***Títol:*** Virtualización de servidores del RDlab

***Volum:*** 1/1
***Alumne:*** Xavier Peralta Ramos

***Director/Ponent:*** Gabriel Verdejo Álvarez/Àlvar Vinacua Pla
***Departament:*** Llenguatges i Sistemes Informàtics
***Data:*** Juny 2012

**DATOS DEL PROYECTO**

| | |
|---|---|
| *Título del proyecto:* | Virtualización de servidores del RDlab |
| *Estudiante:* | Xavier Peralta Ramos |
| *Titulación:* | Enginyeria tecnica en Informàtica de sistemes |
| *Créditos:* | 22,5 |
| *Director/Ponente:* | Gabriel Verdejo Álvarez/Àlvar Vinacua Pla |
| *Departamento:* | Llenguatges i Sistemes Informàtics |

**MIEMBROS DEL TRIBUNAL** *(nombre i firma)*

| | |
|---|---|
| *Presidente:* | Ramon Sangüesa i Sole |
| *Vocal:* | José Ramon Herrero Zaragoza |
| *Secretario:* | Àlvar Vinacua Pla |

**QUALIFICACIÓ**

*Calificación numérica:*

*Calificación descriptiva:*

*Fecha:*

# Virtualización de servidores del RDlab

Xavier Peralta Ramos

Juny 2012

# Agradecimientos

Quisiera agradecer la oportunidad que se me brinda para agradecer a esas personas de mi día a día. En primer lugar, a mis padres, porque siempre estan ahí y me lo han dado todo, lo haga bien o lo haga mal. A Bibiana, que siempre ha sido un ejemplo para mí. A mis abuelos, los que están y los que no, por todo el cariño que me dieron y me dan. A Leti, porque aún desde la distancia se siente cerca, y ejerce esa presión positiva que te anima a hacer las cosas ;).

A los de Barcelona, Albert, Marc, Arnau, Miki, a Pau por sus clases particulares de linux, a los de Alcorisa por hacerme sentir aragonés y a mi compañeros de facultad por preguntar incansablemente cuando presentaba.

Agradecer a Gabriel la oportunidad que me dió en el *RDlab*, por todas las cosas que he ido aprendiendo durante este año. A mis compañeros del RDlab y a los integrantes de la UGDSI por lo bien que me han tratado siempre desde el primer día.

Por último agradecer a Àlvar por su soporte en la realización de la memoria.

De todo corazón,
Gracias!

# Índice general

# Índice de figuras

# CAPÍTULO 1

# Introducción

## 1.1. Origen y motivación del proyecto

El Departamento de *Lenguajes y Sistemas Informáticos (**LSI**)* se creó en 1986 durante una reorganización académica llevada a cabo en la *Universidad Politécnica de Catalunya* (**UPC**) donde se agrupó a los diferentes departamentos relacionados con los ámbitos del software y la informática. De esta forma y ya desde sus inicios, el Departamento de LSI ha destacado por su orientación hacia la investigación potenciando la consolidación de diversos grupos de investigación multidisciplinares.

Un elemento clave para afianzar los grupos de investigación departamentales (siete actualmente que se recogen en la figura 1.1) y conseguir que sean referentes en sus campos de estudio fue la creación de un soporte **TIC** (*Tecnologías de la Información y la Comunicación*, **IT** en inglés) excelente para todo su personal docente e investigador.

Estas tareas de soporte a la investigación, que inicialmente asumió el *Laboratorio de Cálculo de LSI* (**LCLSI**) y posteriormente el *Laboratori de Recerca i Desenvolupament* (**RDlab**), están orientadas a proporcionar a los investigadores el acceso a las tecnologías más innovadoras y punteras. De esta forma, el constante acceso a los nuevos paradigmas y sistemas TIC permite mantener y reforzar el espíritu innovador que necesitan los grupos de investigación del Departamento.

Este constante aprendizaje y adaptación a los nuevos modelos de entender la informática, probablemente una de las ciencias donde más innovación se realiza, conduce necesariamente a contrastar las nuevas propuestas para mantener la calidad de servicio y adaptar nuevos conceptos que aporten una mejora qualitativa.

La informática de sistemas ha cambiado profundamente en los últimos 4 años. Las mejoras introducidas en el hardware (sistemas multicore, soporte físico para operaciones de cambio de contexto...) así como la inmensa expansión de servicios ofrecidos via Internet y las restricciones

| | |
|---|---|
| *ALBCOM* | Grup d'Algorísmia, Bioinformàtica, Complexitat i Mètodes Formals |
| *GIE* | Grup d'Informàtica a l'Enginyeria |
| *GRPLN* | Grup de Recerca en Processament del Llenguatge Natural |
| *KEMLG* | Knowledge Engineering and Machine Learning Group |
| *LARCA* | Laboratori d'Algorísmia Relacional, Complexitat i Aprenentatge |
| *LOGPROG* | Lògica i Programació Informàtica |
| *MOVING* | Grup de recerca en Modelatge, Interacció i Visualització en Realitat Virtual |
| *SOCO* | Soft Computing |

Figura 1.1: Grupos de Investigación en el Departamento de LSI

económicas, han hecho emerger nuevos conceptos TIC como el **GreenIT / Green Computing**, la **virtualización/VM** (*Virtual Machine*) o la **nube** (*Cloud*). Estos conceptos presentan una nueva manera de plantear la organización de los servidores y servicios que muy probablemente nos acompañará las próximas décadas. Este *Proyecto Final de Carrera* (**PFC**) es el estudio y plan de ruta del *RDlab* para adecuar los sistemas TIC del Departamento de LSI a la informática del siglo XXI.

## 1.2. Objetivos

Este estudio se enmarca en el ámbito de los servidores y servicios TIC que proporcionan el soporte a los distintos grupos de investigación. En estos equipos los investigadores realizan todo su trabajo, que cubre aspectos tan diversos como la realización de experimentos y simulaciones, el desarrollo de proyectos docentes o de investigación tanto nacionales como internacionales y de transferencia de tecnología. También se realizan tareas *ad hoc* como la creación de prototipos o el soporte para la realización de congresos y *workshops*.

De esta forma, se pretende que este proyecto inicialmente realice un análisis exhaustivo de los nuevos paradigmas de gestión de sistemas que han aparecido en los últimos años con el objetivo de comprobar su grado de madurez. Posteriormente se comprobará y contrastará efectivamente las posibles mejoras respecto los sistemas tradicionales que se aplican a la gestión TIC. Y finalmente se propondrá un modelo sostenible, razonable y adecuado a la idiosincrasia de los distintos grupos de investigación del Departamento de LSI y compatible con el de la UPC.

Para poder evaluar los puntos anteriores de forma fidedigna, se realizará un modelo o prototipo que permita reproducir un entorno real de los distintos sistemas de investigación. Dentro de este entorno reproducible se calcularán y evaluarán varios aspectos relevantes mediante el uso de programas de *benchmarking*.

## 1.3. El Laboratori de Recerca i Desenvolupament (*RDlab*)

El *RDlab* se ubica dentro de las estructuras de excelencia destinadas a fomentar la investigación y desarrollo de proyectos TIC en los departamentos de la Universitat Politècnica de

Catalunya.

Con el objetivo de proporcionar un mejor apoyo a las tareas de los profesores e investigadores del Departamento de LSI, en el año 1987 se creó el *Laboratorio de Cálculo del Departamento de LSI (LCLSI)*. Este laboratorio, que inicialmente estuvo formado por dos personas, fue creciendo tanto en personal asociado -hasta 12 personas en 2010- como en capacidades y profesionalidad, consiguiendo consolidar un soporte TIC integral para todos los usuarios del Departamento. En este periodo el LCLSI fue asumiendo tanto los servicios básicos o de infraestructura común para todos los usuarios como el soporte especializado en investigación para el personal docente e investigador.

A finales del 2010, después de la creación del Departamento de **ESSI** (*Enginyeria de Serveis i Sistemes d'Informació*), el Departamento de LSI decide hacer una apuesta innovadora y valiente para potenciar sus grupos de investigación y el desarrollo de proyectos: nace el *Laboratori de Recerca i Desenvolupament*.

El *RDlab* es el encargado de proporcionar todo el soporte TIC a los grupos de investigación del Departamento de LSI. Ofrece un soporte personalizado, integral y de máxima calidad a todo el personal docente e investigador así como a sus colaboradores y proyectos. El *RDlab* mantiene un compromiso firme con el Departamento de LSI, aportando su experiencia en el ámbito de las tecnologías de la información para optimizar los procesos de investigación y el desarrollo de proyectos.

## 1.4. Motivación personal

Como estudiante de la FIB accedí a una beca del Laboratorio de Cálculo de *LSI* en el año 2010. Durante este período de tiempo fui aprendiendo diferentes aspectos tecnológicos que fueron reforzados durante mis estudios. Posteriormente, cuando se fundó el *RDlab* se me ofreció la posibilidad de realizar una beca centrada en sistemas operativos, uno de los aspectos que más me interesan de la informática.

El responsable del RDlab me ofreció hacer el proyecto de final de carrera orientado a la virtualización de sistemas, ya que existía la necesidad de ir un paso más adelante dentro de la administración de los servidores que gestionan. Me pareció muy interesante aprender sobre un tema que está teniendo mucho auge durante los últimos años y adquirir nuevos conocimientos y experiencia en este tema. Otra motivación ha sido que siempre he estado muy a gusto en el laboratorio y existe un buen ambiente de trabajo.

La realización de este proyecto también ha presentado varias dificultades. Cabe destacar especialmente la sobreinformación, que realmente ha dificultado encontrar el contenido realmente importante, ya que queda enterrado entre miles de referencias y documentos poco útiles y cientos de foros con preguntas sin respuesta. Además, como suele ser habitual en el mundo de la administración de sistemas, la información se encuentra muy dispersa y es realmente difícil sintetizarla correctamente. También han existido dificultades técnicas en la integración de plataformas que no son totalmente compatibles.

Mención a parte merece la velocidad a la que el software se va actualizando. La temática cubierta en este documento está siendo constantemente revisada y ampliada, de forma que muchas

veces la documentación hace referencia a una versión que por requisitos del sistema (versión de *Kernel* o librerías) no funciona aún en ninguna distribución actual. Esto implica que debe *parchearse* el sistema operativo o el propio *kernel* o que incluso tiene problemas porque algunas opciones o parámetros han desaparecido.

Con todo, se han ido tomando decisiones de diseño que permitiesen avanzar este proyecto creando un marco estable de pruebas con el estado del arte de la tecnología utilizada a inicios de 2012.

## 1.5. Estructura de la memoria

Este documento se ha agrupado en ocho capítulos que permiten seguir la secuencia lógica del trabajo realizado en este proyecto. Con esta organización se pretende facilitar la lectura y seguimiento de este PFC, ofreciendo una aproximación incremental al trabajo realizado.

Los primeros tres capítulos recogen la introducción necesaria para enmarcar este PFC así como su justificación y requisitos. Los capítulos cuatro y cinco presentaran el trabajo de campo realizado, la justificación de los diferentes elementos utilizados así como los resultados obtenidos en las pruebas. Finalmente los capítulos seis y siete contienen las conclusiones obtenidas de este trabajo así como la propuesta modelizada que se realiza.

La planificación seguida en este proyecto y los costes derivados se ha ubicado al final del documento, capítulo 8, junto a los anexos. Esta decisión se fundamenta en que este apartado siempre se (re)calcula *a posteriori* y por tanto, al seguir este documento una secuencia temporal lógica era recomendable situarlo en la parte final del escrito.

Este escrito estará disponible en formato electrónico en las siguientes páginas web:

*http://rdlab.lsi.upc.edu/docu*
*http://gabriel.verdejo.alvarez.googlepages.com/*

### 1.5.1. Convenciones del documento y tipografía

Para una mejor comprensión de este proyecto, dada la elevada naturaleza técnica de su contenido, se seguirán una serie de convenciones tipográficas elegidas para facilitar su lectura.

Los nombres propios de origen extranjero, así como los nombres de instituciones u organizaciones (como en el caso de *Ubuntu* o *RDlab*) aparecen en cursiva. Los acrónimos y nombres de empresas o instituciones apreceran en mayúsculas y sin puntos intermedios (UPC). Así mismo la primera vez que se referencien en este texto, se acompañaran del significado de sus siglas en cursiva y usando negrita en las siglas (**LSI**, *Departamento de Lenguajes i Sistemas Informáticos*), de forma que por motivos de legibilidad puedan ser referenciados posteriormente únicamente por sus siglas.

Las definiciones y citas textuales tanto de personas como de documentos referenciados en este escrito se señalaran usando comillas dobles y utilizando letra en cursiva: "*Any sufficiently*

*advanced technology is indistinguishable from magic*", Arthur C. Clarke.

Igualmente, todos aquellos anglicismos que necesariamente aparecen en este texto, así como cualquier otro vocablo del ámbito de la informática, aparecerán en cursiva la primera vez que se referencien. Lo mismo sucederá con palabras que, si bien son de origen foráneo, forman ya parte de nuestro lenguaje (como por ejemplo *software*).

El uso de la **negrita** queda reservado a palabras o conceptos clave que se desea destacar especialmente debido a su relevancia en el capítulo. Por lo tanto puede aplicarse tanto a texto **en formato normal** o destacado, como por ejemplo ***en cursiva***.

Una parte de este proyecto consiste en código fuente o ficheros de configuración del sistema. Su presentación se realizará delimitada dentro de una figura y con el nombre y/o acceso (*path*) en la descripción resaltado en negrita.

```
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual

auto eth0.XXX
iface eth0.XXX inet manual
vlan_raw_device eth0

auto virbr0
iface virbr0 inet static
address XX.XX.XX.XX
netmask XX.XX.XX.XX
network XX.XX.XX.XX
gateway XX.XX.XX.XX
```

Figura 1.2: Fichero de configuración **/etc/network/interfaces**

# CAPÍTULO 2

# Análisis de requisitos

## 2.1. Introducción

En este capítulo se describirá el resultado del análisis previo a la realización del proyecto dónde se recogieron las distintas restricciones impuestas por el entorno de trabajo. En nuestro caso, principalmente los siete grupos de investigación del Departamento de LSI. La infraestructura existente y las políticas TIC estratégicas del *Departamento de LSI*. También fueron tenidas en cuenta las recomendaciones actuales de la propia *UPC* así como las directrices de la *Generalitat de Catalunya* en política TIC.

Los ***requisitos funcionales*** hacen referencia a las restricciones que marcaran el comportamiento deseado del proyecto, indicando los aspectos relevantes y claves que nuestra solución debe proporcionar.

Los ***requisitos no funcionales*** son aquellos que recogen los aspectos que no definen directamente la funcionalidad o el comportamiento fundamental, pero sí marcar los aspectos cualitativos generales del proyecto.

De esta forma, por tanto, el conjunto total de estos requisitos definidos a continuación permitirá realizar la evaluación de la solución propuesta y establecer el grado de satisfacción final.

## 2.2. Requisitos funcionales

- El proyecto debe proponer una plataforma única de virtualización para los servidores de proyecto administrados por el *RDlab*.

  El sistema escogido debe permitir la ejecución de todos los sistemas existentes actualmente en los servidores de los grupos de investigación. Debido a la singularidad de la

investigación, muchas veces se utilizan una gran variedad de sistemas operativos, programas y servicios que debe mantenerse funcionando varios años.

De esta forma, la solución de virtualización escogida debe permitir la ejecución de todas las configuraciones actuales utilizando una misma plataforma. No se desea contemplar soluciones híbridas basadas en dos o más sistemas de virtualización dependiendo del sistema dónde se ejecutan.

- Los sistemas virtualizados deben permitir ejecutar cualquier Sistema Operativo que funcione en la arquitectura ***x86***.

  En los más de 40 servidores gestionados por el *RDlab* dedicados a la investigación, se encuentran funcionando actualmente distintas versiones de un mismo sistema operativo (*Linux* Debian 4 y 5 por ejemplo). Igualmente también podemos encontrar sistemas de distintos fabricantes, lo que nos obliga a proporcionar una plataforma flexible para la arquitectura *x86*.

  De esta forma es necesario que nuestra plataforma permita virtualizar de una forma completa el hardware existente para soportar las distintas configuraciones que podemos encontrarnos.

- Los sistemas virtualizados deben permitir ejecutar indistintamente sistemas de 32 ó 64 bits.

  Actualmente en los servidores del *RDlab* nos encontramos con la existencia de programas y sistemas operativos que requieren un determinado entorno imposible de cambiar. No siempre se dispone del código fuente de las aplicaciones y por tanto no siempre es posible adaptarlo a los nuevos sistemas. Además, muchas veces los experimentos realizados por los investigadores requieren un entorno concreto en 32 o 64 bits para sus pruebas.

- La solución propuesta debe permitir fácilmente modificar los recursos asignados a las máquinas virtuales (CPU, memória RAM, disco y red).

  Las necesidades de servicio son cambiantes con el tiempo, de esta forma se requiere que la plataforma de virtualización contemple estas modificaciones en la asignación de recursos. La posibilidad de adecuar los parámetros asociados a las máquinas virtuales nos permitirá optimizar el uso del hardware y poder proporcionar un servicio adecuado a las necesidades de cada momento.

  Con este requisito se pretende evitar sistemas dónde la configuración de servidores virtuales sea muy compleja o cause problemas en la estabilidad. Es importante destacar que **esta asignación nueva de recursos no implica que inmediatamente (en caliente) sean efectivos en la máquina virtual**, ya que esta posibilidad dependería en todo caso del propio sistema que se ejecuta virtualizado.

- La plataforma de virtualización debe contemplar la coexistencia de varias máquinas virtuales en un mismo servidor físico.

  Un aspecto importante dentro de este proyecto es la introducción de criterios energéticos más racionales (***GreenIT***) en las políticas TIC del *Departamento de LSI*. Estos hacen referencia a un aprovechamiento más eficientemente de los servidores físicos adecuándolos a las necesidades de trabajo y minimizando el consumo energético.

  De esta forma, la existencia de más de una máquina virtual funcionando en un servidor físico es un aspecto necesario para la plataforma propuesta. La gestión de los recursos debe ser eficiente y contemplar el aprovechamiento de los nuevos procesadores *multicore*.

## 2.3. Requisitos no funcionales

- La plataforma de virtualización propuesta debe ser libre.

  El compromiso con el software libre es un aspectos importante dentro de los grupos de investigación del *Departamento de LSI* y de la UPC. La posibilidad de acceder al código libre y gratuitamente para adaptar o corregir diferentes aspectos, o realizar su compilación en distintas plataformas es vital para los investigadores. También debemos señalar que en la coyuntura económica actual, las limitaciones presupuestarias refuerzan este requisito.

  Este PFC debe por tanto señalar la plataforma libre más adecuada para la virtualización de los sistemas así como sus limitaciones de uso. En el caso de existir fuertes restricciones o imposibilidad de asumir todos los requisitos descritos, se deberían contrastar con las alternativas comerciales existentes.

- La solución propuesta debe contemplar un sistema de gestión remoto de las máquinas virtuales.

  Debido a la gran cantidad de servidores gestionados en el *RDlab* y su continuo crecimiento, es absolutamente necesario que la plataforma virtual permita el uso de un sistema centralizado que facilite su gestión.

  Si bien no se requiere una solución completa y profesional de gestión, sí que se solicita una plataforma que permita al menos el control básico de encender y apagar las máquinas virtuales en los diferentes servidores físicos. De esta forma la plataforma presentada en este estudio ha de ser funcional, dejando para posibles proyectos posteriores la mejora o ampliación del esta herramienta, puesto que su código debe ser público.

- La solución propuesta debe contemplar mecanismos de seguridad y auditoria.

  La seguridad y los mecanismos de control de acceso son aspectos fundamentales para poder controlar un sistema de estas dimensiones. La posibilidad de definir políticas de seguridad comunes a todas las máquinas virtuales facilitará su uso y gestión.

En un entorno dónde varias máquinas virtuales coexisten en un mismo servidor físico, cada una con sus usuarios y servicios, la abstracción y encapsulación de estas debe ser asegurada por el propio sistema.

- La solución propuesta debe contemplar mecanismos de alta disponibilidad para las máquinas virtuales.

  La virtualización de servidores aporta una gran flexibilidad en el uso de los sistemas. Por este motivo la existencia de mecanismos que posibiliten añadir robustez a la plataforma presentada permitirá proporcionar un mejor servicio a los grupos de investigación.

  El esquema de virtualización escogido debe permitir el uso de una arquitectura flexible para aquellos servicios que requieran de una alta disponibilidad. La aplicación de estas políticas quedaría fuera de este proyecto.

- La solución propuesta debe contemplar mecanismos para la realización de copias de seguridad de las máquinas virtuales.

  Aunque no es un requisito directo de la propia virtualización, sí es necesario para la completitud de la plataforma presentada. La solución propuesta debe permitir añadir de una forma sencilla un sistema de gestión de copias de seguridad que le permita aumentar la seguridad y robustez del servicio ofrecido.

  Como en otros requisitos no funcionales, la descripción detallada y la implementación efectiva del sistema queda fuera de este PFC.

# CAPÍTULO 3

## Conceptos y paradigmas TIC del siglo XXI

En este capítulo profundizaremos en los nuevos conceptos que dominan actualmente la informática. Repasaremos brevemente la historia para ubicar como surgió la virtualización e introduciremos las partes básicas que componen una plataforma de virtualización, su taxonomía y cómo funcionan.

## 3.1. Perspectiva histórica

La virtualización es uno de esos conceptos que puede parecer reciente pero en realidad se desarrolló en los años 60, concretamente en el año 1964 de manos de IBM. En 1967, se desarrolló el primer ***hypervisor*** (*hipervisor*), que es el la capa que realiza una abstracción del hardware. La segunda version llegó en 1968 (*CP-67*), donde se habilitó la memoria compartida entre las distintas máquinas virtuales, permitiendo a cada usuario tener su propio espacio de memoria. En 1970 IBM continuó mejorando su tecnología, permitiendo ejecutar máquinas virtuales con distintos sistemas operativos [HOV].

No fue hasta la década de los 90 cuando aparecieron en el mercado las primeras plataformas de virtualización. En 1997 Connectrix saca a la luz *Virtual PC* para *Macintosh* que tuvo un moderado éxito entre los usuarios de *Apple*. En 1999 la plataforma ***VMware*** *Virtual Platform* para arquitectura *IA-32* se convirtió rápidamente el software de virtualización comercial de referencia. En 2003 sale la primera versión del hipervisor open-source ***Xen*** como respuesta de la comunidad libre.

En el año 2005 se extendió el uso y creció la popularidad de la virtualización entre los usuarios de PC al aparecer *VMware workstation*, una versión gratuita para virtualización de escritorio. A partir de 2006 *Intel* y *AMD* producen comercialmente lo primeros procesadores con soporte *hardware* a la virtualización, hecho que ayudó siginificativamente a la actualización de las plataformas para aprovechar las nuevas ventajas que brindaba esta tecnología.

Hoy en día tenemos infinidad de plataformas disponibles para todas las arquitecturas y cada año se van implementando soluciones con un rendimiento más próximo a la ejecución nativa [TVD].

## 3.2. Virtualización

M.Tim Jones, arquitecto de firmware de IBM, en el artículo del año 2007 *Discover the Linux Kernel Virtual Machine* define la virtualización como: "*Virtualization it's the process of taking something and making it look like something else.*"[DKV].

El concepto ha ido evolucionando y adecuándose a las demandas de los usuarios de forma que recientemente la virtualización ha empezado a tomar importancia en el mundo de las tecnologías de la información (se estima que el 60 % de los servidores se virtualizarán entre 2009 y 2014 segun *Gartner*) [SVS].

Una plataforma de virtualización se compone básicamente de 3 elementos: el ***Hipervisor*** o ***VMM*** *(Virtual Machine Monitor)* el ***guest*** y el ***host*** (ver figura 3.1).

El sistema *host*, anfitrión en inglés, es la máquina o el sistema operativo donde está alojado el hipervisor y la máquina virtual. El sistema *guest*, huésped en inglés, es la máquina virtual que se ejecuta en el *host* y proporciona los servicios a los usuarios.

M. Tim Jones define el *hipervisor* como: "*Hypervisors do for operating systems what operating systems roughly do for processes. They provide isolated virtual hardware platforms for execution that in turn provide the illusion of full access to the underlying machine*". En la figura 3.1 se muestra un esquema de la posición que ocupa el hipervisor dentro de una plataforma de virtualización.

Como se puede observar, éste se encuentra un nivel por encima del hardware y hace de capa de abstracción dando a entender a los *guest* que sólo ellos estan ejecutándose en el *host* [AHL].

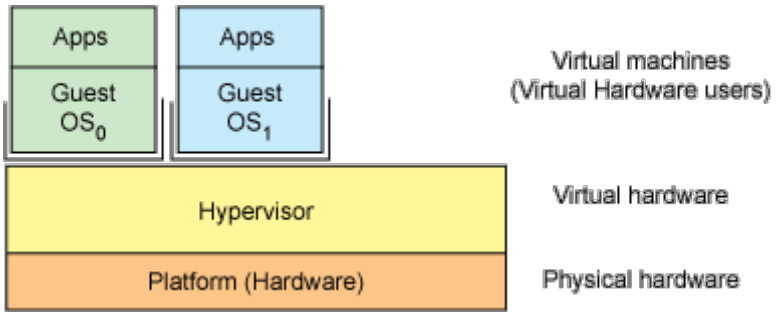


Figura 3.1: Esquema del hipervisor dentro de una plataforma de virtualización

Como se ha indicado anteriormente, la virtualización juega un papel muy importante que ha incrementado exponencialmente en estos últimos años. Los motivos principales para esta rápida expansión son principalmente:

- **Consolidación de servidores**

  La *U.S. Environmental Protection Agency* (*EPA* por sus siglas en inglés) afirma que el aprovechamiento de los servidores en EE.UU es aproximadamente del 5 % del total de los recursos hardware disponibles, el resto del tiempo el servidor se mantiene inactivo. De esta forma, si quisiéramos aprovechar mejor los recursos podríamos ejecutar más aplicaciones en estos equipos pero con la consecuencia que una caída del servidor hiciese caer todos los servicios que se ejecutan en él.

  Gracias a la virtualización se pueden ejecutar diferentes servicios en diferentes máquinas aisladas las unas de las otras de este modo el aprovechamiento puede llegar al 80 % reduciendo el consumo eléctrico, el número de servidores, el espacio para almacenarlos y los costes de mantenimiento sin perder ningún servicio, ya que al ser sistemas independientes pueden ser migrados y ejecutados en otros servidores físicos distintos [DRC].

- **Aislamiento y seguridad**

  Ligada con el anterior punto tenemos esta ventaja significativa. La virtualización permite ejecutar diferentes máquinas virtuales aisladas esto significa que ninguna puede acceder a los recursos de las demás y que si una de ellas cae por cualquier motivo (error en la configuración, ataque externo, etc.) las otras máquinas que están funcionando seguirán ejecutándose sin que haya daños colaterales.

- **Portabilidad y alta disponibilidad**

  Otra característica de la virtualización es la gran flexibilidad que ofrece. A parte de la posibilidad que brinda en cuanto a poder virtualizar diferentes sistemas operativos y arquitecturas, tenemos la posibilidad de mover máquinas virtuales entre diferentes máquinas físicas.

  Varias plataformas de virtualización tienen soporte para realizar una migración en caliente *(Live migration)* que permite mover máquinas virtuales entre distintos servidores físicos sin que éstas se paren. Esta funcionalidad se consigue mediante ***snapshots***, que son copias exactas del estado actual en que se encuentra una máquina virtual en un instante dado. Los *snapshots* también permiten recuperar una máquina virtual que ha dejado funcionar ya que contienen todo el sistema y sus datos.

- **Desarrollo de software y banco de pruebas**

  Esta característica es también una gran baza de la virtualización y podría ser una aplicación derivada de las dos características anteriores.

  Aprovechando las características de aislamiento y tolerancia a fallos una màquina virtual es muy útil a la hora de probar nuevas actualizaciones del sistema. Mediante el mecanismo de *clonación* (*clone*) se puede comprobar que estas no entorpecen en el buen funcionamiento del sistema pudiéndose extrapolar a nuevas aplicaciones.

Cabe destacar, por contra, que la virtualización también tiene inconvenientes. Uno de ellos, quizá el más importante, es que se perderá un porcentaje de rendimiento comparado con el que

podría alcanzar la máquina sin virtualizar debido al *overhead* que se introduce al virtualizarla, aunque la tendencia es a ir mejorando cada día este rendimiento.[GOV]

Otra desventaja es que la virtualización hoy en día necesita procesadores con soporte hardware a la virtualización para que su rendimiento sea aceptable. Además, tiene una dificultad añadida que es la configuración del sistema base (*host*) donde se ejecutarán las máquinas virtuales.

### 3.2.1. Tipos de virtualización

Una vez explicados los elementos que forman una plataforma de virtualización vamos a introducir los diferentes tipos que existen. Primero definiremos los hipervisores (ver figura 3.2). Actualmente existen dos tipos de *hipervisores* que se clasifican según en qué ámbito del sistema se ejecutan:

- **Hipervisores tipo 1 o *bare metal***: Los hipervisores de este tipo se ejecutan directamente sobre el hardware al mismo nivel que se ejecutaría un sistema operativo. Como ejemplos de hipervisores tipo 1 tenemos *KVM*, *Xen* y *VMware*.

- **Hipervisores tipo 2 o *hosted***: Los hipervisores tipo *hosted* se ejecutan, a diferencia de los anteriores, sobre un sistema operativo base que a su vez se ejecuta sobre el mismo hardware físico. Como ejemplos de hipervisores de este tipo tenemos *Qemu* y *Wine*.

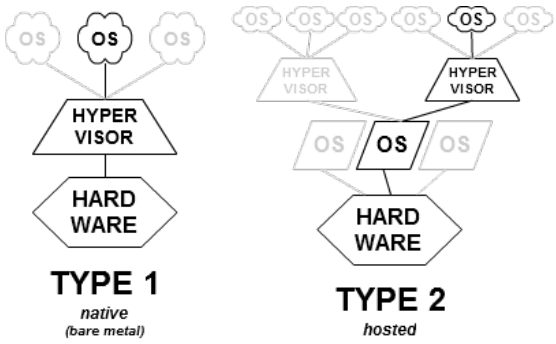


Figura 3.2: Estructura de los dos tipos de hipervisores

Podemos decir que, independientemente de su tipo, un *hipervisor* es una capa que abstrae el hardware de la máquina de sus *guests*. De esta manera cada *guest* ve *hardware virtual* en lugar de *hardware real* del sistema.

A continuación procederemos a explicar las características de los tipos de plataforma de virtualización más importantes que existen y se utilizan habitualmente.

### 3.2.2. Paravirtualización

De la palabra griega "$\pi\alpha\rho\alpha$", que significa aproximación, surge el prefijo "*para*"de la palabra paravirtualización cuyo significado sería próximo a la virtualización. Se puede usar esta extrapolación puesto que la paravirtualización es el estado intermedio entre la virtualización y la ejecución nativa. Como plataforma de paravirtualización podemos destacar a *Xen* que es el máximo exponente de esta aproximación. A continuación nos centramos en esta implementación de la paravirtualización.

La paravirtualización (figura 3.3) implica modificar el kernel tanto del *host* como del *guest*. De esta forma el *kernel* del *host* se modifica debido a que hasta hace poco por defecto el *kernel* de *Linux* no estaba preparado para realizar las funciones de hipervisor. Igualmente es necesario modificar el kernel del *guest* para que pueda saber que se está ejecutando sobre un entorno paravirtualizado y contribuya a que el rendimiento sea óptimo. Como es necesario que se modifiquen los kernels del *host* y el *guest*, esta aproximación a la virtualización sólo se puede utilizar en kernels de código libre. Igualmente esta limitación explica que la paravirtualización no permita virtualizar sistemas *Windows*.

En la paravirtualización el único componente que tiene todos los privilegios de acceso al hardware es el hipervisor. No obstante, existe un *guest* privilegiado en el que el hipervisor puede confiar llamado *Domain 0*, *driver Domain* o *dom0*. Este *guest* privilegiado es el único con permiso para controlar las funciones de administración como serian encender *guests* y acceder a los dispositivos hardware ya que tiene los drivers usuales para exportar las abstracciones al hipervisor.

El concepto de *Domain 0* es complejo de explicar y poco intuitivo, por ello resulta de ayuda utilizar la analogía siguiente: "Se debe pensar en el hipervisor como un coche en el que viajan los pasajeros que son los *guests* y dónde el *domain 0* hace de conductor"[TC10].

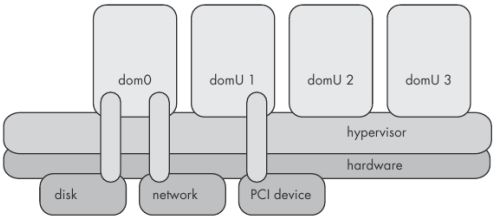


Figura 3.3: Estructura de la paravirtualización.

A continuación describiremos los principales componentes que encontramos en la paravirtualización:

- **CPU:** A pesar de la paravirtualización, el procesador sigue siendo un objeto físico sujeto a la realidad de la física. Esto quiere decir que la CPU sigue aceptando solamente una instrucción en un instante de tiempo como si se tratara de un sistema sin virtualizar.

  Cada *guest* mantiene su propia cola interna de instrucciones y notifica al hipervisor cúal es la próxima instrucción que debe tratarse, pudiendo también pedir al hipervisor que se

trate una interrupción posteriormente gracias al *domain-virtual timer* o al *system timer*. Con el *system timer* los *guests* pueden liberar el procesador mientras se escribe en el buffer de E/S y una vez lleno se vuelve a hacer una petición para ocupar la CPU.

En el hipervisor tenemos el *domain-virtual timer* que se usa como programador interno de los procesos de los *guests* y el *system timer* que se encarga de los eventos que son sensibles a tiempo real como la red.

- **Memoria:** Aunque el hipervisor tiene el control sobre la memoria que se le asigna a cada *guest*, éstos manejan todas las funciones que gestionan la memoria asignada (ver firgura 3.4).

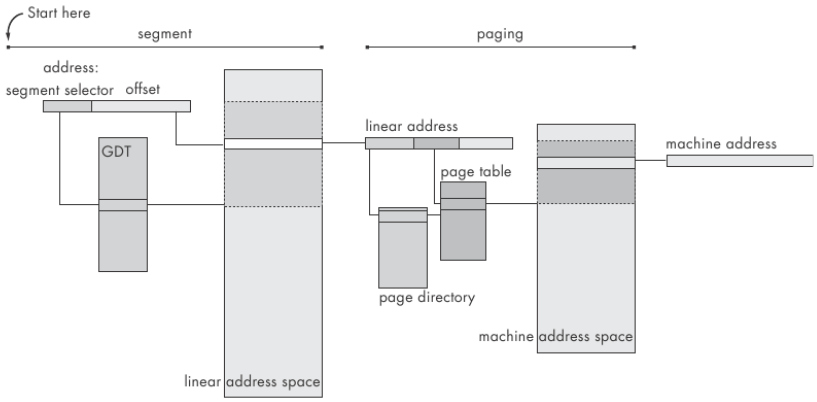


Figura 3.4: Esquema de acceso a memoria

En una máquina sin virtualizar el procesador implementa una memoria virtual que ofrece dos ventajas: La primera es permitir que cada proceso acceda a su propia memoria como si no hubiese nada mas ejecutándose en el *host*. La Segunda es permitir a los procesos acceder a más memoria de la que hay físicamente disponible, moviéndola a *swap* si es necesario.

Este mecanismo se aplica tanto si el sistema se ejecuta virtualizado como si es nativo y sirve para proteger a la memoria de forma que un proceso no acceda a una posición de memoria que no le pertenezca (Figura 3.4). Como los procesos deben actualizar sus mapas de memoria a través del hipervisor, este debe asegurarse que los *guests* solamente acceden a memoria dentro de su reserva. Cabe señalar que en el caso de los *guest* la memoria que no tienen reservada no existe.

Finalmente se añade otra capa a la gestión de memoria para asegurarse que ésta sea contigua al asignarse a un *guest*. Por eso se crea una una tabla para traducir la memoria fisíca real del *host* en la memoria del *guest*. De esta manera el *guest* ve la memoria contigua (Figura 3.5).

- **Dispositivos de Entrada/Salida:**

  Por dispositivos de **E/S** (*Entrada/Salida*) nos referimos a dispositivos de bloque y a dispositivos de red. La E/S siempre es el cuello de botella de la virtualización pero con la

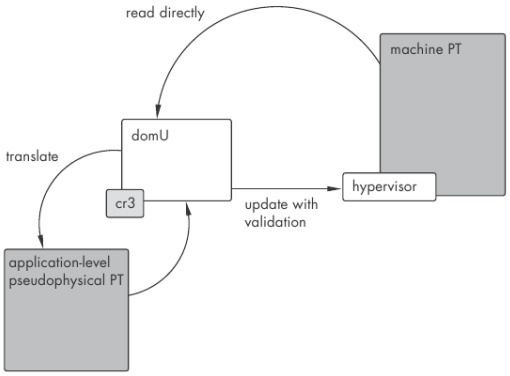


Figura 3.5: Esquema del uso de memoria en la paravirtualización.

paravirtualización se ofrece un rendimiento más próximo a la ejecución nativa.

En la paravirtualización el hipervisor se encarga de gestionar la E/S con la ayuda del *Dom0*. Ésta se gestiona mediante mecanismos que se denominan *device channels y virtual devices*. Los *device channels y virtual channels* son enlaces punto a punto entre el dispositivo *frontend* del *guest* y el dispositivo *backend* del *Dom0* implementados en un *buffer* en forma de anillo como se puede ver en la figura 3.6.

Este anillo está formado por regiones de memoria previamente seleccionada, de esta forma los *guests* trabajan directamente en la memoria sin tener que utilizar al hipervisor constantemente. En ciertos momentos los *guest* notifican al hipervisor que el anillo se ha actualizado y entonces el hipervisor ejecuta la acción que corresponde (enviar un paquete, escribir en disco, etc) a través del *dom0*.

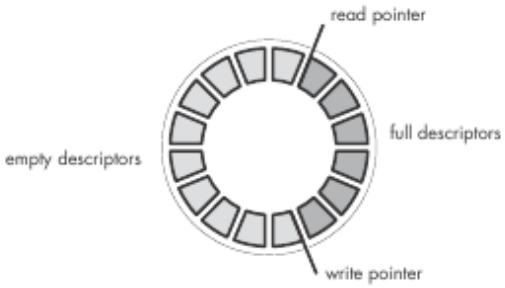


Figura 3.6: Estructura del buffer de E/S en paravirtualización.

- **Red:** En la comunicación con la red *(Network)* de datos, el hipervisor hace de canal de comunicación entre el dispositvo virtual del *guest* y el dispositivo virtual del *Dom0*. Una vez el *Dom0* recibe el paquete este lo envía hacia donde corresponde (ver figura 3.7).

- **Dispositivos de bloque:**

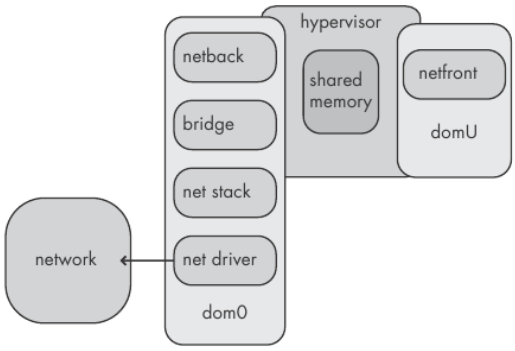


Figura 3.7: Esquema de red en la paravirtualización.

Entendemos como dispositivos de *bloque* (*block devices*), todos los dipositivos que pueden hacer funciones similares a las de un disco duro (particiones, *arrays*, imágenes de sistemas de ficheros, etc).

Los dispositivos de bloque se controlan igual que con los dispositivos de red, gestionando su acceso via *Dom0*. Se crea un dispositivo virtual en el *guest*, el cual es utilizado por el hipervisor como canal de comunicación y un dispositvo virtual en el *Dom0*. Cuando el *Dom0* recibe datos los escribe en el disco (ver figura 3.8) [TC10].

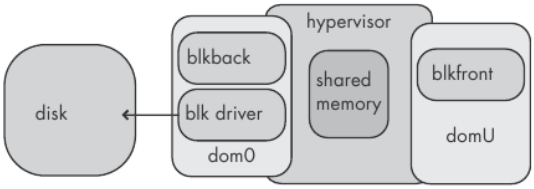


Figura 3.8: Esquema de acceso a dispositivo de bloque en la paravirtualización

### 3.2.3. Virtualización Completa

En inglés *Full virtualization/native virtualization*, se define como aquella que permite virtualizar todo tipo de sistemas operativos sin ser modificados y sin que los *guests* sepan que estan ejecutándose en un entorno virtualizado (ver figura 3.9). Esta última característica hace que el rendimiento sea más bajo porque los *guest* no colaboran con el hipervisor. Por el contrario este tipo de virtualización nos da la flexibilidad de poder virtualizar cualquier sistema operativo (*Windows*, *UNIX*,...).

Debido a su flexibilidad y utilidad constantemente surgen diferentes herramientas que hacen que los *guest* tengan un rendimiento más alto, sobretodo en Entrada/Salida.

La virtualización completa ha cambiado sustancialmente desde el año 2006 cuando aparecieron los primeros procesadores (*intel VT* y *AMD-V*) que permitian ejecutar el hipervisor

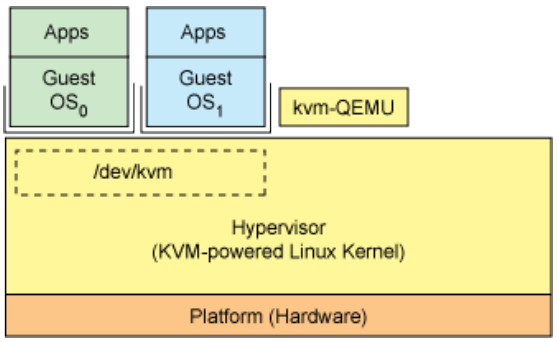


Figura 3.9: Esquema de una plataforma de virtualización completa

en un nuevo nivel privilegiado en el *Ring 0* gracias a la tecnología que incluían con soporte hardware a la virtualización. De esta forma, en este proyecto nos centraremos en explicar la virtualización completa ya que actualmente todos los procesadores integran asistencia *hardware* a la virtualización.

Definiremos inicialmente cómo gestiona la virtualización completa los recursos que proporciona a los *guests*:

- **CPU:** La gestión de la CPU en la virtualización completa es igual que en la paravirtualización. El *scheduler* o planificador del hipervisor del sistema gestiona en que momento un *guest* puede utilizar la CPU. En caso de que haya más de un núcleo en la cpu se puede reservar uno o más núcleos para un *guest* concreto, tarea que realiza de forma transparente el hipervisor.

- **Memoria:** La gestión de la memoria también es muy parecida a la que se hace en un entorno sin virtualizar. Cuando se ejecuta un *guest*, el hipervisor crea un espacio de direcciones contiguo para ésta. Este espacio de direcciones tiene las mismas propiedades que el espacio de memoria virtual de un sistema no virtualizado. De esta forma, este tipo de gestión permite al hipervisor ejecutar simultáneamente múltiples máquinas protegiendo la memoria de cada una de ellas. En la figura 3.11 podemos ver un esquema.

Como en todas las CPU modernas, la arquitectura *x86* contiene un sistema que traduce las direcciones virtuales de memoria a direcciones físicas de memoria. De esto se encarga la *Memory Management Unit (MMU)*. La virtualización de la MMU es posible pero ésta solo traduce de memoria virtual del *guest* a memoria física del *guest* y no tiene en cuenta la traducción de memoria física del *guest* a memoria física del *host*, que también es necesaria.

Una de las posibles soluciones es crear una tabla de páginas que traduce de memoria virtual del *guest* a memoria física del *host* llamada *shadow page table*. La *shadow page table* se construye a medida que hay fallos en la traducción, se reporta al hipervisor y éste las añade (figura 3.11). Otro inconveniente que aparece es que la tabla de páginas del *guest* y la *shadow page table* tienen que estar sincronizadas. De esta forma cada vez

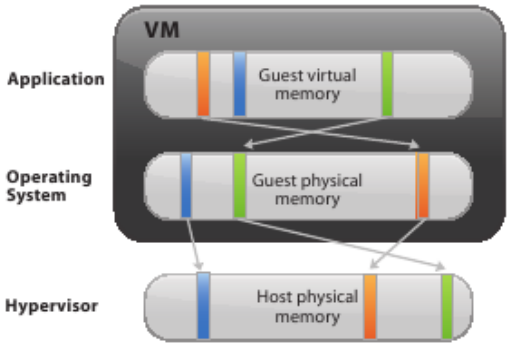


Figura 3.10: Esquema de memoria en virtualización completa

que el *guest* escribe en una página el cambio también debe hacerse en la *shadow page table*[KKL+07][VMw09].

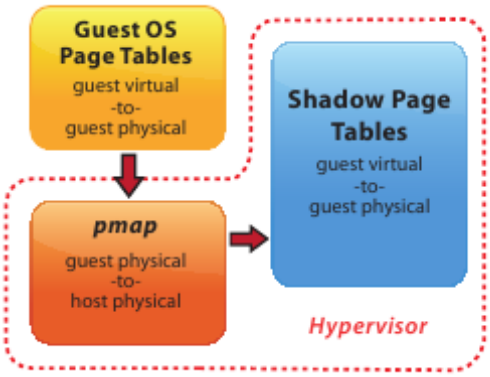


Figura 3.11: Esquema de la *shadow page table*

Las dos soluciones más importantes de este tipo de virtualización son *KVM* y *VMware*, que junto con *Xen* tienen diferentes optimizaciones para que se aproveche mejor la memoria física. Se trata de *Kernel Samepage Merging* (KSM) para KVM *Transparent Page Sharing* (TPS) para VMware y *memory Ballooning* en el caso de *Xen*:

- ***KSM/TPS***: Es la misma solución implementada en cada sistema con distinto nombre según el fabricante. Se trata de un sistema donde el hipervisor va mirando las páginas de cada una de los *guests* y une las páginas cuyo contenido es el mismo. Así pues, cuando un *guest* accede a memoria, en realidad está accediendo a una memoria compartida con otros *guests*.

  Este sistema de *pool* hace posible que efectivamente haya más memoria libre, debido a que compacta las partes comunes de código de sistema cuando estamos ejecutando máquinas virtuales con el mismo sistema operativo *guest* [KSM].

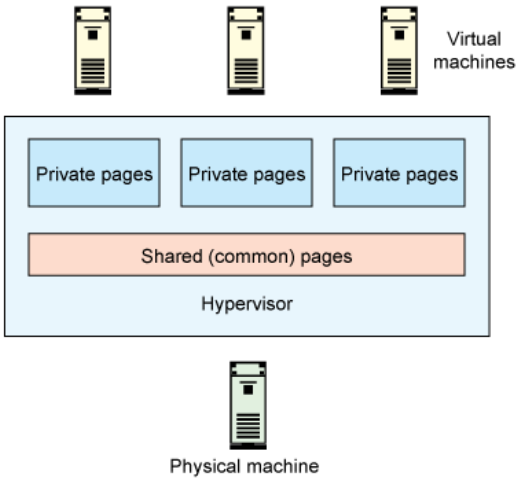


Figura 3.12: Esquema de KSM o TPS

- **Memory Ballooning**: Los sistemas *guest* están aislados por completo de las otras máquinas virtuales y del hipervisor. De esta forma no pueden saber lo que esta ocurriendo. En el caso de que la memoria del host se reduzca mucho entonces entra en escena el *ballooning*. Este mecanismo hace que la máquina esté atenta por si el hipervisor necesita más memoria. Entonces mediante un parámetro definido anteriormente cede memoria que no usa al hipervisor para que esta se la ceda a otro *guest* que tiene déficit de memoria siempre y cuando el *guest* que cede memoria tenga poca carga (figura 3.13).

  Este sistema se debe usar con mucho cuidado ya que podría darse el caso de que también el *guest* se quede sin memoria y se ralentice todo el sistema.

- **Dispositivos de E/S**: Antes de que salieran al mercado los procesadores con soporte a la virtualización, en la virtualización completa, los *guests* accedían a los dispositivos de E/S mediante la emulación que el hipervisor hacía del hardware. Ésta era una buena solución por el hecho de que abstraía el hardware a la perfección. La emulación en este caso se hacía tratando la interrupción del *guest* e imitando los comportamientos del hardware real gracias a la traducción binaria de instrucciones, hecho que daba una gran flexibilidad pero también era muy ineficiente.

  Con la llegada de los nuevos procesadores, las plataformas punteras de virtualización excepto en el caso *Xen*, han cambiado la forma de gestionar los recursos de E/S. Se han ido añadiendo controladores de paravirtualización en los *guests* y en el *hipervisor* para acceder a estos dispositivos, *virtio* para *KVM* y *guest-tools* para *VMware*. De esta forma se permite que los controladores del guest y del hipervisor se comuniquen directamente en el espacio de usuario hecho que facilita la E/S y mejora enormemente el rendimiento.

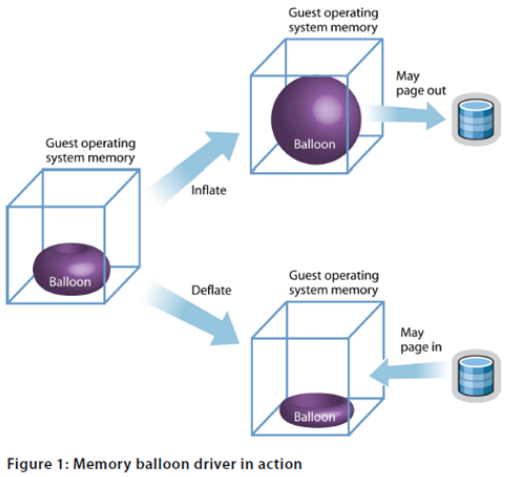


Figura 3.13: Esquema de gestión de memory ballooning

Igual que en la paravirtualización, con estos nuevos *drivers* se crean unos *buffers* en forma de anillo para comunicar *guest* e *hipervisor*. En *KVM* han avanzado un poco más el estado del arte y recientemente han implementando un sistema llamado *vhost-net* que reduce la latencia y aumenta el ancho de banda hasta 8 veces [LVH]. Estos *drivers* estan disponibles solamente para sistemas *guest Linux* a partir del kernel 2.6.30 y *Windows*.

### 3.2.4. Soporte hardware

Entendemos por soporte *hardware* a la virtualización el conjunto de mejoras que se implementaron a partir del año 2006 en los procesadores de *Intel* y *AMD*. En los procesadores actuales existen diferentes capas de privilegios llamadas *rings* o anillos. Los rings van desde el 0 (modo más privilegiado), al 3 (modo menos privilegiado). En estos rings se ejecutan diferentes partes del sistema operativo, por ejemplo en el *ring 0* se ejecuta el *kernel*, en los *rings 1* y *2* el interfaz para los dispositivos y en el *ring 3* las aplicaciones.

Uno de los elementos principales de la primera generación de procesadores con soporte a la virtualización fue la creación de una nueva capa de privilegios en la arquitectura de la CPU conocida como *ring -1* [RCS]. La ventaja de esta nueva capa de abstracción es que los hipervisores que soportan esta tecnología pueden ejecutarse en el *ring -1* y así los *guests* pueden acceder a la CPU en el *ring 0* justo como lo harían si la máquina se ejecutara sin virtualizar[EPT].

En la segunda generación implementaron las *Extended Page Tables* en *Intel* y *Rapid Virtualization Indexing* en *AMD*. Estos mecanismos permiten la creación de tablas de páginas que sirven para traducir las direcciones físicas del guest a las direcciones físicas del host. Así, el *guest* puede modificar su propia tabla y manejar directamente los fallos de página. Esto evita al *guest* tener que salir para que el hipervisor trate el fallo, que es el mayor *overhead* que produce en la virtualización cuando se accede a memoria[HAV].

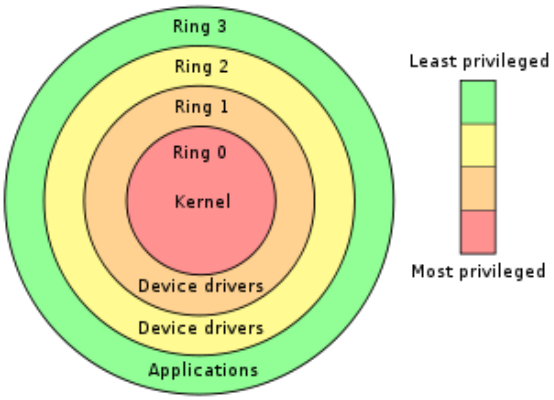


Figura 3.14: Esquema de la jerarquía de anillos de privilegios

### 3.2.5. Portabilidad y flexibilidad vs rendimiento de sistemas virtualizados

La virtualización, como hemos explicado anteriormente, aporta ventajas muy importantes entre las que podemos destacar la portabilidad, que puede ser muy útil a la hora de migrar un *guest* de un *host* a otro. Para que exista una verdadera portabilidad necesitamos que todos los sistemas ataquen a un mismo *pool* o sistema de almacenamiento. De esta forma, lo único que se necesita para migrar una máquina virtualizada es mover su configuración y arrancar la máquina en el otro *host*.

El poder aplicar esta característica presenta también algún inconveniente. El precio que hay que pagar es perder rendimiento de acceso a disco. En un *pool* de almacenamiento los discos que se crean para las máquinas virtuales son ficheros que contienen la imagen del sistema virtualizado. Los accesos a ficheros son más lentos que los accesos directos a dispositivos de bloques (*raw access*), por culpa entre otros aspectos de la fragmentación del disco. Además, los sistemas *guest* que atacan directamente a un dispositivo de bloques tienen mejor rendimiento debido a que el número de capas software por las que pasa para acceder a los datos es menor.

Otra ventaja que aporta el uso de ficheros sobre el acceso directo a dispositivos de bloque es la capacidad de aumentar el tamaño si es necesario, lo que nos permite crecer si se requiere más espacio de almacenamiento en la máquina virtual. Esta ventaja también puede obtenerse en los dipositivos de bloque creando un ***LVM*** *(logical volume manager)* [LVM]. Este sistema nos permite aumentar el espacio asignado en caliente, además de otras ventajas que explicaremos más adelante como el uso de *snapshots*.

## 3.3. Cloud

El concepto de *cloud* o *nube* se suele confundir con el concepto de virtualización. Si bien los dos conceptos están relacionados al poder coexistir, no son lo mismo. El *cloud* es un modelo de prestación de servicios alojados de forma externa y trasparente a los que se accede a través de la red. Es decir, se hace uso de un servicio sin conocer dónde o cómo funciona.

Estos servicios pueden ser software y entonces hablaríamos de *Software as a Service (**SaaS**)* e incluyen todo tipo de aplicaciones como podrían ser un *webmail* o plataformas que permiten prototipar, desarrollar, etc. Los *Platform as a Service (**PaaS**)* hardware, que se denominan también *Infraestructure as a Service (**IaaS**)* incluyen todo tipo de infraestructura hardware como servidores, espacio de disco, equipamiento de red, etc [CCV].

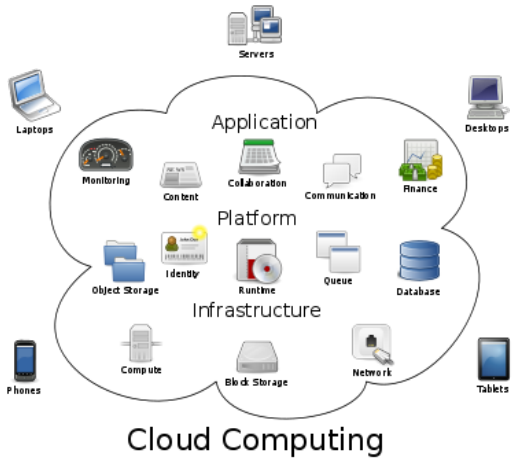


Figura 3.15: Visualización de un esquema cloud

La relación entre el *cloud* y la virtualización la encontramos sobretodo en el caso de *IaaS*[1]. Normalmente cuando se contrata un servicio de este tipo (como podría ser un servidor) el proveedor del servicio, mediante un interfaz web, crea una máquina virtual que corresponderá al nuevo servidor. La virtualización permite que los servicios que se contraten se hagan efectivos más rápido, dinámicamente ajustando de forma automática las necesidades y demandas del usuario. Sus principales ventajas son:

- **Implementación casi instantánea**: Cuando se decide contratar un servicio en la *nube*, este servicio estará disponible a los pocos minutos de haberlo contratado gracias al uso de plantillas y sistemas de clonación que aceleran el proceso de creación de una máquina virtual.

- **Escalabilidad**: Generalmente, el uso de la infraestructura de una organización sufre valles y picos de trabajos. Una buena solución es externalizar parte de los servicios para que absorban el trabajo que se produce en estos picos. Esto reduce significativamente los costes de mantener una infraestructura propia que en los valles está infrautilizada pero que es necesaria en los picos de trabajo.

- **GreenIT**: El consumo energético derivado de las TIC representa un gran gasto tanto económico como ecológico. El uso de sistemas *cloud* dónde unos pocos centros de datos (*CPD*) puedan contener centenares o miles de servidores virtualizados, que de otra forma

[1] El cloud se podria implementar perfectamente sin utilizar la virtualización

se encontrarían dispersas en cientos de pequeñas salas mal acondicionadas, supone un enorme ahorro energético y económico.

Por supuesto el uso del *cloud* también presenta algunos inconvenientes. Podemos destacar principalmente los siguientes:

- **Dependencia de la red**: El sistema *cloud*, al prestar servicios externos que no se encuentran en local tiene una total dependencia de la conexión que comunica con los servicios. Esto quiere decir que si en una organización falla la conexión de red o un *ISP* da problemas, todos los servicios externos quedaran inaccesibles hasta que la conexión vuelva a estar disponible. Asimismo, la velocidad de la conexión con *Internet* marcará la velocidad de acceso a los servicios.

- **Protección de datos**: Quizá el más importante de los inconvenientes si se trata de datos confindenciales. La ubicación de los datos que se guardan en la *nube* suele ser difusa y genera desconfianza. La legislación que pueda regir en el país donde están ubicados los datos puede no ser la misma que la que rige en el propio país del usuario. A esto también hay que añadirle que muchas veces los datos están en países donde los costes son más bajos, hecho que crea inquietud por problemas de seguridad física.

- **Costes por crecimiento**: Si la organización crece mucho y cada vez necesita replicar más servicios, el cloud se convierte en una opción cara comparada con comprar una infraestructura propia y virtualizar la infraesctrutura para prestar estos servicios a medio y largo plazo. El motivo es obvio, ya que los costes de mantener un CPD propio se reparten entre los servidores alojados. Si se tienen cincuenta el precio puede ser muy alto, pero con quinientos el coste queda mucho más repartido y es más asumible [CCP][CCD].

# CAPÍTULO 4

# Definición del entorno de pruebas

En este capítulo vamos a explicar qué infraestructura *software* y *hardware* hemos utilizado para confeccionar el entorno de pruebas. Definiremos sus características principales y explicaremos porqué se han escogido.

## 4.1. Objetivos de las pruebas

En este PFC queremos hacer un estudio del coste en términos de rendimiento que tiene virtualizar los servidores y ver qué ganancias en flexibilidad y portabilidad obtenemos. Ciertamente el uso de técnicas de virtualización penaliza el rendimiento final debido al *overhead* de gestión del sistema. Sin embargo la ganancia en flexibilidad, optimización del uso de recursos y la mejora constante del soporte a la virtualización compensan estos inconvenientes.

Con estas pruebas pretendemos medir el grado de pérdida que produce la virtualización así como elegir la mejor plataforma para su uso.

Los requisitos que debemos cubrir son los siguientes:

1. **Debemos poder virtualizar cualquier sistema operativo**

   Es un objetivo esencial. Pueder que los usuarios necesiten utilizar un tipo de plataforma concreta que sólo se encuentre en un sistema operativo determinado. Independientemente de qué plataforma de virtualización utilicemos, ésta debe poder ejecutar cualquier sistema operativo.

   De esta forma y tras el análisis realizado en el capítulo anterior optaremos por sistemas que permitan realizar una **virtualización completa**.

2. **La pérdida de rendimiento debe ser la mínima posible**

   Debemos tener en cuenta que los servidores deben prestar los servicios con una eficiencia similar a los no virtualizados. Es decir, que el usuario final prácticamente no note que la máquina esta virtualizada. En servidores con poca carga de uso puede no ser un elemento clave, pero en servidores con un gran uso de recursos es muy importante que el **rendimiento sea óptimo**.

3. **La plataforma de virtualización debe proporcionar flexibilidad y portabilidad**

   Junto con el gestor del entorno virtualizado, la plataforma de virtualización nos debe aportar flexibilidad y portabilidad para que la administración de servidores sea mucho más ágil comparándola con el sistema de administración actual. El uso de éstas técnicas duplica en el mejor caso el número de servidores a gestionar (todos los reales más sus correspondientes virtuales), lo que implica que necesitamos herramientas de gestión.

## 4.2. Recursos hardware

### 4.2.1. Criterios de selección

Los criterios de selección de *hardware* deben ajustarse a las características de una máquina moderna y similar en prestaciones a los servidores utilizados actualmente por los grupos de investigación.

Actualmente todos los servidores utilizados en el RDlab se encuentran ubicados en el ***CPD*** (*Centro de Procesamiento de Datos*) que la *UPC* tiene ubicado en el Edificio Omega, situado en el *Campus Nord*. Todos los servidores se encuentran dispuestos en armarios o *racks* para mejorar la eficiencia de almacenamiento y refrigeración. De esta forma los servidores deben cumplir unos requisitos físicos para su instalación así como unos criterios de eficiencia energética.

Figura 4.1: Instalaciones del CPD de la UPC

### 4.2.2. Equipamiento

Una vez fijados los criterios de selección de hardware, definiremos el equipo que utilizaremos para realizar las pruebas. Utilizaremos un servidor *Dell Poweredge r310* con procesador *Intel Xeon X3450* de cuatro núcleos, 8GBytes de RAM, cuatro discos *NearLine-SAS* de 1TByte y una tarjeta de red a 1 Gbyte.

Figura 4.2: Imágenes del servidor *Dell R310*

## 4.3. Recursos software

### 4.3.1. Criterios de selección

Los criterios de selección de software se han basado principalmente en consultar los distintos foros y artículos para obtener los programas de referencia. De esta forma los programas utilizados permiten comparar sus resultados con los ya existentes en Internet. Además estas herramientas deben cumplir los requisitos funcionales y no funcionales que hemos descrito en el capítulo dos.

El software utilizado no debe añadir ningún coste económico al proyecto. De esta forma, en el caso de los sistemas operativos, de la gran cantidad de posibilidades existentes obtenemos un subconjunto más manejable. En el Departamento de *LSI* la plataforma de uso común para los servidores utilizados en investigación es Linux. Además prácticamente todas las aportaciones que se realizan en temas de virtualización aparecen inicialmente en este sistema, por lo que se decidió escoger esta familia de los sistemas Unix.

Cabe destacar que existen algunas alternativas híbridas en su modelo de negocio como ***Red Hat Enterprise***. Este sistema operativo basado en GNU/Linux puede descargarse gratuitamente por la web sin pagar ningún coste. Sin embargo todos los aspectos relativos al soporte de este sistema son de pago. Las actualizaciones o cualquier duda sobre su instalación o uso es atendida únicamente si se ha contratado el soporte correspondiente.

En el ámbito de las plataformas de virtualización hablaremos de las tres más punteras que existen actualmente. Se trata de *KVM*, *Xen* y *VMware*. Estas tres alternativas permiten el tipo de virtualización completa que coincide con nuestros requisitos.

*VMware* es la plataforma más extendida para grandes organizaciones, añade un coste económico muy alto por cada licencia de uso. Una licencia básica para 3 servidores tiene un coste de 455€.

*Xen* es un software libre y gratuito debido a que es una implementación que ha desarrollado la comunidad de software libre mediante una modificación del *Kernel* de *Linux*. Este programa añade las funciones de un hipervisor al sistema para gestionar el uso de *guests*.

Por lo que respecta a la plataforma *KVM*, es otra solución gratuita y libre a la que gran parte de la comunidad está dando apoyo. En los dos últimos años ha creciendo significativamente su uso hasta convertirse en la alternativa mayoritaria. Actualmente esta plataforma de virtualización se incluye en todos los repositorios oficiales de las principales ditribuciones *Linux*.

Las tres plataformas descritas tienen características similares en cuanto a su funcionalidad. Permiten la migración de máquinas virtuales, pueden modificar los recursos asignados a una máquina virtual y permiten ejecutar sistemas *guests* en virtualización completa. Su gestión de disco permite también el uso tanto de imágenes como particiones físicas de disco. De estas tres plataformas propuestas evaluaremos *KVM* y *Xen* y analizaremos cual de estas dos ofrece un mejor rendimiento y se adapta mejor a nuestros objetivos.

Un gestor de entornos virtualizados es una aplicación o plataforma que presta los servicios para gestionar un conjunto de máquinas virtuales y reales. Esto significa que mediante esta aplicación o plataforma se puede controlar al menos las acciones de crear, modificar los recursos que utiliza y encender o apagar una máquina virtual. También suele añadir funciones para gestionar el sistema *host*.

*Convirt*, *Ovirt*, *virt-manager* y *open-stack*, son los gestores gratuitos más importantes y populares que existen en la actualidad. Soportan la gestión tanto de sistemas basados en *KVM* como *Xen*. En el caso de *Convirt* cabe señalar que también dispone de una versión de pago, denominada *Enterprise*, que incluye algunas funcionalidades extra y el soporte al usuario. En el caso de *VMware* éste dispone de su propio gestor de sistemas virtuales que está incluido en el paquete de compra de su licencia.

### 4.3.2. Sistemas operativo Host

Como ya hemos comentado anteriormente tenemos múltiples opciones a la hora de escoger los sistemas operativos que vamos a utilizar. La selección ha dependido además de los criterios anteriores de la disponibilidad de versiones existentes en cada distribución Linux a la hora de realizar el proyecto [WLD].


Figura 4.3: Logos de las pricipales distribuciones *Linux*

Es importante destacar que durante el primer semestre del año 2012 varias distribuciones han ido actualizando sus versiones. Desgraciadamente estas no han podido ser siempre incorporadas debido a las restricciones temporales y a la complejidad técnica de su preparación.

Consecuentemente, los tres sistemas sistemas escogidos para evaluarse como *host* fueron los siguientes: *Ubuntu Server 10.04 LTS*, *Debian 6 Squeeze* y *CentOS 5*.

### *Ubuntu Server 10.04 LTS*

*Ubuntu* es un sistema operativo muy popular gracias principalmente al éxito de su versión para equipos portátiles y de sobremesa. Su primera versión se liberó en el año 2004 y año tras año ha ido aumentando su base de usuarios.

Debido a que originariamente estaba orientada al usuario novel, con un enfoque al pragmatismo y la mejora de la experiencia del usuario, este sistema diferenció dos gamas según su uso. La versión *desktop* es la encargada de incorporar rápidamente todos los avances que mejoren la experiencia de usuario, su ciclo de vida es anual. La gama *server* ***LTS*** (*Long Time Service*) está orientada a los servidores. Incorpora un ciclo de vida mucho más largo, 5 años, y libera nuevas versiones cada 2 años.

Esta distribución tiene su repositorio principal mantenido por canonical, una empresa privada y financiada por el empresario sudafricano *Mark Shuttleworth*, y por una gran comunidad de desarrolladores que aportan su tiempo y trabajo gratuitamente.

Utiliza un *kernel* linux y su origen es una bifurcación del código base de la distribución *Debian*. El objetivo inicial era hacer de *Debian* una distribución más fácil de usar y entender para los usuarios finales, corrigiendo errores y haciendo más asequibles algunas tareas como la gestión de aplicaciones. Al ser un sistema gratuito se nutre principalmente de las aportaciones libres que realiza la comunidad de desarrolladores así como de donativos y la venta de soporte técnico para empresas.

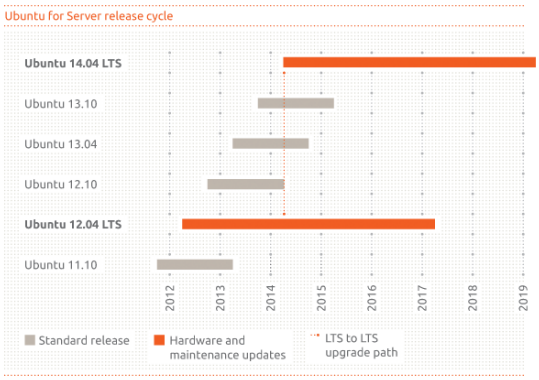


Figura 4.4: Ciclo de versiones LTS

*Ubuntu* está disponible para procesadores *x86* de *32* y *64 bits*. Ofrece soporte para la mayoría de *hardware* que se encuentra en el mercado y periódicamente realiza actualizaciones como se puede observar en la figura 4.4.

Durante estos cinco años canonical ofrece soporte para nuevas actualizaciones de aplicaciones, parches de seguridad (*bugs*) y mejora del soporte *hardware*. Esto permite mantener el

sistema actualizado añadiendo las nuevas mejoras que puedan surgir. La virtualización es actualmente una una tecnología puntera y por tanto podemos obtener una ventaja en su gestión si las mejoras que se produzcan se incorporan automáticamente en la propia distribución [USO].

*Ubuntu* ofrece soporte propio para *KVM* en sus repositorios, y permite el uso de *Xen* gracias al soporte externo de algunos programadores. Éstos ha modificado un kernel para *Ubuntu* de forma que contemple el soporte al hipervisor de *Xen*.

En los últimos años este sistema ha tenido una tendencia creciente en el uso de administración de servidores web según *w3techs.com* y actualmente tiene una cuota de uso de un 20 % de los servidores que utilizan *Linux* en todo el mundo [WTU]. Como apunte decir que *wikipedia* tiene en sus servidores web un sistema *Ubuntu*.

#### *Debian 6 Squeeze*

*Debian* es un sistema operativo de código libre desarrollado por una comunidad de más de mil voluntarios alrededor del mundo. Este sistema nació en el año 1993 gracias a un proyecto específico de código libre con el mismo nombre. Su nombre se debe a su fundador, *Ian Murdock*, que combinó su nombre con el de su novia, *Debra Lyn*. La idea inicial con la que nació este proyecto fue la de crear un sistema *GNU* usando el kernel de linux como núcleo.

*Debian* tiene un modelo de desarrollo independiente de las empresas creado por y para los propios usuarios. Esto es posible debido a que no existe ninguna dependencia comercial puesto que no vende directamente su software. Éste se encuentra disponible gratuitamente en *Internet* y permite a personas o empresas distribuirlo comercialmente siempre que se respete la licencia *GNU*.

Este sistema operativo proporciona soporte para más de 10 arquitecturas entre las que podemos destacar: *i386*, *amd64*, *alpha*, *sparc*, *powerpc* y *arm*. La versión utilizada en las pruebas, la número 6 que se denomina *Squeeze*, cuenta con casi 30.000 paquetes en sus repositorios de software. Cabe señalar que los nombres de las versiones se eligen utilizando personajes de la película "*Toy Story*".

*Debian* siempre mantiene 3 versiones disponibles, la versión *oldstable* es la versión anterior y carece de soporte. La versión *stable* es la que se mantienen y recibe soporte oficial y la *unstable* recoge las últimas actualizaciones de los programas y sirve de núcleo para la próxima versión estable.

Los desarrolladores de este sistema operativo suelen liberar cada cuatro o cinco años una versión *stable* con las nuevas actualizaciones. Estos ciclos de vida tan largos se justifican entre otros motivos por la gran cantidad de arquitecturas soportadas. Debido a que únicamente se recogen actualizaciones de versiones de aplicaciones con los nuevos lanzamientos, *Debian* es una distribución muy robusta pero atrasada tecnológicamente.

Actualmente la distribución tiene soporte tanto para *Xen* como para *KVM* desde los repositorios oficiales. Hoy en día *Debian* es la distribución más utilizada en servidores *Linux* con una cuota del 30 % según w3techs.com [WTD].

#### *CentOS 5.6*

*CentOS (Community ENterprise Operating System)* es una distribución libre de Linux basada en la de pago *Red Hat Enterprise Linux*. Es el fruto de la compilación llevada a cabo por voluntarios del código fuente que libera *Red Hat* bajo la licencia *GNU* de *Linux*. En el año 2004 crearon su primera versión basada en *RedHat 2*.

Su distribución base (*Red Hat*) se compone de software libre y código abierto. Todo el código se publica en formato binario juntamente con las actualizaciones para los suscriptores de pago. Sin embargo, como es requerido en la licencia *GNU*, deben libera todo el código fuente basado en sistemas abiertos. Los desarrolladores de *CentOS* usan ese código fuente para crear el producto final que es muy similar a l original y está a disposición del público. Sin embargo, carece del soporte técnico para la administración que sí brinda *RedHat*.

*CentOS* está disponible para la mayoría de arquitecturas que *Red Hat* soporta. Podemos destacar principalmente las siguientes: *x86*, *amd64*, *itanium*, *PowerPC*. *CentOS* es un sistema que se basa en las actualizaciones que *Red Hat* libera, lo que implica que muchas aplicaciones deban ser descargadas mediante el código fuente y compiladas explícitamente por el usuario en su sistema.

Tanto *KVM* como *Xen* pueden ejecutarse en *CentOS* mediante el uso de los repositorios oficiales. Cabe señalar que recientemente *Red Hat* anunció que no va a seguir dando apoyo a *Xen* y se centrarán en desarrollar y mejorar el soporte para *KVM* [RHX].

Esta distribución es la segunda más utilizada en el mundo de los servidores *Linux* con una cuota de uso de casi el 29 % [WTC].

### 4.3.3. Sistemas operativos Guest

La elección de un sistema operativo *guest* no tiene relevancia hablando en términos de rendimiento. La virtualización completa deber de ser igual de efectiva sea cual sea el sistema operativo *guest*.

Si bien es cierto que el uso de controladores (*drivers*) específicos de paravirtualización pueden incrementar el rendimiento, es razonable pensar que éstos se encontrarán disponibles para los sistemas guest más habituales. De esta forma al prescindir de ellos nos centraremos en el rendimiento puro de la virtualización y no en las optimizaciones que pueda aportar el sistema *guest*.

Por este motivo se ha escogido como sistema representativo *Ubuntu 10.04 LTS x64 Server* para hacer las pruebas de rendimiento en las máquinas virtuales. Casi todas las máquinas que se administran en el *RDlab* ejecutan *Ubuntu* como sistema operativo con una imagen configurada y preparada por el personal del *RDlab*.

### 4.3.4. Plataforma de virtualización

Después de descartar la plataforma *VMware* por no ser gratuita, nos centraremos en describir las dos plataformas libres que implementan la virtualización completa: *KVM* y *Xen*.

### *KVM*

***KVM*** *(Kernel-based Virtual Machine)* es una plataforma de virtualización que implementa el paradigma de la virtualización completa y está totalmente desarrolada usando código libre. Esta plataforma fue creada y esta mantenida por la empresa *Qumranet*, recientemente comprada por *Red Hat*.

*KVM* esta formada por un módulo del *kernel* de *Linux* denominado *kvm.ko* y un conjunto de herramientas que se ejecutan en el espacio de usuario. El componente para el núcleo está incluido a partir del de la versión **2.6.20**.

El módulo del *Kernel* le permite realizar las funciones de hipervisor tratando a las máquinas virtuales como si fuesen un proceso más del sistema *host* (ver figura 4.5). Esta característica hace que la virtualización se beneficie de las mejoras de rendimiento que van surgiendo en el continuo desarrollo del *kernel*. Aspectos como por ejemplo el planificador de procesos o la gestión de la memoria tienen un gran impacto en el rendimiento.

El hecho de que el hipervisor se ejecute como un proceso en el espacio del kernel y no de aplicaciones, hace que éste forme parte de los hipervisores tipo 1 o *Bare Metal* que se describieron en el capítulo anterior.

Esta plataforma introduce un nuevo modo de ejecución a los ya existentes (*Kernel* y usuario) en el sistema. Este nuevo modo de ejecución se llama *guest* y se usa para ejecutar el código del sistema operativo de la máquina virtual que no es E/S. En el momento que se deben ejecutar instrucciones que impliquen E/S, se sale del modo *guest* y se utiliza el modo usuario mediante el software *QEMU*. *QEMU* es una plataforma que intercepta todas las peticiones de E/S de los *guest* y las encamina hacia el modo usuario dónde se emulan para ser ejecutadas.

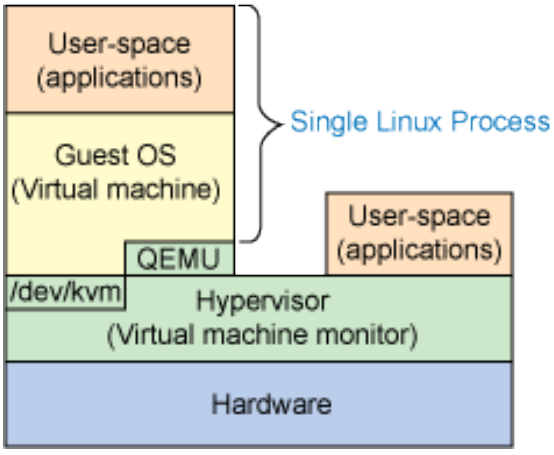


Figura 4.5: Esquema de *KVM* dentro del sistema operativo

Como se puede ver en la figura 4.5, *KVM* crea un fichero especial de dispositivo denominado */dev/kvm*. Mediante este dispositivo se provee memoria a las máquinas virtuales y su gestión. También es dónde se crean las tablas para traducir la memoria física del *guest* a memoria física del *host* [DKV].

Actualmente se estan introduciendo mejoras para aumentar el rendimiento de E/S como por ejemplo el uso de nuevos controladores de paravirtualización como *virtio* para las operaciones de E/S y *vhost-net* para la red de datos.

*KVM* necesita ejecutarse en procesadores de la familia *x86* que tengan soporte hardware para la virtualización (*Intel VT o AMD-V*) para poder proporcionar una virtualización completa al sistema *host*.

Esta es la plataforma que recientemente ha dado un salto cualitativo más grande respecto al resto de plataformas competidoras como *Xen* o *VMware*. Una prueba de ello es que *Red Hat* compró *Qumranet* y dejó de dar soporte para *Xen* centrándose solamente en desarrollar y mejorar esta plataforma de virtualización.

#### *Xen*

*Xen* es una plataforma de virtualización que salió a la luz en el año 2003 de la mano de la Universidad de *Cambridge*. Inicialmente esta plataforma se había desarrollado para ejecutar máquinas virtuales con paravirtualización. Con la llegada de los procesadores que incorporaban soporte a la virtualización, se añadió el soporte a la virtualización completa.

Esta plataforma está basada en código libre y se desarrolla gracias a la comunidad de usuarios. Es una de las plataformas más extendidas y el hecho de que sea una plataforma tan madura hace que sea robusta y sólida.

*Xen* necesita modificar el *kernel* de *Linux* para que este pueda realizar las tareas de hipervisor. Esta modificación que se hace sobre el *kernel* cambia la estructura del sistema operativo base. De esta forma la estructura se compone del hipervisor (capa de abstracción de hardware) y el *Dom0* (gestor de recursos hardware) tal y como se explicó anteriormente [XEO].

Recientemente *Ubuntu y Red Hat* anunciaron que no seguirían dando soporte para *Xen* y se centrarían en desarrollar *KVM* por lo que su futuro es incierto.

### 4.3.5. Gestor de entornos virtualizados

El gestor de entornos virtualizado o monitor de máquinas virtuales es una parte muy importante en una infraestructura virtualizada. La posibilidad de crear varias máquinas virtuales en un mismo servidor físico dispara el número de sistemas a monitorizar. Estos gestores existen para facilitar la administración de toda la infraestructura asociada.

Los tres gestores gráficos más importantes actualmente son: *virt-manager*, *Ovirt* y *Convirt*.

- *Virt-manager* es una aplicación desarrollada en *python*, que se encuentra disponible en los repositorios oficiales de las tres distribuciones probadas. Es un gestor muy básico que puede administrar máquinas remotamente mediante conexiones *ssh*.

*Ovirt* es una aplicación web gratuita con un alto grado de complejidad. Está pensada para grandes infraestructuras con centenares de *pools* de almacenamiento y máquinas virtuales. Este sistema permite utilizar todas las opciones que brindan tanto *KVM* como *Xen*.

*convirt open-source*. Esta aplicación web escrita en *python* y desarrollada por la empresa *convirture*. Permite la ejecución y control tanto de plataformas *KVM* como *Xen* así como parametrizar todas las funciones. También realiza las operaciones de migración, encendido, apagado y modificación que ofrecen las plataformas *KVM* y *Xen*.

Además permite crear para el soporte de disco sistemas *lvm* desde el propio interfaz lo que nos confiere gran flexibilidad. Contempla la posibilidad de crear plantillas de máquinas virtuales y tiene una interfaz agradable a la vista y muy intuitiva. Recientemente ha lanzado la versión 2.1.1 que ha mejorado mucho el consumo de recursos que precisa y ha resultado un salto cualitativo importante.

Después de instalar y probar las distintas opciones, por facilidad de uso, instalación y características que se ha considerado que la mejor plataforma es actualmente *convirt*.

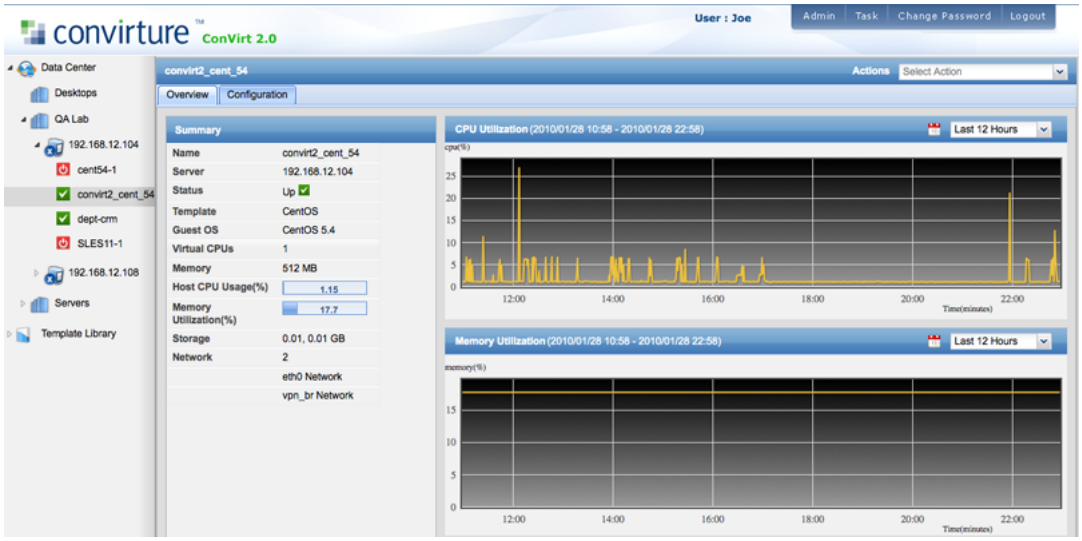


Figura 4.6: Captura de pantalla del interfaz de convirt

## 4.4. Arquitectura del entorno de pruebas

Nuestro entorno de pruebas está formado por un servidor *Dell poweredge r310* donde hemos instalado en distintas particiones primarias de disco cada uno de los sistemas operativos anteriormente mencionados. Después hemos realizado la configuración y parametrización del sistema para permitir el uso de los distintos sistemas de virtualización.

Una vez instaladas las plataformas se han creado cuatro máquinas virtuales con acceso directo a un partición de disco mediante *lvm* y cuatro máquinas con fichero de imagen de disco.

El objetivo de esta configuración es certificar que la velocidad de acceso directo a la partición es mejor que el acceso via fichero a una imagen de disco. Las máquinas virtuales se han creado todas iguales, asignándoles 2 Gbytes de RAM y 1 core del procesador.

La red está configurada mediante lo que se denomina modo ***bridge*** de forma que tenga acceso directo a la red. De esta forma la máquina virtual dispondrá de acceso propio a la red independiente del que usa el *host*. Se ha configurado de esta forma debido a que necesitamos que las máquinas virtuales sean totalmente accesibles desde fuera sin necesidad de un redireccionamiento de puertos que implicaría el utilizar una configuración tipo ***NAT*** (*Network Address Translation*).

A continuación definiremos las cuatro aplicaciones de *benchmarking* que nos permitirán estudiar como afecta la virtualización al rendimiento de un sistema. Además nos dará un criterio para poder determinar cual es la mejor solución cuantitativamente.

Las pruebas de rendimiento se realizarán en cada sistema operativo, en cada plataforma y en todas las máquinas virtuales con el fin de saber como escala el rendimiento a medida que aumentamos el número de máquinas virtuales en ejecución. La aplicaciones de benchmarking que vamos a utilizar son las siguientes:

1. **CPU:**

   Para estudiar el rendimiento de la CPU hemos utilizado la aplicación de referencia ***Linpack***[1]. Este benchmark o prueba de rendimiento desarrollado en 1976 por Jack Dongarra es una de las herramientas mas extendidas en sistemas científicos y supercomputación. *Linpack* calcula el rendimiento puro de la CPU resolviendo un problema de matrices generadas por un generador de números aleatorios.

   La métrica que utiliza linpack son los ***FLOPS*** (*FLoating Point Operations Per Second*) que determin a el número de operaciones en coma flotante por segundo. En nuestro caso los valores seran de GFlops (mil millones de operaciones de coma flotante por segundo) debido a la gran potencia de los procesadores Xeon [LIN].

2. **RAM:**

   Para medir el rendimiento de la memoria RAM hemos utilizado el programa RAMspeed, un benchmark openSource muy extendido que forma parte de la suite *phoronix*, una de las más famosas web de análisis de dispositivos.

   RAMspeed puede utilizar dos tipos de benchmarks, uno para medir el ancho de banda máximo y otro mucho mas próximo a la realidad que es sensible a latencias. La métrica que utiliza RAMspeed son los MBytes/s [RAM].

3. **Disco:**

   En cuanto al rendimiento de disco utilizaremos una de las más famosas aplicaciones para sistemas Unix de rendimiento de disco, *bonnie++*. Este programa es la evolución del programa *bonnie* escrito por Tim Bray.

[1] En http://www.top500.org/project/linpack se informa del top500 basándose en linpack

*Bonnie++* permite medir la velocidad de lectura y escritura, el número de búsquedas por segundo y la latencia de los accesos secuenciales o aleatorios. La métrica que utiliza son los KBytes/s [DIS].

4. **Red:**

   Para realizar el test de rendimiento de la red de datos utilizaremos la herramienta *IPerf*. Este programa escrito en c++ se encuentra en los repositorios de todas las distribuciones Linux y permite crear flujos TCP y UDP. Nos permite medir de una forma real el ancho de banda y la latencia existente entre dos puntos de red (direcciones IP).

# CAPÍTULO 5

## Benchmarking y resultados

En este capítulo describiremos los resultados que se han obtenido al ejecutar las diferentes aplicaciones de *benchmarking* seleccionadas. El objetivo de este capítulo es evaluar el comportamiento real de cada uno de los componentes que conformarán la solución propuesta.

Los resultados obtenidos se presentarán en gráficos tridimensionales de barras para facilitar su comparación, ya que debido a la gran cantidad de tests realizados pueden ser difíciles de seguir. Igualmente la estructura que se seguirá en este capítulo es la de realizar una breve explicación de las pruebas realizadas y adjuntar sus resultados correspondientes. En los anexos de este documento pueden consultarse las pruebas realizadas con detalle en forma de tablas de texto así como los parámetros de configuración utilizados.

## 5.1. Sistemas hosts

En esta sección se presentarán los resultados obtenidos de las ejecuciones en todos los sistemas *hosts* seleccionados. De esta forma obtendremos una medida cuantitativa del rendimiento bruto de nuestro servidor físico y nos permitirá comparar las mejoras que pueda introducir el uso de una u otra distribución *Linux*.

- **CPU**

  Para la realización del test de CPU se ha seleccionado el programa *linpack* que realiza cálculos sobre matrices. Este software realiza un cálculo intensivo de la CPU utilizando todos los cores disponibles simultáneamente. Los resultados se han obtenido compilando el código fuente de la aplicación para sistemas de 64 bits y ejecutando directamente el programa mediante el comando:

  *xlinpack_xeon64.*

En la gráfica de CPU de las figuras 5.1, 5.3 y 5.5 podemos observar como el sistema operativo ejecutándose en el servidor físico escala casi perfectamente. También se puede observar que aparece un cierto *overhead* de gestión a medida que se van ejecutando un mayor número de procesos simultáneamente.

Como se puede comprobar claramente en las gráficas referenciadas la diferencia entre los distintos sistemas evaluados no es significativa. Esto es debido a que todos los sistemas *Linux* comparten gran parte del código fuente. De esta forma el rendimiento bruto no es un criterio válido para seleccionar a un sistema sobre otro.

- **RAM**

  Para realizar los test de rendimiento del acceso a memoria RAM se ha utilizado el programa *RAMspeed*. Este programa nos permite calcular el ancho de banda existente mediante el uso de uno o varios procesos simultáneos. Los resultados mostrados en las gráficas se obtuvieron utilizando la siguiente llamada al programa:

  *ramsmp -b 3 -g 2 -p1*

  En la gráfica de uso de memoria *RAM* de las figuras 5.1 y 5.3, 5.5 podemos observar que el aumento de procesos que utilicen la memoria física permite maximizar su rendimiento. Sin embargo cabe destacar que cuanto el sistema hace uso del *SWAP* el rendimiento decrece significativamente. El gran ancho de banda que proporciona el bus de conexión y a las tecnologías actuales para memorias *DDR3-1333Mhz* escalan adecuadamente.

- **Acceso a disco**

  El test del E/S sobre los discos del sistema se realizó utilizando el programa *bonnie++*. Para obtener unos resultados fiables que no estén condicionados por el uso de las *caches* del sistema, se recomienda utilizar como parámetro el doble de la memoria física disponible. En nuestro caso el comando utilizado en el test fue:

  *bonnie++ -b*

  En la gráfica de *E/S de disco* de las figuras 5.2, 5.4 y 5.6 podemos observar que el rendimiento del disco decrece muy rápidamente al aumentar el número de procesos. Este comportamiento es debido a que los discos magnéticos siguen siendo dispositivos muy lentos en comparación con la *CPU* o la memoria *RAM*.

  Si bien tecnologías como los discos sólidos (***SSD**, Solid State Disk*) mejoran substancialmente el acceso, su coste hoy por hoy los descarta como opciones reales. De esta forma tenemos que el acceso al disco es uno de los cuellos de botella respecto el rendimiento.

- **Red de datos**

  Debido a que el tráfico de red es difícilmente predecible y deseamos obtener unos valores reproducibles y estables, se optó por utilizar dos tipos de simulaciones para las comunicaciones.

El primer test de rendimiento se realizó a través del dispositivo interno de red *loopback*, lo que permite establecer la cantidad de *bits* por segundo que puede manejar el kernel. En la segunda prueba se evaluó la transmisión de datos entre el sistema real (*host*) y diferentes máquinas virtuales, con el objetivo de comprobar la escalabilidad del sistema.

El comando utilizado para los test fue el siguiente:

*iperf -c <direccion_ip> -w 65536 -t 60*

En la gráfica de *red de datos* de las figuras 5.2, 5.4 y 5.6 podemos observar que el comportamiento es lineal y por tanto el rendimiento se mantiene. El objetivo de esta prueba no es obtener el rendimiento bruto de la comunicación de red, si no poder compararlo con el obtenido en las máquinas virtuales.

### 5.1.1. Ubuntu 10.04 LTS

Tal y como se ha comentado anteriormente las diferencias existentes entre las distintas alternativas *Linux* para el sistema *host* son mínimas. De esta forma realizaremos una breve lectura de los resultados obtenidos en los distintos test para ver cómo se comporta el sistema *host* al aumentar el número de máquinas virtuales que gestiona.

Es importante hacer notar que las pruebas se han realizado de forma que los recursos físicos existentes (cores del procesador, memoria física) son superiores a los consumidos por las máquinas virtuales en ejecución, de forma que no hay competición entre los distintos procesos por un mismo recurso. Esta configuración del sistema es la deseable prácticamente en todos los escenarios ya que en otro caso es totalmente imposible garantizar un rendimiento mínimo de la máquina virtual.

Podemos observar en los test de uso de los cores de la ***CPU*** (ver figuras 5.1, 5.3 y 5.5) cómo el rendimiento obtenido escala proporcionalmente. Consecuentemente observamos que el planificador del sistema (*scheduler*) asigna los recursos (cores) de ejecución del sistema equitativamente. Esta política de planificación nos permite asegurar que las distintas máquinas virtuales tendrán un rendimiento sostenido y no se verán afectadas entre sí.

En el caso de la utilización de la memoria RAM física observamos que también se cumple el criterio de proporcionalidad en la asignación del rendimiento bruto. De esta forma las diferentes máquinas virtuales mantienen de forma constante el rendimiento tanto en los accesos a memoria como en uso de los cores de la CPU.

Podemos destacar que el rendimiento total bruto de la memoria aumenta conforme sube el número de procesos simultáneos alcanzando el máximo con 3. Esto es debido a la tecnología y ancho de banda de la memoria utilizada, que varia en cada caso, y que en nuestro servidor es DDR3 1333Mhz.

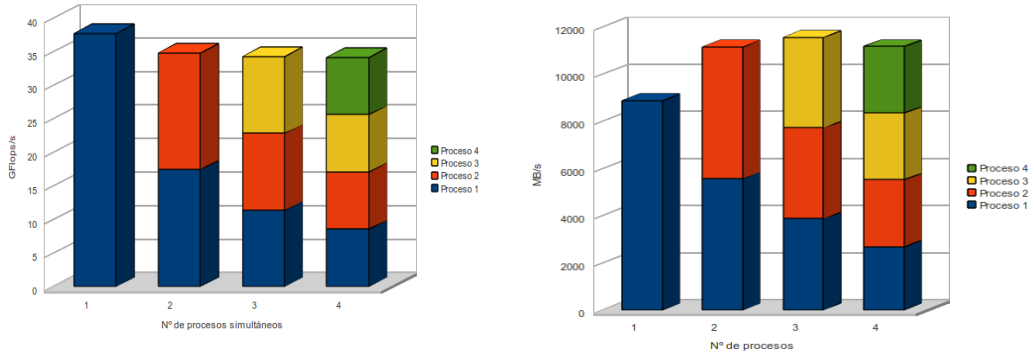


Figura 5.1: Resultados de los test de CPU y RAM realizados en el sistema *host Ubuntu* (4 cores)

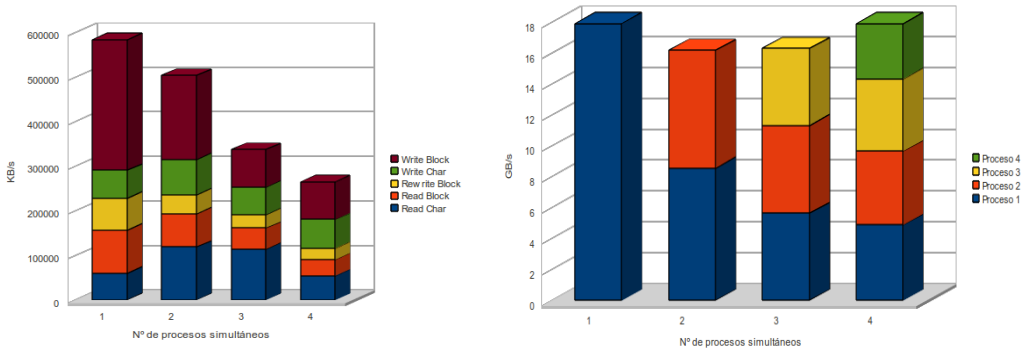


Figura 5.2: Resultados de los test de disco y red realizados en el sistema *host Ubuntu* (4 cores)

### 5.1.2. Debian 6

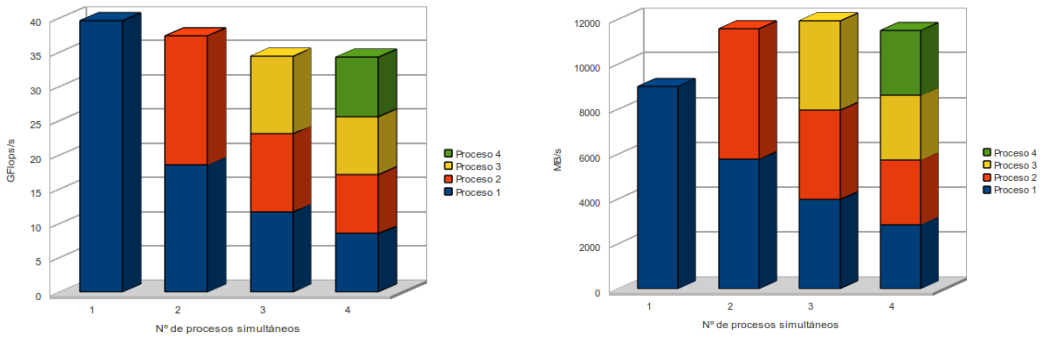


Figura 5.3: Resultados de los test de CPU y RAM realizados en el sistema *host Debian 6.0* (4 cores)

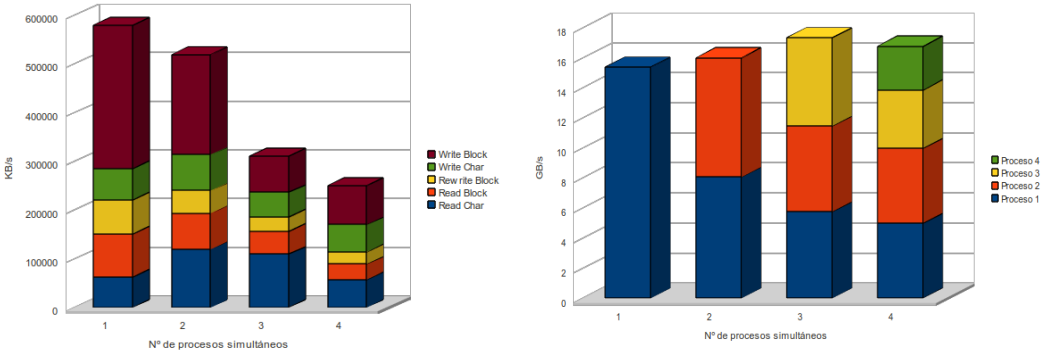


Figura 5.4: Resultados de los test realizados de disco y red en el sistema *host Debian 6.0* (4 cores)

### 5.1.3. CentOS 5

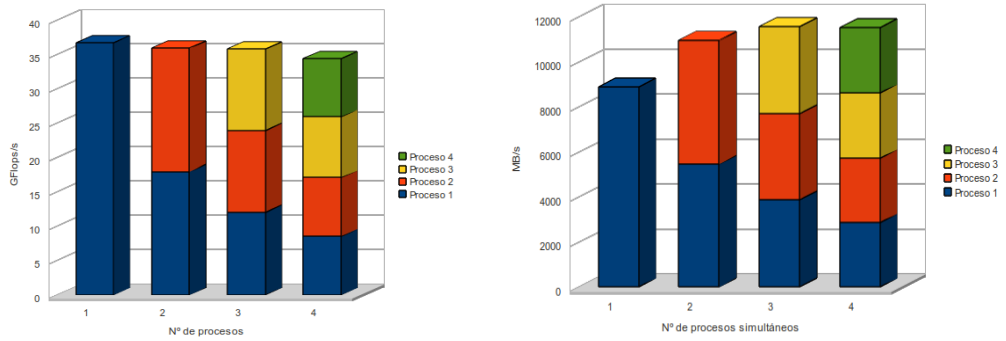


Figura 5.5: Resultados de los test realizados de CPU y RAM en el sistema *host CentOS 5.0* (4 cores)

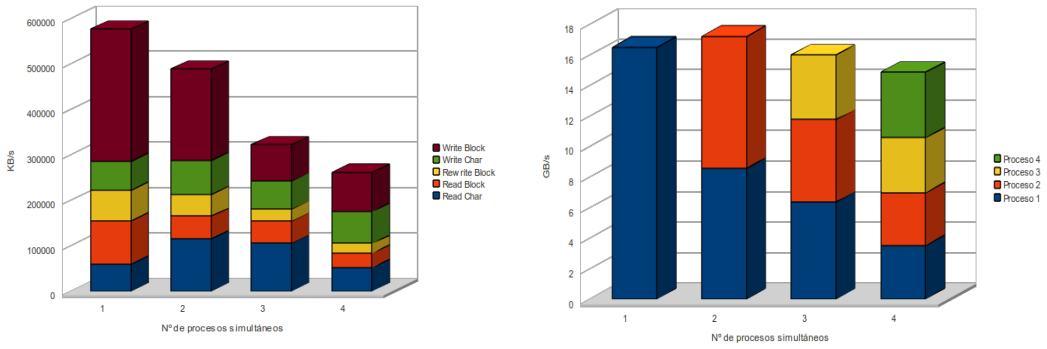


Figura 5.6: Resultados de los test realizados disco y red en el sistema *host CentOS 5.0* (4 cores)

## 5.2. Sistemas guests

Es esta sección describiremos los resultados que se han obtenido ejecutando los diferentes test en los sistemas *guests*. Los sistemas de virtualización seleccionados para realizar estas pruebas han sido *KVM* y *Xen*.

Al igual que sucedía en el apartado anterior presentaremos las gráficas de los resultados agrupadas para facilitar el seguimiento de los valores obtenidos. En el anexo de este documento se presentan con más detalle todas las tablas y datos que las han generado.

Las pruebas que se han realizado en este punto han sido diseñadas para utilizar un máximo del 80 % de los recursos físicos disponibles. Esto es necesario para evitar que el sistema *host* incurra en los *overheads* producidos por el uso de *SWAP* o falta de cores de ejecución.

Como ya hemos constatado en el punto anterior el uso de uno u otro sistema *Linux* como sistema *host* no produce resultados diferentes. De esta forma en este apartado, para no añadir varias hojas con gráficas que muestran prácticamente los mismos resultados, incluiremos únicamente las realizadas bajo *Ubuntu LTS 10.04*. La elección de este sistema se debe, entre otros aspectos, a que es el mismo sistema que el que se utiliza en el *guest* y su configuración es más sencilla para la evaluación de las pruebas.

Al observar las gráficas del rendimiento de *KVM* y *XEN* en las figuras 5.7 y 5.9 podemos comprobar que los resultados son similares para la gestión de *CPU* y memoria *RAM*. Sin embargo, en el caso del acceso a disco que podemos observar en las figuras 5.8 y 5.10 sí podemos observar una diferencia significativa entre ambas plataformas. En el caso de las pruebas de red podemos observar en las mismas gráficas que no existe una diferencia importante en los rendimientos obtenidos.

El sistema *KVM* obtiene un rendimiento de acceso a disco de prácticamente un 20 % respecto *Xen*. Esta diferencia se mantiene prácticamente igual conforme se aumenta el número de máquinas virtuales en el sistema *host*. Como ya se constató en las pruebas realizadas en el servidor físico, el acceso a disco es uno de los cuellos de botella más grandes en virtualización, de esta forma cualquier optimización nos permite obtener un a gran mejora.

### 5.2.1. Ubuntu 10.04 LTS

En las figuras 5.7 y 5.9 podemos observar el rendimiento bruto que obtenemos dentro de cada máquina virtual. En este caso la configuración del planificador (*scheduler*) de procesos del sistema host es crítico. Cuanto más frecuentemente se recalculen las prioridades de los procesos, más se notará el *overhead* que añade el sistema a la virtualización. En este caso observamos que las máquinas virtuales escalan de forma proporcional en el sistema.

En el caso de la memoria RAM también podemos observar que tanto en la plataforma KVM como en XEN se reparte equitativamente el rendimiento entre los diferentes procesos existentes en el *guest*. De esta forma queda equilibrado el rendimiento bruto entre los distintos procesos que se ejecutan en la máquina virtual copiando el comportamiento de un sistema no virtualizado.

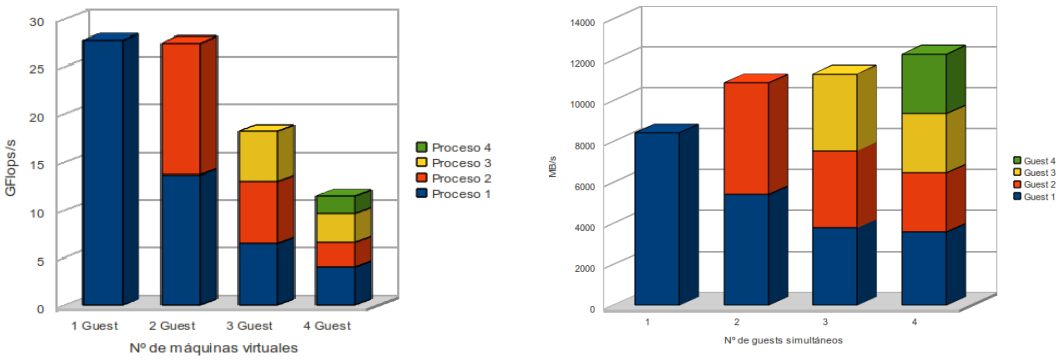


Figura 5.7: Resultados pruebas de CPU y RAM *KVM* en *Ubuntu* (4 cores)

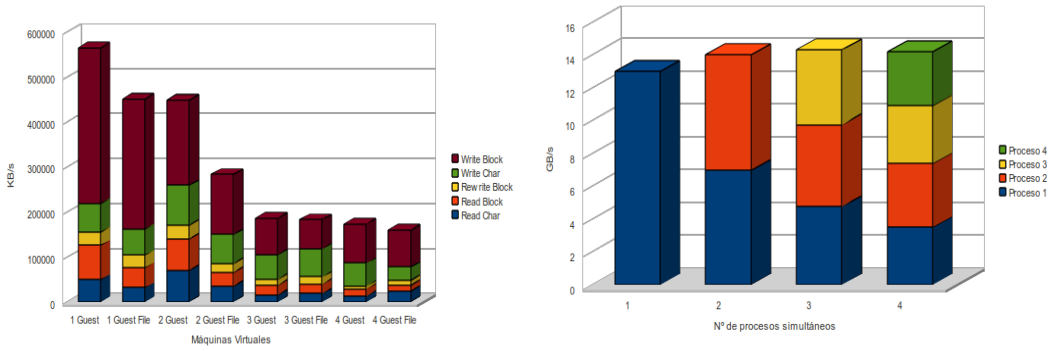


Figura 5.8: Resultados pruebas de disco y red *KVM* en *Ubuntu* (4 cores)

En las gráficas de acceso a disco (figuras 5.8 y 5.10) podemos observar los resultados obtenidos en la ejecución del test con una máquina virtual que utiliza un acceso directo a una partición de disco (*Guest*) y una máquina virtual que utiliza un fichero único que contiene su imagen (*Guest file*).

Para realizar la simulación en un entorno lo más real posible, se ha utilizado una partición (o fichero de imagen) de 200Gbytes de tamaño ocupando mediante ficheros *dummy* 180Gbytes. De esta forma los test se realizan en los últimos 20Gbytes lo que permite poner de manifiesto el *overhead* por indirecciones del sistema de ficheros (ver punto 6.1.2).

En las gráficas obtenidas (figura 5.10) podemos ver rápidamente que el rendimiento absoluto que obtiene cualquier plataforma virtualizada es claramente superior accediendo directamente a la partición de disco. En las pruebas realizadas podemos observar que existe una penalización de acceso de hasta un 23 % en ambas plataformas de virtualización.

La penalización que añade el modelo de fichero es debida a que para acceder a un bloque virtual de disco el sistema real (*host*) debe realizar varias indirecciones en su *file system*, proceso

que no existe en el caso de acceder directamente a una partición.

Además, cabe destacar que este *overhead* se agrava conforme crezca el tamaño del fichero de la máquina virtual. Cuanto mayor sea el tamaño del fichero del sistema *guest*, menor será el rendimiento obtenido de acceso a disco penalizando enormemente al sistema.

El uso de discos *SAS*, discos de estado sólido (**SSD**, *Solid State Disk*) o sistemas de *arrays de disco* minimizan el impacto de este overhead debido al gran rendimiento que proporcionan, pero a costa de aumentar enormente el coste económico de la inversión a realizar.

También se puede observar claramente como la plataforma *KVM* obtiene un mejor rendimiento, prácticamente un 25 %, en el acceso a disco del obtenido con *Xen*. Esta gran diferencia se fundamenta principalmente en el uso **drivers paravirtualizados** (*virtio*) que actualmente proporciona *KVM*. Estos *drivers* permiten un modo de acceso a los dispositivos prácticamente transparente para la máquina virtual, lo que minimiza enormemente el *overhead* que produce la virtualización. Actualmente estos *drivers* se encuentran disponibles para sistemas *Linux* y *Microsoft Windows*.

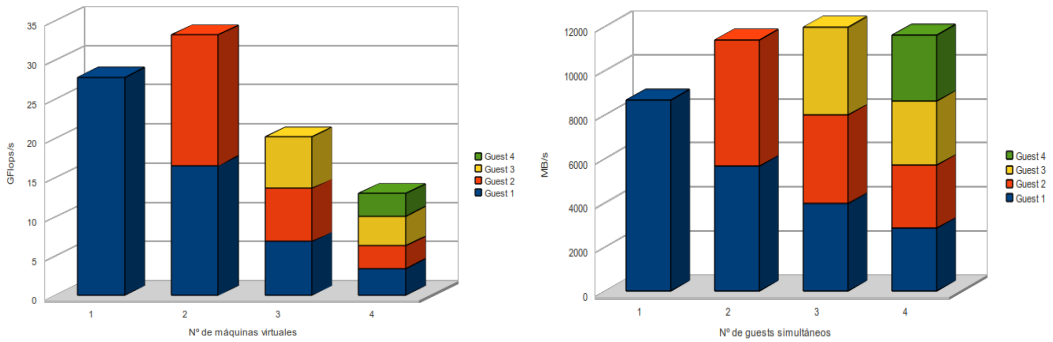


Figura 5.9: Resultados pruebas de CPU y RAM *Xen* en *Ubuntu* (4 cores)

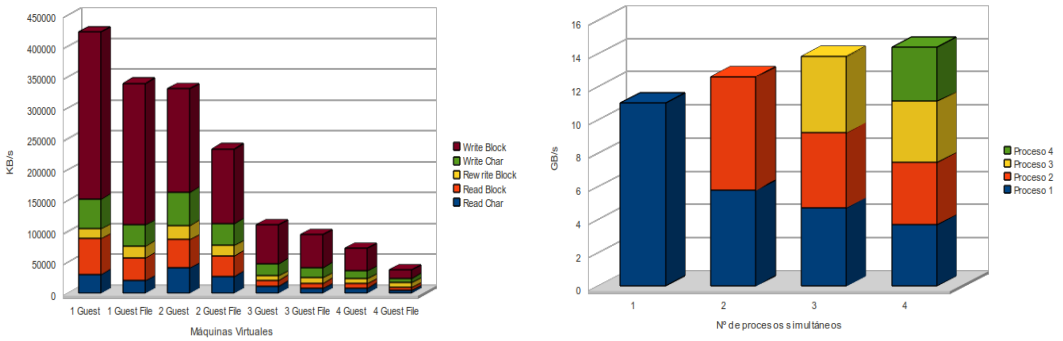


Figura 5.10: Resultados pruebas de disco y red *Xen* en *Ubuntu* (4 cores)

### 5.2.2. Debian 6

Ver los resultados en el Anexo de este documento.

### 5.2.3. CentOS 5

Ver los resultados en el Anexo de este documento.

### 5.2.4. Comparativa de resultados *Host* y *Guest*

A continuación contrastaremos los resultados obtenidos tanto en el sistema *host* como en el sistema *guest* con el objetivo de evaluar la pérdida bruta de rendimiento al utilizar una plataforma de virtualización *KVM*.

#### Resultados de rendimiento CPU

En la figura 5.11 podemos observar el resultado obtenido del rendimiento de CPU en los sistemas *host* y *guest* utilizando 1, 2 y 4 cores. Es importante señalar que el rendimiento de las máquinas virtuales escala linealmente en el sistema, lo que nos permite asegurar que mientras existan cores físicos libres podemos aumentar el rendimiento sin perjudicar al resto del sistema.

En las configuraciones con 1 o 2 cores observamos que el rendimiento bruto de la máquina virtual es el 70 % del obtenido en el sistema *host*, mientras que en configuraciones con más cores reales se iguala hasta prácticamente el 90 % de la potencia real. Pese a que varios estudios confirman que el rendimiento debería ser superior ([VSS][ECP][CXK]) nuestro *benchmark* no ha obtenido esos resultados.

Tras examinar el tema en profundidad llegamos a la conclusión de que estas desviaciones son debidas a la configuración **preemptiva** del planificador (*scheduler*) *kernel* que afecta al *overhead* en la asignación de procesos a los cores físicos y a la configuración del procesador. Varios modelos de procesadores actuales añaden a los cores físicos reales unidades extra de ejecución de intrucciones (*hypertheading*, [IHT]) que a nivel de sistema aparecen como cores reales, pero en verdad no lo son. De esta forma cuando una máquina virtual queda asignada a una unidad *hypethreading*, su rendimiento baja considerablemente.

**Nuestras recomendación en este aspecto es que en caso de utilizar este tipo de tecnología se añadan más cores a la máquina virtual o se deshabilite directamente en la BIOS. Esto nos permite asegurar el uso de cores físicos reales exclusivamente.**

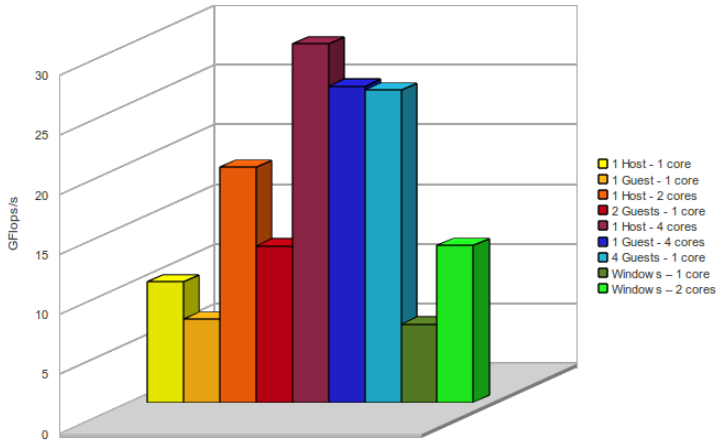


Figura 5.11: Comparativa de resultados de CPU en Ubuntu y Windows con *KVM*

### Resultados de rendimiento RAM

En nuestro sistema de pruebas nos aseguramos que todas las máquinas virtuales estuvieran siempre en la memoria física del sistema. De esta forma se puede evaluar realmente el acceso a memoria *RAM* evitando las penalizaciones que añade cuando el sistema *host* o *guest* utilizan el área de intercambio o *SWAP*.

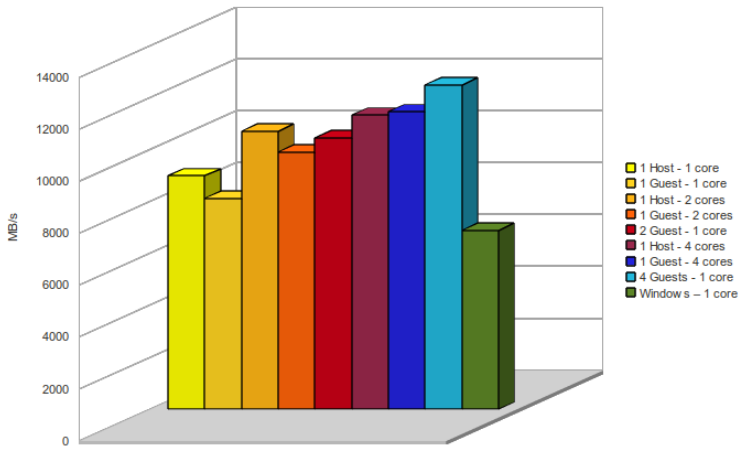


Figura 5.12: Comparativa de resultados de RAM en Ubuntu y Windows con *KVM*

Los resultados muestran claramente que el rendimiento bruto de una máquina virtual en acceso a memoria RAM es prácticamente el mismo que el de la máquina real, superando el 96 %. Cabe señalar que la arquitectura del servidor real (tipo de memoria, velocidad de conexión del bus) condiciona totalmente los resultados, y en nuestro caso escala perfectamente hasta las 4 máquinas virtuales simultáneas.

### Resultados de rendimiento disco

El rendimiento de disco es la parte que más penalización sufre en la virtualización de sistemas. El motivo principal es que la virtualización completa no permite un acceso directo al dispositivo de bloques, ya que éste debe ser emulado por el kernel y por tanto se añade un gran *overhead* para cada acceso.

En las gráficas 5.13, 5.14 y 5.15 podemos observar cómo se penaliza enormemente el rendimiento en el caso de sistemas con grandes tasas de *entrada/salida* simultánea. El programa utilizado en las pruebas realiza varios subtests exhaustivos tanto de lectura como de escritura que comprueban el acceso a nivel de caracter y bloque. Estos subtests se repiten tanto para accesos aleatorios como secuenciales.

De esta forma, en principio, debería evitarse la virtualización de sistemas con altas tasas de E/S como por ejemplo bases de datos o servidores de ficheros dentro de un mismo servidor real.

Sin embargo existen métodos para hacer asumible su utilización y minimizar el impacto del acceso a disco: Balancear servidores virtualizados con alta E/S entre diferentes servidores físicos, el uso de acceso directo a disco (*direct block device*) en vez de ficheros (*file*) o externalizar el sistema de disco a arrays especializados dónde esta penalización queda compensada por sus altas capacidades *hardware*.

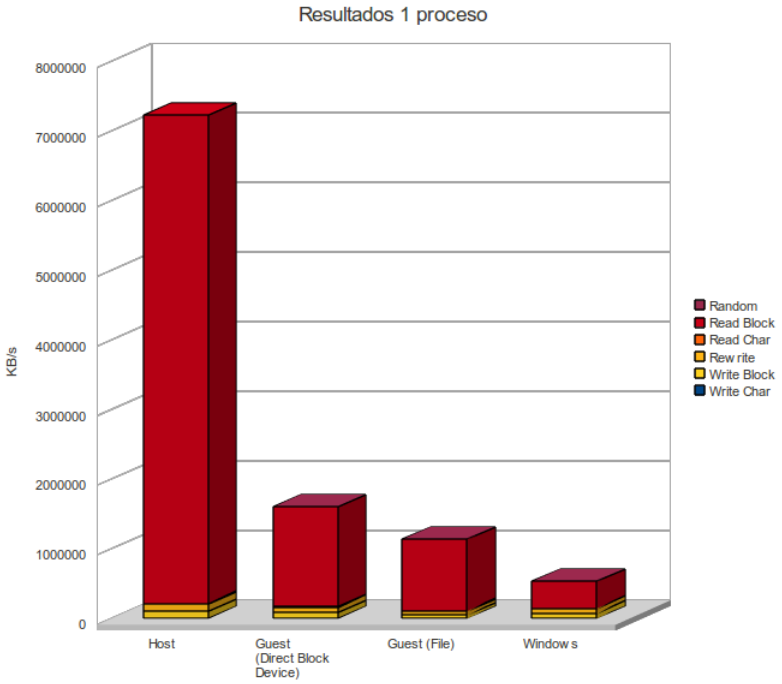


Figura 5.13: Comparativa de resultados de disco en Ubuntu y Windows con *KVM*

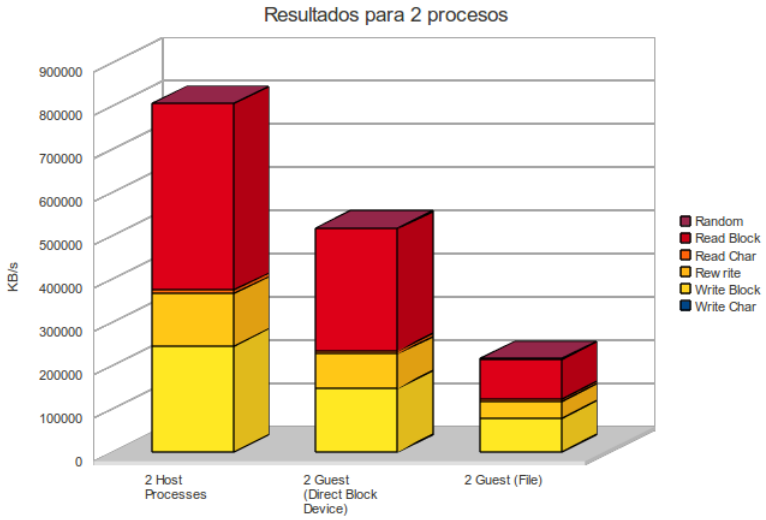


Figura 5.14: Comparativa de resultados de disco con dos procesos simultáneos en Ubuntu y Windows con *KVM*

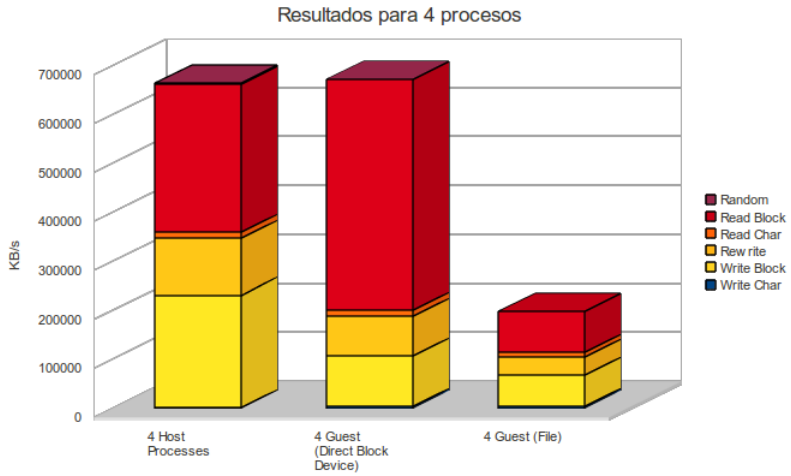


Figura 5.15: Comparativa de resultados de disco con cuatro procesos simultáneos en Ubuntu y Windows con *KVM*

### Resultados de rendimiento red

Finalmente en este apartado mostramos los resultados obtenidos en la simulación de tráfico de red entre el sistema real y las máquinas virtuales.

Debido a que en las transferencias dentro de una red de datos quedan afectadas por el tráfico existente, se decidió evaluar la transferencia interna mediante el *interface loopback* y la transferencia *IP* entre el sistema *host* y las máquinas virtuales.

Para nuestras pruebas hemos utilizado el modo *bridge* utilizando los *interfaces* del sistema *virbr0/tap0*) de forma que cada *host* tiene acceso directo a la red. Otra opción seria mediante un esquema tipo **NAT** (*Network Address Translation*) dónde todas las máquinas virtuales comparten una misma dirección *IP*.

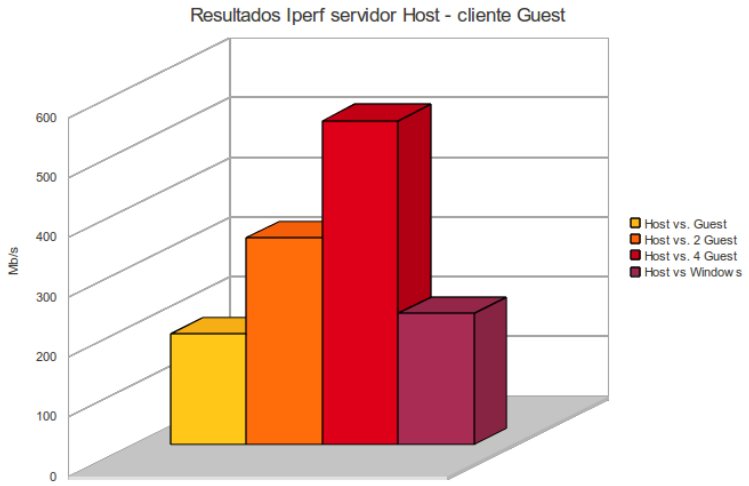


Figura 5.16: Comparativa de resultados de red en Ubuntu y Windows con *KVM*

En las transferencias internas de datos entre el sistema *host* y el sistema *guest* (ver figura 5.16), podemos observar claramente que el rendimiento de la red escala linealmente de forma que no existe un cuello de botella significativo en las comunicaciones al aumentar el número de sistemas virtualizados en el *host*.

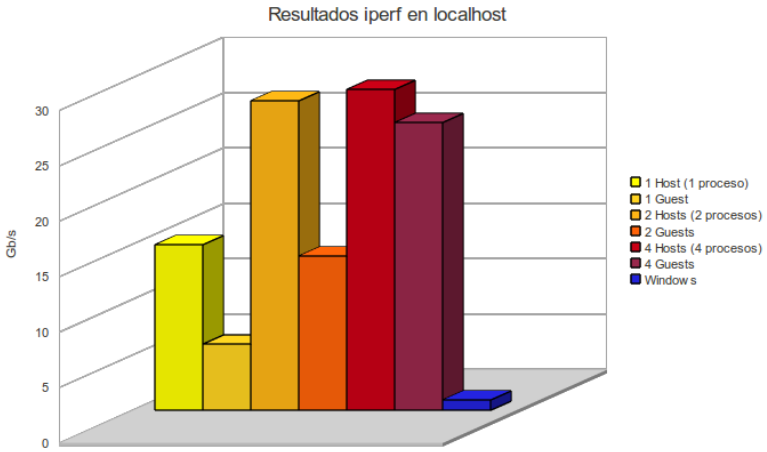


Figura 5.17: Comparativa de resultados de red en localhost en Ubuntu y Windows con *KVM*

En el caso de las comunicaciones utilizando el interface *loopback* sí podemos observar una pérdida de rendimiento importante al aumentar la concurrencia de procesos en el sistema *guest*. Este comportamiento se puede explicar por la magnitud de las transferencias realizadas (*Gbits/s*). De esta forma un pequeño *overhead* producido por la plataforma de virtualización en las comunicaciones, se convierte en una gran pérdida al repetirse millones de veces por segundo.

De todas formas debemos señalar que incluso en el caso más sencillo (1 sistema *guest* con 1 core asignado), la transferència registrada es de unos 5Gbits/s lo que garantiza que las comunicaciones no serán un cuello de botella para el sistema virtualizado.

# CAPÍTULO 6

# Conclusiones

En este capítulo se tratarán las conclusiones obtenidas a partir de los tests de rendimiento realizados. Se evaluarán los resultados de los distintos sistemas operativos y las plataformas de virtualización propuestos.

Finalmente, propondremos el nuevo modelo de administración para los servidores de proyecto gestionados por el *RDlab* y comentaremos las diferentes opciones de ampliación o futuras mejoras.

## 6.1. Evaluación de resultados

Después de realizar en el capítulo anterior las pruebas de los distintos sistemas operativos y plataformas de virtualización, extraeremos las conclusiones que nos deben permitir la selección del sistema más adecuado a nuestros intereses.

Debemos destacar que los sistemas evaluados parten de la misma familia *Unix*, lo que necesariamente nos lleva a que no existan grandes diferencias en los resultados.

### 6.1.1. Resultados host

De los resultados obtenidos de los tres sistemas operativos *Ubuntu*, *Debian* y *CentOS*, podemos comprobar como esperábamos que no existen grandes diferencias. Los distintos tests no han sido un elemento definitivo en este caso debido a la similitud del código que presentan todos los sistemas *Linux*.

Pese a ser un resultado esperable y que puede parecer no relevante, sí que era necesario realizar esta prueba. El cariz eminentemente práctico de este estudio nos obligaba a comparar

las distintas alternativas desde una óptica cuantificable y demostrable.

De esta forma, para realizar la elección del sistema *host* que utilizaremos en nuestra propuesta utilizaremos aspectos de gestión y configuración del sistema. Las diferencias entre las distintas distribuciones *Linux* son principalmente las herramientas de gestión que ofrecen así como los ciclos de vida y actualización. La mejor opción es aquella que nos facilite su administración e incluya por defecto el soporte de los sistemas virtualizados.

### 6.1.2. Resultados guest

Como ya se detalló en el capítulo anterior la elección del sistema *guest* fue la misma para todos los sistemas evaluados. Esta elección se debe a que en la virtualización completa el sistema es totalmente independiente del servidor físico.

Los aspectos más relevantes que pueden obtenerse en las pruebas realizadas hacen referencia al sistema de gestor virtual (hipervisor) y el sistema de acceso a disco para los sistemas *guest*.

De las plataformas de virtualización evaluadas observamos que la E/S con *KVM* es ligeramente más rápida que la realizada usando *Xen*. Esto es debido a que con el tipo de virtualización completa que usa el hipervisor *Xen* no se utiliza ninguna optimización específica para E/S. En cambio, con el uso de los controladores paravirtualizado que incorpora *KVM*, la mejora de los resultados es sustancial.

Respecto a la gestión del disco por parte de los sistemas *host* sí podemos extraer algunas conclusiones relevantes. En el caso de utilizar un acceso directo por parte del sistema virtualizado a la partición del disco, obtenemos un mayor rendimiento que accediendo a un único fichero dónde se encuentre la imagen de los datos.

Para entender el motivo por el cual empeorará cada vez más el rendimiento cuanto mayor sea el fichero que contenga los datos del sistema virtualizado, debemos analizar cómo se accede a los datos en los sistemas de ficheros (ver figura 6.1).

Podemos ver claramente que cuanto mayor sea el tamaño del fichero más indirecciones se deben realizar para encontrar la posición real de los datos. De esta manera, con imágenes que ocupan Gigabytes o incluso Terabytes, el acceso a un bloque de datos por parte del *host* requiere varias lecturas reales en el disco.

Como contrapartida, al utilizar el acceso directo a las particiones del disco por parte del hipervisor se pierde la flexibilidad que ofrece mover la máquina virtual como un único fichero entre diferentes *hosts*. Sin embargo, nos permite optimizar el rendimiento del sistema y mejorar la velocidad de respuesta del sistema virtalizado. Para dotar este sistema de la flexibilidad necesaria podemos utilizar gestores de volúmenes como *LVM* que encapsulan las particiones ofreciendo un acceso transparente y homogéneo a los datos.

Actualmente el RDlab no dispone de un array de discos que pueda servir de espacio común entre servidores físicos, por lo que no resultaría difícil adaptar esta solución que aúna rendimiento y flexibilidad.

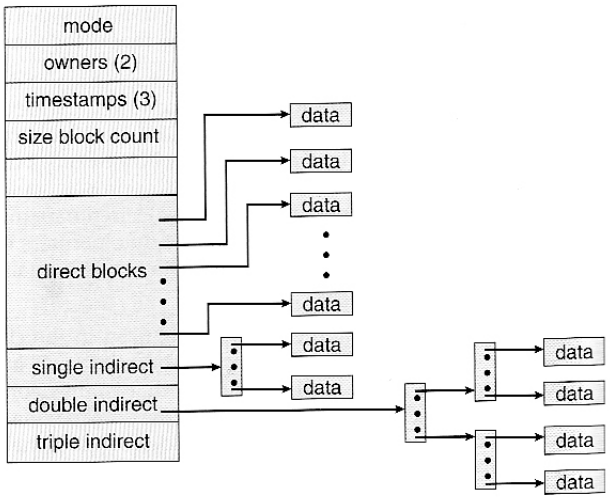


Figura 6.1: Esquema de acceso a los inodos de fichero

## 6.2. Modelo propuesto

Una vez implementado el prototipo y realizadas las pruebas de rendimiento y estabilidad, realizaremos la descripción del modelo de virtualización propuesto para el *RDlab*. La arquitectura que se presenta a continuación debe marcar la hoja de ruta a seguir para la adecuación de los sistemas existentes al nuevo paradigma de gestión de servidores y servicios TIC para los años venideros.

Este modelo cumple con los objetivos iniciales de este trabajo así como con los requisitos recogidos en el capítulo 2. Las características concretas del mundo universitario, el funcionamiento interno de los grupos de investigación y la infraestructura existente condicionan la solución seleccionada.

De esta forma debemos destacar que **esta solución que se ajusta perfectamente a las necesidades de los grupos de investigación de LSI, puede no ser válida para otros departamentos o entornos**. Es importante hacer notar que, por un lado, cada organización tiene su propia manera de funcionar, lo que dificulta la estadarización de soluciones. Y, por otro, en cada entorno de investigación la solución debe ser ajustada y personalizada.

### 6.2.1. Plataforma de virtualización

La solución que se propone para realizar la virtualización de los servidores es la basada en *KVM* (ver figura 6.2). Además de cumplir todos los requisitos y proporcionar una virtualización completa de los sistemas hosts, nos evita tener que modificar el *Kernel* del sistema como pasa con *Xen*.

En las pruebas de rendimiento realizadas no se ha observado una diferencia significativa entre las plataformas de paravirtualización y de virtualización completa. El gran trabajo realizado

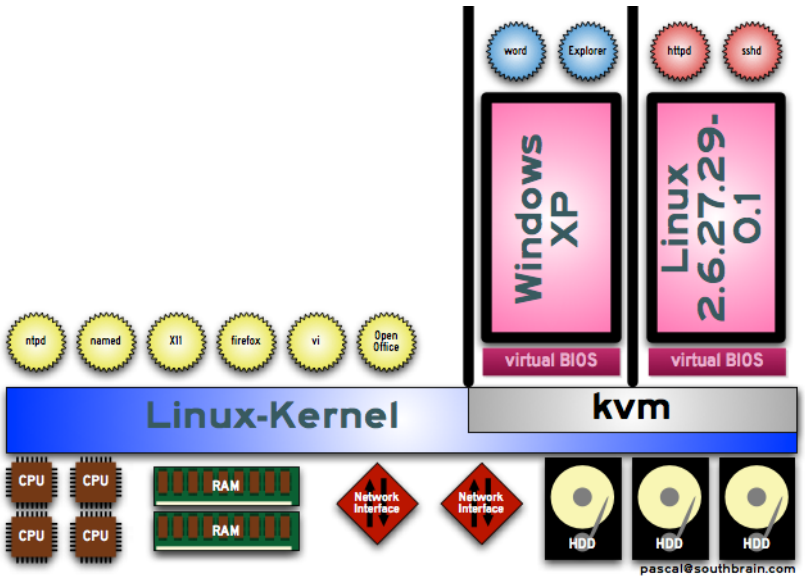


Figura 6.2: Esquema de la plataforma de virtualización con KVM

para mejorar el sistema *KVM* aparición de *drivers* avanzados como *virtio*, ha permitido mejorar espectacularmente el rendimiento y minimizar el *overhead* de la virtualización. Consecuentemente, en el criterio de selección se ha valorado la sencillez de configuración e integración dentro de los sistemas ya existentes. Debemos señalar que con más de 50 servidores reales dedicados a tareas de investigación, cualquier facilidad técnica resulta una gran ventaja.

Respecto a la utilización del disco se ha decidido optar por utilizar un esquema flexible de uso mediante la implementación de un sistema *LVM*. De esta forma, el sistema virtualizado funcionará directamente sobre particiones de disco que permitirán aprovechar al máximo el rendimiento. El uso de ficheros presenta varios inconvenientes como penalizar la velocidad de acceso o la dificultad de realizar copias de seguridad puesto que disponemos de un único volumen de datos. Además, debido al gran tamaño que pueden alcanzar (centenares de Gigas o incluso Terabytes) su gestión es difícil si no se dispone de un array de discos.

Aceptamos sin embargo la conveniencia de utilizar ficheros en el caso de disponer de un sistema común de almacenamiento para facilitar mecanismos de alta disponibilidad. Actualmente el *RDlab* carece de un array de discos dedicado a esta tarea, motivo por el que se prefiere la flexibilidad de una gestión con *LVM*. Este tema se trata con más profundidad en las soluciones de backup, punto 6.2.4.

Respecto la configuración de la red del sistema *host* y los mecanismos de seguridad y acceso, se propone un mecanismo flexible que combine varias técnicas. Generalmente los sistemas de virtualización existente utilizan una configuración simple en modo *bridge* como la presentada en la figura 6.3. El modo *bridge* se utiliza usualmente para dotar de acceso total a la red a los sistemas *guests*, ya que utilizar técnicas como *NAT* imposibilita acceder a varios sistemas a los mismos puertos simultáneamente, ya que comparten la misma dirección *IP*.

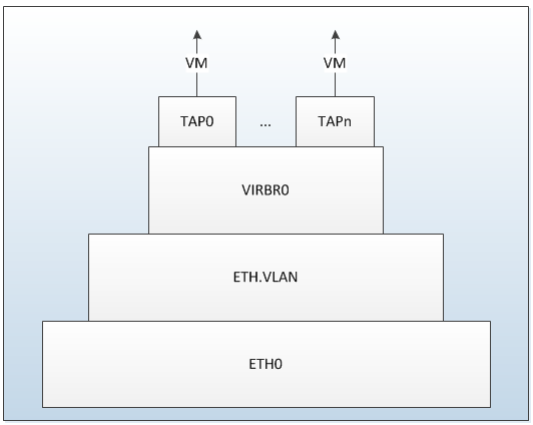


Figura 6.3: Esquema básico de red para entornos virtualizados

En una configuración básica tenemos que el interfaz de acceso a red gestionado por el hipervisor funciona directamente sobre el dispositivo de red (*eth0*) o el dispositivo *VLAN* si existiese. Sobre este interfaz real definimos el interface de red virtual utilizado por el hipervisor denominado *virbr0*. De esta forma, cada una de las máquinas virtuales creará su propio interfaz (*tap*) que le permite tener acceso directo al sistema *bridge* virtualizado.

Con esta configuración básica nos encontramos que todas las máquinas virtuales deben compartir un mismo esquema de red. Por tanto, obtenemos un sistema menos flexible ya que entre otras limitaciones tenemos que a nivel de seguridad, todas las reglas de filtrado se aplican sobre el mismo interfaz.

En la configuración de red avanzada propuesta para el RDLAB (ver figura 6.4) se crean dos interfaces *bridge* denominados *virbr0* y *virbr1*. De esta manera podemos tener máquinas virtuales que sirvan a diferentes rangos de red y VLAN de forma sencilla y flexible. Todo este esquema sigue funcionando sobre un interface de red real, lo que permite aplicar reglas de *firewall* (IPtables) específicas para cada sistema.

Además, con estas configuraciones basadas en *bridge* tenemos un sistema *guest* totalmente autónomo a nivel de red. Cada máquina virtual incluye su configuración de red (dirección *IP*) de forma que si es ejecutada en otro servidor físico sigue funcionando sin necesidad de realizar ningún cambio de configuración en el *host* o en el *guest*.

### 6.2.2. Sistema *host*

Tal y como hemos constatado en las pruebas que hemos realizado no existe una gran diferencia entre las distintas alternativas. Todos los sistemas *Linux* comparten la mayoría del código, existiendo pequeñas diferencias debido a parámetros de configuración o a las versiones de los programas. De esta forma la elección de un sistema u otro debe basarse en criterios como la facilidad de configuración, estabilidad del sistema y rapidez de incorporación de nuevas tecnologías.

Si bien *Debian* es probablemente el sistema más robusto y un buen candidato, la lentitud en la evolución de sus versiones le resta mucho atractivo. La virtualización es un campo en

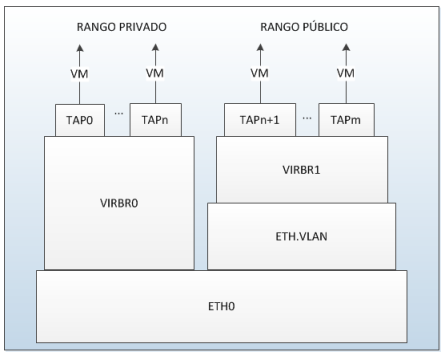


Figura 6.4: Esquema de red para entornos virtualizados en el *RDlab*

constante movimiento y poder aprovechar rápidamente estas aportaciones redunda directamente en el rendimiento de la plataforma y por tanto en una mejor experiencia de uso para los investigadores.

*CentOs* es un sistema que se basa en Redhat y por tanto no tiene desarrollo propio. La dependencia de una tercera empresa comercial para el mantenimiento del sistema plantea serias dudas a medio y largo plazo. La apuesta por KVM en este sistema parece definitiva pero al no ser subscriptores de pago siempre se queda a expensas de la evolución empresarial que decida *RedHat*.

Ubuntu es una plataforma joven y con relativa poca experiencia en el soporte a servidores. Sin embargo cuenta con una gran comunidad de desarrolladores y usuarios que la han llevado crecer cada año. Los ciclos de vida y actualización son bastante adecuados y el soporte existente es satisfactorio. Actualmente grandes proyectos libres ya han migrado a esta distribución (Wikipedia) e incluso grandes sistemas comerciales como Amazon Cloud (EC2) la soportan.

Por tanto, se adoptará como sistema *Ubuntu 10.04 LTS x64 server*. Esta decisión no compromete a largo plazo, ya que hemos visto que nuestra plataforma permite independizar los sistemas *host* y *guest*. Así, en caso de necesidad, podríamos cambiarlos sin excesivo esfuerzo y de forma transparente para las máquinas virtuales.

### 6.2.3. Gestor de entornos virtualizados

Para la gestión de entornos virtualizados proponemos un esquema seguro como el que recoge la figura 6.5. De esta forma buscamos un único gestor que nos permita manejar tanto los sistemas reales como los virtuales.

La comunicación entre los distintos sistemas que componen este esquema se realiza mediante una conexión segura vía *ssh*. De esta forma mantenemos el esquema de seguridad recogido en los requisitos del proyecto. Este sistema evita que la comunicación entre el gestor virtual y los sistemas *host* o *guest* circule sin cifrar por la red y que un eventual atacante pueda obtener información relevante (opciones de configuración o *passwords*) de nuestro sistema que le permitiese tomar el control de nuestros servidores.

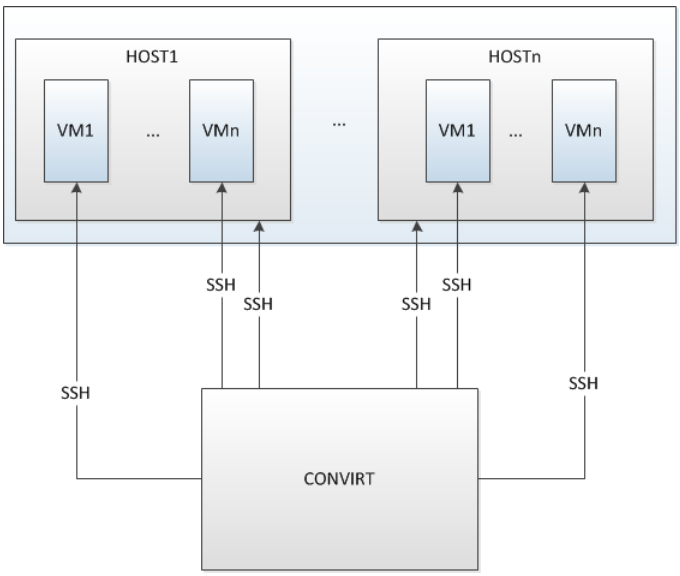


Figura 6.5: Esquema de gestión de entornos virtualizados para el *RDlab*

De los tres sistemas de gestión probados en este proyecto, *Convirt* destaca claramente por su estabilidad, facilidad de configuración y entorno web gráfico. Las alternativas como *ovirt* resultan muy complejas de instalar y son poco flexibles ya que requiren el uso de zonas compartidas entre los sistemas para su comunicación. El *virt-manager* es aún un sistema poco maduro y no carente de *bugs*, por tanto no puede ser utilizado en un entorno de producción.

En la figura 6.6 podemos observar la instalación del gestor de entornos virtualizados *convirt* así como el sistema *host* utilizados en las pruebas y la fase de pilotaje.

### 6.2.4. Imágenes de los sistemas (*backup)*

En los puntos anteriores hemos ido definiendo incrementalmente una arquitectura que nos permite realizar una virtualización total de los servidores de investigación gestionados por el *RDlab*. Se ha fundamentado mediante el anterior estudio técnico la viabilidad de este sistema y, por tanto, de cara a su implantación real debería proporcionar una solución a todos los aspectos existentes en la gestión de servidores y servicios.

El requisito de permitir la creación de imágenes de los sistemas, tanto *host* como *guest*, recoge la necesidad de contar con un sistema de *backup* fiable. Esto es especialmente crítico en los sistemas virtualizados puesto que es donde los investigadores realizan su trabajo y la posibilidad de recuperar la información en caso necesario es fundamental.

Actualmente los servidores de investigación albergan más de 20 Terabytes de información de los cuales únicamente de una parte se hace copia. Por un lado, nos encontramos con que lógicamente cada año el espacio consumido por parte de los proyectos y la investigación crece. Por otro, observamos una disminución del coste económico de los discos, lo que retroalimenta la primera parte.

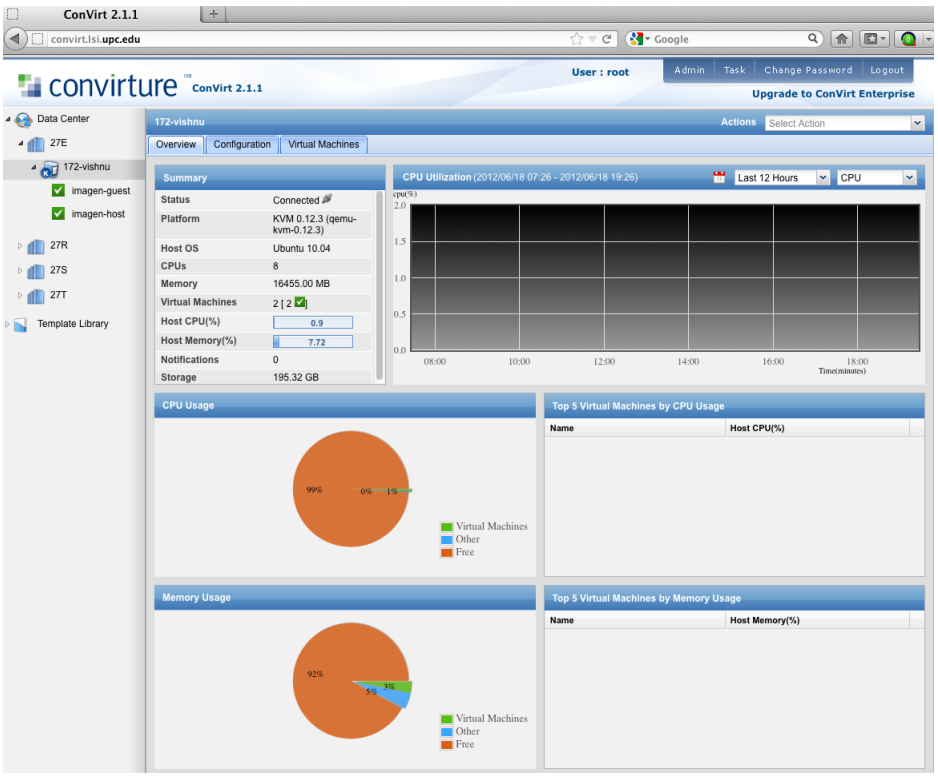


Figura 6.6: Gestor de entornos Convirt para el *RDlab*

Gran parte de la motivación de este estudio consistía en poner las bases del modelo TIC para los próximos años. De esta forma, nuestra solución debe contemplar la gestión del espacio y las copias de seguridad. Actualmente los servidores se compran con dos discos que se configuran en *RAID 1* para minimizar el impacto de los fallos físicos. Con la virtualización y la flexibilidad para crear sistemas virtuales el consumo del espacio crecerá por lo que no podemos confiar en seguir las mismas recetas de décadas pasadas.

La solución que se presenta en este documento relativa a la gestión del espacio en disco está pensada para que sea escalable y flexible. De esta forma puede crecer con las necesidades y aprovechar los recursos existentes en la medida de lo posible.

En el sistema *host* utilizaremos la tecnología de gestión de volúmenes que nos brinda *LVM*. De esta forma para cada máquina virtual el sistema *host* definirá un volumen *LVM* donde se encontraran sus datos. Este modelo nos permite tener diferentes máquinas virtuales en un mismo servidor real cada una ocupando el espacio que necesite. Además *LVM* permite crecer "**en caliente**", lo que nos proporciona la flexibilidad necesaria para aumentar el espacio asignado a un *guest* si fuera necesario.

Otra característica que nos aporta el uso del *LVM* y que aprovecharemos para la realización de las copias de seguridad de los sistemas es la posibilidad de usar *snapshots*. Los *snapshot* son imágenes estáticas o "fotos"del sistema LVM en un momento concreto.

De esta forma podemos obtener una copia estable del sistema de ficheros (recordemos que las máquinas virtuales ¡están funcionando!) y realizar una copia en un soporte externo, generalmente un array de discos (ver figura 6.7). Una vez realizada la copia del sistema, se elimina el *snapshot* y el sistema queda como estaba.

Debemos señalar también que este esquema permitiría una gestión más sencilla de los *backups* realizando diferentes *snapshots* dentro del *LVM* sin borrarlos. Es importante destaca,sin embargo, que esta gestión puede ser peligrosa si el espacio destinado a almacenar los cambios se llena.

La eliminación del *snapshot* viene determinada porque éste consume espacio en disco. Al realizar un *snapshot* de un volumen se ha de indicar qué cantidad de espacio se reserva para contener los cambios que se producen en el sistema. Al corresponder a una máquina virtual ejecutándose, el contenido del volumen irá cambiando con el tiempo y antes o después se consumirá el espacio consignado e invalidará el *snapshot*. Igualmente, si no existe el espacio libre suficiente en el volumen para almacenar los cambios, este no se creará y por tanto fallará la copia.

Es necesario indicar que el uso de estas técnicas de *backup* basadas *LVM* no son incompatibles con los sistemas tradiciones. En éstos se suele instalar un cliente de *backup* que realiza una copia de los ficheros en un servidor remoto como por ejemplo en el software *Legato* o en el sistema libre *Bacula*. Es más, el uso de copias basadas en *snapshots* presenta la ventaja de que permite recuperar el sistema sin necesidad de instalar ningún cliente y, además, recupera todo el sistema *guest* que puede ser directamente ejecutado en cualquier sistema *host*.

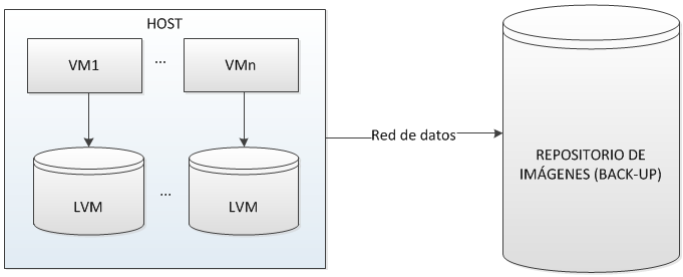


Figura 6.7: Sistema de *backup* de servidores para el *RDlab*

### 6.2.5. Sistema de alta disponibilidad

Al igual que en el punto anterior, la alta disponibilidad del sistema es un requisito fundamental para proporcionar una solución adecuada que cubra todos los aspectos importantes de la gestión TIC.

El uso de técnicas de virtualización aporta una gran flexibilidad al independizar la ejecución del sistema *guest* del servidor físico asociado. Si bien no es necesario actualmente que todos los servicios y servidores del *RDlab* esten configurados en alta disponibilidad, sí que deben ser capaces de estarlo en caso necesario.

Las necesidades de la investigación y la realización de proyectos varían con el tiempo, de forma que durante un período de tiempo un servicio puede pasar a ser crítico (por ejemplo durante el período de tiempo cercano al *deadline* de un congreso) y viceversa.

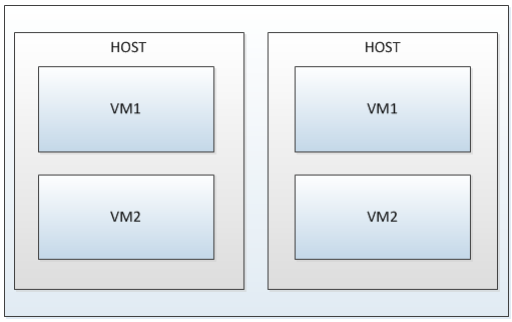


Figura 6.8: Esquema básico sin alta disponibilidad real para el *RDlab*

En el modelo básico de virtualización tenemos una serie de servidores físicos que ejecutan cada uno su máquina virtual (ver figura 6.8). En este esquema no existe un espacio compartido, por lo que el fallo del *host* deja inaccesible todas las máquinas virtuales que contenga. Con una configuración como ésta no es posible recuperar la ejecución de la máquina virtual debido a que perdemos el acceso a sus datos.

Sin embargo, sí podríamos llegar a obtener un cierto grado de tolerancia y volver a tener el sistema *guest* funcionando a partir de una copia (*backup*) anterior. En algunos servicios la volatilidad de datos es muy baja (por ejemplo un servidor web para un congreso) y si las copias se realizan con suficiente frecuencia podrían permitir la recuperación del servicio.

Para dotar a nuestra solución de un sistema real de alta disponibilidad se debe proporcionar una zona común dónde residan todas las máquinas virtuales (ver figura 6.9). En este caso se recomienda que el modelo de almacenamiento físico utilizado para las máquinas virtuales sea el uso de ficheros. De esta forma cuando el sistema detecte un fallo de servicio en un servidor físico o en una máquina virtual concreta, puede lanzar el servicio en cualquiera de los sistemas *host* disponibles.

En este esquema se asume que el sistema de array de disco es fiable y tolerante a fallos. Al compartir todas las máquinas un único espacio en disco, un problema en el sistema de discos nos introduce un punto de fallo (***SPOF**, Single Point of Failure*) que haría peligrar todo el sistema.

Cabe destacar también que una arquitectura tolerante a fallos como la que hemos presentado permite también la migración de máquinas virtuales. De esta forma podríamos mover los distintos servicios virtualizados "*en caliente*"de un *host* a otro. Esta característica está contemplada en *KVM* y la soporta el gestor *Convirt*.

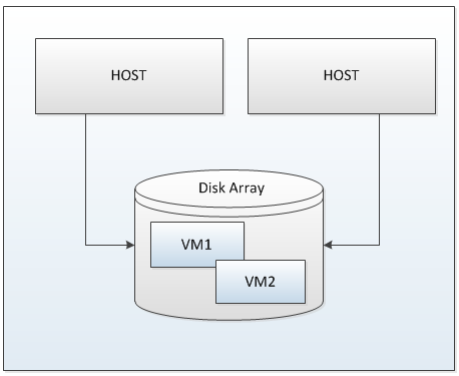


Figura 6.9: Esquema básico de alta disponibilidad para el*RDlab*

## 6.3. Ampliaciones futuras

Como se han recogido en los apartados anteriores de este capítulo existen varios aspectos de ampliación y mejora que si bien si han sido expuestos, quedan fuera del ámbito de este estudio. Las limitaciones temporales y la necesidad de completar en PFC obligan a desarrollarlas en otros trabajos futuros.

En el punto 6.2.4 hemos presentado el **sistema de *backup*** encargado de realizar las copias tanto de los sistemas *host* como *guest*. Se han explicado las herramientas técnicas que pueden utilizarse así cómo su integración en la plataforma de virtualización presentada. Sin embargo, queda pendiente de realizar el sistema que realice y gestione estas copias. Definir la política de *backup* deseada (diaria/semanal/mensual) tanto de las máquinas virtuales como de los servidores físicos. El tipo de copia a realizar de los datos (totales *full* o incrementales) así como la ventana temporal de validez de la copia.

Otro aspecto importante a desarrollar seria la incorporación del **sistema de alta disponibilidad** descrito en el punto 6.2.5. Actualmente en el *RDlab* no se dispone de ningún espacio de almacenamiento común a los servidores debido entre otros motivos a que cada proyecto financia únicamente sus recursos.

Quedaría pendiente realizar un estudio detallado sobre los sistemas y servicios existentes actualmente en el *RDlab*. Identificar los servicios críticos así como los sistemas *host* candidatos a ejecutarlos. Posteriormente realizar una propuesta a los grupos de investigación y al Departamento para financiar la compra de un array de discos que permitiese implementarla.

Respecto a las sistemas y mecanismos de seguridad, una posible ampliación futura podría ser la creación de un **repositorio centralizado de ficheros log** para la auditoria de los sistemas *host* y *guest*. El uso de la virtualización aumenta enormemente la cantidad de sistemas a gestionar, fragmentando la información de auditoria del sistema entre decenas de equipos. Conseguir centralizar toda esta información permite realizar una gestión más eficiente y aplicar mecanismos de seguridad y control a todos los sistemas *hosts* y *guest* simultáneamente.

Actualmente existen herramientas recolectoras de esta información (*logstash* y *graylog* por ejemplo) que nos permiten realizar esta tarea. Este aspecto queda totalmente fuera del ámbito de este PFC y por tanto no ha sido desarrollado en los puntos anteriores.

Finalmente, destacaremos que **la virtualización es un ámbito vivo y en constante evolución**. De tal forma se hace necesario mantener el contacto con la comunidad de desarrolladores y consultar artículos periódicamente para que las nuevas mejoras tecnológicas puedan ser incorporadas a nuestro sistema.

# CAPÍTULO 7

## Gestión del proyecto

En este último capítulo del proyecto final de carrera presentaremos y detallaremos aquellas fases relacionadas con la gestión del proyecto. Se evaluará la planificación realizada, la consecución de los objetivos planteados así como la utilización eficiente de los distintos recursos (temporales, físicos o económicos).

Para la realización de este PFC, que empezó a tomar forma en el mes de Septiembre de 2011, se realizó una planificación inicial previa durante la fase de toma de requisitos (finales del mes de Octubre). Una vez enmarcados los objetivos y con un cierto conocimiento previo de las tecnologías a utilizar, se definieron las distintas fases para finalizarlo, y por tanto, poder ser presentado en Junio de 2012.

Sin embargo, desde el punto de vista económico esta aproximación no fue posible. Debido a la complejidad de todo el trabajo que debía realizarse (familiarizarse con los sistemas existentes en el *RDlab*, analizar tecnologías punteras en constante desarrollo, implementar modelos y realizar pruebas sobre ellos) se requería un compromiso constante de trabajo. De esta forma se decidió asignar una partida presupuestaria del *RDlab* para el pago de una beca que cubriese la totalidad del proyecto.

El diseño anual de los presupuestos departamentales obligaba a consignar la beca asociada hasta el 31 de Diciembre de 2011, realizando una renovación de ésta en Enero de 2012. Los brutales recortes que sufrieron tanto el propio Departamento de *LSI* como sus grupos de investigación, dificultaron enormemente la financiación de la segunda parte de este proyecto (Enero - Junio de 2012). Finalmente, gracias a la implicación de los grupos de investigación y la generación de recursos propios por parte del *RDlab*, este proyecto ha podido finalizarse de forma satisfactoria.

## 7.1. Planificación

La resolución correcta de un proyecto comporta dividir el trabajo a realizar en un conjunto de tareas más sencillas que permitan facilitar su elaboración y control. En el caso de un proyecto tan complejo y largo como éste, la planificación ha sido un elemento imprescindible que ha permitido realizar un seguimiento semanal y tomar decisiones que han ayudado a minimizar las desviaciones existentes.

En la figura 7.2 se presenta el diagrama de *Gantt* correspondiente a la realización de este PFC. Para facilitar su lectura se presenta en una misma figura la planificación inicial (referenciadas con el prefijo **Pre** y en color rojo) y la planificación final o real una vez finalizado este trabajo. Se ha decidido presentar este diagrama dividido en cinco grandes fases que agrupan las tareas claves que se han realizado de forma que cada subtarea recogida sea al menos de una semana de duración.

- **Toma de requisitos**: En esta fase se recogen las tareas encargadas de enmarcar el proyecto. En este período de tiempo se realizaron las definiciones de los *requerisitos funcionales* y *no funcionales*. Si bien hay que reconocer que conforme se avanzaba en el proyecto éstos quedaron más ajustados a la realidad, cabe destacar que dichos cambios fueron mínimos y no obligaron a replanteárselos.

  Como se puede observar en el diagrama de *Gantt*, no hubo desviación respecto la planificación.

- **Análisis, estudio y recolección de la información** : Las tareas dedicadas a la consulta de documentación y análisis del estado del arte han consumido, como es lógico en este tipo de proyectos punteros, gran parte del tiempo asignado al PFC. Este es uno de los puntos que justifican su duración debido a que los conceptos presentados y las habilidades necesarias para implementarlos y probarlos requieren de un alto nivel técnico.

  Cabe destacar, como se puede observar en la desviación de la planificación inicial, que en el transcurso de los seguimientos semanales se constató el gran dinamismo existente en el campo de la virtualización. La gran cantidad de empresas e instituciones que lo utilizan refuerzan una gran comunidad de desarrolladores que presenta aportaciones y mejoras constantemente. De esta forma se decidió seguir con la planificación prevista pero realizando adecuaciones técnicas en los momentos clave. Ejemplos de esta revisión ocurrieron con la salida (*release*) de nuevas versiones del hipervisor *KVM* que incorporaban cambios tecnológicos fundamentales (*major releases*) o la aparición de versiones estables de plataformas gráficas de gestión.

  De esta forma, a diferencia de lo inicialmente planificado, las tareas de documentación y seguimiento se han extendido a lo largo de todo el proyecto. Esto ha permitido incorporar las mejoras fundamentales al proyecto para presentar una solución razonable, eficiente y adecuada tecnológicamente al presente.

  También podemos observar la existencia de una desviación en las fechas destinadas a realizar el análisis de los sistemas gráficos de gestión del entorno virtual y las posibles

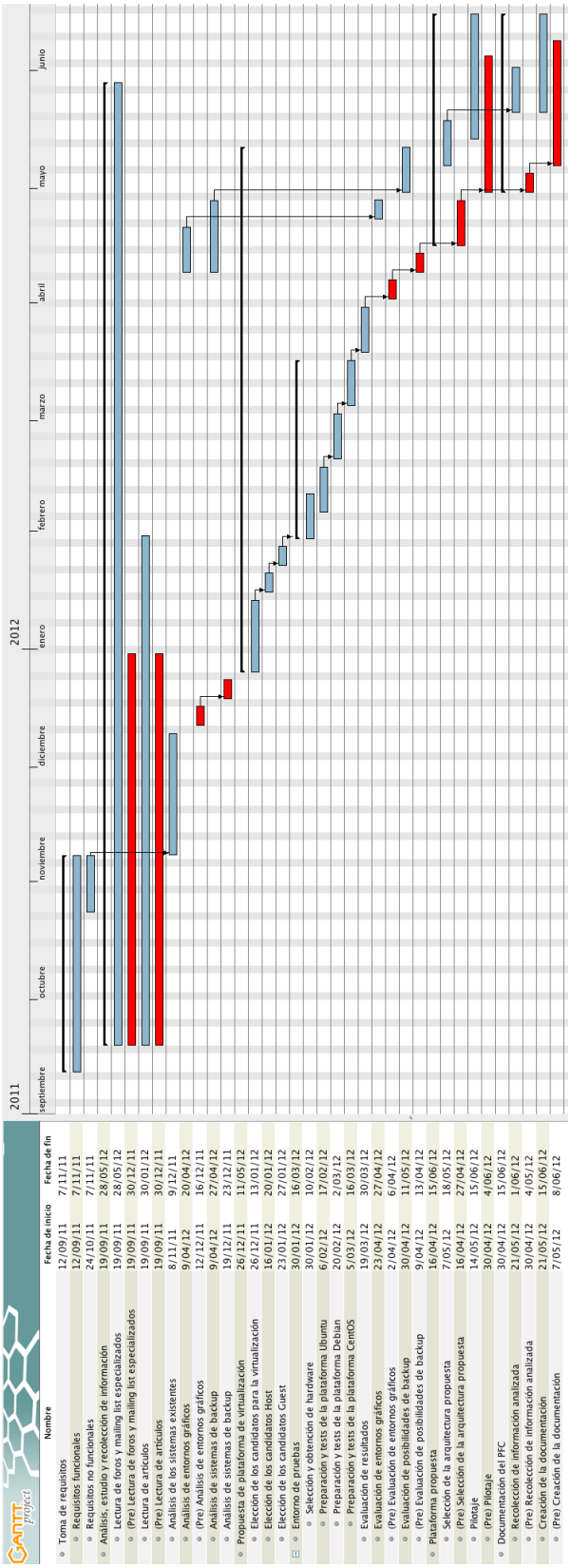


Figura 7.1: Diagrama de Gantt con la planificación **(Pre)**via y real del proyecto

soluciones de *backup*. Esto fue debido al retraso que sufrieron en finalizar el desarrollo de las versiones estables varios programas de referencia. Estas nuevas versiones incorporaban mejoras significativas y debido a que no influían en el resto de la plataforma se decidió postergar su análisis.

- **Propuesta de plataforma de virtualización**: Esta fase recoge el análisis realizado de las diferentes alternativas, la elección de los candidatos y la realización de las pruebas comparativas. El tiempo dedicado a las distintas tareas prácticas ha consumido la mayor parte del proyecto debido a la eminente naturaleza técnica que requiere la solución propuesta.

  Las desviaciones temporales existentes en esta fase correspondientes a las tareas de evaluación de sistemas gráficos de gestión y sistemas de copias de seguridad son consecuencia del punto anterior.

- **Plataforma propuesta**: La solución presentada en este proyecto ha sido inferida fácilmente de los resultados obtenidos y los requisitos iniciales. Sin embargo, la adecuación de la plataforma seleccionada al modelo existente en el *RDlab* ha obligado a definir y procedimentar los pasos necesarios para su aplicación.

  La realización de una fase de pilotaje de la arquitectura resultante se corresponde al test de estabilidad del modelo propuesto. Se ha considerado que un mes de funcionamiento de seis servidores virtualizados en producción nos permite validar esta arquitectura.

- **Documentación del PFC**: La realización de esta documentación, desgraciadamente, se ha visto afectada directamente por los retrasos acumulados. De esta forma nos hemos visto obligados a realizar un ajuste importante en las fechas destinadas a la creación de esta memoria y por tanto retrasar su presentación.

## 7.2. Cálculo de costes

En la realización de todo proyecto se suele incluir el cálculo de los costes asociados. Éstos generalmente suelen hacer referencia a los recursos económicos empleados en su realización ya que suelen ser fácilmente cuantificables. Sin embargo, suele quedar oculta una gran parte referente a la implicación en el trabajo realizado y a las horas dedicadas fuera del ámbito de la beca que deseamos destacar.

Como se ha indicado anteriormente, para la realización de este proyecto se decidió otorgar una beca estándar *Univers-UPC* de 20 horas semanales. Esta beca de colaboración, por motivos administrativos y presupuestarios, se realizó en dos períodos que se corresponden a los meses de Septiembre a Diciembre de 2011 y se renovó de Enero a Junio de 2012.

También se han añadido a los costes económicos el del personal del *RDlab* y el asociado al equipamiento hardware utilizado. Quedan fuera de este análisis los costes de infraestructura básica de trabajo como por ejemplo el de las impresoras o el sistema de correo departamental.

| GRUPO | CONCEPTO | HORAS | PRECIO | SEMANAS | TOTAL |
|---|---|---|---|---|---|
| Personal | Beca correspondiente al 2011 | 20 | 6,30 | 16 | 2.016,00 |
| | Beca correspondiente al 2012 | 20 | 6,50 | 24 | 3.120,00 |
| | Soporte Técnico: Personal IC nivel 1 | 5 | 15,00 | 40 | 3.000,00 |
| | Tutorización: Responsable TIC | 2 | 30,00 | 40 | 2.400,00 |
| Hardware | Servidor Dell R310 | | 1.900,00 | | 1.900,00 |
| | PC Nec powermate ml470 | | 800,00 | | 800,00 |
| | | | | | **13.236,00 €** |

Figura 7.2: Costes económicos del PFC

Para la realización de este proyecto se ha contado con la ayuda experta de los técnicos del *RDlab*. Su aportación ha consistido principalmente en la resolución de dudas técnicas y tecnológicas que han ido surgiendo así cómo el soporte en el análisis de los sistemas en producción que utilizan los distintos grupos de investigación. La tutorización de este PFC, tal y como ya se ha comentado, se realizó mediante reuniones semanales.

La infraestructura hardware utilizada para este trabajo ha constado de un servidor y una estación de trabajo. El servidor, modelo *Dell* R310, como los utilizados habitualmente para los grupos de investigación, nos permitió implementar un prototipo válido y evaluarlo adecuadamente. El PC utilizado, un *Nec* nl470, sirvió de estación de trabajo y quedó destinado como equipo de sobremesa.

## 7.3. Objetivos conseguidos

Una vez finalizado el proyecto es necesario recapitular a los objetivos iniciales de forma que podamos evaluar la consecución o no de estos. Este apartado recogerá la evaluación *a posteriori* del proyecto con el ánimo de establecer qué grado de realización se ha logrado mediante el trabajo realizado.

Los objetivos principales de este PFC (recogidos en el capítulo 1) eran la realización de un estudio en profundidad de los nuevos paradigmas TIC y la realización de una propuesta que permitiese incorporarlos al *RDlab*. De estos nuevos modelos de gestión, la virtualización ha resultado ser la piedra angular sobre la que construir una nueva manera de gestionar tanto los servidores como los servicios.

En este documento se recoge el estudio y evaluación de las distintas tecnologías que nos permiten llevar a cabo la virtualización así como una propuesta en firme de cómo deben ser implementadas en el Departamento de *LSI*. Esta propuesta ha sido contrastada mediante la realización de un prototipo totalmente funcional que ha estado en producción durante más de un mes.

A continuación repasaremos brevemente los *requisitos funcionales* y *no funcionales* descritos en el capítulo 2 para contrastarlos y comprobar qué grado de satisfacción proporciona la solución elegida en este PFC.

- **Requisitos funcionales**: La plataforma propuesta anteriormente se compone de un único sistema de virtualización completa. De esta forma cualquier sistema existente puede ser virtualizado ya que independiza al *guest* tanto del hardware como del sistema operativo del *host*. Esta independencia queda condicionada a que el sistema operativo *host* sea de 64 bits, puesto que un sistema host de 32 bits no puede virtualizar un sistema de 64 bits.

  La plataforma de virtualización *KVM* permite la ejecución de tantas máquinas virtuales en el sistema *host* como soporte su hardware. Además, también contempla de una manera sencilla la modificación de los recursos asignados a la máquina virtual. Esto puede realizarse directamente mediante la línea de comandos (***CLI**, Command Line Interface*) incluyendo un parámetro con el valor asignado o mediante el uso de un sistema gráfico de gestión de máquinas virtuales.

- **Requisitos no funcionales**: Todos los elementos evaluados en este documento así como los que integran la propuesta de la plataforma de virtualización son libres. Además se incluye dentro de la propuesta una plataforma gráfica web denominada *Convirture* que permite de forma sencilla la gestión de todos los sistemas *host* y *guest* existentes.

  La gestión de los mecanismos de seguridad viene determinada principalmente por el sistema *host* y la arquitectura de red utilizada. El sistema *host* ejecuta una virtualización completa, lo que independiza a los distintos sistemas entre sí, evitando posibles problemas de seguridad. Además, como se utiliza un sistema Linux podemos usar de todos los mecanismos de seguridad que proporciona (*IPtables*, *SELinux*, Logs,...). La arquitectura de red utilizada, mediante el uso de tecnología *VLAN* y rangos IP diferentes para los sistemas *host* y *guest* permite monitorizar y restringir todo el tráfico del sistema con las herramientas de red estándar.

  Nuestra arquitectura soporta el uso de mecanismos de alta disponibilidad mediante la incorporación de una zona común entre los distintos *hosts*, por ejemplo mediante el uso de un array de discos. De esta forma cualquier máquina virtual es susceptible de ser ejecutada en otro *host* en caso de fallo.

  Para la realización de las copias de los sistemas guests (*backup*) se propone el uso de un sistema como *LVM*. De esta forma se puede utilizar el mecanismo de *snapshots* que permite obtener una imagen estable del sistema en ejecución y realizar una copia de éste.

Como ya hemos detallado anteriormente, tanto los objetivos como los requisitos de este proyecto han sido satisfechos. En el punto 6.3 de este documento se recogen las distintas posibilidades existentes de mejora así como las ampliaciones futuras que han quedado fuera del ámbito de este **P**royecto **F**inal de **C**arrera de Ingeniería Técnica de sistemas.

# APÉNDICE A

## Anexo

Archivo: /home/xperalta/Escritorio/ficheros_host/interfaces Página 1 de 1

```
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual

auto eth1
iface eth1 inet manual

auto eth0.129
iface eth0.129 inet manual
        vlan_raw_device eth0

auto virbr0
iface virbr0 inet static
        address 172.29.XXX.XXX
        netmask 255.255.XXX.XXX
        network 172.29.XXX.XXX
        gateway 172.29.XXX.XXX
        broadcast 172.29.XXX.XXX
        bridge_ports eth0
        bridge_maxwait 5
        bridge_fd 1
        bridge_stp on

auto virbr1
iface virbr1 inet manual
        bridge_ports eth0.129
        bridge_maxwait 5
        bridge_fd 1
        bridge_stp on
```

Figura A.1: Fichero de configuración de la red del *host*

```
kernel = "/usr/lib/xen-4.0/boot/hvmloader"
vif = ['bridge=virbr1']
auto_start_vm = 0
on_shutdown = 'destroy'
arch_libdir = 'lib64'
disk = ['phy:/dev/rdlabvg/xen-debian1,hda,w', 'file:/home/root-rdlab/ubuntu-10.04.4-desktop-
amd64.iso,hdc:cdrom,r']
name = "xen-debian1"
usb = 1
platform = 'kvm'
on_reboot = 'restart'
boot = 'cd'
os_version = u'10.04'
pae = 1
memory = 1500
acpi = 1
vnc = 1
device_model = "/usr/lib64/xen/bin/qemu-dm"
on_crash = 'destroy'
network_mode = 'tap'
apic = 1
sdl = 0
bootloader = ''
arch = 'x86_64'
usbdevice = 'tablet'
builder = 'hvm'
vcpus = 4
vfb = [ 'type=vnc,vncdisplay=0']
keymap='es'
usbdevice='table'
```

Figura A.2: Fichero de configuración de un *guest Xen*

Archivo: /home/xperalta/Escritorio/PFC/XEN/grub.cfg-xen Página 1 de 1

```
menuentry 'Ubuntu, with Linux 2.6.32.41' --class ubuntu --class gnu-linux --class gnu --class os {
        recordfail
        insmod raid
        insmod mdraid
        insmod ext2
        set root='(md0)'
        multiboot /boot/xen.gz noreboot dummy=dummy mem_set=2097152
        module /boot/vmlinuz-2.6.32.41 dummy=dummy root=UUID=5449ef6c-2692-4c80-9d40-e394076c920f ro
rootdelay=90  rootdelay=90
        module  /boot/initrd.img-2.6.32.41
}
```

Figura A.3: Fichero de configuración del grub *Dom0 Xen*

```
<domain type="kvm">
  <name>osmgmt02</name>
  <uuid>c533fb5c-71b7-2e4c-8b9a-7742ba3ada03</uuid>
  <memory>20971520</memory>
  <currentmemory>20971520</currentmemory>
  <vcpu>4</vcpu>
  <os>
    <type arch="x86_64" machine="rhel5.4.0">hvm</type>
    <boot dev="hd">
  </boot></os>
  <features>
    <acpi>
    <apic>
    <pae>
  </pae></apic></acpi></features>
  <clock offset="utc">
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type="file" device="disk">
      <driver name="qemu" cache="none">
      <source file="/usr/local/imgs/osmgmt02_20g.img">
      <target dev="hda" bus="ide">
    </target></driver></disk>
    <disk type="file" device="disk">
      <source file="/usr/local/imgs/osmgmt02_swap_20g.img">
      <target dev="hdb" bus="ide">
    </target></disk>
    <disk type="block" device="disk">
      <source dev="/dev/cciss/c0d2">
      <target dev="hdd" bus="ide">
    </target></disk>
    <interface type="bridge">
      <mac address="54:52:00:14:1e:c1">
      <source bridge="br3">
      <model type="e1000">
    </model></mac></interface>
    <interface type="bridge">
      <mac address="54:52:00:14:1e:d4">
      <source bridge="br1">
      <model type="e1000">
    </model></mac></interface>
    <serial type="pty">
      <source path="/dev/pts/5">
      <target port="0">
    </target></serial>
    <console type="pty" tty="/dev/pts/5">
      <source path="/dev/pts/5">
      <target port="0">
    </target></console>
    <input bus="ps2" type="mouse">
    <graphics type="vnc" port="-1" autoport="yes" keymap="en-us">
  </graphics></devices>
</clock></domain>
```

Figura A.4: Fichero de configuración de un *guest KVM*

UBUNTU
KVM

| | 1 | 1 | 2 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|
| Proceso 1 | 37,7434 | 27,7916 | 17,4201 | 13,69 | 11,3871 | 6,4765 |
| Proceso 2 | | | 17,3989 | 13,8019 | 11,4354 | 6,4793 |
| Proceso 3 | | | | | 11,4759 | 5,2585 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 27,8564 | 16,5654 | 6,8765 | 3,4761 |
| Proceso 2 | | 16,6781 | 6,7845 | 2,8764 |
| Proceso 3 | | | 6,5631 | 3,7694 |
| Proceso 4 | | | | 2,8976 |

DEBIAN
KVM

| | Host | 1 Guest | 2 Host | 2 Guest | 3 Host | 3 Guest |
|---|---|---|---|---|---|---|
| Proceso 1 | 39,6743 | 31,3797 | 18,6346 | 14,8646 | 11,7854 | 6,3895 |
| Proceso 2 | | | 18,7786 | 14,8674 | 11,3986 | 6,5834 |
| Proceso 3 | | | | | 11,3496 | 6,4875 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 30,6842 | 15,8241 | 7,6847 | 3,4961 |
| Proceso 2 | | 15,0274 | 6,9749 | 3,9624 |
| Proceso 3 | | | 7,328 | 3,0872 |
| | | | | 3,9692 |

CENTOS
KVM

| | Host | 1 Guest | 2 Host | 2 Guest | 3 Host | 3 Guest |
|---|---|---|---|---|---|---|
| Proceso 1 | 36,689 | 28,7851 | 17,9542 | 14,729 | 12,067 | 6,6834 |
| Proceso 2 | | | 17,9834 | 14,7675 | 11,8963 | 6,6794 |
| Proceso 3 | | | | | 11,8358 | 6,599 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 27,4953 | 17,4963 | 11,8053 | 3,4075 |
| Proceso 2 | | 17,5841 | 11,816 | 3,3909 |
| Proceso 3 | | | 11,802 | 3,8364 |
| Proceso 4 | | | | 3,7385 |

Figura A.5: Datos RAMspeed sistemas

UBUNTU
KVM

| | 1 | 1 | 2 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|
| Proceso 1 | 37,7434 | 27,7916 | 17,4201 | 13,69 | 11,3871 | 6,4765 |
| Proceso 2 | | | 17,3989 | 13,8019 | 11,4354 | 6,4793 |
| Proceso 3 | | | | | 11,4759 | 5,2585 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 27,8564 | 16,5654 | 6,8765 | 3,4761 |
| Proceso 2 | | 16,6781 | 6,7845 | 2,8764 |
| Proceso 3 | | | 6,5631 | 3,7694 |
| Proceso 4 | | | | 2,8976 |

DEBIAN
KVM

| | Host | 1 Guest | 2 Host | 2 Guest | 3 Host | 3 Guest |
|---|---|---|---|---|---|---|
| Proceso 1 | 39,6743 | 31,3797 | 18,6346 | 14,8646 | 11,7854 | 6,3895 |
| Proceso 2 | | | 18,7786 | 14,8674 | 11,3986 | 6,5834 |
| Proceso 3 | | | | | 11,3496 | 6,4875 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 30,6842 | 15,8241 | 7,6847 | 3,4961 |
| Proceso 2 | | 15,0274 | 6,9749 | 3,9624 |
| Proceso 3 | | | 7,328 | 3,0872 |
| | | | | 3,9692 |

CENTOS
KVM

| | Host | 1 Guest | 2 Host | 2 Guest | 3 Host | 3 Guest |
|---|---|---|---|---|---|---|
| Proceso 1 | 36,689 | 28,7851 | 17,9542 | 14,729 | 12,067 | 6,6834 |
| Proceso 2 | | | 17,9834 | 14,7675 | 11,8963 | 6,6794 |
| Proceso 3 | | | | | 11,8358 | 6,599 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 27,4953 | 17,4963 | 11,8053 | 3,4075 |
| Proceso 2 | | 17,5841 | 11,816 | 3,3909 |
| Proceso 3 | | | 11,802 | 3,8364 |
| Proceso 4 | | | | 3,7385 |

Figura A.6: Datos iperf sistemas

UBUNTU
KVM

| | 1 | 1 | 2 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|
| Proceso 1 | 37,7434 | 27,7916 | 17,4201 | 13,69 | 11,3871 | 6,4765 |
| Proceso 2 | | | 17,3989 | 13,8019 | 11,4354 | 6,4793 |
| Proceso 3 | | | | | 11,4759 | 5,2585 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 27,8564 | 16,5654 | 6,8765 | 3,4761 |
| Proceso 2 | | 16,6781 | 6,7845 | 2,8764 |
| Proceso 3 | | | 6,5631 | 3,7694 |
| Proceso 4 | | | | 2,8976 |

DEBIAN
KVM

| | Host | 1 Guest | 2 Host | 2 Guest | 3 Host | 3 Guest |
|---|---|---|---|---|---|---|
| Proceso 1 | 39,6743 | 31,3797 | 18,6346 | 14,8646 | 11,7854 | 6,3895 |
| Proceso 2 | | | 18,7786 | 14,8674 | 11,3986 | 6,5834 |
| Proceso 3 | | | | | 11,3496 | 6,4875 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 30,6842 | 15,8241 | 7,6847 | 3,4961 |
| Proceso 2 | | 15,0274 | 6,9749 | 3,9624 |
| Proceso 3 | | | 7,328 | 3,0872 |
| | | | | 3,9692 |

CENTOS
KVM

| | Host | 1 Guest | 2 Host | 2 Guest | 3 Host | 3 Guest |
|---|---|---|---|---|---|---|
| Proceso 1 | 36,689 | 28,7851 | 17,9542 | 14,729 | 12,067 | 6,6834 |
| Proceso 2 | | | 17,9834 | 14,7675 | 11,8963 | 6,6794 |
| Proceso 3 | | | | | 11,8358 | 6,599 |
| Proceso 4 | | | | | | |

XEN

| | 1 Guest | 2 Guest | 3 Guest | 4 Guest |
|---|---|---|---|---|
| Proceso 1 | 27,4953 | 17,4963 | 11,8053 | 3,4075 |
| Proceso 2 | | 17,5841 | 11,816 | 3,3909 |
| Proceso 3 | | | 11,802 | 3,8364 |
| Proceso 4 | | | | 3,7385 |

Figura A.7: Datos linpack sistemas

Ubuntu disco

| | Read Char | Read Block | Rewrite Block | Write Char | Write Block |
|---|---|---|---|---|---|
| Host 1 | 60335 | 96630 | 70982 | 62792 | 292863 |
| Guest1 | 49679 | 76489 | 29336 | 63448 | 345620 |
| Guest file | 31388 | 44737 | 27428 | 58409 | 288568 |
| Host 2-1 | 59792 | 37995 | 21168 | 39793 | 118199 |
| Host 2-2 | 59376 | 36542 | 21221 | 39794 | 69618 |
| Guest2-1 | 34220 | 33721 | 22895 | 68424 | 68424 |
| Guest2-2 | 35756 | 35788 | 8128 | 19954 | 120559 |
| Guest file 2-1 | 14479 | 22219 | 11728 | 22529 | 39294 |
| Guest file 2-2 | 20596 | 8549 | 7884 | 42208 | 93962 |
| Host 3-1 | 38072 | 15257 | 9938 | 21119 | 28401 |
| Host 3-2 | 37489 | 15077 | 9878 | 20903 | 27904 |
| Host 3-3 | 38397 | 16941 | 9943 | 20472 | 27544 |
| Guest3-1 | 6098 | 4826 | 1948 | 11464 | 38958 |
| Guest3-2 | 5811 | 2577 | 3696 | 13162 | 14983 |
| Guest3-3 | 3946 | 13867 | 7827 | 29310 | 27784 |
| Guest file 3-1 | 4889 | 10848 | 7080 | 16575 | 16393 |
| Guest file 3-2 | 7066 | 4577 | 3250 | 22069 | 21296 |
| Guest file 3-3 | 6518 | 5224 | 6123 | 23442 | 27442 |
| Host 4-1 | 13742 | 8614 | 6429 | 16347 | 21399 |
| Host 4-2 | 13724 | 9704 | 6438 | 15639 | 20864 |
| Host 4-3 | 13515 | 8629 | 6428 | 16300 | 21271 |
| Host 4-4 | 13641 | 8796 | 6488 | 16219 | 21201 |
| Guest4-1 | 3176 | 1808 | 1504 | 3710 | 15630 |
| Guest4-2 | 3519 | 1404 | 1746 | 4730 | 27898 |
| Guest4-3 | 3404 | 1666 | 1440 | 4391 | 17595 |
| Guest4-4 | 2437 | 7770 | 4649 | 1840 | 2867 |
| Guest file 4-1 | 1828 | 6070 | 2268 | 4346 | 3842 |
| Guest file 4-2 | 4259 | 1364 | 1201 | 29896 | 54714 |
| Guest file 4-3 | 2707 | 7462 | 1414 | 2303 | 6127 |
| Guest file 4-4 | 4046 | 1371 | 1643 | 14484 | 21518 |
| 2 host | 119168 | 74537 | 42389 | 79587 | 187817 |
| 2 guest | 69976 | 69509 | 31023 | 88378 | 188983 |
| 2 guest file | 35075 | 30768 | 19612 | 64737 | 133256 |
| 3 host | 113958 | 47275 | 29759 | 62494 | 83849 |
| 3 guest | 15855 | 21270 | 13471 | 53936 | 81725 |
| 3 guest file | 18473 | 20649 | 16453 | 62086 | 65131 |
| 4 host | 54622 | 35743 | 25783 | 64505 | 84735 |
| 4 guest | 12840 | 16267 | 6526 | 51029 | 86201 |
| 4 guest file | 23740 | 13674 | 11178 | 29050 | 82324 |
| 2 guest xen | 41014 | 45668 | 23519 | 53310 | 167459 |
| 2 guest xen file | 27242 | 33237 | 16854 | 35615 | 120810 |
| 3 guest xen | 11869 | 9053 | 6976 | 20475 | 62528 |
| 3 guest xen file | 7976 | 8059 | 9037 | 16499 | 53499 |
| 4 guest xen | 8581 | 7298 | 7309 | 12823 | 36950 |
| 4 guest xen file | 4495 | 5040 | 7212 | 7576 | 13585 |
| Guest xen 1 | 30765 | 57832 | 15481 | 48345 | 270386 |
| Guest xen file 1 | 20482 | 37041 | 19306 | 34074 | 229083 |
| Guest xen 2-1 | 21739 | 22142 | 12871 | 27912 | 80213 |
| Guest xen 2-2 | 19275 | 23526 | 10648 | 25398 | 87246 |
| Guest xen file 2-1 | 13639 | 16756 | 7203 | 17053 | 59563 |
| Guest xen file 2-2 | 13603 | 16481 | 9651 | 18562 | 61247 |
| Guest xen 3-1 | 4035 | 2387 | 849 | 5873 | 12531 |

Página 1

Figura A.8: Datos de bonnie++ *Ubuntu*

CentOS disco

| | Read Char | Read Block | Rewrite Block | Write Char | Write Block |
|---|---|---|---|---|---|
| 1 Host | 58426 | 95623 | 68312 | 63578 | 290564 |
| Guest1 | 50153 | 74803 | 26592 | 60531 | 318356 |
| Guest file | 32519 | 50714 | 41863 | 24032 | 214375 |
| Host 2-1 | 57832 | 24889 | 23678 | 37223 | 71498 |
| Host 2-2 | 57453 | 24576 | 23992 | 37823 | 129504 |
| Guest2-1 | 36054 | 23164 | 20670 | 31086 | 50785 |
| Guest2-2 | 35961 | 24651 | 20412 | 30952 | 71543 |
| Guest file 2-1 | 15662 | 17531 | 12804 | 21753 | 45173 |
| Guest file 2-2 | 20675 | 20094 | 14725 | 17426 | 76341 |
| Host 3-1 | 35598 | 16543 | 8954 | 20761 | 26786 |
| Host 3-2 | 35456 | 15894 | 8643 | 20378 | 26872 |
| Host 3-3 | 34987 | 16134 | 8721 | 20572 | 25786 |
| Guest3-1 | 7634 | 4193 | 2981 | 18356 | 19541 |
| Guest3-2 | 8652 | 7541 | 3865 | 16535 | 24835 |
| Guest3-3 | 4641 | 5802 | 6295 | 17652 | 13580 |
| Guest file 3-1 | 4207 | 3654 | 4375 | 14420 | 25436 |
| Guest file 3-2 | 5283 | 3106 | 4792 | 18655 | 26504 |
| Guest file 3-3 | 4609 | 3285 | 4378 | 16491 | 13428 |
| Host 4-1 | 12981 | 7542 | 5382 | 17643 | 21345 |
| Host 4-2 | 12876 | 7763 | 5862 | 17463 | 20876 |
| Host 4-3 | 13042 | 7923 | 5523 | 17891 | 20923 |
| Host 4-4 | 12643 | 7978 | 5426 | 17632 | 22425 |
| Guest4-1 | 4518 | 3598 | 2065 | 3692 | 18543 |
| Guest4-2 | 3761 | 3562 | 2265 | 3765 | 25683 |
| Guest4-3 | 5762 | 4867 | 2653 | 5629 | 24684 |
| Guest4-4 | 2415 | 3178 | 2348 | 2764 | 16536 |
| Guest file 4-1 | 1756 | 2746 | 1376 | 2672 | 7362 |
| Guest file 4-2 | 1825 | 5195 | 3061 | 1970 | 12468 |
| Guest file 4-3 | 1563 | 4507 | 3652 | 2644 | 11874 |
| Guest file 4-4 | 2655 | 4356 | 1675 | 1503 | 3261 |
| 2 Hosts | 115285 | 49465 | 47670 | 75046 | 201002 |
| Total 2 guest | 72015 | 47815 | 41082 | 62038 | 122328 |
| Tota 2 guest file | 36337 | 37625 | 27529 | 39179 | 121514 |
| 3 Hosts | 106041 | 48571 | 26318 | 61711 | 79444 |
| Total 3 guest | 20927 | 17536 | 13141 | 52543 | 57956 |
| Total 3 guest file | 14099 | 10045 | 13545 | 49566 | 65368 |
| 4 Hosts | 51542 | 31206 | 22193 | 70629 | 85569 |
| Total 4 guest | 7799 | 16804 | 9764 | 8789 | 34965 |
| Total 4 guest file | 16456 | 15205 | 9331 | 15850 | 85446 |
| Total 2 guest xen | 45412 | 44339 | 32170 | 62163 | 246271 |
| Total 2 guest xen file | 34373 | 46135 | 25570 | 39924 | 112020 |
| Total 3 guest xen | 13157 | 6601 | 5429 | 10539 | 30739 |
| Total 3 guest xen file | 11467 | 6300 | 7674 | 8642 | 20774 |
| Total 4 guest xen | 8527 | 5740 | 8636 | 6291 | 19602 |
| Total 4 guest xen file | 5616 | 5211 | 7042 | 6485 | 12352 |
| Guest xen 2-1 | 23648 | 21486 | 14528 | 35672 | 138629 |
| Guest xen 2-2 | 21764 | 22853 | 17642 | 26491 | 107642 |
| Guest xen file 2-1 | 18742 | 20861 | 13725 | 20861 | 46739 |
| Guest xen file 2-2 | 15631 | 25274 | 11845 | 19063 | 65281 |
| Guest xen 3-1 | 5638 | 2763 | 1003 | 4926 | 16482 |
| Guest xen 3-2 | 3652 | 1765 | 1863 | 3072 | 9521 |
| Guest xen 3-3 | 3867 | 2073 | 2563 | 2541 | 4736 |

Página 1

Figura A.9: Datos de bonnie++ *CentOS*

CentOS disco

| | Read Char | Read Block | Rewrite Block | Write Char | Write Block |
|---|---|---|---|---|---|
| 1 Host | 58426 | 95623 | 68312 | 63578 | 290564 |
| Guest1 | 50153 | 74803 | 26592 | 60531 | 318356 |
| Guest file | 32519 | 50714 | 41863 | 24032 | 214375 |
| Host 2-1 | 57832 | 24889 | 23678 | 37223 | 71498 |
| Host 2-2 | 57453 | 24576 | 23992 | 37823 | 129504 |
| Guest2-1 | 36054 | 23164 | 20670 | 31086 | 50785 |
| Guest2-2 | 35961 | 24651 | 20412 | 30952 | 71543 |
| Guest file 2-1 | 15662 | 17531 | 12804 | 21753 | 45173 |
| Guest file 2-2 | 20675 | 20094 | 14725 | 17426 | 76341 |
| Host 3-1 | 35598 | 16543 | 8954 | 20761 | 26786 |
| Host 3-2 | 35456 | 15894 | 8643 | 20378 | 26872 |
| Host 3-3 | 34987 | 16134 | 8721 | 20572 | 25786 |
| Guest3-1 | 7634 | 4193 | 2981 | 18356 | 19541 |
| Guest3-2 | 8652 | 7541 | 3865 | 16535 | 24835 |
| Guest3-3 | 4641 | 5802 | 6295 | 17652 | 13580 |
| Guest file 3-1 | 4207 | 3654 | 4375 | 14420 | 25436 |
| Guest file 3-2 | 5283 | 3106 | 4792 | 18655 | 26504 |
| Guest file 3-3 | 4609 | 3285 | 4378 | 16491 | 13428 |
| Host 4-1 | 12981 | 7542 | 5382 | 17643 | 21345 |
| Host 4-2 | 12876 | 7763 | 5862 | 17463 | 20876 |
| Host 4-3 | 13042 | 7923 | 5523 | 17891 | 20923 |
| Host 4-4 | 12643 | 7978 | 5426 | 17632 | 22425 |
| Guest4-1 | 4518 | 3598 | 2065 | 3692 | 18543 |
| Guest4-2 | 3761 | 3562 | 2265 | 3765 | 25683 |
| Guest4-3 | 5762 | 4867 | 2653 | 5629 | 24684 |
| Guest4-4 | 2415 | 3178 | 2348 | 2764 | 16536 |
| Guest file 4-1 | 1756 | 2746 | 1376 | 2672 | 7362 |
| Guest file 4-2 | 1825 | 5195 | 3061 | 1970 | 12468 |
| Guest file 4-3 | 1563 | 4507 | 3652 | 2644 | 11874 |
| Guest file 4-4 | 2655 | 4356 | 1675 | 1503 | 3261 |
| 2 Hosts | 115285 | 49465 | 47670 | 75046 | 201002 |
| Total 2 guest | 72015 | 47815 | 41082 | 62038 | 122328 |
| Tota 2 guest file | 36337 | 37625 | 27529 | 39179 | 121514 |
| 3 Hosts | 106041 | 48571 | 26318 | 61711 | 79444 |
| Total 3 guest | 20927 | 17536 | 13141 | 52543 | 57956 |
| Total 3 guest file | 14099 | 10045 | 13545 | 49566 | 65368 |
| 4 Hosts | 51542 | 31206 | 22193 | 70629 | 85569 |
| Total 4 guest | 7799 | 16804 | 9764 | 8789 | 34965 |
| Total 4 guest file | 16456 | 15205 | 9331 | 15850 | 85446 |
| Total 2 guest xen | 45412 | 44339 | 32170 | 62163 | 246271 |
| Total 2 guest xen file | 34373 | 46135 | 25570 | 39924 | 112020 |
| Total 3 guest xen | 13157 | 6601 | 5429 | 10539 | 30739 |
| Total 3 guest xen file | 11467 | 6300 | 7674 | 8642 | 20774 |
| Total 4 guest xen | 8527 | 5740 | 8636 | 6291 | 19602 |
| Total 4 guest xen file | 5616 | 5211 | 7042 | 6485 | 12352 |
| Guest xen 2-1 | 23648 | 21486 | 14528 | 35672 | 138629 |
| Guest xen 2-2 | 21764 | 22853 | 17642 | 26491 | 107642 |
| Guest xen file 2-1 | 18742 | 20861 | 13725 | 20861 | 46739 |
| Guest xen file 2-2 | 15631 | 25274 | 11845 | 19063 | 65281 |
| Guest xen 3-1 | 5638 | 2763 | 1003 | 4926 | 16482 |
| Guest xen 3-2 | 3652 | 1765 | 1863 | 3072 | 9521 |
| Guest xen 3-3 | 3867 | 2073 | 2563 | 2541 | 4736 |

Página 1

Figura A.10: Datos de bonnie++ *Debian*

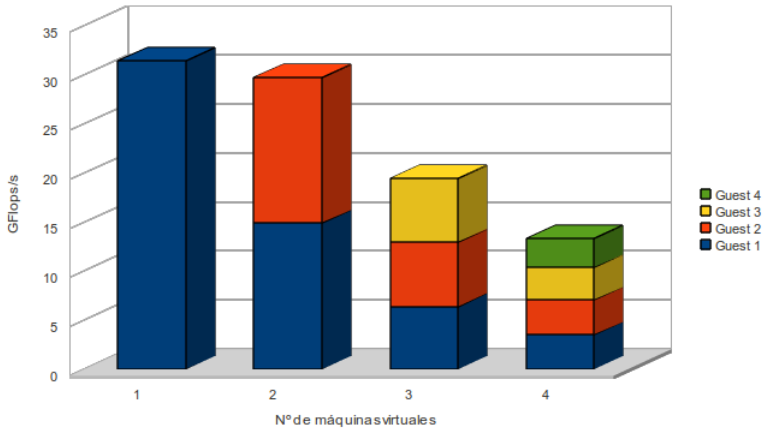


Figura A.11: Resultados de las pruebas de CPU *KVM* en *Debian*

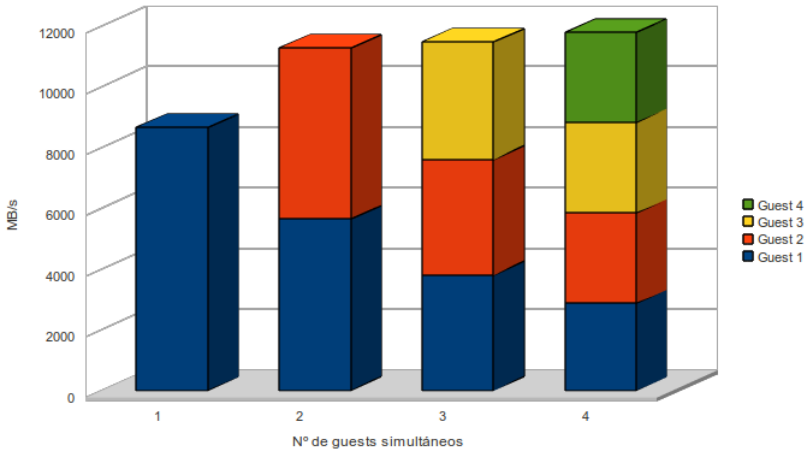


Figura A.12: Resultados de las pruebas de RAM *KVM* en *Debian*

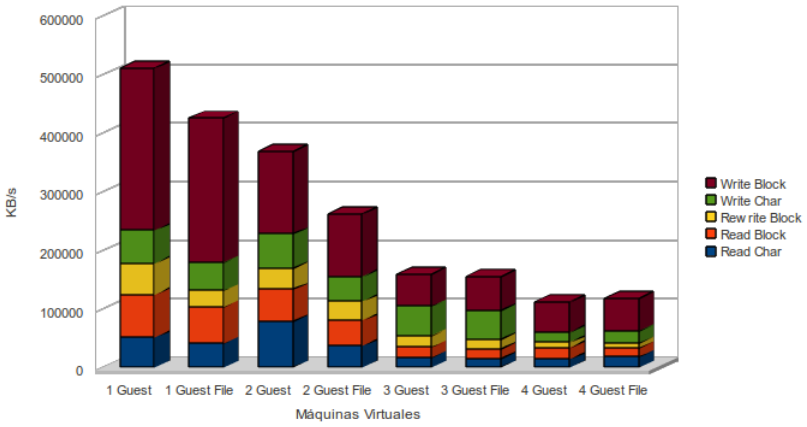


Figura A.13: Resultados de las pruebas de disco *KVM* en *Debian*

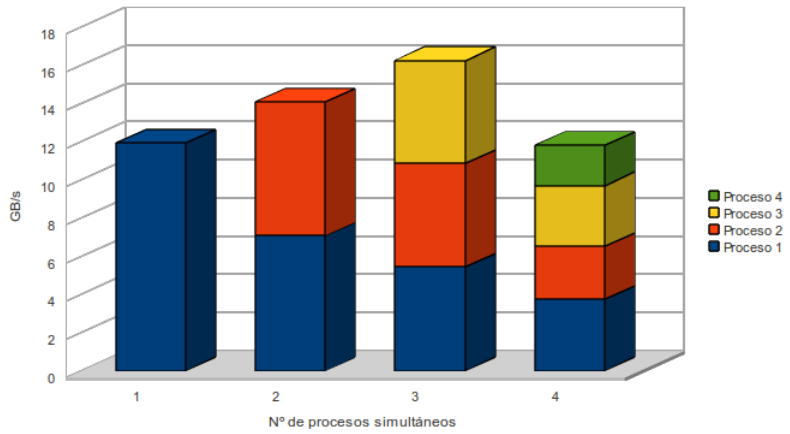


Figura A.14: Resultados de la pruebas de red *KVM* en *Debian*

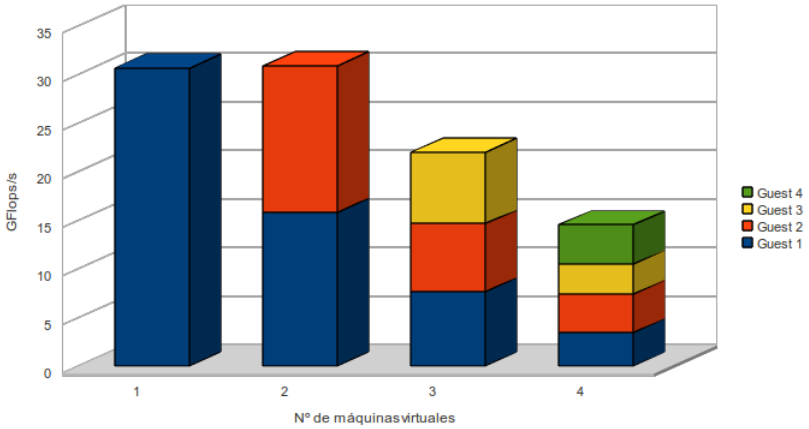


Figura A.15: Resultados de las pruebas de CPU *Xen* en *Debian*

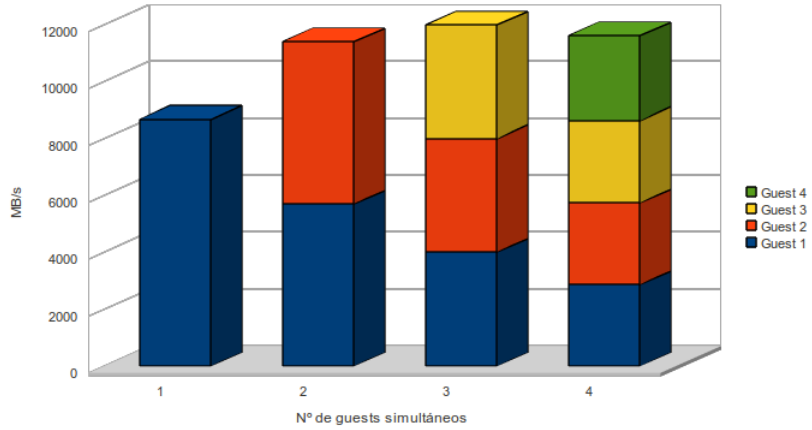


Figura A.16: Resultados de las pruebas de RAM *Xen* en *Debian*

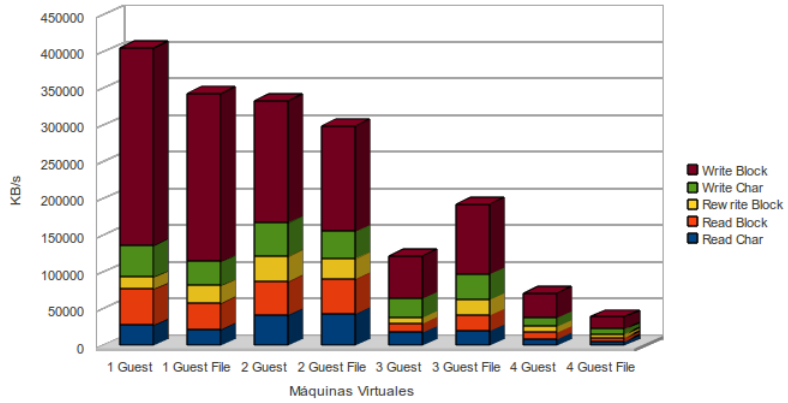


Figura A.17: Resultados de las pruebas de disco *Xen* en *Debian*

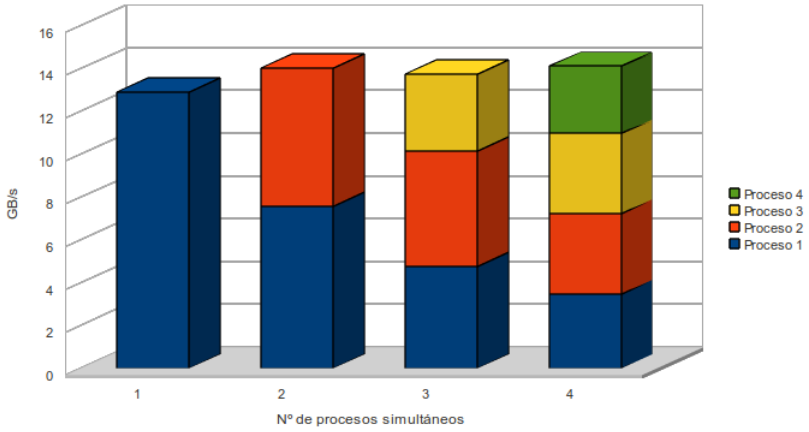


Figura A.18: Resultados de las pruebas de red en *Xen* en *Debian*

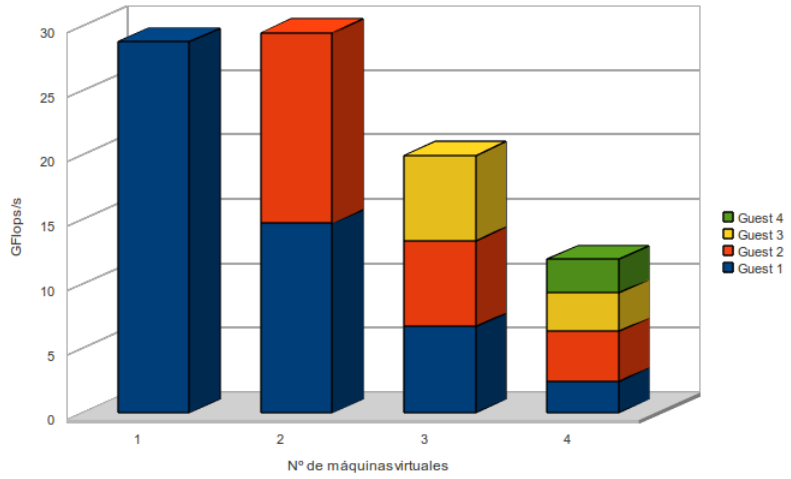


Figura A.19: Resultados de las pruebas de CPU en *KVM* en *CentOS*

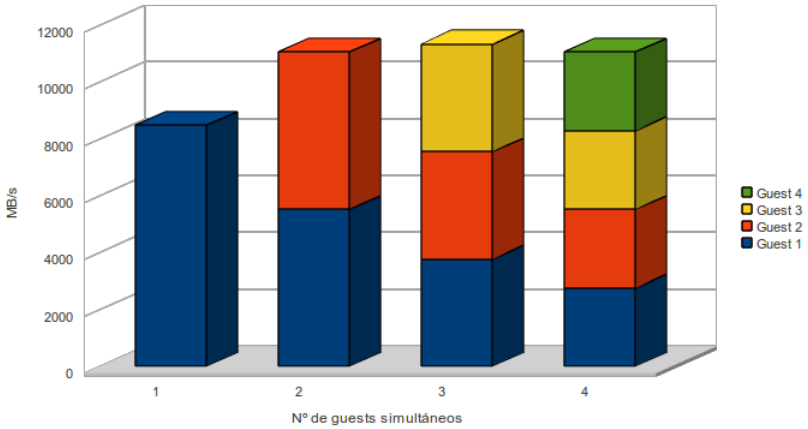


Figura A.20: Resultados de las pruebas de RAM en*KVM* en *CentOS*

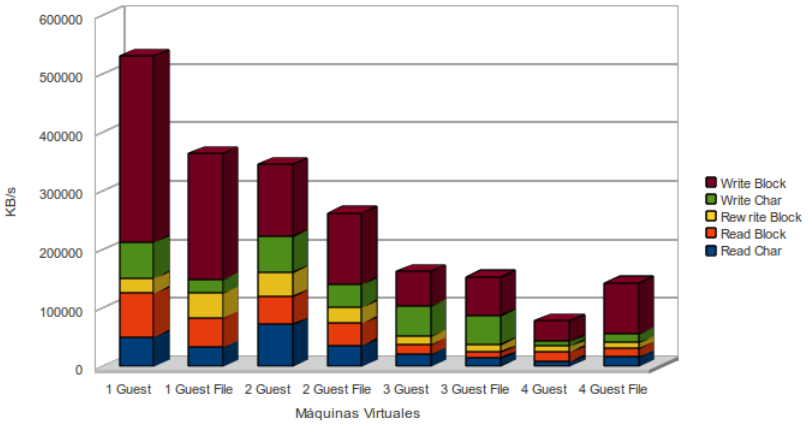


Figura A.21: Resultados de las pruebas de disco en *KVM* en *CentOS*

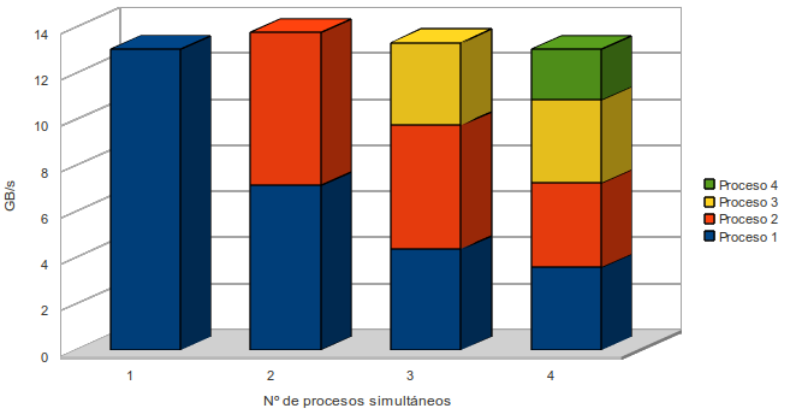


Figura A.22: Resultados de las pruebas de red en*KVM* en *CentOS*

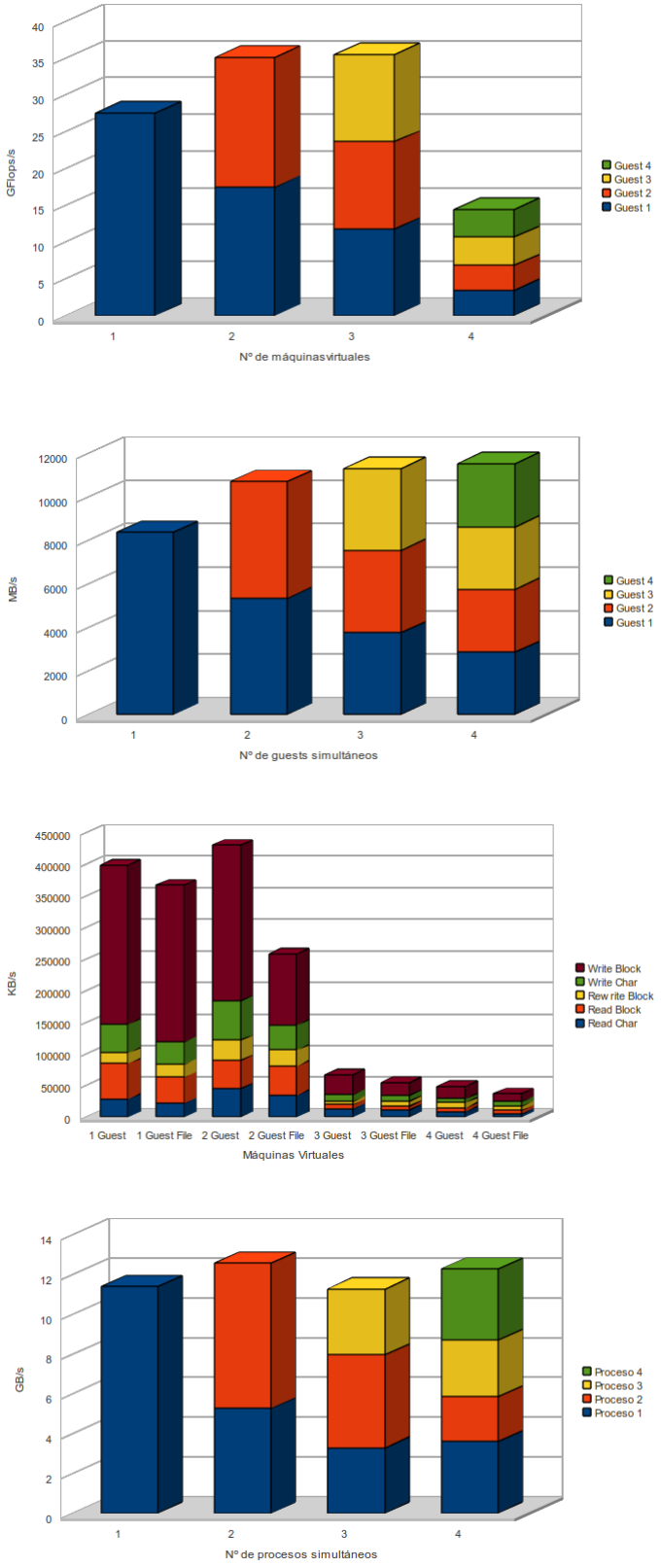


Figura A.23: Resultados pruebas *Xen* en *CentOS*

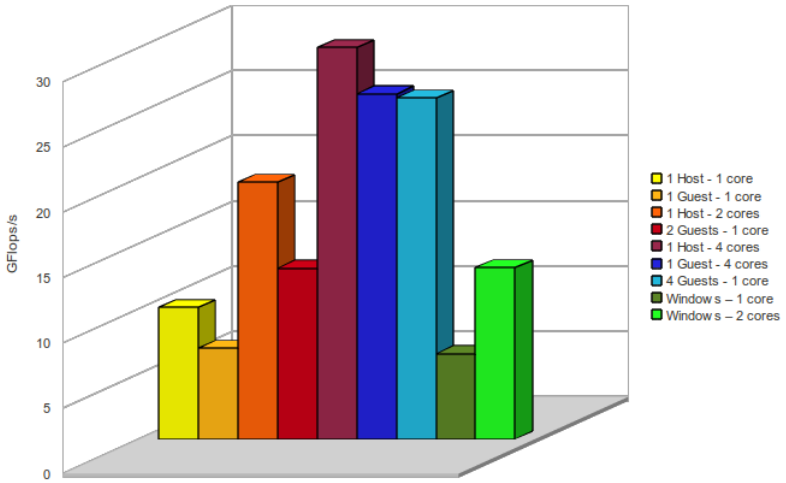


Figura A.24: Comparativa de resultados de disco de CPU en Ubuntu y Windows con *KVM*

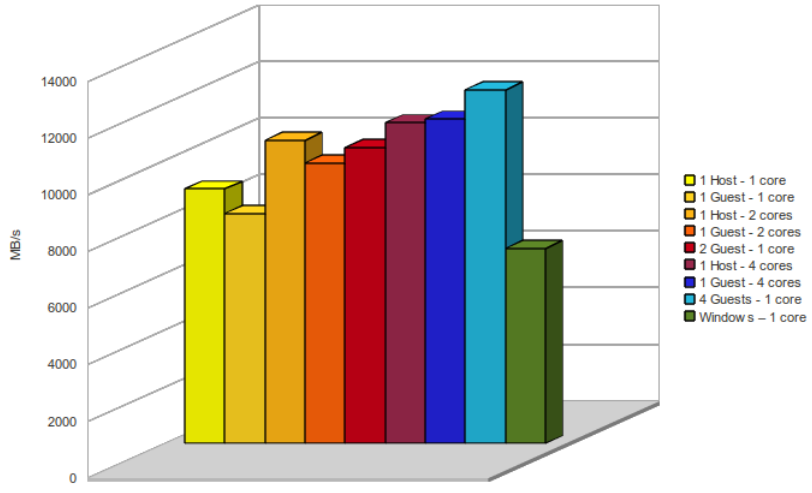


Figura A.25: Comparativa de resultados de RAM en Ubuntu y Windows con *KVM*

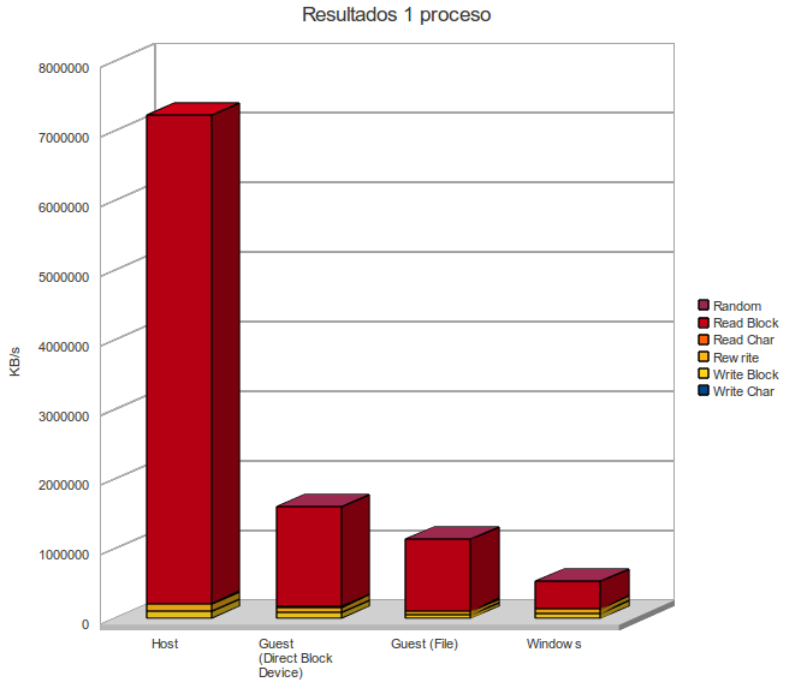


Figura A.26: Comparativa de resultados de disco en Ubuntu y Windows con *KVM*

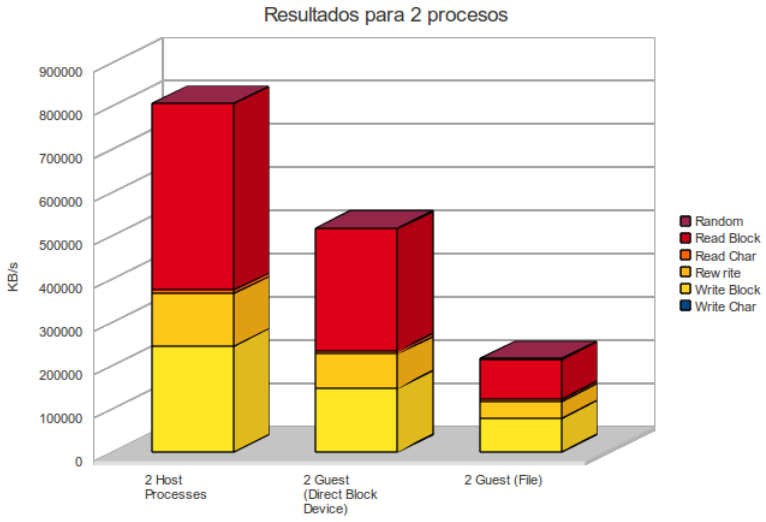


Figura A.27: Comparativa de resultados de disco con dos procesos simultáneos en Ubuntu y Windows con *KVM*

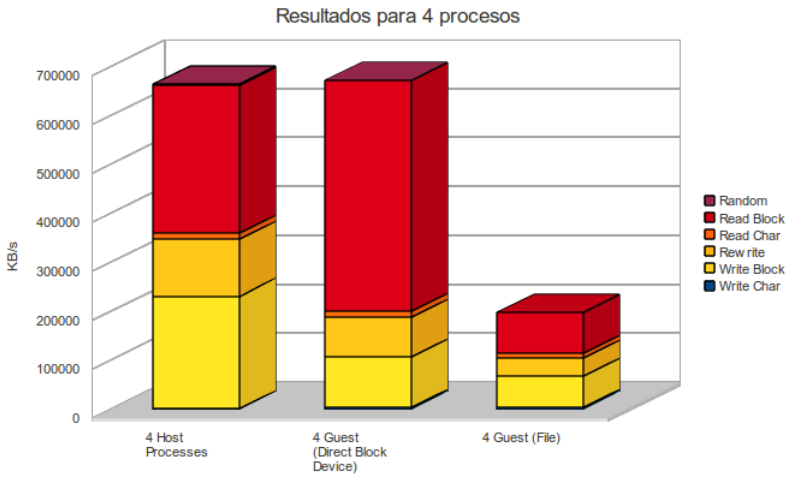


Figura A.28: Comparativa de resultados de disco con cuatro procesos simultáneos en Ubuntu y Windows con *KVM*

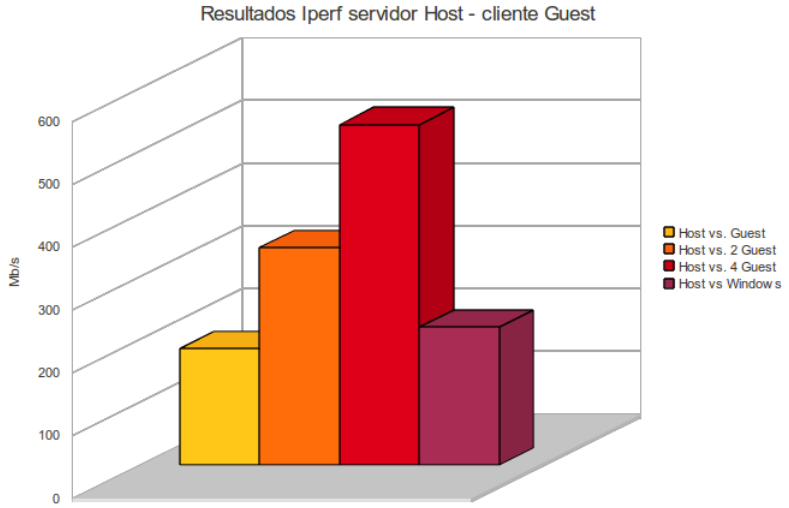


Figura A.29: Comparativa de resultados de red en Ubuntu y Windows con *KVM*

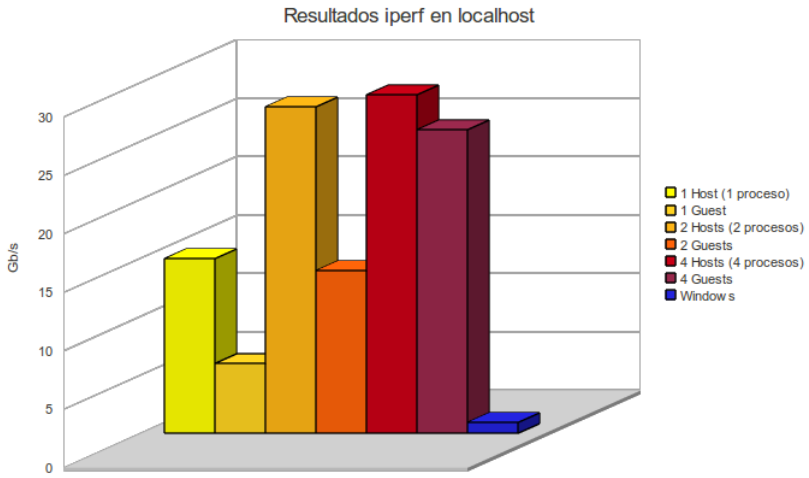


Figura A.30: Comparativa de resultados de red en localhost en Ubuntu y Windows con *KVM*

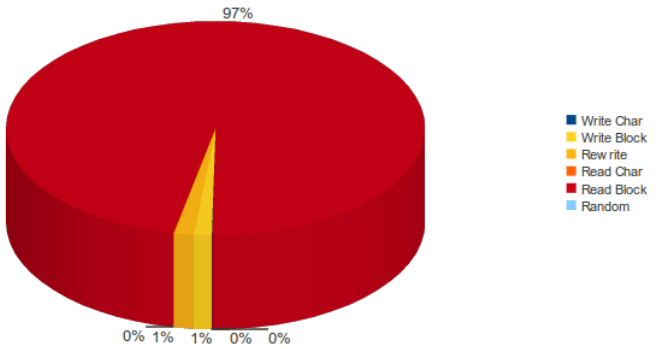


Figura A.31: Peso de cada test de disco en *Host* Ubuntu

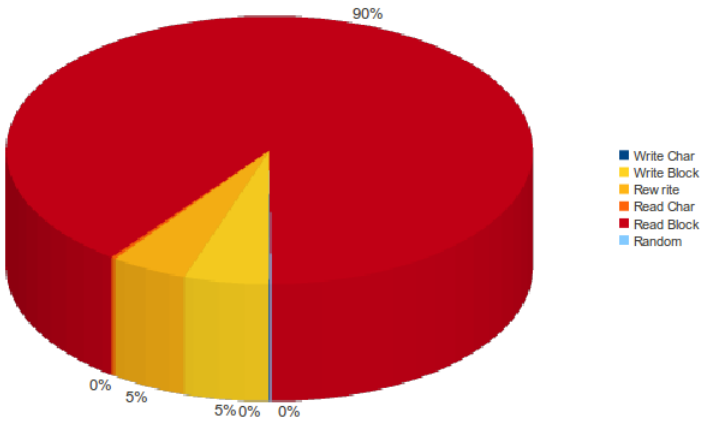


Figura A.32: Peso de cada test de disco en *Guest* Ubuntu con acceso a partición

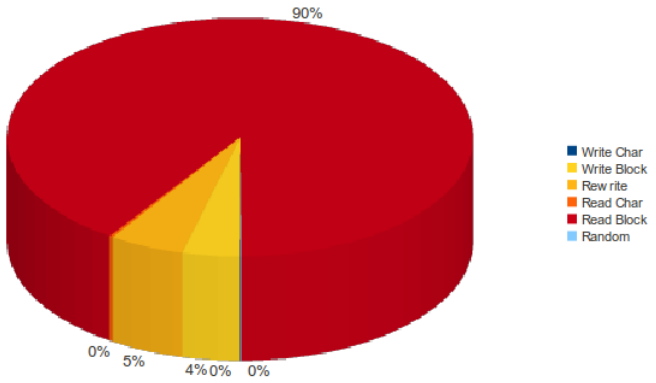


Figura A.33: Peso de cada test de disco en *Guest* Ubuntu con acceso a fichero

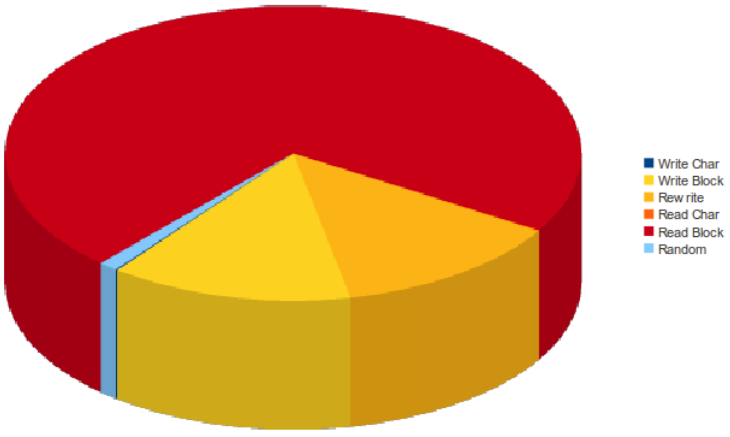


Figura A.34: Peso de cada test de disco en *Guest* Windows con acceso a fichero

# Bibliografía


[AHL] *Anatomía de un Hipervisor Linux*, http://www.ibm.com/developerworks/linux/library/l-hypervisor/index.html.

[CCD] *Cloud Computing for Dummies: Saas, paas, iaas and all that was*, http://cloudvirtualization.wordpress.com/tag/iaas/.

[CCP] *Cloud Computing: Pros and cons*, http://www.networkworld.com/supp/2009/ndc3/051809-cloud-pro-con.html.

[CCV] *Cloud Computing vs Virtualization*, http://www.ehrscope.com/cloud-computing-vs-virtualization.

[CXK] *Quantitative Comparison of Xen and Kvm*, http://www.ibm.com/developerworks/linux/library/l-linux-kvm/.

[DIS] *Web de bonnie++*, http://www.coker.com.au/bonnie++/.

[DKV] *Discover the linux Kernel Virtual Machine*, http://www.ibm.com/developerworks/linux/library/l-linux-kvm/.

[DRC] *Datacenter Report Congres*, http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf.

[ECP] *Evaluating Cascade Performance*, http://www.singlehop.com/blog/evaluating-cascade-performance-part-i-cpu-ram/.

[EPT] *Intel Virtualization Technology*, http://www.intel.com/technology/itj/2006/v10i3/1-hardware/8-virtualization-future.htm.

[GOV] *Guide Of Virtualization*, http://arstechnica.com/hardware/news/2008/08/virtualization-guide-1.ars.

[HAV] *Hardware-Assisted Virtualization*, http://virtualizationreview.com/articles/2009/01/01/hardwareassisted-virtualization.aspxm.

[HOV] *History Of Virtualization*, http://www.ibm.com/developerworks/aix/library/au-aixhpvirtualization/index.html.

[IHT] *Intel® Hyper-threading Technology*, http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html.

[KKL+07] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori, *Kvm: the Linux Virtual Machine Monitor*, Linux Simposium, 2007.

[KSM] *Anatomía of Linux Kernel Shared Memeory*, http://www.ibm.com/developerworks/linux/library/l-kernel-shared-memory/index.html.

[LIN] *Documentación de Linpack*, http://www.netlib.org/linpack/.

[LVH] *Linux kernel Vhost-net*, http://www.linux-kvm.org/page/VhostNet.

[LVM] *Logical Volume Manager*, http://tldp.org/HOWTO/LVM-HOWTO/.

[RAM] *Web de RAMspeed*, http://alasir.com/software/ramspeed/.

[RCS] *Ring Computer Security*, http://en.wikipedia.org/wiki/Ring_%28computer_security%29.

[RHX] *Red Hat drops Xen from rhel*, http://www.computerworld.com/s/article/9175883/Red_Hat_drops_Xen_from_RHEL.

[SVS] *Server Virtualization to Skyrocket*, http://www.crn.com/news/applications-os/214000129/server-desktop-virtualization-to-skyrocket-by-2013-report.htm.

[TC10] Chris Takemura and Luke S. Crawford, *The Book of Xen*, No starch press, 2010.

[TVD] *Timeline of Virtualization Development*, http://en.wikipedia.org/wiki/Timeline_of_virtualization_development.

[USO] *Ubuntu Server bussiness Overview*, http://www.ubuntu.com/business/server/overview.

[VMw09] VMware, *Understanding Memory Resource Management in Vmware esx server*, VMware, Inc, 2009.

[VSS] *Virtualization Studies*, http://vmstudy.blogspot.com.es/.

[WLD] *Wiki Linux Distributions*, http://en.wikipedia.org/wiki/Comparison_of_Linux_distributions.

[WTC] *Usage Statistics and Market Share of Centos for Websites*, http://w3techs.com/technologies/details/os-centos/all/all.

[WTD] *Usage Statistics and Market Share of Debian for Websites*, http://w3techs.com/technologies/details/os-debian/all/all.

[WTU] *Usage Statistics and Market Share of Ubuntu for Websites*, http://w3techs.com/technologies/details/os-ubuntu/all/all.

[XEO] *Xen Overview*, http://wiki.xen.org/wiki/Xen_Overview.

# Glosario de términos

**Anillo de privilegios** : División conceptual de los niveles de permisos de ejecución de un sistema operativo, identificados por un valor numérico. El valor más bajo indica mayor nivel de privilegios.

**Benchmarking** : Aplicación o conjunto de aplicaciones orientadas a medir el rendimiento del sistema en base a ciertos parámetros, siendo los más habituales memoria (RAM), procesador (CPU), acceso a disco o acceso a la red.

**Bridge** (*conexión de red puente*) : Mecanismo de interconexión de dos interfaces de red. Permite la transmisión entre el sistema conectado al puente y un tercer sistema sin que el sistema puente reciba los datos.

**Buffer** : Región de memoria usada para albergar datos mientras son movidos a otra ubicación.

**Cloud Compunting** (*Computación en la nube*) : Paradigma de prestación de servicios de computación alojados de forma externa y trasparente, a los que se accede a través de la red.

**Cloud** (*Nube*) :Ver Cloud Computing.

**Convirt** (Open Source) : Software orientado a la administración de máquinas virtuales creadas en KVM o Xen.

**Device channels** : Enlace punto a punto entre el dispositivo frontend del *guest* y el dispositivo backend del *Dom0* implementados en un buffer en forma de anillo.

**Domain 0** (*driver Domain o dom0*) : *Guest* privilegiado en el que el hipervisor puede confiar. Este *guest* privilegiado es el único con permisos para controlar las funciones de administración como serían encender *guests* y acceder a los dispositivos *hardware*.

**Entrada/Salida** (*E/S*) : Conjunto de interfaces destinados a la comunicación entre sistemas o dispositivos. También referido a las señales enviadas a través de dichos sistemas.

**Extended Page Tables** : Mecanismo de Intel destinado a la creación de tablas de páginas que traducen direcciones físicas del guest a direcciones físicas del host, permitiendo al primero manejar los fallos de página.

**GreenIT** : Nombre por el que se conocen las tecnologías de la información sostenibles y/o respetuosas con el medio ambiente.

**Guest Additions** (*guest-tools*) : Conjunto de controladores de dispositivo diseñados para ser instalados dentro de una máquina virtual creada con Virtual Box que optimizan la comunicación e interacción con el host.

**Guest** : Sistema instalado en una máquina virtual.

**Hipervisor** : Administrador de máquinas virtuales.

**Host** : Sistema anfitrión. Máquina física que contiene máquinas virtuales.

**Kernel Samepage Merging** (*KSM*) : Sistema para *KVM* donde el hipervisor consulta las páginas de cada una de los guests y une las páginas cuyo contenido es el mismo.

**Kernel** : Núcleo central de un sistema operativo. Puente software que conecta las aplicaciones con el procesado de datos de bajo nivel realizado por el hardware.

**KVM** : Hipervisor de tipo 1 o *bare meta*l, el cual se ejecuta sobre el *hardware* al mismo nivel que lo hace un sistema operativo.

**Librería** (Biblioteca) : Colección de recursos (código y/o datos) usados en el desarrollo de software, que ofrecen servicios a otros programas.

**Logical Volume Manager** (*LVM*) : Gestor de volúmenes lógicos. Permite combinar varias particiones de disco en un solo volumen accesible por el sistema operativo.

**Memory Ballooning** : Sistema de compartición de memoria entre los *hosts* y los *guest*. Los *guest* contienen un *balloon* (globo) gestionado por el *host*, que permite a este obtener memoria de un *guest* que no la esté usando para distribuirla a otros *guest* que lo necesiten.

**Memory Management Unit** (*MMU*) : Componente hardware que traduce direcciones lógicas de memoria en direcciones físicas.

**Multicore** : Procesador con más de un núcleo de ejecución.

**NAT** (*Network Address Translation*) : Sistema de traducción de direcciones para comunicar redes con rangos de direcciones incompatibles. Habitualmente usado para conectar sistemas con direcciones públicas con privadas y viceversa.

**Nearline SAS** : Implementación de una interfaz **SAS** (*Serial Attached SCSI*) que permite el uso de discos *SATA* convencionales como si de discos SAS se tratase.

**Open source** : Traducido como código abierto, hace referencia a aquel software cuyo código y detalles de implementación son distribuidos al gran público.

**Overhead** : Sobrecoste, consumo de recursos o tiempo añadido a la ejecución de una tarea.

**Rapid Virtualization Indexing** : Implementación equivalente de las Extended Page Tables de Intel por AMD.

**Raw access** : Traducido como acceso en crudo, hace referencia al accesso directo a los dispositivos de bloque.

**Repositorio** : Almacén del que se pueden descargar paquetes de software para ser instalados en un sistema.

**Scheduler** : Traducido como planificador, es un sistema que gestiona el uso de la CPU por parte del *guest*.

**Shadow page table** : Implementación del accesos a la tabla de paginación de memoria mediante el *guest* realiza las escrituras en una tabla secundaria (*shadow*) y el *host* realiza los cambios en la tabla de paginas principal, evitando el acceso a ésta por parte del *guest*.

**Snapshot** : Copia del estado de un sistema en un momento en particular, que permite su restauración a *posteriori*.

**Transparent Page Sharing** (*TPS*) : Equivalente de Kernel Samepage Merging para *VMware*.

**Virtio** : Conjunto de controladores de dispositivo diseñados para ser instalados dentro de una máquina virtual que optimizan la comunicación e interacción con el host.

**Virtual channels** : Ver *Device Channels*.

**Virtualizacion** : Creación mediante software de una versión de un recurso físico de una máquina.

**VMWare ESX** : Hipervisor de tipo 1 o *bare metal*, el cual se ejecuta sobre el *hardware* al mismo nivel que lo hace un sistema operativo.

**Xen** : Hipervisor de tipo 1 o *bare metal*, el cual se ejecuta sobre el *hardware* al mismo nivel que lo hace un sistema operativo.