual Machine Migration*

- Julio Gómez López, Nicolás Padilla Soriano. *Manual de Administración de Sistemas Operativos en I.T.I. de Sistemas.* Universidad de Almería, 2005

- Jorge Fuertes. *LVM2: Logical Volume Manager. Curso de administración de sistemas GNU/Linux*. Mayo de 2009

- Kirill Kolyshkin, openvz.org. *Virtualization in Linux*. Septiembre de 2006

- Alberto Molina Coballes, José Domingo Muñoz Rodríguez. *Instalación y configuración de Xen 3.0 en Debian GNU/Linux*. Mayo de 2007

- Saúl Ibarra Corretgé. *Virtualización con Xen. Revisión 3*

- Ian Pratt, Keir Fraser, Steven Hand, Christian Limpach, Andrew Warfield, University of Cambridge. Dan Magenheimer, Hewlett-Packard Laboratories. Jun Nakajima, Asit Mallick, Intel Open Source Technology Center. *Xen 3.0 and the Art of Virtualization*. Proceedings of the Linux Symposium, Volume Two, Pages 65-78. 20-23 de Julio de 2005, Otawa, Ontario, Canadá

- XenSource. *Xen – the Art of Virtualization: a XenSource White Paper*.

- Alfredo Alcayde García, Raúl Baños Navarro, Jesús Camacho Rodríguez, Juan Antonio García Moreno, Consolación Gil Montoya, Francisco Gil Montoya, María Dolores Gil Montoya, Julio Gómez López, Saúl Ibarra Corretgé, Francisco José Méndez Cirera, David Prieto Carrellán. *VoIP y Asterisk: Redescubriendo la telefonía*. Ra-Ma. 2008. ISBN: 978-84-7897-902-8

URLs

- http://www.virtualizate.es/
  Web con conceptos básicos sobre virtualización. Ventajas de la virtualización

- http://www.virtualizados.com/
  Portal Web con noticias e información general sobre virtualización

- http://www.techweek.es/virtualizacion/tech-labs/1003109005901/ventajas-desventajas-virtualizacion.1.html
  Ventajas y desventajas de la Virtualización. La virtualización y la empresa. Implantación de la virtualización: consolidación de servidores. Isabel Martín, HP España. 2008

- http://www.intel.com/technology/virtualization/
  Web oficial de la tecnología Intel VT en Intel

- http://www.amd.com/us/products/technologies/virtualization/
  Web oficial de la tecnología AMD-V en AMD

- http://es.wikipedia.org/
  La enciclopedia libre

- http://www.adminso.es/wiki/
  Wiki sobre la Administración de Sistemas Operativos

- http://www.chilton-computing.org.uk/acl/technology/atlas/overview.htm
  Sitio Web con información sobre el Proyecto Atlas

- http://www.virtualbox.org/
  Sitio Web oficial de Virtual Box

- http://www.virtualbox.org/
  Sitio Web oficial de la solución de sistemas operativos invitados VirtualBox

- http://bochs.sourceforge.net/
  Sitio Web oficial del emulador Bochs

- http://www.qemu.org/
  Sitio Web oficial del emulador Qemu

- http://www.vmware.com/
  Sitio Web oficial de VMWare

- http://www.citrix.com/English/ps2/products/feature.asp?contentID=1686939
  Sitio Web oficial de Citrix XenServer

- http://www.vm.ibm.com/
  Sitio Web oficial de la solución de virtualización completa z/VM

- http://www.xen.org/
  Sitio Web oficial del hipervisor Xen

- http://wiki.openvz.org/
  Sitio Web oficial de la solución de virtualización a nivel del sistema operativo OpenVZ

- http://linux-vserver.org/
  Sitio Web oficial de Linux V-Server, solución de virtualización a nivel del sistema operativo

- http://user-mode-linux.sourceforge.net/
  Sitio Web oficial de User-mode Linux, virtualización a nivel del kernel

- http://www.linux-kvm.org/
  Sitio Web oficial de KVM, solución de virtualización a nivel del kernel

- http://sourceware.org/lvm2/
  Sitio Web oficial de LVM2

- http://www.nfsv4.org/
  Sitio Web oficial de NFS versión 4

- http://www.samba.org/
  Sitio Web oficial de Samba

- http://www.openafs.org/
  Sitio Web oficial de OpenAFS

- http://www.drbd.org/
  Sitio Web oficial de DRBD

- http://freenas.org/doku.php?lang=es
  Sitio Web oficial en español de la distribución FreeNAS

- http://opennas.codigolivre.org.br/
  Sitio Web oficial de la distribución OpenNAS

- http://www.serverelements.com/naslite.php
  Sitio Web oficial de NASLite

- http://wiki.xensource.com/
  Xen *Wiki* con todo tipo de información compartida por la comunidad Xen Open Source

- http://stacklet.com/
  Sitio Web con imágenes de máquinas virtuales en descarga directa para Xen, KVM, VMWare y Qemu

- http://www.oszoo.org/
  Sitio Web con imágenes de sistemas operativos listos para ser usados en Qemu

- http://www.xen-tools.org/
  Sitio Web oficial de la herramienta Xen-Tools

- http://www.xen-tools.org/software/rinse/
  Sitio Web oficial de la herramienta Rinse, usada para el aprovisionamiento de máquinas virtuales con sistemas operativos de tipo RPM

- https://fedorahosted.org/cobbler/
  Sitio Web oficial de Cobbler, herramienta para el aprovisionamiento de máquinas virtuales

- http://libvirt.org/
  Sitio Web oficial de libvirt, API de virtualización

- http://libvncserver.sourceforge.net/
  Sitio Web oficial de LibVNCServer/LibVNCClient

- http://www.convirture.com/
  Sitio Web oficial de ConVirt, herramienta para el aprovisionamiento y monitorización de máquinas virtuales

- http://sourceforge.net
  Sitio para el desarrollo y descarga de software libre

- http://sourceforge.net/projects/xenman/
  Sitio Web oficial para la descarga de XenMan en sourceforge

- http://sourceforge.net/tracker/index.php?func=detail&aid=1738580&group_id=168929&atid=848458
  URL para la descarga del parche aplicable a XenMan para su uso en Debian 5 Lenny

- http://virt-manager.et.redhat.com/
  Sitio Web oficial de Virt-Manager, herramienta para la puesta en marcha y monitorización de máquinas virtuales

- http://www.nagios.org/
  Sitio Web oficial de Nagios, aplicación para la monitorización de equipos y redes

- http://oss.oetiker.ch/mrtg/
  Sitio Web oficial de MRTG, aplicación para la monitorización de interfaces de red, equipos y redes

- http://www.asterisk.org/
  Sitio Web oficial de Asterisk

- http://sipp.sourceforge.net
  Sitio Web oficial de SIPp, herramienta para el *benchmarking* de centralitas VoIP Asterisk

- http://asterisk.hosting.lv/
  Sitio Web con los códecs g729 y g723.1 Linux y FreeBSD para Asterisk

- http://www.linux-ha.org/
  Sitio Web oficial de la solución de alta disponibilidad software Linux-HA

- http://www.linux-ha.org/wiki/Heartbeat
  Sitio Web oficial de Heartbeat en la web de Linux-HA

- http://hg.linux-ha.org/
  Repositorios Mercurial para el proyecto Linux-HA

- http://clusterlabs.org/
  Sitio Web oficial de Pacemaker, herramienta administrativa de los recursos de un clúster

- http://hg.clusterlabs.org/
  Repositorios Mercurial para el proyecto Pacemaker

- http://fedoraproject.org/
  Sitio Web oficial de la distribución GNU/Linux Fedora

- http://www.debian.org/
  Sitio Web oficial de la distribución GNU/Linux Debian

- http://www.google.es/
  Buscador Google

- http://www.gnu.org/home.es.html
  Sitio Web oficial en español del Proyecto GNU

- http://www.saghul.net/blog/
  Mi Brain-Training Personal, *blog* de Saúl Ibarra, uno de los más importantes sobre VoIP en castellano

- http://delicious.com/saghul/xen
  Web con ejemplos y tutoriales sobre Xen. Saúl Ibarra.

- http://www.josemariagonzalez.es/
  El Blog de Virtualización en Español, una de las Webs y *blogs* más importantes sobre virtualización en español

- http://lists.xensource.com/mailman/listinfo/xen-users
  Lista de correo para usuarios de Xen

- http://lists.xensource.com/mailman/listinfo/xen-devel
  Lista de correo para los desarrolladores del hipervisor Xen

- http://www.ual.es/
  Sitio Web de la Universidad de Almería

- http://masteracsi.ual.es/
  Sitio Web del Máster Propio en Administración, Comunicaciones y Seguridad Informática por la Universidad de Almería

- http://www.yellow-bricks.com/
  Blog sobre virtualización mantenido por Duncan Epping: VMWare, HA, Cloud Computing

- http://blogs.technet.com/b/davidcervigon/
  Blog sobre virtualización por David Cervigón: Hiper-V

- http://blog.scottlowe.org/
  Weblog para profesionales en virtualización, almacenamiento y servidores. Scott Lowe

- http://www.redhat.com/virtualization/rhev/
  Sitio Web oficial de Red Hat Enterprise Virtualization

- http://www.linux-kvm.com/
  Noticias, *blogs* y recursos sobre KVM

- http://www.linuxinsight.com/
  Interesante Web con noticias sobre proyectos software libre

- http://www.cl.cam.ac.uk/research/srg/netos/xen/
  Sitio Web oficial del Proyecto Xen en la Universidad de Cambridge

- http://www.virtual-strategy.com/
  Magazine *online* sobre la tecnología de virtualización

- http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=462
  Visión general sobre las máquinas virtuales y su aplicación práctica en aulas. Ministerio de Educación, Política Social y Deporte

- http://www.alcancelibre.org/
  Noticias, manuales, software, capacitación y recursos Linux.

- http://houseofsysadmins.ning.com/
  Red social de Administradores de Sistemas

- http://techthrob.com/2009/03/02/virtualization-in-linux-a-review-of-four-software-choices/
  Jonathan DePrizio. *Virtualization in Linux: a review of four software choices*.

- http://virt.kernelnewbies.org/
  Linux Virtualization *Wiki*

- http://virtualization.com/
  Sitio Web de la Universidad de Almería

- http://www.vi-pedia.com/
  Vi-Pedia, *wiki* con información general sobre virtualización

- http://virtualization.com/
  Página Web con recursos sobre virtualización: noticias, artículos, entrevistas, trabajo, etc.