Universidad de Almería

Titulación de Ingeniero en Informática

CLÚSTER DE ALTO RENDIMIENTO EN UN CLOUD: EJEMPLO DE APLICACIÓN EN CRIPTOANÁLISIS DE FUNCIONES HASH

Autor:
Gabriel Molero Escobar

Directores:
Julio Gómez López
Eugenio Eduardo Villar Fernández

Almería, septiembre 2011

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

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

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

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

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

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

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

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

En estas primeras líneas me gustaría destacar la importancia de ciertas personas que han hecho posible la realización de este proyecto.

A los tutores de este proyecto, Julio y Eugenio. Os tengo que agradecer el apoyo constante, el tiempo empleado, que ha sido mucho y los conocimientos compartidos, pero por encima de todo, vuestra calidad personal.

A mis padres, Gabriel e Isabel, por darme esta gran oportunidad sin ponerme ningún impedimento, a mi hermana Isa, que siempre está ahí, y como no, mi abuela Isabel, por su ánimo constante con su sonrisa de oreja a oreja.

A mis compañeros de carrera: Tone, Ferre, Cobo, Aranzazu, Carlos, Noé, José, Fran, Mari Carmen y alguno más que no recuerde…

Y mis amigos de toda la vida: Toni, Fran, Guille, Juanjo, Anatael, Medina y Padilla, sencillamente por ser como son.

A todos ellos, gracias de todo corazón.

ÍNDICE

INTRODUCCIÓN

Es indudable el dinamismo que presentan las tecnologías de la información en la actualidad en lo que a su influencia en nuestras costumbres y los modelos de negocio de las empresas se refiere.

Dos modelos de computación dominan la aplicación de las tecnologías de la información: el modelo de ordenador central, de eficacia largamente demostrada, y el más reciente modelo cliente-servidor. Además de ellos, recientemente y con el beneficio de los avances *hardware* en los recursos informáticos disponibles, surge el novedoso paradigma de computación denominado ***Cloud Computing*** **o computación en la nube,** que aprovecha las bondades de la virtualización de servidores y responde al explosivo aumento del número de dispositivos con conexión a Internet.

Con el usuario como objetivo principal, **la funcionalidad del cloud computing se fundamenta en la disposición de servicios** –*infraestructuras informáticas completas, plataformas para el desarrollo de aplicaciones, o software, de forma general-* **en red**, **posibilitando el rápido aprovisionamiento de recursos** en una potente prestación de servicios IT y de negocio. Se trata de una **tendencia de computación fiable y segura, con una escalabilidad elástica capaz de atender exigentes cambios en la demanda sin que ello suponga un incremento en los costes** de administración y gestión. Su introducción ha permitido definir un nuevo modelo en la consumición de servicios adaptando nuevas formas para el *cobro por recursos consumidos*.

Como usuario final es posible disfrutar de las ventajas proporcionadas por el cloud computing en el uso de servicios y recursos que ofrecen las denominadas **nubes públicas**; en este caso **tanto los recursos como los servicios son accesibles vía proveedores** que desarrollaron y mantienen el entorno cloud computing: soluciones de mensajería y colaboración, entornos de desarrollo de aplicaciones, infraestructuras informáticas bajo demanda, software, etc.

Por otro lado encontramos las denominadas **nubes privadas** en aquellas organizaciones que sopesan la evolución de su infraestructura de servidores hacia el modelo cloud computing, permitiéndoles obtener una **infraestructura virtual**

dinámica que les ayuda a mejorar los servicios prestados, **reduciendo costes** administrativos y **controlando los riesgos** asociados.

Otra forma de administrar los servidores de una infraestructura avanzada es la utilización de clústeres de alto rendimiento. Básicamente, un ***clúster*** es un **conjunto de computadores que se comporta como si de uno sólo se tratara, con el objetivo de ejecutar aplicaciones que precisan gran cantidad de recursos de procesamiento**. Hoy en día desempeñan un papel importante en la solución de problemas científicos, relacionados con la ingeniería y el comercio moderno, evolucionando como apoyo de actividades que van desde aplicaciones de supercómputo, software en misiones críticas, servidores web y comercio electrónico, hasta bases de datos de alto rendimiento.

Existen numerosas **tecnologías, librerías e interfaces** que nos permiten explotar las capacidades de procesamiento de un clúster; siendo las más relevantes analizadas en el presente proyecto, tales como, **MPICH, OpenMPI o LAM/MPI**.

El objetivo principal de este proyecto fin de carrera consiste en la **implementación de un clúster de alto rendimiento (*High Perfomance Cluster*) basado en el paradigma cloud computing**. Para ello, en primer lugar se establecerá una infraestructura de computación en una nube privada, y sobre ella se implementará de forma dinámica un clúster de alto rendimiento GNU/Linux.

Para determinar la tecnología que mejor se adapta a las necesidades del proyecto se llevarán a cabo diferentes benchmarks de rendimiento, cuyos resultados serán analizados. Una vez decidido la mejor tecnología, se utilizará el clúster virtual para la resolución de un problema que precisa de alto rendimiento debido a su elevado coste computacional: el criptoanálisis de funciones hash., cuyos resultados compararemos con otras tecnologías como **CUDA** y **ejecución de hebras CPU** para obtener una conclusión final.

Capítulo 1 - SISTEMAS DE ALTO RENDIMIENTO

Los **sistemas de alto rendimiento** (HPC, *High Performance Computing*) hacen referencia a una rama de la computación aplicada que se centra fundamentalmente en la **solución de problemas que hacen un uso intensivo del cálculo**. Hace unos años, los sistemas de alto rendimiento, a los que se hacía referencia habitualmente como *supercomputación*, estaban compuestos por grandes sistemas, especializados y de elevado coste, que podíamos encontrar principalmente en centros de investigación.

Con el paso del tiempo, el desarrollo del hardware ha traído consecuentemente la mejora de éste y abaratamiento de costes, dando paso a tecnologías que antes eran impensables de emplear, debido a los costes asociados o al bajo rendimiento que ofrecían, como sucede en el caso de los **clústeres**. Esta tecnología surge como una nueva alternativa muy recomendable en el campo de la **computación**, **tanto paralela como distribuida**, donde un conjunto de computadores se unen a través de la red para resolver un mismo problema. La principal **ventaja** de esta tecnología es la mejor relación **coste-rendimiento**, sin embargo, ante una gran demanda de computación, se requiere bastante **espacio y gestión del almacenamiento**, lo que requiere más equipos, y por tanto, un **mayor coste en el mantenimiento**. Ante esto se motiva la aparición de nuevas tecnologías, como es el caso del ***Grid computing* o el *cloud computing***, que ofrecen **soluciones escalables, potentes y flexibles**, logrando **integrar sistemas y dispositivos heterogéneos**, sin necesidad de estar centrados geográficamente para resolver problemas de gran escala, con un mantenimiento menos costoso.

En este proyecto nos centraremos tanto en la tecnología clúster como cloud computing para resolver problemas de elevado coste computacional, ofreciendo así un sistema híbrido como solución de alto rendimiento.

1. Tecnología Cloud Computing

Hoy en día la tecnología *cloud computing* (computación en la nube) esta cobrando gran importancia en el sector de las tecnologías de la información y comunicación. De forma general, cuando hacemos referencia a esta tecnología, estamos hablando de servicios alojados en la red. Habitualmente identificaremos la red como Internet, de ahí el nombre de *computación en la nube*. Así, cloud computing consiste en un paradigma novedoso en la oferta de servicios de computación a través de Internet.

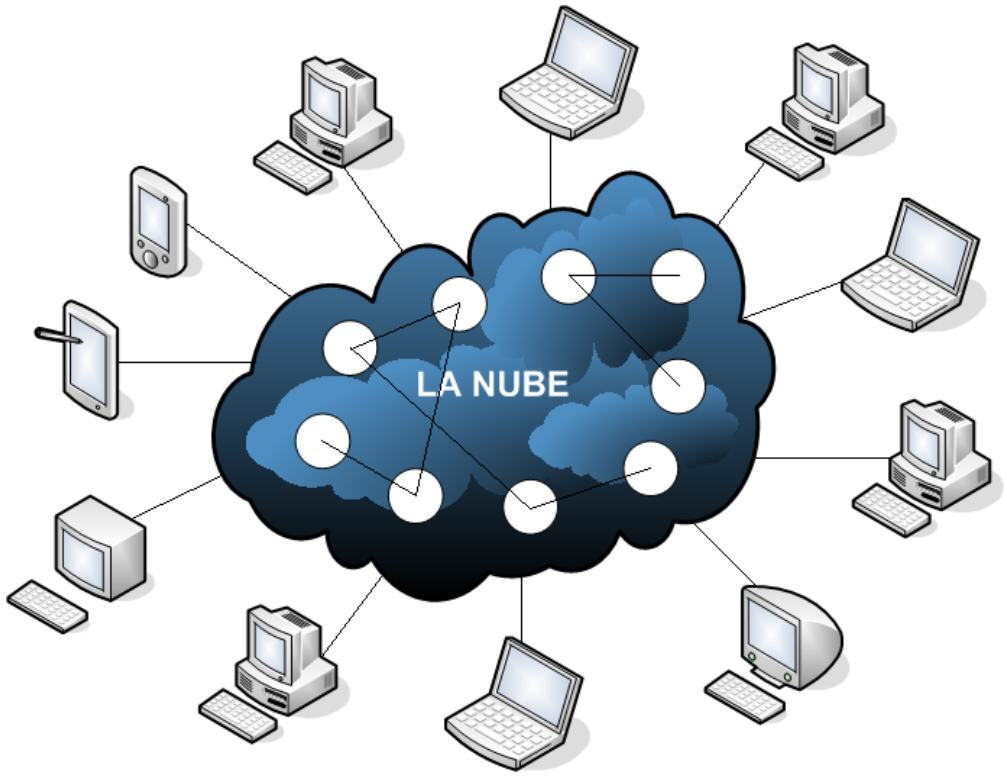

Figura 1.1 - Concepto de computación en la nube

El **concepto fundamental** sobre el que se basa el cloud computing es el de **ofrecer** cualquiera de los **recursos de un sistema como un servicio, negocio o tecnología alojados en la red**, incluyendo software, a los cuales tienen acceso clientes y usuarios sin que éstos deban tener conocimientos avanzados sobre esos recursos, su configuración o procesos de administración y mantenimiento. Estos servicios se ofrecen de forma que es posible pagar en función del consumo realizado, de forma flexible y completamente adaptable a las necesidades de los usuarios.

La mayoría de las ventajas que aporta la aplicación del cloud computing vienen derivadas del uso de la virtualización y, por tanto, de una infraestructura de servicios dinámica, escalable y automatizable. En la nube existe una gran capacidad para migrar los servicios de una localización a otra y es adaptable en función de la demanda registrada para los servicios que ofrece.

1.1. Características

A la hora de estudiar las características de la tecnología cloud computing nos centraremos principalmente en las diferentes topologías que nos podemos encontrar, así como los servicios ofrecidos por esta tecnología.

1.1.1. Tipos de nubes

En función de quién es el que administra y utiliza la nube podemos clasificar los diferentes modelos disponibles en cinco categorías:

- **Nubes públicas o externas**. Estas nubes son completamente administradas por una tercera parte, el proveedor del servicio, por lo que su puesta en marcha, administración y mantenimiento no suponen un esfuerzo para sus usuarios; el acceso y utilización de los servicios es prácticamente inmediato. Como contrapartida, los recursos de computación, almacenamiento,… que ofrece el *data center* sobre el que está instalada la nube del proveedor son compartidos entre sus usuarios, aunque éstos no son conscientes de su utilización.

  Los recursos se aprovisionan de forma dinámica automáticamente a través de Internet, vía aplicaciones y servicios web. La utilización de la nube por parte de los usuarios se registra para realizar la facturación del servicio en función del consumo.

- **Nubes privadas o internas**. Al contrario que las nubes públicas o externas, las nubes privadas o internas son administradas por el propio usuario u organización, que tiene todo el control sobre la nube. Las ventajas iniciales que aportan las nubes públicas desaparecen por tanto, y el usuario debe realizar un esfuerzo inicial de instalación y configuración de la nube en su propio *data center*, además de la administración y mantenimiento posterior.

  La ventaja principal es obvia: el usuario posee en propiedad la nube y puede hacer uso y administrar todos los recursos que ofrece, controlando las aplicaciones que se ejecutan en la nube y quién tiene autorización para trabajar en ella. Este tipo de nubes se utilizan cuando se requiere protección de datos a nivel de los servicios.

  También se pueden considerar como soluciones de virtualización automatizada ya que consisten básicamente en la ejecución de aplicaciones y servicios en máquinas virtuales alojadas dinámicamente en un conjunto de servidores o *hosts* dentro del clúster de la organización. Permiten, por tanto, compartir los recursos hardware del clúster entre las máquinas virtuales, recuperación efectiva ante desastres y gran escalabilidad.

Debido a sus características y el aún presente recelo al uso de nubes públicas, el resto de la documentación de este proyecto esta enfocada al despliegue de este tipo de nubes en organizaciones privadas.

- **Nubes híbridas.** Como su nombre indica se trata de una solución cloud intermedia entre nubes públicas y privadas, combinándolas. En este tipo de nubes, cuyo uso actualmente no es muy extendido –a pesar de que se prevé que sean las más utilizadas en unos años-, el cliente tiene el control solamente sobre las aplicaciones y servicios principales de la nube mientras que la administración del resto es delegada (también de forma controlada). Esto, precisamente, es lo que puede añadir cierta complejidad a su implantación, por lo que su uso actual se encuentra en período de prueba sobre todo para aplicaciones y servicios que precisan sincronización o el uso de sistemas de almacenamiento y bases de datos complejos. Sí son más comunes los casos de utilización de nubes de almacenamiento híbridas o de nubes de *hosting* Web híbridas (en las que existen equipos del clúster que son físicos mientras otros son máquinas virtuales).

- **Nubes combinadas.** Consiste en la combinación de dos o más nubes privadas o públicas, administradas por diferentes usuarios y proveedores. Gracias a esta integración sus usuarios pueden cambiar a servicios proporcionados por nubes públicas con mayor facilidad.

- **Nubes comunitarias.** Este modelo surge de la posibilidad de que varias organizaciones compartan sus recursos de computación y tecnológicos al compartir negocios, servicios y objetivos, y por tanto deciden tomar ventaja de la aplicación del cloud computing conjuntamente. Con menos usuarios que una nube pública y, quizás resultando más costosa su implantación, ofrece mayores niveles de privacidad y seguridad.

En la figura 1.2 podemos ver de forma esquemática en qué consisten los tres primeros tipos de nubes.

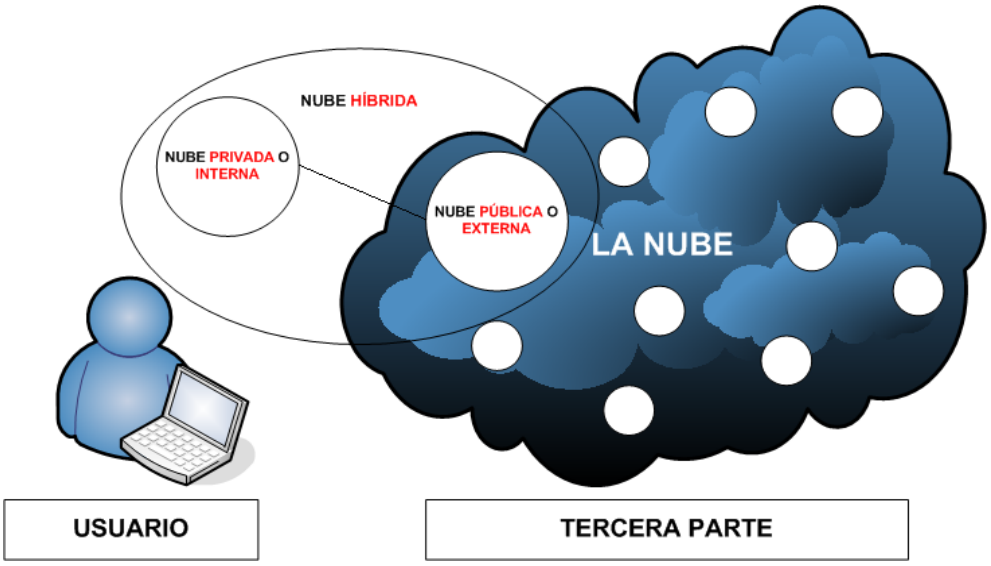

Figura 1.2 – Tipos de nubes. Computación en la nube

1.1.2. Tipos de servicios ofrecidos

Los servicios que podemos ofrecer mediante la aplicación de cloud computing dependen de las necesidades, negocios, objetivos, etc., de la organización en la que se desea implantar. Generalmente son clasificados en tres tipos diferentes:

- **Infraestructura como servicio** (*Infrastructure as a Service - IaaS*). El servicio ofrecido a los usuarios de la nube consiste en la infraestructura necesaria para utilizar el entorno de ejecución y software deseados. Fundamentalmente consta de recursos de capacidad de computación, conexión y almacenamiento como servicio (servidores, sistemas de almacenamiento local y compartido, conexiones de red, routers, etc.), por lo que en ocasiones también es denominado como *HaaS* (*Hardware as a Service*).

- **Plataforma como servicio** (*Platform as a Service - PaaS*). El servicio proporcionado por la nube es un entorno completo sobre el que podemos trabajar con el software, aplicaciones, servicios, etc., que consideramos necesarios en la organización. Es decir, la abstracción de un ambiente de desarrollo junto a una serie de servicios es encapsulada y entregada al cliente/usuario, permitiendo llevar a cabo cada una de las fases de desarrollo y prueba del software tanto de forma genérica como especializada.

- **Software como servicio** (*Software as a Service - SaaS*). Es el caso más conocido de todos y por el que más se debate en la actualidad sobre el cloud computing. El software es entregado al usuario o cliente en forma de aplicaciones finales listas para trabajar con ellas. Como parte del servicio se incluye el soporte y mantenimiento, además del aseguramiento de la disponibilidad del mismo en cualquier momento.

En la tabla 1.1 podemos ver de forma resumida los tres tipos de servicios que se pueden ofrecer con cloud computing, además de **las soluciones más representativas** que permiten desplegar nubes para prestar esos servicios.

Tipo de servicio		Servicios ofrecidos	Ejemplos
	Infraestructura como servicio (IaaS)	*Hosting*: capacidad de cómputo, conexión y almacenamiento	GoGrid, Amazon Web Services EC2 y S3, Enomaly´s ECP, OpenNebula, Eucalyptus, Ubuntu Enterprise Cloud (UEC), OpenStack.
	Plataforma como servicio (PaaS)	Construcción: entorno de desarrollo y prueba de software	Microsoft Windows Azure, Google App Engine, Force.
	Software como servicio - SaaS	Consumo: aplicaciones finales	Microsoft Business Productivity Online Standard, Salesforce, Basecamp, Google Apps, BitNami

Tabla 1.1 - Tipos de servicios ofrecidos con cloud computing

1.2. Ventajas y desventajas

A primera vista una tecnología como la computación en la nube modifica los modelos de negocio existentes aportando numerosas ventajas tanto a los proveedores de los servicios de la nube como a los usuarios de esos servicios. Por ejemplo, a los primeros permitiéndoles ofrecer un mayor número de servicios en la red, de forma estandarizada, rápida y eficiente en su despliegue, y a los segundos facilitándoles un acceso a servicios de forma inmediata, configurados y de forma transparente.

Las ventajas más importantes que reporta el modelo de cloud computing son:

- **Ahorro generalizado costes**: administración, mantenimiento, monitorización, recuperación ante desastres, políticas de seguridad, etc.
- Gran **escalabilidad** debido al aprovisionamiento dinámico de recursos.
- **Fiabilidad** gracias a la presencia de recursos redundantes.
- **Abstracción del hardware.**
- Sistema de **cobro proporcional al consumo.**
- **Agilidad** en la producción y puesta en marcha de servicios. Los recursos pueden ser aprovisionados sin coste y de forma rápida.
- **Integración** garantizada de servicios de red.
- Capacidad de **adaptación** de los modelos de negocio.
- Efectiva **recuperación ante desastres.**
- **Alta disponibilidad y alto rendimiento.**
- **Simplicidad** en la instalación y mantenimiento por parte del usuario, no precisa de un tipo de hardware específico. Además, proporcionar soporte y actualizaciones resulta más sencillo y directo.
- **Reducida inversión inicial** para su puesta en marcha por parte del usuario.
- **Aprovisionamiento dinámico y rápido** de servicios y sus actualizaciones.
- Aumento de la **eficiencia energética**.
- **Independencia de la localización y de los dispositivos** para los usuarios de la nube, que sólo precisan acceso Web.

Como contrapartida, las desventajas más importantes que presenta:

- **Sentimiento de inseguridad o vulnerabilidad** en el usuario de la nube al no tener el almacenamiento físico de los datos sensibles del negocio a su alcance (para solventarlo las nubes se suelen dotar de eficientes y robustos sistemas de seguridad).
- **Dependencia de una conexión a Internet** debido a la localización en red de los servicios (se recomienda disponer de conexiones adicionales a usar en caso de fallo de la principal).
- **Falta de desarrollo en algunas de las soluciones y servicios cloud** en comparación a los mismos en el modelo clásico de computación.
- **Dependencia de los proveedores del servicio** al centralizar datos y aplicaciones.

- La **confiabilidad del servicio** prestado depende del estado tecnológico y financiero del proveedor.
- Es **posible que servicios altamente especializados no estén disponibles** en la red a corto plazo.
- Aparición de **posibles amenazas de seguridad** debido a la arquitectura multinodo de la nube.
- **Disminución de la velocidad si se sobrecarga la red** utilizando protocolos para el encriptado de las comunicaciones, por ejemplo.
- **Degradación del servicio** si la infraestructura de la nube no escala a medida que crece el número de usuarios de la misma.

Como podemos observar la lista de ventajas de cloud computing es larga, aunque no debemos descuidar y prevenir la posible aparición de imprevistos o situaciones problemáticas si tienen lugar ciertas circunstancias o configuraciones.

Si queremos implantar una infraestructura cloud es necesario realizar un estudio previo que permita estimar los riesgos del proceso dependiendo de las características de nuestra organización. No debemos dejar de tener presente que la computación en la nube es una tecnología ampliamente expuesta en la actualidad a debate en términos tan diversos como privacidad, adecuación a regulaciones y términos legales, licenciamiento, seguridad o sostenibilidad.

1.3. Herramientas

A continuación vamos a presentar y conocer las soluciones y herramientas software libre desarrolladas a día de hoy con mayor proyección y extendida utilización para la puesta en marcha y administración de nubes privadas o internas.

1.3.1. Ubuntu Enterprise Cloud

Una de las opciones más utilizadas actualmente a la hora de poner en marcha una solución basada en la tecnología cloud computing es Ubuntu Enterprise Cloud (UEC). UEC está basada en la solución de computación en la nube Eucalyptus, una de las más importantes para la comunidad *open source*.

Eucalyptus (*open.eucalyptus.com*), acrónimo de "*Elastic Utility Computing Architecture Linking Your Programs To Useful Systems*", es una de las herramientas por excelencia para la implementación de sistemas de computación cloud en clústeres. Una de las características más importantes es que **su interfaz dispone de una gran compatibilidad**, además de con múltiples interfaces cliente, **con interfaces de nubes públicas como EC2 y S3 de Amazon o EBS**. Eucalyptus utiliza herramientas GNU/Linux y tecnologías básicas para servicios Web por lo que los procesos de instalación, puesta en marcha y mantenimiento de la nube resultan muy sencillos. Cabe destacar también la calidad de los servicios de consultoría, preparación y soporte ofrecidos por Eucalyptus a

sus usuarios. Además, soporta los más populares hipervisores para implementar y desplegar tanto nubes privadas como híbridas.

Eucalyptus es la base cloud de la distribución Ubuntu Enterprise Cloud (UEC) incluida en la edición servidor de Ubuntu y que **utiliza KVM como hipervisor**. Gracias a UEC podemos desplegar infraestructuras de cloud computing privado o interno de una forma fácil y rápida.

Los distintos componentes que intervienen en su arquitectura son:

- **Cloud Controller** o controlador de la nube. Es el servidor principal y con el que el usuario interactúa para administrar la nube y todos sus componentes. Dispone de una interfaz compatible con la API EC2 de Amazon, la interfaz web de UEC (accesible al realizar la instalación) y otras interfaces como euca2ools en línea de comandos (que debemos utilizar en ocasiones para crear nuevas máquinas virtuales o monitorizarlas) o Hybridfox vía web (que veremos más adelante).

- **Walrus Storage Controller** (WS3) o controlador de almacenamiento Walrus. Compatible con protocolos de Amazon, permite almacenar las máquinas virtuales instanciadas además de otros datos. Normalmente se ejecuta en el mismo equipo que el Cloud Controller.

- **Elastic Block Storage Controller** (EBS) o controlador de almacenamiento en bloque elástico. Se instala en el mismo equipo que el Cloud Controller y se configura durante el proceso de instalación. Su funcionalidad principal es la de crear dispositivos de bloques susceptibles de ser montados por máquinas virtuales en ejecución, por ejemplo, para crear un sistema de ficheros. Entre otras muchas cosas permite también la creación de instantáneas de volúmenes que almacena en WS3.

- **Cluster Controllers** o controladores de clúster. Como se aprecia en la Figura 1.3, operan entre el Cloud Controller y los Node Controllers o controladores de nodo. Se encargan de recibir las peticiones de parte del Cloud Controller para la asignación de las instancias de máquinas virtuales decidiendo, en función de su estado, en qué Node Controller deben hacerlo. Además colabora con el Cloud Controller intercambiando información sobre los recursos que están siendo consumidos y controla el tráfico de las redes virtuales e instancias conectadas a ellas.

- **Node Controllers** o controladores de nodo. Se ejecutan en las máquinas físicas que se encargan de alojar las instancias de las máquinas virtuales. Se encuentra en comunicación con el hipervisor y sistema operativo instalados en el nodo. Debe identificar los recursos utilizados por las máquinas virtuales así como los disponibles (memoria, espacio en disco, tipo de procesador y número de núcleos) en todo momento.

Los Node Controllers reciben órdenes del Cloud Controller para la manipulación de las instancias y responder a consultas de disponibilidad. Para instanciar una imagen primero verifica la autenticidad de la petición, y después descarga la imagen del WS3 (las cachea, por si debe instanciar varias veces una máquina virtual), crea la interfaz virtual de red y la inicia. Si la orden es detener dicha máquina virtual, se realizan las mismas acciones pero en orden inverso.

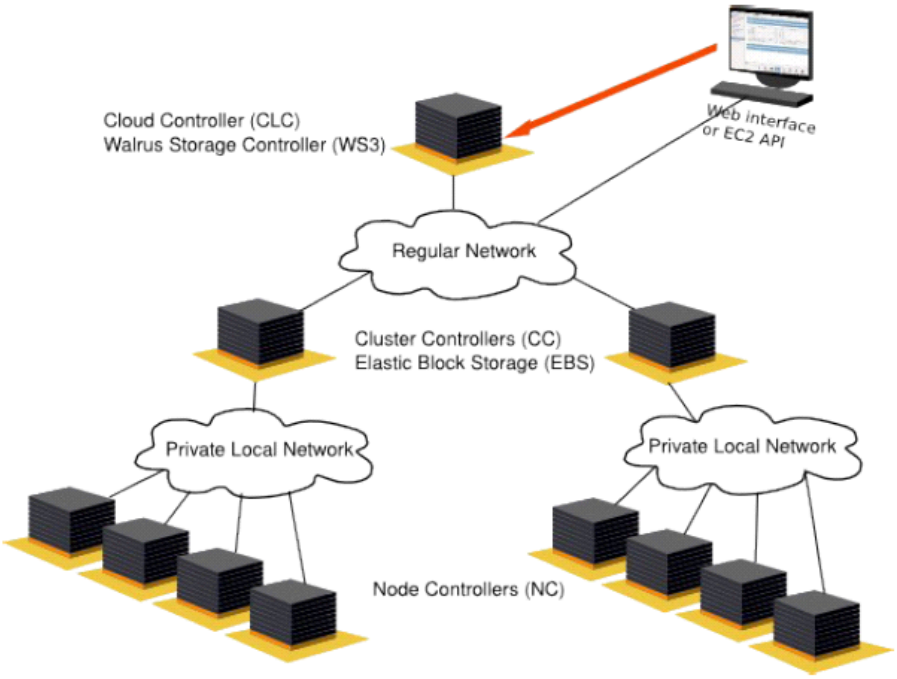

Figura 1.3 - Arquitectura de Ubuntu Enterprise Cloud

En esta gráfica la 'Regular Network', no tiene porque significar Internet, sino una red que sea generalmente accesible por los usuarios del cloud.

1.3.2. *Opennebula*

OpenNebula es una plataforma software que permite el desarrollo de cloud públicos, privados y mixtos de infraestructuras como servicio (IaaS). Se trata de un producto software bajo licenciamiento Apache 2.0, desarrollado en la Universidad Complutense de Madrid y financiado por varios proyectos europeos (Reservoir, 4CaasT, ...) y nacionales(Nuba, HPCcloud, Medianet, ...) Como características técnicas destacadas, OpenNebula implementa, además de una interfaz propia, los interfaces OGF OCCI y un subconjunto de los interfaces EC2-Query de Amazon

Con OpenNebula podemos implementar **cualquier tipo de desarrollo cloud** y administrar su infraestructura virtual en nuestro *data center* o clúster, o **integrarla y combinarla con otras infraestructuras de nubes públicas** o externas. Esto último permite un **gran escalado de los entornos y servicios virtualizados**. El desarrollo de nubes públicas es soportado gracias a la exposición de la funcionalidad de la nube con interfaces que permiten la administración tanto de las máquinas virtuales desplegadas en la infraestructura como del almacenamiento o conexiones de red.

La arquitectura presentada por OpenNebula, que podemos ver en la figura 1.4, está basada en la perspectiva clásica de un grupo de nodos que forma un clúster, actuando uno de ellos como equipo *Front-end* (en el cual se encuentran instalados los drivers de virtualización) para la administración del resto, que ejecutan las instancias o imágenes de las máquinas virtuales creadas y desplegadas por el *Front-end*. Es requisito **indispensable** que exista una **conexión de red entre los nodos del clúster y el *Front-end***, ya que las comunicaciones entre este equipo y el resto de nodos se realizan mediante SSH.

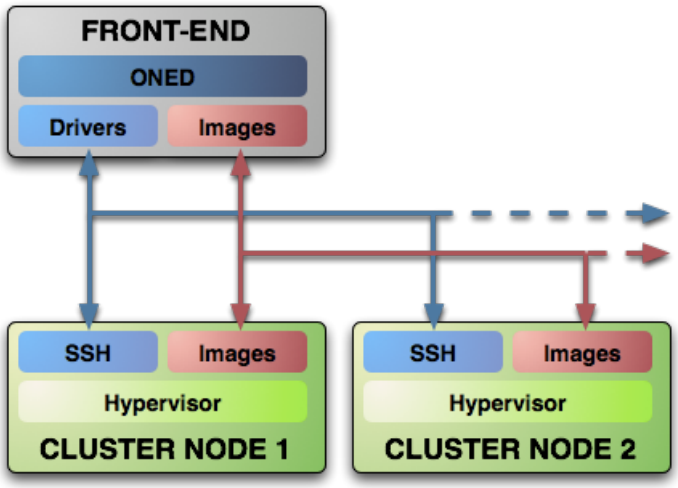

Figura 1.4 - Arquitectura del clúster OpenNebula para nubes privadas

Los componentes básicos de un sistema cloud OpenNebula son los siguientes:

- **Front-end.** Ejecuta el servidor OpenNebula y los servicios del clúster.
- **Nodos.** Son equipos o servidores con un hipervisor instalado, encargados de proporcionar los recursos y el entorno de ejecución de las máquinas virtuales.
- **Repositorio de imágenes.** Se trata de un medio de almacenamiento, local en el equipo Frontend o compartido a través de un sistema avanzado de almacenamiento (por ejemplo, una unidad NAS), para las imágenes o patrones base de las máquinas virtuales del cloud.
- **Servicio OpenNebula.** Se trata del demonio principal que sirve el sistema. Administra todos los subsistemas del clúster (red, almacenamiento e hipervisores de los nodos) y las máquinas virtuales en su ciclo de vida.
- **Drivers.** Permite que el servidor OpenNebula interactúe con los subsistemas del clúster, por ejemplo un sistema de almacenamiento o un hipervisor.
- **Usuario oneadmin**. Es el usuario administrador de la nube privada, encargado de ejecutar cualquier operación relacionada con las máquinas virtuales, redes virtuales, nodos del clúster o usuarios.
- **Usuarios.** Utilizan OpenNebula para crear y administrar sus propias máquinas y redes virtuales.

1.3.3. *Enomaly's* Elastic Computing Platform (ECP)

Permite crear infraestructuras cloud computing para el despliegue de negocios y servicios tanto pequeños como medianos o grandes.

Con ECP es posible diseñar, desplegar y administrar aplicaciones en la nube reduciendo significativamente los costes y cargas de trabajo asociados a aspectos tan relevantes como la administración de la infraestructura mejorando la disponibilidad de los servicios y aplicaciones virtuales.

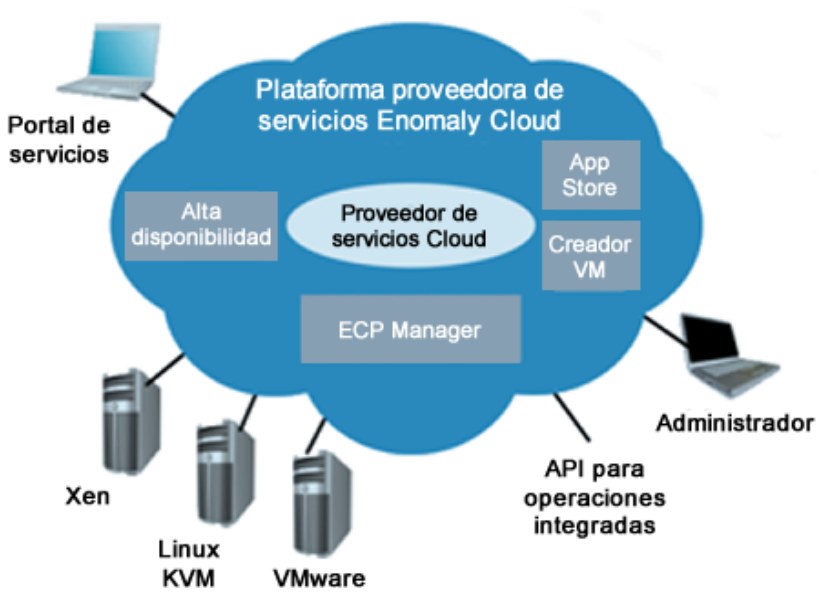

Figura 1.5 - Arquitectura de Enomaly's Elastic Computing Platform

ECP incluye interfaces web que permiten al personal encargado de su administración planear los despliegues de las máquinas virtuales de forma eficiente, rápida y simple, pudiendo automatizar procesos complejos como el escalado de las máquinas virtuales, el balanceo de la carga de trabajo registrada, o el análisis, configuración y optimización de la capacidad de la nube. El mayor objetivo de ECP es, sin duda, **proporcionar una infraestructura para el desarrollo de la computación en nube centrada en el ahorro de costes** al mismo tiempo que proporciona un gran valor adicional.

ECP se ha propuesto desarrollar una plataforma centrada en la aplicación de su infraestructura, no en la gestión, proporcionando todas las herramientas necesarias para implementar complejas aplicaciones cloud, sin necesidad de utilizar complicadas herramientas.

El motor de aprovisionamiento de reglas de ECP proporciona la inteligencia necesaria para determinar en cada momento la localización óptima, tanto física como virtual, de cada componente de la aplicación virtual.

1.3.4. *BitNami*

El objetivo principal de BitNami, soportado por Bitrock, es la simplificación del proceso de creación y puesta en marcha de aplicaciones y portales web de forma virtual en la nube. Los servicios que ofrece BitNami, entre ellos paquetes con numerosos gestores de contenido, son completamente integrados en entornos que contienen todo el software necesario para su correcto funcionamiento.

Las denominadas pilas de BitNami incluyen instaladores, las imágenes de las máquinas virtuales y las plantillas utilizadas en la nube de forma gratuita. Entre los gestores de contenido más utilizados en las nubes BitNami tenemos Drupal, Joomla!, Wordpress, Alfresco, Subversion, MediaWiki o Redmine.

1.3.5. *OpenQRM*

OpenQRM es una plataforma software que permite el desarrollo de cloud públicos, privados y mixtos de infraestructuras como servicio (IaaS). Se trata de un producto software bajo licenciamiento GPL v2, desarrollado por la compañía openQRM Enterprise. Como características técnicas destacadas, openQRM ofrece soporte para numerosas distribuciones de GNU/Linux, permite monitorización mediante Nagios y cuenta con una gestión de almacenamiento integrada en la plataforma. En configuraciones híbridas cuenta con adaptadores para los servicios cloud EC2 de Amazon. Soporta máquinas virtuales XEN, VMWare, Citrix XenServer y KVM.

1.3.6. *CloudStack*

CloudStack es una solución software open source desarrollada por Cloud.com, que permite efectuar el despliegue, configuración y gestión de entornos de computación elástica, tanto para hipervisores Xen Server como KVM. Además de la versión CloudStack Community Edition, soportada por la comunidad, se ofrecen dos versiones comerciales, CloudStack Enterprise Edition, para empresas y CloudStack Service Provider Edition, para proveedores de servicios.

CloudStack puede desplegarse en uno o más servidores de gestión de tal forma que se conectarían a una única base de datos MySQL. Opcionalmente, se podrían distribuir las peticiones Web mediante el empleo de gestores de balance de carga. Además, una copia de seguridad de la base de datos del servidor de gestión podría desplegarse empleando la replicación MySQL en un sitio remoto. Como se puede observar en la figura 1.6, existen varios servidores de gestión que reciben las peticiones que gestionan los distribuidores de carga. Además, la base de datos se encuentra replicada.

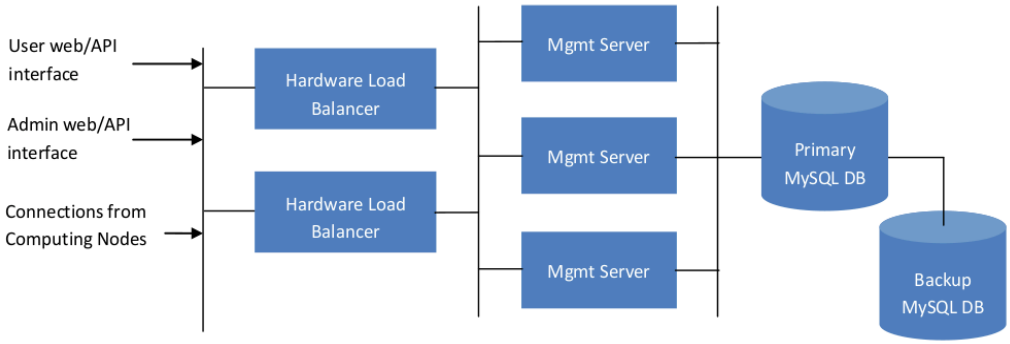

Figura 1.6 - Arquitectura con balanceo de carga y replicación de base de datos

La infraestructura de despliegue se basa en la utilización de nodos de computación, PODS (conjunto de nodos de computación) y zonas de disponibilidad (conjunto de PODS y almacenamiento secundario).

Los nodos de computación constituyen el bloque básico para efectuar el escalamiento de la plataforma CloudStack. Se pueden añadir nodos de computación adicionales en cualquier momento para proporcionar mayor capacidad a las máquinas virtuales huésped. Estos nodos no son visibles para el usuario final y, por tanto, no podrán determinar en qué nodo de computación se ejecutará su máquina virtual.

El administrador del sistema puede determinar:

- Cuántos nodos de computación debe agrupar en un POD.
- Cuántos servidores primarios de almacenamiento debe ubicar en un POD y la capacidad total de los servidores de almacenamiento.
- Cuántos PODS debe ubicar en una zona de disponibilidad.
- Qué cantidad de almacenamiento secundario debe desplegar en una zona de disponibilidad.
- Si se empleará balanceo de carga.
- Cuántos servidores de gestión se desplegarán.
- Si se habilitará la replicación MySQL como medio para evitar desastres.

1.3.7. *OpenStack*

OpenStack es una innovadora solución cloud computing que permite la creación de infraestructuras virtuales de gran fiabilidad y escalabilidad en la red. Aunque es muy reciente su desarrollo, dispone de un gran soporte por parte de grandes compañías como la NASA o Rackspace, Citrix, Dell, AMD, Intel, Cloud.com, Puppet Labs y Zenoss, entre otros. Es de resaltar para este gran proyecto que sea software libre, en contraposición a otros propietarios.

En la actualidad OpenStack engloba dos proyectos relacionados: OpenStack Compute y OpenStack Object Storage. El primero consiste en software dedicado al aprovisionamiento y administración de un gran número de servidores privados virtuales, mientras que el segundo consiste en software para la creación de almacenamiento redundante, escalable y de alta disponibilidad e integridad para terabytes o petabytes de datos utilizando clústeres de servidores.

2. Tecnología Clúster

El origen del término "clúster" y del uso de este tipo de tecnología es desconocido pero se puede considerar que comenzó a emplearse a finales de los años 50 y principios de los años 60.

El término clúster hace referencia a una agrupación de dos o más computadores que se encuentran interconectados y que trabajan de forma conjunta con algún propósito concreto. Desde fuera, en la mayoría de los casos, el clúster es visto como un único equipo que ofrece unos determinados servicios.

Figura 1.7 - Ejemplo de arquitectura de clúster

El cómputo con clústeres surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que así lo requieren.

Los clústeres son normalmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador, típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables.

2.1. Características

La construcción de un clúster es sencilla y económica debido a la gran flexibilidad que éste nos ofrece, ya que los nodos pueden tener la misma configuración hardware y sistema operativo (*clúster homogéneo*), diferente rendimiento pero con arquitecturas y sistemas operativos similares (*clúster semi-homogéneo*), o tener diferente hardware y sistema operativo (*clúster heterogéneo*).

Para que un clúster funcione como tal, no basta solo con conectar entre sí los equipos, sino que es necesario proveer un sistema de administración del clúster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento.

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

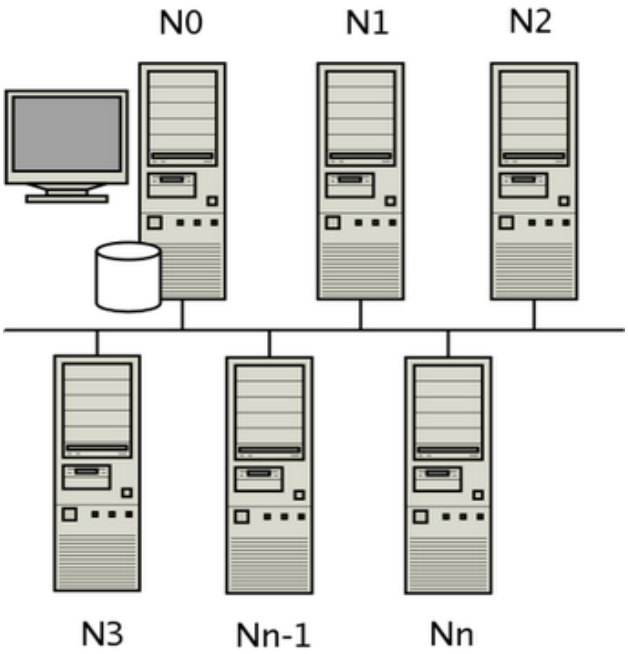

Figura 1.8 - Diagrama de un clúster de N nodos

En general, un clúster necesita de varios componentes software y hardware para poder desempeñar su función:

- **Nodos:** Pueden ser simples ordenadores, sistemas multi-procesador o estaciones de trabajo (*workstations*). El clúster puede estar conformado por nodos dedicados o por nodos no dedicados. En un clúster con nodos dedicados, los nodos no disponen de teclado, ratón ni monitor y su uso está exclusivamente dedicado a realizar tareas relacionadas con el clúster, mientras que en un clúster con nodos no dedicados, los nodos disponen de teclado, ratón y monitor y su uso no está exclusivamente dedicado a realizar tareas relacionadas con el clúster,

sino que el clúster hace uso de los ciclos de reloj que el usuario del computador no está utilizando para realizar sus tareas.

Cabe aclarar que a la hora de diseñar un clúster lo recomendable es que los nodos tengan características similares, es decir, guarden cierta similitud en arquitectura y sistema operativo, ya que si se conforma un clúster con nodos totalmente heterogéneos (existe una diferencia grande entre capacidad de procesadores, memoria, disco duro) será ineficiente debido a que el *middleware* delegará o asignará todos los procesos al nodo de mayor capacidad de cómputo y solo distribuirá cuando éste se encuentre saturado de procesos; por eso es recomendable construir un grupo de nodos lo más parecidos posible.

- **Almacenamiento:** El almacenamiento puede consistir en un sistema NAS (*Network Attached Storage*), una SAN (*Storage Area Network*), o almacenamiento interno en el servidor. El protocolo más comúnmente utilizado es NFS (*Network File System*), que permite compartir el sistema de ficheros entre servidor y los nodos. También existen sistemas de ficheros específicos para clústeres como Lustre (CFS) y PVFS2.

- **Sistema operativo**: El sistema operativo es el software destinado a permitir una gestión eficaz de sus recursos. En este caso debe ser multiproceso y multiusuario.

- **Conexiones de red:** Los nodos de un clúster pueden conectarse mediante una simple red Ethernet con placas comunes (adaptadores de red o NICs), o utilizar tecnologías especiales de alta velocidad como Fast Ethernet, Gigabit Ethernet, Myrinet, InfiniBand, etc.

- **Middleware**: software que generalmente actúa entre el sistema operativo y las aplicaciones con la finalidad de proveer al clúster de:
  - Una interfaz única de acceso al sistema, SSI (*Single System Image*), la cual genera la sensación al usuario de que utiliza un único ordenador muy potente.
  - Herramientas para la optimización y mantenimiento del sistema: migración de procesos, checkpoint-restart (congelar uno o varios procesos, mudarlos de servidor y continuar su funcionamiento en el nuevo host), balanceo de carga, tolerancia a fallos, etc..
  - Escalabilidad: debe poder detectar automáticamente nuevos servidores conectados al clúster para proceder a su utilización.

  Existen diversos proyectos middleware, tales como MOSIX, Cóndor, OpenSSI, etc. El middleware recibe los trabajos entrantes al clúster y los distribuye de manera que el proceso se ejecute más rápido y el sistema no sufra sobrecargas en un servidor. Esto se realiza mediante la aplicación de políticas definidas en el sistema (automáticamente o por un administrador) que le indican dónde y cómo

debe distribuir los procesos, por un sistema de monitorización, el cual controla la carga de cada CPU y la cantidad de procesos que se ejecutan en él.

- **Ambientes de programación paralela:** Los ambientes o entornos de programación paralela permiten implementar algoritmos que hagan uso de recursos compartidos: CPU, memoria, datos y servicios.

2.2. Beneficios de la tecnología Clúster

Las aplicaciones paralelas escalables requieren buen rendimiento, baja latencia, comunicaciones que dispongan de gran ancho de banda, redes escalables y acceso rápido a archivos. Un clúster puede satisfacer estos requerimientos usando los recursos que tiene asociados a él.

La tecnología clúster permite a las organizaciones incrementar su capacidad de procesamiento usando tecnología estándar, tanto en componentes hardware como software que pueden adquirirse a un coste relativamente bajo.

El término clúster tiene diferentes connotaciones para distintos grupos de usuarios. Los tipos de clústeres, establecidos de acuerdo su uso y los servicios que ofrecen, determinan el significado del término para el grupo que lo utiliza.

Los clústeres pueden clasificarse según las características que presentan: clústeres de alto rendimiento (HPCC – *High Performance Computing Clusters*), clústeres de alta disponibilidad (HA – *High Availability*) o clústeres de alta eficiencia (HT – *High Throughput*).

- **Alto rendimiento** (HPCC – *High Performance Computing Clusters*): Son clústeres en los cuales se ejecutan tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez. El llevar a cabo estas tareas puede comprometer los recursos del clúster por largos periodos de tiempo.

  En este tipo de clústeres el objetivo principal es proporcionar altas prestaciones de capacidad de cómputo superior a las que pudiera ofrecer un único servidor. El alto rendimiento puede ser aplicado en distintos niveles también, siendo especialmente interesante en nuestro caso enfocado a servicios, en especial en aquellos que han de soportar una mayor congestión o tráfico.

- **Alta disponibilidad**: Son clústeres diseñados con el objetivo de proveer disponibilidad y confiabilidad. Estos clústeres tratan de brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad es obtenida mediante software que detecta fallos y permite la recuperación efectiva de los mismos, y replicación hardware para evitar tener un único punto de fallo.

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

- **Alta eficiencia**: Son clústeres que presentan un diseño enfocado a la ejecución de la mayor cantidad de tareas en el menor tiempo posible. Existe independencia de datos entre las diferentes tareas individuales. El retardo entre los nodos del clúster no es considerado un gran problema.

Los clústeres pueden ser catalogados de forma diferente como Clústeres de IT Comerciales (Alta disponibilidad, Alta eficiencia) y Clústeres Científicos (Alto rendimiento). A pesar de las discrepancias a nivel de requerimientos de las aplicaciones, muchas de las características de las arquitecturas hardware y software, situadas en un nivel inferior al de las aplicaciones en todos estos clústeres, son las mismas. Más aún, un clúster de un determinado tipo puede también presentar características de los otros.

2.3. Familias y herramientas

En función de las características anteriormente mencionadas y los tipos de clúster derivados de ellas, podemos encontrarnos diferentes familias de clústeres que utilizan una serie de herramientas que a continuación vamos a detallar y que aprovecharemos para la implementación de nuestro clúster en un cloud.

2.3.1. Cluster Beowulf

En 1994 la NASA construyó un clúster bajo Linux con hardware barato, con 16 procesadores 486 conectados mediante una red local Ethernet, con el objetivo de conseguir alto rendimiento. Para ello se utilizó computación paralela por lo que las aplicaciones (escritas en C y Fortran) estaban paralelizadas, es decir, utilizaban librerías de paso de mensaje (PVM y MPI) para que los procesos se ejecuten en múltiples procesadores siguiendo el paradigma cliente/servidor. El proyecto se denominó *Beowulf* y fue un gran éxito, por lo que actualmente cualquier sistema similar se denomina clúster tipo Beowulf. Suelen estar destinados a centros de cálculo, cuyo objetivo principal es la computación de alto rendimiento.

A continuación mostramos las principales características de dichas librerías de paso de mensajes:

- **PVM (Maquina virtual paralela).** Esta librería está destinada al cómputo paralelo en sistemas distribuidos. Permite que una red de computadoras heterogénea comparta sus recursos de cómputo, tales como la memoria RAM o el procesador por ejemplo, consiguiendo disminuir el tiempo de ejecución de un programa al distribuir la carga de trabajo en varios equipos.

  La última versión de esta librería, desarrollada por la Universidad de Tennessee, fue lanzada en marzo de 1993 con mejoras en la tolerancia a fallos y portabilidad.

  PVM permite implementar computación paralela formada por todos los nodos en los que esté activo el demonio "pvmd". Consta de los siguientes componentes:

  - El demonio pvmd.
  - El archivo de configuración ~/pvm.hosts.
  - La consola interactiva de PVM (pvm) y su interfaz gráfica XPVM (xpvm).
  - Las librerías PVM.
  - Las herramientas de desarrollo (aimk, ...).

- **MPI (Interfaz de Paso de Mensajes).** Es un estándar que define la sintaxis y la semántica de las funciones contenidas en una librería de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

  Su principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación de sistemas distribuidos.

  En programación concurrente, el paso de mensajes es una técnica muy empleada para aportar sincronización entre procesos y permitir la exclusión mutua, de una forma similar al uso de semáforos o monitores.

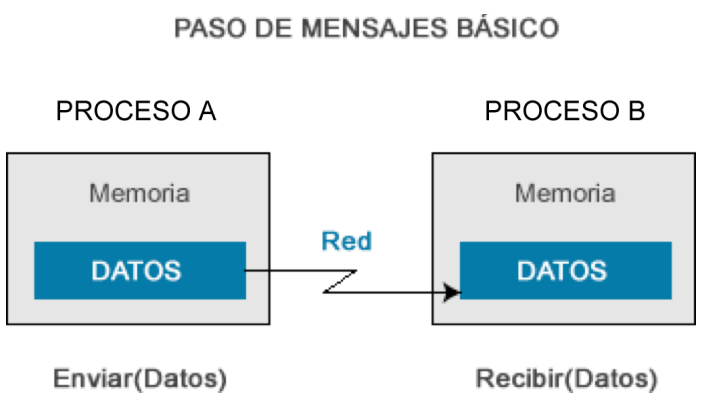

Figura 1.9 - Envío y recepción de un mensaje mediante MPI

Dependiendo de si el proceso emisor espera a que el mensaje sea recibido, podemos hablar de paso de mensajes síncrono o asíncrono. En este último, el proceso que envía no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje y a enviarlo antes de que se haya recibido el anterior. Debido a esto se suelen utilizar buzones, en los que se almacenan los mensajes a espera de que un proceso los reciba. Por tanto empleando el paso de mensajes asíncrono, el proceso emisor sólo se bloquea o para cuando finaliza su ejecución, o si el buzón está lleno. Por su contra, en el paso de mensajes síncrono, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución. Por esto se suele llamar a esta técnica encuentro, o "*rendezvous*".

Las implementaciones en MPI consisten en un conjunto de bibliotecas de rutinas que se pueden utilizar en programas escritos en los lenguajes de programación C, C++, Fortran y Ada. La ventaja de MPI sobre otras bibliotecas de paso de mensajes, es que los programas que utilizan la biblioteca son portables (dado que MPI ha sido implementado para casi toda arquitectura de memoria distribuida), y rápidos, (porque cada implementación de la biblioteca ha sido optimizada para el hardware en la cual se ejecuta).

Existen múltiples implementaciones libres de MPI tales como MPICH, OpenMPI o LAM/MPI. Veamos brevemente a continuación en qué consiste cada una de ellas.

- **MPICH** (MPI Chameleon). Funciona sobre una capa de abstracción del hardware que le permite ser independiente de la arquitectura y fácilmente portable. Esta capa, llamada ADI (*Abstract Device Interface*, Interfaz de Dispositivo Abstracto) se encarga de facilitar el acceso al hardware mientras que el resto de MPICH por encima de la ADI se encarga de la sintaxis y la semántica MPI.

- **OpenMPI** (open-mpi.org). Es el proyecto resultado de la unión de varias implementaciones de MPI, como LAM/MPI, FT-MPI y LA-MPI. Es muy similar a MPICH, salvo que necesita del paquete "*openmpi-bin*", y cambia la sintaxis del fichero de máquinas para nodos con múltiples procesadores.

- **LAM/MPI** (lam-mpi.org). Es una implementación de MPI que utiliza un demonio *lamd* activo en cada nodo. Para poder utilizar esta implementación es necesario el paquete *lam-runtime*, así como que el denominado "fichero de máquinas" contenga los nombres completos de las máquinas que formen el clúster.

  LAM/MPI proporciona la herramienta XMPI (paquete *xmpi*), que permite ejecutar programas paralelos MPI y monitorizarlos. Sin embargo, XMPI no puede activar el clúster, éste tiene que haber sido activado con *lamboot* antes de lanzar XMPI.

- **OpenMP.** Es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución *fork-join*, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (*fork*) con menor peso, para luego "*recolectar*" sus resultados al final y unirlos en un solo resultado (*join*).

  Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influencian el comportamiento en tiempo de ejecución.

  Definido por un grupo de proveedores de hardware y de software mayores, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas para las plataformas que van desde las computadoras de escritorio hasta las supercomputadoras.

  Una aplicación construida con un modelo de programación paralela híbrido se puede ejecutar en un clúster de computadoras utilizando ambos OpenMP y MPI, o más transparentemente a través de las extensiones de OpenMP para los sistemas de memoria distribuida.

2.3.2. *Clusters* HA con LVS

Para implementar en Linux clústeres HA (alta disponibilidad), utilizados principalmente en *granjas de servidores*, tenemos que hacer trabajar conjuntamente varias piezas (al estilo UNIX). Este tipo de clústeres consta de tres componentes:

- **Los servidores reales**. Son las máquinas que realmente prestan el servicio a los clientes, en las que corre Apache. Para asegurar un servicio 24x7 tendremos varios servidores reales.
- **El balanceador de carga virtual.** Reparte las peticiones de los clientes entre los servidores reales. Para que el balanceador de carga no sea un SPOF (*Single Point Of Failure*, Punto único de fallo) utilizaremos un balanceador *master* junto con un balanceador *backup*, conformando un balanceador virtual.
- **Los datos**. Los servidores reales deben servir los mismos datos.

Para que todo funcione necesitamos software que cumpla las siguientes funciones:

1. **Balancear la carga.** El software que se encarga de balancear la carga entre los servidores reales es **LVS** (Linux Virtual Server, linuxvirtualserver.org), instalado en ambos balanceadores (también se encarga de mantener los balanceadores sincronizados entre sí).
2. **Monitorizar los balanceadores (*failover*).** Para monitorizar los balanceadores y activar el balanceador *backup* si el balanceador *master* falla podemos utilizar **HeartBeat** (desarrollado por Linux-HA, linux-ha.org/Heartbeat/) o **KeepAlived** (keepalived.org).
3. **Monitorizar los servidores reales (*health checks*).** Para monitorizar los servidores reales y excluirlos si fallan (y reinsertarlos cuando vuelven a estar en línea) podemos utilizar **Ldirectord** (vergenet.net/linux/ldirectord/) o **KeepAlived** (monitoriza tanto los balanceadores como los servidores reales).
4. Que los **servidores reales sirvan los mismos datos**. Para que los servidores reales sirvan los mismos datos tenemos varias opciones, siendo lo más recomendable algún sistema de almacenamiento compartido como **NFS** (junto con el sistema de *mirror* en red DRBD) o **AoE** (junto con un sistema de ficheros diseñado para clústeres como Coda). En algunos casos puede ser suficiente aplicar *mirroring*, en cuyo caso cada servidor real tendrá sus propios datos que serán sincronizados periódicamente (por ejemplo con *rsync*).
5. Para **monitorizar el sistema** es habitual instalar algún software que monitorice el sistema (carga, rendimiento...) como por ejemplo **MON** (Service Monitoring Daemon, escrito en Perl, monitoriza el sistema y ejecuta acciones cuando un servicio deja de funcionar, mon.wiki.kernel.org) o **HAPM** (High Availability Port Monitor, hapm.sourceforge.net).
6. **Hora NTP**. Para asegurarnos de que todos los nodos tienen la misma hora del sistema (*System Time*) se suelen configurar para que se conecten al mismo servidor de hora NTP (Network Time Protocol), que puede ser local.

Hacer funcionar este tipo de clúster no es tarea sencilla ya que hay muchas posibilidades de configuración y topologías (véase figura I.10).

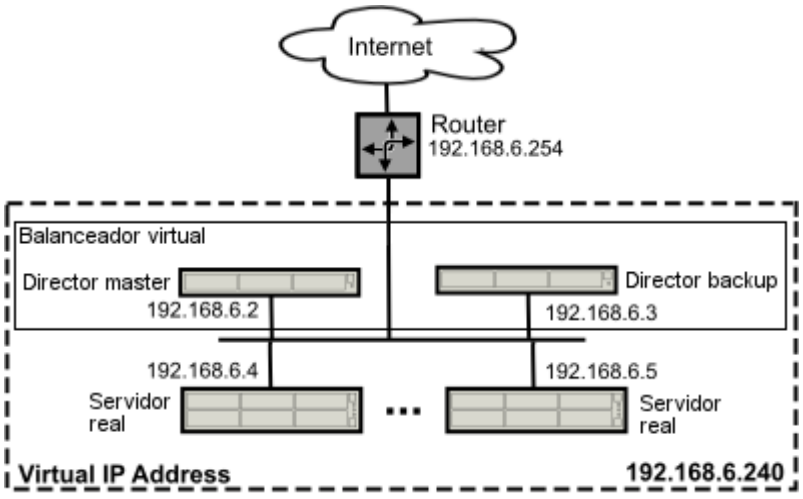

Figura 1.10 Ejemplo de arquitectura clúster HA

Por ejemplo, dos opciones muy utilizadas son:

- **UltraMonkey:** proyecto que simplifica la instalación y configuración de un clúster HA integrando las herramientas LVS, HeartBeat y Ldirectord.
- **LVS + Keepalived:** proyecto que permite instalar un clúster HA integrando estas herramientas encargándose de monitorizar los balanceadores de carga (mediante el protocolo VRRP, Virtual Router Redundancy Protocol) y los servidores reales (permite chequeos TCP_CHECK, HTTP_GET y SSL_GET), reemplazando así a HeartBeat y Ldirectord, utilizadas en la opción anterior.

2.3.3. *Clústeres* SSI

En un SSI (*Single System Image*) todas las computadoras vinculadas dependen de un sistema operativo común, diseñado a tal efecto.

Una imagen única del sistema es una propiedad de un sistema que oculta la naturaleza heterogénea y distribuida de los recursos, y los presenta a los usuarios y a las aplicaciones como un recurso computacional unificado y sencillo. SSI significa que el usuario tiene una visión global de los recursos disponibles independientemente del nodo al que están físicamente asociados esos recursos. Además, SSI puede asegurar que un sistema continuará funcionando después de algún fallo (alta disponibilidad) así como asegurar que el sistema tenga una carga uniforme y multiprocesamiento común (gestión y planificación de recursos).

Las metas más importantes que se persiguen en el diseño de un SSI son, principalmente, la completa transparencia en la gestión de recursos, escalabilidad, y la capacidad de soportar aplicaciones de usuario, centrándose sobre todo en la alta disponibilidad. Por ello son muy utilizados en *granjas de render*.

Algunas herramientas empleadas en este tipo de clúster son:

- **OpenMosix** (openmosix.sourceforge.net). Se trata de un parche para el kernel Linux que permite a varias máquinas actuar como un sistema multiprocesador mayor. OpenMosix balancea la carga de trabajo entre todos los nodos que forman el clúster: migra los procesos, independientemente de en qué nodo se han originado, al nodo con menos carga de trabajo.

  Su mayor ventaja es que las aplicaciones no tienen que estar programadas específicamente para OpenMosix ya que trabaja con aplicaciones normales (no paralelizadas), siendo su funcionamiento transparente al usuario. Pero tiene una limitación: sólo migra procesos que no usen memoria compartida, por lo que no migra procesos multi-hilo.

  OpenMosix está formado por los siguientes componentes:

  - Un parche para el kernel Linux.
  - Herramientas para la línea de comandos y para el entorno gráfico.

- El script de inicio */etc/init.d/openmosix*.

- **Kerrighed** (kerrighed.org). Es un software que permite disponer de un clúster SSI (*Single System Image*) bajo Linux, similar a la herramienta anterior pero con algunas importantes **diferencias**:

  - Permite migrar procesos multi-hilo (bueno).
  - No tiene herramientas de usuario gráficas (malo).
  - Funciona sobre kernel Linux 2.6 y es un proyecto activo (muy bueno).

  Kerrighed está formado por los siguientes **componentes**:

  - Un parche para el kernel Linux.
  - Módulo *kerrighed*.
  - Herramientas de usuario para la línea de comandos.
  - Script de inicio */etc/init.d/kerrighed*.
  - Archivos de configuración: */etc/kerrighed_nodes* y */etc/default/kerrighed*.
  - Librería *libkerrighed*.

  Kerrighed necesita tener exactamente los mismos binarios, librerías, usuarios y grupos en cada nodo, así como un sistema de ficheros compartido accesible por todos los nodos bajo el mismo directorio, para que los procesos que abren archivos puedan migrar.

Capítulo 2 - CLUSTER DE ALTO RENDIMIENTO EN UN CLOUD

1. Antecedentes

Ahora que nos hemos introducido en los conceptos fundamentales relacionados con la tecnología clúster podemos comenzar a ver cuál es la relación existente entre virtualización y clustering, antes de entrar en profundidad con la **tecnología Cloud Computing**, que a grandes rasgos podemos decir que consiste en el establecimiento de clústeres virtuales cuyos nodos son máquinas virtuales ubicadas en el mismo o distintos equipos.

Si tenemos presente el concepto de **virtualización** no es difícil pensar que las técnicas clustering se pueden aplicar en entornos de virtualización con el objetivo de aprovechar todas las ventajas particulares que éstos nos ofrecen.

En el caso de los clústeres, la virtualización nos aporta importantes ventajas, tales como el ahorro en espacio y energía, mayor utilización del hardware de los servidores, menor coste en la administración, etc., pero sobre todo nos beneficiamos en que los nodos de nuestro sistema pasan a ser entes lógicos consiguiendo una rápida recuperación ante posibles desastres, mejora de las políticas y rapidez de procesos como puesta en marcha, recuperación y copias de seguridad, gran escalabilidad, aumento de la seguridad, flexibilidad, etc.

De esta forma es posible implementar clústeres con tan sólo un único servidor físico, o varios si se deseara mayor redundancia y seguridad, y los nodos físicos de un clúster real pasan a ser máquinas virtuales (nodos virtuales) alojadas en el servidor. Éstas máquinas virtuales desempeñan su función como si se trataran de nodos de servidores, ya que es posible la instalación y configuración en ellas de software de alta disponibilidad, alto rendimiento, balanceo de carga, etc. Pero hemos de ser coherentes a la hora de establecer y configurar las máquinas virtuales que formarán el clúster virtual,

ya que si no valoramos los recursos disponibles físicamente en el servidor que las aloja , el rendimiento del sistema no será óptimo.

Prácticamente todas las soluciones de virtualización que conocemos y que ofrecen las funcionalidades necesarias para su aplicación en entornos empresariales con grandes exigencias, como Xen, KVM o VMware como los ejemplos más extendidos hoy en día, permiten crear clústeres de una forma sencilla, ofreciendo mecanismos de configuración, monitorización, recuperación y migración de máquinas virtuales, etc.

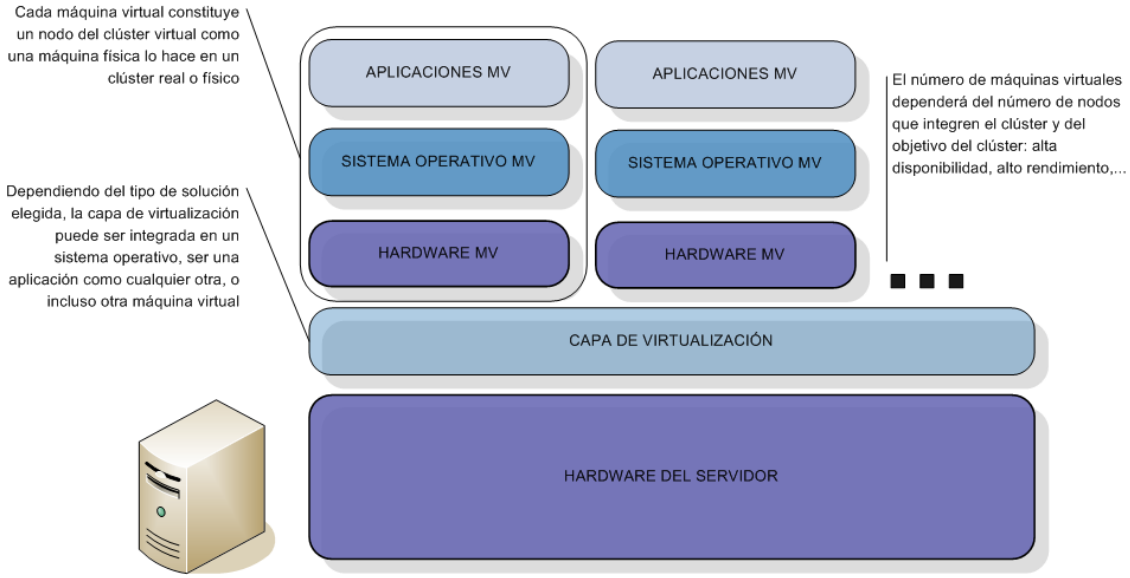

Figura 2.1 - Cluster de máquinas virtuales

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

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

virtuales que aloja caerían también. Por este motivo es totalmente recomendable que un clúster virtual no sólo disponga de un servidor físico como anfitrión.

2. Estado actual

En la investigación con clústeres de alto rendimiento (HPC) existe una gran tradición por la integración de tendencias y tecnologías para la consecución de un mayor poder computacional en la resolución de problemas importantes en cuanto a volumen de computación en temas relacionados con la ciencia e ingeniería. Por ejemplo, las GPUs (*Graphical Process Units*) se han convertido en uno de los componentes más utilizados recientemente en los sistemas HPC modernos.

Existen importantes avances en la integración conjunta de clústeres de alto rendimiento con grid computing y cloud computing, basada en la virtualización, en términos de motivación, puntos fuertes y debilidades, combinando lo mejor de cada una de estas tecnologías (HPC centralizado –local o remotamente-, compartición de recursos, y oferta de servicios bajo demanda mediante cloud computing) en entornos de computación híbridos, obteniendo arquitecturas que soportan funcionalidades no disponibles de otra forma, de forma escalable y tolerante a fallos, dando lugar a clústeres de alto rendimiento implementados en nubes públicas o privadas, siendo éste un modelo de recursos computacionales autoajustable (en tamaño) que permite operar en un entorno de computación híbrido para la consecución de sistemas administrativos de procesos a gran escala.

Algunas de las características fundamentales de este tipo de entornos, aportadas, como no, de la combinación de los diferentes paradigmas que lo conforman, son:

- **Propiedad de los recursos** bien de forma local o de forma remota.
- **Recursos accesibles** de forma pública o privada.
- **Escalado de recursos** casi **estático** (compra de hardware) o **dinámico** (mediante la adquisición pública de los mismos).
- **Políticas de asignación de recursos** de forma exclusiva o compartida entre organizaciones.
- **Portabilidad de aplicaciones**, en base a plataforma o no.

Algunos proyectos relacionados con este paradigma son Globus Toolkit, UNICORE, y la infraestructura NSF Teragrid. **El modelo IaaS de la computación cloud es el más apropiado y flexible para el balanceo de cargas de trabajo en clústeres del alto rendimiento**, por ejemplo como es el caso de Amazon EC2 o Amazon S3.

La motivación en la implantación de estos entornos híbridos reside en la posibilidad de mantener o incluso mejorar los puntos fuertes de cada uno de los paradigmas de computación que los componen.

Debido a las diferentes posibilidades y el solapamiento entre las características y funcionalidades ofrecidas por cada uno de estos paradigmas, no es raro que en la actualidad se proponga la combinación de estos sistemas, por ejemplo algunos investigadores recomiendan la utilización de arquitecturas como el *Grid-of-Cloud* o *Cloud-of-Clouds*, mientras que otros se posicionan al lado de los clouds orientados al HPC. También, la combinación de HPC y Grid es común para el trabajo científico, o la combinación de Grids (GridX y GridWay). Para cargas de trabajo limitadas en capacidad pueden ser usados los Grids-of-Clouds, como los Grids de Nimbus Science Clouds.

De esta forma, HPC, grid computing y cloud computing deben ser vistos como capas en una jerarquía de recursos disponibles y no como enfoques dispares, teniendo en cuenta que es tarea del diseñador de este modelo valorar la combinación de sus puntos fuertes así como de sus debilidades.

Si aprovechamos la flexibilidad que proporciona cloud computing para desplegar clústeres de alto rendimiento, entonces podremos poner en marcha clústeres flexibles, adaptables a nuestras necesidades y a un bajo coste. Este tipo de entorno de computación híbrido es denominado ***Elastic Cluster***, formando una infraestructura dinámica para la administración de servicios, que incluye servicios a nivel de clúster como la administración de cargas de trabajo así como módulos inteligentes que actúan como puente entre los servicios a nivel de clúster y los servicios de administración de la infraestructura dinámica. Este modelo soporta tanto recursos virtualizados como físicos.

Por ejemplo, Amazon solucionó los problemas acaecidos en la ejecución de aplicaciones HPC en sus máquinas virtuales (problemas de *overhead* introducidos por la virtualización y la no existencia de almacenamiento e interconexiones de alto rendimiento) utilizando servidores HPC así como sistemas de almacenamiento en red como EBS. Ahora, sus máquinas virtuales se encuentran preparadas para la ejecución de cargas de trabajo HPC tolerantes a latencias.

Un Elastic Cluster o cloud clúster es autoajustable, en cuanto recursos, adaptándose a la carga de trabajo presente y alcanzando así los requisitos de tiempo para las aplicaciones, ofreciendo una mejor disponibilidad y en definitiva un mejor rendimiento. A diferencia de los clústeres HPC tradicionales, en los que los nodos que trabajan son aprovisionados de forma estática, los clústeres elásticos reservan dinámicamente recursos adicionales pertenecientes a clouds privados o públicos, presentando soporte para la integración con servicios de alto nivel y su asociación con otros clústeres, pudiendo ser los nodos activos de un HPC máquinas físicas o virtuales.

El tamaño máximo de un Elastic Cluster depende de factores de limitación como su capacidad, ejecución y coste. Múltiples Elastic Clusters se pueden utilizar para afrontar requisitos como alta disponibilidad o seguridad, siendo probable que múltiples Elastic Clusters deban ser utilizados conjuntamente para ejecutar trabajos con una carga computacional elevada.

Proyectos relacionados o similares son Oracle Grid Engine (también conocido como SGE), Nimbus Elastic Side y Virtual Grid Execution System.

El escalado tanto para aumentar como para disminuir recursos en la infraestructura es conocido como *cloud bursting*. Otras estrategias que están presentes en este tipo de modelos a parte del *cloud bursting*, son la distribución de la carga entre varios HPC, y la creación de clústeres virtuales personales, proporcionando recursos por trabajo/usuario.

3. Descripción del modelo propio

Tal y como se ha mencionado en apartados anteriores, un clúster es una agrupación de varios equipos que se encuentran interconectados y que trabajan de forma conjunta con algún propósito determinado. De esta forma, las desventajas de un clúster son su alto coste y su poca flexibilidad.

Si aprovechamos la flexibilidad que nos proporciona cloud computing para desplegar clústeres de alto rendimiento, entonces tendremos clústeres flexibles, adaptables a nuestras necesidades y a un bajo coste. Pero no todo son ventajas: la creación de una infraestructura cloud aporta una pérdida de rendimiento que es necesario cuantificar para ver si es conveniente o no su uso, aspecto en el que nos centraremos en el próximo capitulo.

3.1. Arquitectura del modelo

Para nuestro modelo de Elastic Cluster, vamos a necesitar de una infraestructura física sobre la que se establezca una capa de abstracción donde podamos desplegar las máquinas virtuales que conforman nuestro clúster de alto rendimiento y que se ejecutan en el cloud. Estas máquinas virtuales se crean bajo una plataforma de virtualización, normalmente XEN o KVM; en este caso nos decantaremos por el segundo por la sencillez en las tareas de configuración y administración.

En la Figura 2.2 podemos ver el esquema que se va a implementar para el clúster de alto rendimiento en un cloud.

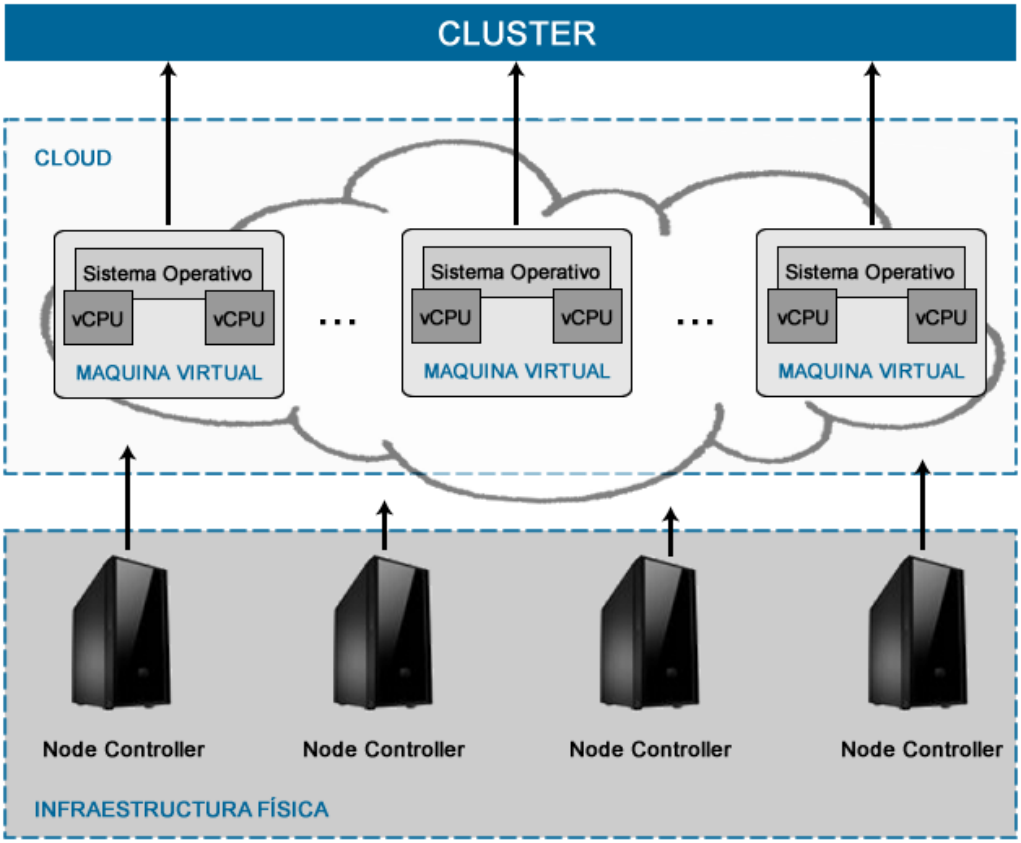

Figura 2.2 - Esquema de un Cluster utilizando infraestructura Cloud

Es evidente que el número de máquinas virtuales que nosotros podamos desplegar en nuestro cloud depende de los recursos físicos que dispongamos y sus características, es decir, del número de nodos físicos que dispongamos y las características de estos.

3.2. Implementación del modelo

Para la implementación de nuestro clúster de alto rendimiento en un cloud debemos tener en cuenta las dos infraestructuras necesarias que ya hemos citado, por un lado la infraestructura física, formada por los diferentes servidores donde se puede desplegar nuestra nube privada, y la infraestructura virtual (la propia nube), donde se ejecutan las máquinas virtuales que forman el clúster.

Para la configuración de los sistemas se ha utilizado la distribución Ubuntu Maverick Meerkat 10.10 en los servidores, Ubuntu Lucid Lync 10.04 en las instancias de las máquinas virtuales en el cloud y Mpich2 1.2.1 para el clúster.

3.2.1. Implementación de la nube.

A la hora de implementar nuestro cloud, hemos decidido utilizar la distribución Ubuntu Enterprise Cloud, que como ya se analizó en el primer capítulo, está basada en la solución Eucalyptus, una de las herramientas por excelencia para la implementación clústeres en sistemas cloud.

Así dispondremos de una arquitectura software *open source* basada en Linux con la que implementar clouds privados e híbridos y que proporciona la infraestructura como un servicio (IaaS), de tal forma que los usuarios de nuestro entorno puedan aprovisionar sus propios recursos (hardware, almacenamiento y red) en función de sus necesidades.

UEC está diseñada para ser fácil de instalar, de la forma menos intrusiva posible, proporcionando una capa de red virtual de tal forma que se aísla el tráfico de red de diferentes usuarios y permite que uno o más clústeres parezcan pertenecer a la misma LAN. Además, tiene la capacidad de interactuar con Amazon EC2 y los servicios S3 del cloud público ofreciendo la posibilidad de crear un cloud híbrido.

De esta forma, de acuerdo con esta herramienta, nuestra plataforma ofrecerá las siguientes funcionalidades:

- API compatible con Amazon Web Service (AWS).
- Arquitectura agnóstica en relación al hipervisor (soporta Xen y KVM).
- Gestor Walrus de almacenamiento compatible con S3 (gestiona el almacenamiento de los datos de usuario, así como las imágenes de los sistemas de archivos de las máquinas virtuales, los *kernels y los ramdisks*).
- Soporte EBS (Elastic Block Store) sobre AoE e iSCSI.
- Múltiples modos de funcionamiento de red para adaptarse a diferentes arquitecturas.
- Interfaz Web y herramientas CLI para la administración y la configuración del cloud.
- Arquitectura escalable: las peticiones del cloud se sirven de forma asíncrona.

Los usuarios que interactúen con el cloud dispondrán de diferentes funciones para implementar, gestionar y mantener sus propias colecciones de recursos virtuales. Entre ellas se incluye:

- **Gestión de claves SSH**. Se emplean claves privadas y públicas para validar la identidad del usuario cuando se conecta a las máquinas virtuales por medio de SSH. Los usuarios pueden añadir, obtener información y borrar pares de claves.
- **Gestión de imágenes.** Antes de poder ejecutar una máquina virtual, éstas deben ser preparadas para poder emplearse en el cloud. Los usuarios pueden

empaquetar, subir, registrar, obtener información, descargar, desempaquetar y cancelar el registro de las imágenes de las máquinas virtuales.

- **Gestión de máquinas virtuales.** Los usuarios pueden ejecutar sus propias máquinas virtuales en el cloud. Los usuarios pueden ejecutar, obtener información, apagar, y reiniciar una gran variedad de máquinas virtuales Linux definidas empleando las funciones de gestión de imágenes.
- **Gestión de direcciones IP.** Dependiendo del modo de gestión de red establecido, los usuarios podrán tener acceso al sistema de IPs elásticas, lo que les permitirá asignar, asociar, obtener información y liberar direcciones IP.
- **Gestión de grupos.** "*Security groups*" son conjuntos de reglas de firewall aplicadas a las instancias de las máquinas virtuales asociadas a un grupo, permitiendo crear, obtener información, borrar, autorizar y revocar grupos de seguridad.
- **Gestión de volúmenes y *snapshots*.** Eucalyptus permite a los usuarios crear bloques de volúmenes de forma dinámica que podrán ser empleados con las máquinas virtuales. Los usuarios pueden crear, asignar, liberar, obtener información y borrar volúmenes. Los usuarios pueden crear y borrar *snapshots* y crear nuevos volúmenes.

a) Arquitectura interna

La arquitectura de nuestra nube será modular y sus componentes internos emplearán servicios Web, lo que hace factible reemplazarlos con facilidad e incluso expandirlos.

Como ya se mencionó en el capitulo 1, el hipervisor utilizado por Ubuntu Enterprise Cloud es KVM, y sus componentes son:

- **Cloud Controller** o controlador de la nube. Es el servidor principal y con el que el usuario de la nube interactúa para administrar la nube y todos sus componentes. Dispone de una interfaz compatible con la API EC2 de Amazon, la interfaz web de UEC (accesible al realizar la instalación) y otras interfaces como euca2ools en línea de comandos (que debemos utilizar en ocasiones para crear nuevas máquinas virtuales o monitorizarlas) o Hybridfox vía navegador web.
- **Walrus Storage Controller (WS3)** o controlador de almacenamiento Walrus. Compatible con protocolos de Amazon, permite almacenar las máquinas virtuales instanciadas además de otros datos. En nuestro cloud se ejecuta en el mismo equipo que el Cloud Controller.
- **Elastic Block Storage Controller (EBS)** o controlador de almacenamiento en bloque elástico. Se instala en el mismo equipo que el Cloud Controller y se configura durante el proceso de instalación. Su funcionalidad principal es la de crear dispositivos de bloques susceptibles de ser montados por máquinas virtuales en ejecución, por ejemplo, para crear un sistema de ficheros. Entre otras

muchas cosas permite también la creación de instantáneas de volúmenes que almacena en WS3.

- **Cluster Controllers** o controladores de clúster. Se encarga de recibir las peticiones de parte del Cloud Controller para la asignación de las instancias de máquinas virtuales decidiendo, en función de su estado, en qué Node Controller deben hacerlo. Además colabora con el Cloud Controller intercambiando información sobre los recursos que están siendo consumidos y controla el tráfico de las redes virtuales e instancias conectadas a ellas. Se encuentra instalado en el mismo equipo que los anteriores.
- **Node Controllers** o controladores de nodo. Se ejecutan en las máquinas físicas que se encargan de alojar las instancias de las máquinas virtuales. Se encuentra en comunicación con el hipervisor y sistema operativo instalados en el nodo. Debe identificar los recursos ocupados por las máquinas virtuales así como los disponibles (memoria, espacio en disco, tipo de procesador y número de núcleos) en todo momento.

Los Node Controllers reciben órdenes del Cloud Controller para la manipulación de las instancias y responder a consultas de disponibilidad. Para instanciar una imagen primero verifica la autenticidad de la petición, y después descarga la imagen del WS3 (las cachea, por si debe instanciar varias veces una máquina virtual), crea la interfaz virtual de red y la inicia. Al detenerla opera en orden inverso.

En la Figura 2.3, puede observar el esquema de la infraestructura física de la nube, donde la Regular Network, no tiene porque significar Internet, sino una red que sea generalmente accesible por los usuarios del cloud.

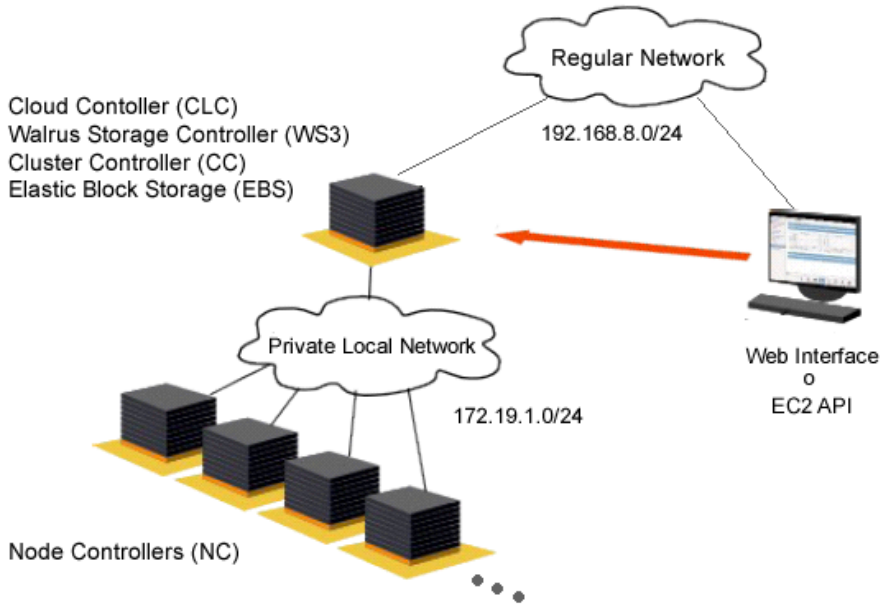

Figura 2.3 - Arquitectura física de nuestra nube

b) Instalación

En este apartado se introducirán las características de la instalación de Ubuntu Enterprise Cloud para desplegar una nube donde se instancian las máquinas virtuales que conformen el clúster de alto rendimiento. Para más detalles, puede consultar el Anexo I: Puesta en marcha del Cloud.

La tabla 2-1, refleja los requisitos mínimos necesarios en los equipos físicos para poder desplegar nuestra nube.

	Controlador de la Nube		**Nodos del Clúster**	
Hardware	**Mínimo**	**Recomendado**	**Mínimo**	**Recomendado**
CPU	1Ghz	2x2 Ghz	Extensiones VT	VT, 64-bit, Multicore
Memoria	2 Gb	4 Gb	1Gb	4 Gb
Disco duro	40 GB	200 GB	40GB	100 GB
Red	100 Mbps	1000 Mbps	100 Mbps	1000 Mbps

Tabla 2.1 - Requisitos para la instalación de Ubuntu Entrerprise Cloud

De acuerdo con esta información, el equipamiento para el desarrollo de este proyecto es el que se muestra en la tabla 2-2.

Modelo	**CPU**	**Memoria**	**Disco Duro**	**Unidades**
Dell R210	4 x INTEL XEON X340 - 2,40 GHz	4GB	250 GB HDD	5
Dell R200	4 x INTEL XEON X310 - 3,13 GHz	4GB	250 GB HDD	1
HP DL120 G5	2 x INTEL XEON E3110 - 3,00 GHz	4GB	250 GB HDD	3

Tabla 2.2 - Especificaciones del equipamiento disponible

De los cuales seleccionamos un HP DL120 G5 que actua como Cloud Controller, el resto desempeñan la función de Node Controllers, por lo que como máximo, tendremos disponibles 28 CPUs para desplegar las máquinas virtuales.

El primer paso para desplegar UEC es la instalación del Cloud Controller, en el cual instalamos los diferentes componentes. En nuestro caso, tal y como muestra la Fig. 2-3, instalamos el Cloud Controller, Walrus Controller, Cluster Controler y Storage Controller.

Una vez instalado el controlador de la nube, se procede a instalar los Node Controllers con la misma distribución. Éstos detectan que en la red ya existe un Cloud Controller y se registran los complementos necesarios de los nodos. Si se realiza sobre una distribución Ubuntu existente debe realizarse manualmente.

Tras realizar la instalación del controlador y de los distintos nodos que componen el clúster UEC, se procede a realizar las primeras tareas de configuración de las comunicaciones del controlador y el clúster antes de poder comenzar a trabajar con la nube. Toda esta información puede ser consultada en detalle en el Anexo I: Puesta en marcha del Cloud.

3.2.2. Implementación del clúster.

Una vez puesta en marcha la nube con UEC, el siguiente paso es implementar el clúster de alto rendimiento utilizando instancias de imágenes de máquinas virtuales del cloud.

Una EMI (*Eucalyptus Machine Image*) es una combinación de una imagen de disco, un kernel, un *ramdisk* y un archivo XML que contiene los metadatos acerca de la imagen. Las imágenes EMI están almacenadas en el Walrus Controller y se emplean como plantillas para crear las instancias en el cloud.

Cada una de las imágenes tiene asignado un identificador único que puede emplearse para ejecutar las instancias, así como un tipo de máquina virtual que impone los límites para la utilización de los recursos. Hay que señalar que estos límites son configurables por parte del administrador de UEC y permiten especificar el número de CPUs/Cores a emplear (Eucalyptus no efectúa distinción entre CPUs y Cores), la capacidad del disco, así como la cantidad de memoria RAM. Los tipos de instancias predefinidos son:

Tipo	CPUs	Memoria (MB)	Disco (GB)
m1.small	1	128	2
c1.medium	1	256	5
m1.large	2	512	10
m1.xlarge	2	1024	20
c1.xlarge	4	2048	20

Tabla 2.3 - Tipos de instancias predefinidos en UEC

Así el ciclo de vida de una instancia viene definido por la Figura 2.4.

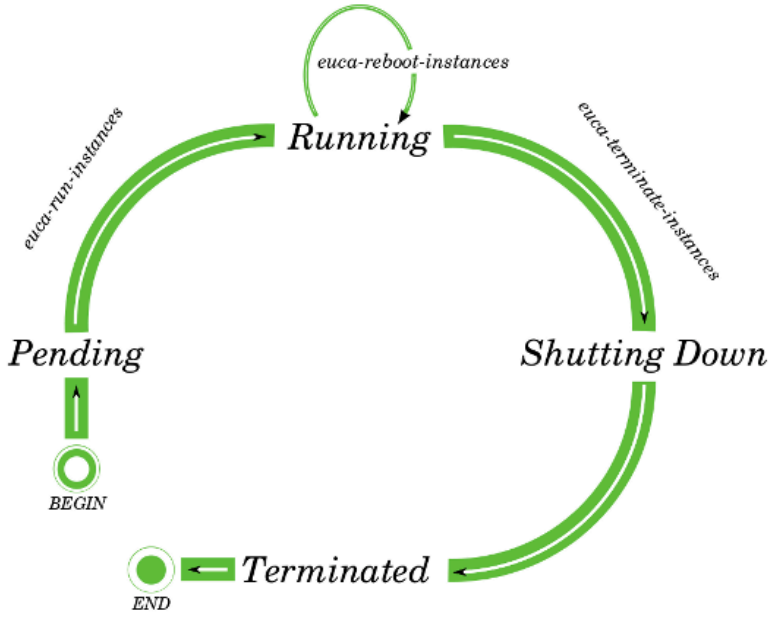

Figura 2.4 - Ciclo de vida de una instancia de UEC

Durante el estado *Pending* se efectúa el despliegue de la máquina virtual en el nodo (se prepara la transferencia desde el Walrus Controller, se prepara la interfaz de red virtual, etc.). El estado *Running* indica que la máquina se encuentra en ejecución. El estado *Shutting Down* indica que la máquina virtual se está apagando (a petición del usuario). Finalmente, el estado *Terminated* indica que la instancia ha sido eliminada del nodo en el que se encontraba alojada. Mientras que una instancia está en ejecución, puede reiniciarse.

3.2.3. Instalación y configuración del clúster

Una vez instanciadas las diferentes imágenes de las máquinas virtuales en el cloud, el siguiente paso consiste en la puesta en marcha del clúster, para el cual, al igual que en el paso anterior, puede encontrar amplia información técnica en el Anexo II: Puesta en marcha del Clúster.

En este caso se ha implementado un clúster tipo Beowulf para conseguir alto rendimiento, basándonos en la computación paralela por lo que los programas, escritos en lenguaje C, se encuentran paralelizados, es decir, utilizan librerías de paso de mensaje (MPI) para que los procesos se ejecuten en múltiples procesadores siguiendo el paradigma cliente/servidor.

Se ha utilizado la implementación MPICH de MPI, concretamente la versión MPICH2 1.2.1, optimizada para entornos homogéneos, lo que proporciona un mayor rendimiento en el paso de mensajes entre nodos.

Esta versión funciona sobre una capa de abstracción del hardware que le permite ser independiente de la arquitectura y fácilmente portable. Esta capa, llamada ADI (*Abstract Device Interface*) se encarga de facilitar el acceso al hardware mientras que el resto de la implementación por encima de la ADI se encarga de la sintaxis y la semántica MPI.

Una vez instalado MPICH2 en cada uno de los nodos del clúster (instancias de máquinas virtuales), debemos configurarlos para indicarles cuáles son los nodos disponibles para ejecutar programas MPICH a través del fichero de máquinas del servidor.

Del mismo modo, para que el nodo servidor pueda ejecutar comandos remotos en los esclavos, MPICH utiliza por defecto SSH (Secure Shell). Por lo tanto instalamos tanto en el servidor como en los esclavos SSH y los configuramos para que acepten conexiones desde el nodo servidor sin solicitar la introducción de contraseña, utilizando el mismo usuario en cada una de las instancias.

Otro aspecto interesante y que hay que tener en cuenta durante el estudio de rendimiento a realizar es la utilización de directorios compartidos utilizando NFS.

Capítulo 3 - RENDIMIENTO DEL ELASTIC CLOUD

Como se ha mencionado anteriormente, es posible aprovechar la flexibilidad que proporciona la tecnología cloud computing para desplegar clústeres virtuales de alto rendimiento, permitiendo entonces la puesta en marcha de clústeres flexibles, adaptables a nuestras necesidades y a un bajo coste, denominados *Elastic Cluster*s. Estos entornos de computación híbridos permiten obtener arquitecturas computacionales que soportan funcionalidades no disponibles de otra manera, de forma autoajustable (en tamaño y recursos), escalable y tolerante a fallos.

Como se ha demostrado no todo son ventajas; la creación de una infraestructura cloud conlleva de forma inherente una pérdida de rendimiento que es necesario cuantificar para ver si es conveniente o no su uso. Éste será el hilo argumental de este capítulo, en el cual se analizará el rendimiento obtenido para nuestro clúster de alto rendimiento implantado en un cloud utilizando *Ubuntu Enterprise Cloud*.

1. Entorno de prueba

Para poder analizar las ventajas que ofrece el despliegue de clústeres en una infraestructura cloud es necesario realizar el análisis del rendimiento de un clúster físico frente a un clúster cloud utilizando el mismo equipamiento. Para analizar el rendimiento de ambos sistemas se utilizan varios *benchmarks* que vemos a continuación.

Para realizar estas pruebas de rendimiento se ha utilizado el equipamiento descrito en el capítulo anterior (véase la tabla 3.1).

Modelo	CPU	Memoria	Disco Duro	Unidades
Dell R210	4 x INTEL XEON X340 - 2,40 GHz	4GB	250 GB HDD	5
Dell R200	4 x INTEL XEON X310 - 3,13 GHz	4GB	250 GB HDD	1
HP DL120 G5	2 x INTEL XEON E3110 - 3,00 GHz	4GB	250 GB HDD	3

Tabla 3.1 - Especificaciones del equipamiento disponible

Del conjunto anterior de nueve servidores totales se han empleado los cinco servidores DELL 210 como nodos del clúster y de la nube, y un servidor HPDL120 G5 como controlador de la nube.

Así los dos escenarios creados para las pruebas de rendimiento han sido:

- **Clúster de Hosts Físicos** (PHC, *Physical Host Cluster*). Se ha creado el clúster utilizando cinco servidores Dell R210 con el sistema operativo Ubuntu Server 10.10.
- **Clúster de Máquinas Virtuales** (VMC, *Virtual Machine Cluster*). Primero se ha creado la infraestructura cloud utilizando el servidor HP DL120 como *Cloud Controller* y se han utilizado cinco servidores Dell R210 como *Node Controller*. Una vez creada la infraestructura cloud se han desplegado diez instancias de dos procesadores cada una para la creación del clúster. Aunque es posible crear una configuración diferente en lo que a instancias se refiere (por ejemplo, cinco instancias de cuatro procesadores cada una), los resultados obtenidos son prácticamente idénticos.

En ambos casos, el clúster dispone de un total de veinte procesadores y la infraestructura física se compone de cinco servidores Dell R210.

Es conveniente recordar que para la configuración de los sistemas se ha utilizado la distribución Ubuntu Maverick Meerkat 10.10 en los servidores físicos, Ubuntu Lucid Lync 10.04 en las instancias de las máquinas virtuales en el cloud y Mpich2 1.2.1 para el clúster.

2. Ejecución

El objetivo de las pruebas de rendimiento es determinar cómo afecta al rendimiento del sistema la creación del clúster en la infraestructura Cloud. Para analizar el rendimiento del Clúster de Hosts Físicos (PHC) y del Clúster de Máquinas Virtuales (VMC) han sido utilizados los siguientes *benchmarks*::

- **Benchmark Básico**. El objetivo es analizar el rendimiento de los elementos básicos del sistema: sistema de ficheros, memoria RAM y potencia de procesamiento. Para medir el rendimiento del sistema de ficheros y RAM se ha creado un *benchmark* propio, mientras que para medir la potencia del sistema se ha utilizado *MPI Benchmark.*
- **Benchmark NPB**. Los *NAS Parallel Benchmarks* han sido desarrollados por la NASA para el estudio del rendimiento de supercomputadores paralelos. NPB consiste en un conjunto de *benchmarks*, cada uno de los cuales está centrado en la resolución de problemas de ingeniería aeroespacial que necesitan una gran potencia de procesamiento.

Los *benchmarks NPB*, formados por un conjunto de *kernels* computacionales y aplicaciones, representan las partes con mayor carga computacional de simulaciones CFD (*Computational Fluid Dynamics*) utilizadas por la NASA para la evaluación del rendimiento de sus computadores actuales y futuros en el ámbito de sus necesidades de supercomputación.

Así, podemos distinguir los siguientes *kernels*:

- MG - *MultiGrid Method*
- IS - *Integer Sort*
- FT - *Fast Fourier Transformbased Spectral Method*
- CG - *Conjugate Gradient*

Asimismo son aplicaciones:

- BT - *Block Tridiagonal*
- LU - *Block Lower and Upper Triangular Systems*
- SP - *Scalar Pentadiagonal Systems*

El conjunto de *benchmarks NPB* constituye un estándar de facto en la evaluación del rendimiento de sistemas paralelos para computación científica. Estos *benchmarks* fueron originalmente implementados usando los lenguajes C y Fortran, usando distintas formas de paralelismo, como OpenMP para paralelizar a nivel de *thread*, o MPI para paralelizar la ejecución entre varias máquinas.

2.1. Benchmark Básico

Para medir el rendimiento del sistema de ficheros y memoria RAM se ha comparado el rendimiento de un servidor físico respecto a una instancia de máquina virtual en el cloud utilizando un *benchmark* propio escrito en lenguaje C; para ello hemos usado un servidor DELL R210 de cuatro núcleos y una instancia de máquina virtual en el cloud con el mismo número de procesadores, desplegada en dicho servidor físico para compartir las mismas características.

La comparativa del rendimiento del sistema de ficheros se ha obtenido a partir de la escritura y lectura de un fichero de disco secuencial de 100 MB para las dos opciones, mientras que el rendimiento en memoria RAM se ha obtenido al medir los tiempos en la creación de un vector de caracteres de 1MB, el cual es escrito y leído 1000 veces.

Tal y como se muestra en la tabla 3.2, la virtualización del sistema provoca un pérdida del 23,3% de rendimiento en el sistema de ficheros y 1,11% de rendimiento en el uso de la memoria RAM.

Modelo	PHC	VMC	%
Sistema de ficheros	52,87 Mbyte/s	43,22 Mbyte/s	23,3
Memoria RAM	332,63 Mbytes/s	328,86 Mbyte/s	1,11

Tabla 3.2 - PHC vs VMC: Rendimiento del sistema de ficheros y memoria RAM

Para medir la potencia de procesamiento del sistema, en ambos entornos, se ha ejecutado el *benchmark MPI* variando el número de procesadores del clúster y se ha calculado el grado de eficiencia del sistema en cada caso.

La eficiencia es un valor típicamente entre cero y uno, permitiendo determinar el grado de tiempo de procesamiento útil que realizan los procesadores para la resolución del problema, en comparación con el esfuerzo realizado en tareas de comunicación y sincronización. La eficiencia ideal de un sistema es 1 y representa que el 100% del tiempo de procesamiento del sistema se utiliza para resolver el problema.

En la figura 3.1 puede ver el grado de eficiencia comparado para el PHC y el VHC. Lógicamente, la eficiencia para p=1 en el caso del VMC se ha calculado sobre la eficiencia para p=1 sobre el PHC.

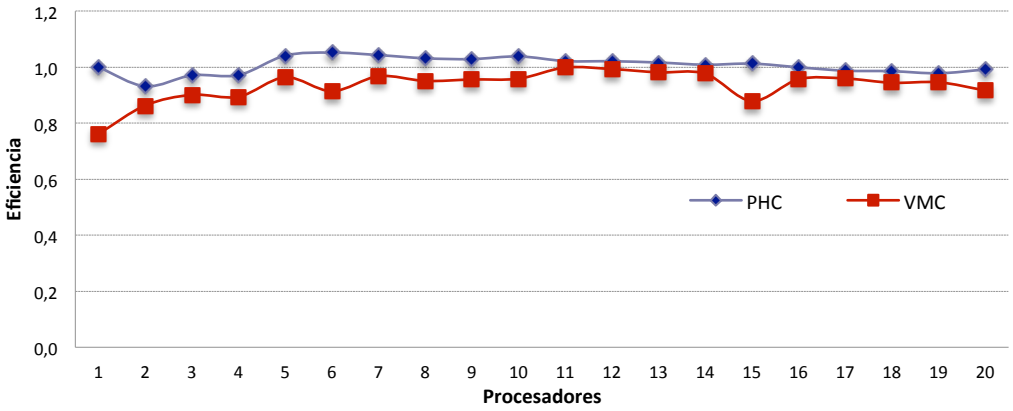

Figura 3.1 - PHC vs VMC: Eficiencia

Tal y como muestra la figura anterior, tanto el PHC como el VMC presentan una eficiencia bastante alta ya que utilizando veinte procesadores la eficiencia es 0,991 y 0,917, respectivamente.

El PHC tiene en todo momento una eficiencia mayor, como era de esperar, y su comportamiento resulta de mayor estabilidad (desviación media menor).

En concreto, el VMC registra una pérdida de rendimiento del 8,07%. Esta pérdida de rendimiento se produce por la penalización introducida por la capa de virtualización del cloud.

	PHC	VMC
Media	1,006	0,934
Desviación Media	0,030	0,055

Tabla 3.3 - PHC vs VMC: Resumen de Eficiencia

Un hecho a tener en cuenta es que en ocasiones la eficiencia aumenta respecto a su ejecución anterior e incluso supera al valor ideal que es 1. Esto ocurre al introducir en el clúster un servidor que presenta un mejor rendimiento que la media de los servidores anteriores que forman el clúster.

2.2. Benchmark NPB

Tal y como se ha indicado anteriormente, los *NAS Parallel Benchmarks* han sido desarrollados por la NASA para el estudio del rendimiento de supercomputadores paralelos. En particular, vamos a utilizar los *benchmarks* LU y SP.

El algoritmo LU, resuelve las ecuaciones de *Navier-Stokes* en 3 dimensiones usando la discretización en diferencias finitas y resolviéndola por el método de descomposición de matrices LU (*lower-upper*).

Por su parte, el programa SP, también resuelven las ecuaciones de *Navier-Stokes*, aunque para ello utiliza la factorización de *Beam-Warming* para hallar la solución mediante Jacobianos totalmente diagonalizados de tamaño 5 por 5.

En las figuras 3.2 y 3.3 se muestran los resultados de la ejecución de los *benchmarks* LU y SP, respectivamente, en los entornos PHC y VHC. El *benchmark* LU se ejecuta para 2^n procesos mientras que el *benchmark* SP se ejecuta para n^2 procesos.

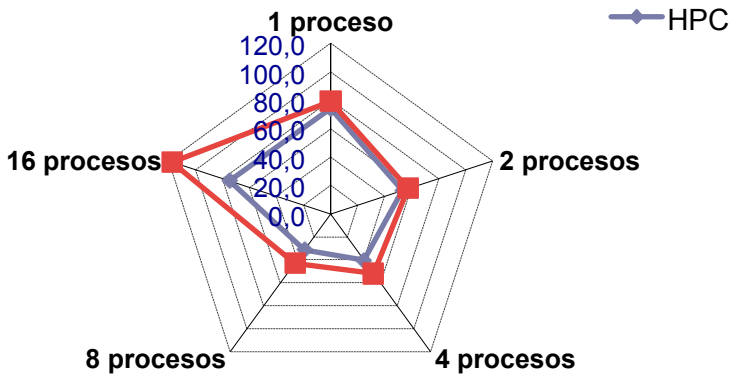

Figura 3.2 - Resultados del Benchmark LU (segundos)

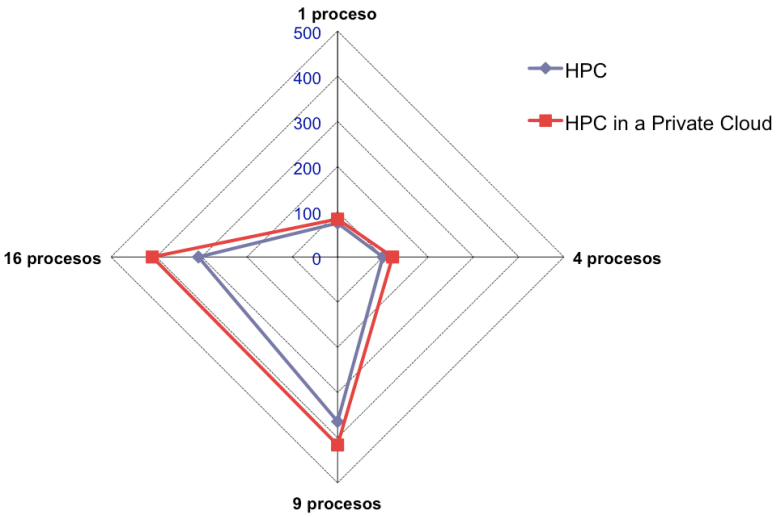

Figura 3.3 - Resultados del Benchmark SP (segundos)

Procesos	PHC	VMC	Penalización rendimiento
1	74,33 s.	78,99 s.	6,27%
2	53,38 s.	57,6 s.	7,91%
4	40,26 s.	51,78 s.	28,61%
8	31,53 s.	42,89 s.	36,03%
16	74,61 s.	117,6 s.	57,62%

Tabla 3.4 - PHC vs VMC: Resultados del Benchmark LU

Procesos	PHC	VMC	Penalización rendimiento
1	75,23 s.	83,49 s.	10,98 %
4	100,79 s.	121,72 s.	20,77 %
9	364,78 s.	416,11 s.	14,07 %
16	306,11 s.	409,38 s.	33,74 %

Tabla 3.5 - PHC vs VMC: Resultados del Benchmark SP

Como es posible apreciar en los gráficos anteriores, el VMC muestra un comportamiento peor que PHC. Concretamente, para la ejecución del *benchmark* LU en el entorno VMC el rendimiento es un 27,29% peor, y en el *benchmark* SP un 19,89%. Por lo tanto, la ejecución del *benchmark* presenta una penalización media en el rendimiento del 23.59%.

Si se analizan los resultados obtenidos, en la gran mayoría de las pruebas realizadas, se puede ver que a medida que aumenta el número de procesos el rendimiento del sistema VMC empeora. Esto es lógico, ya que al aumentar el número de procesos aumentan las comunicaciones del sistema y las diferencias en el rendimiento se hacen más presentes, al existir la capa de virtualización introducida por el cloud.

2.3. Análisis de resultados.

Con los resultados obtenidos en las pruebas de rendimiento anteriores es posible concluir que la creación de clúster en un entorno cloud penaliza el rendimiento entre un 8,07% y un 23,59%, dependiendo del grado de paralelización de la aplicación.

Aunque la creación del clúster utilizando la infraestructura cloud produce una pérdida del rendimiento, ésta tecnología resulta ideal para la puesta en marcha de clústeres de alto rendimiento de bajo coste, adaptables a las necesidades individuales de cada proyecto.

A continuación se va a comprobar que los resultados obtenidos en los entornos de prueba se ajustan a los obtenidos en un problema real. Para ello, se va analizar el rendimiento del sistema en la ejecución de una aplicación que tiene un alto grado de paralelización, como es el caso del criptoanálisis de funciones hash.

Capítulo 4 - EJEMPLO DE APLICACIÓN: CRIPTOANÁLISIS DE FUNCIONES HASH

La constante evolución de las nuevas tecnologías y de las comunicaciones a través de Internet aporta significantes mejoras en cuanto a productividad y mercados de negocio. Esto trae consigo grandes beneficios, pero al mismo tiempo, presenta nuevos retos relacionados con la de seguridad informática.

Uno de los grandes aliados de la seguridad informática es la criptografía gracias a la utilización de funciones hash. Las funciones hash se utilizan en las comunicaciones para almacenar contraseñas, para verificar la integridad de mensajes, etc.

Una función hash permite calcular la huella (o resumen) que identifica de forma unívoca un determinado conjunto de datos. La forma clásica de "romper" las funciones hash es el uso de ataques de fuerza bruta y la utilización de tablas *Rainbow*. Los ataques de fuerza bruta consisten en la generación de todas las posibles soluciones hasta encontrar la correcta, mientras que las tablas *Rainbow* son unas tablas de consulta que ofrecen un compromiso entre tiempo y espacio para obtener claves en texto simple a partir del resultado de una función de hash. La principal desventaja que presenta la aplicación de ambas técnicas es que tienen un coste computacionalmente muy alto, haciendo inviable su uso en un equipo convencional.

1. Criptografía

La criptografía es una de las soluciones más comunes para muchos de los ataques que se pueden producir contra un Sistema Informático. Según la RAE, por definición, es el arte de escribir con clave secreta o de un modo enigmático, sin embargo esta definición arcaica y ambigua queda corta para las necesidades que hoy día se le exige satisfacer. Se define criptografía como la rama de las Matemáticas, la Informática y la Telemática, que hace uso de métodos y técnicas con el objeto principal de cifrar, y por tanto proteger, un mensaje o archivo por medio de un algoritmo, usando una o más claves.

Figura 4.1 - Algoritmo criptográfico

En la figura 4.1 se muestra el funcionamiento básico de un algoritmo de cifrado el cual consta de los siguientes elementos:

- **Texto plano**: Es el mensaje que el emisor envía al receptor, resultando ininteligible para cualquier otra persona.
- **Texto cifrado**: Es el texto obtenido después de codificar el texto en claro.
- **Función de cifrado**: Es el mecanismo a través del cual a partir de un texto claro se obtiene un texto cifrado (cifrar), y a partir de un texto cifrado se consigue un texto en claro (descifrar).
- **Contraseña, llave o clave**: Es la información compartida entre el emisor y receptor mediante la cual es posible cifrar y descifrar.

Relacionado con la criptografía encontramos el término criptoanálisis. Éste es definido como la ciencia que estudia los algoritmos criptográficos, con la intención de detectar cualquier punto débil del mismo para explotarlo y poder obtener el sentido de una información cifrada, sin acceso a la información secreta requerida para obtener este sentido normalmente. Típicamente, esto se traduce en conseguir la clave secreta. En el lenguaje no técnico, se conoce esta práctica como "romper" o "forzar" el código, aunque esta expresión tiene un significado específico dentro del argot técnico.

Criptografía y criptoanálisis son las dos principales áreas de la criptología, que se define de forma general como el estudio de todo lo relacionado con la ocultación y desocultación de la información.

1.1. Criptosistemas

No se puede entender la criptografía moderna sin antes tratar el concepto de criptosistema. Un criptosistema es definido como la conjunción de:

- El conjunto de los posibles mensajes que se podrían llegar a cifrar.
- El conjunto de los posibles mensajes que podrían provenir de la realización del cifrado.
- El conjunto de todas las posibles claves que se podrían emplear en el criptosistema.
- El conjunto de transformaciones de cifrado que se podrían aplicar a cada mensaje sin cifrar para obtener el mensaje cifrado.
- El conjunto de transformaciones de descifrado que se podrían aplicar a cada mensaje cifrado para obtener el mensaje sin cifrar.

Además, todo criptosistema debe cumplir la condición por la cual dado un mensaje cifrado, tras la correcta aplicación de su clave correspondiente, podamos hallar el mensaje original. En la figura 4.2 podemos observar el funcionamiento básico de forma esquemática de un criptosistema.

Figura 4.2 - Criptosistema clásico

Existen diversas formas de clasificar los criptosistemas, siendo las dos más importantes según el tratamiento de la información y según el tipo de claves que usan, que presentamos a continuación.

1.1.1. Criptosistemas según el tratamiento de la información.

Existen dos tipos de criptosistemas a la hora de realizar el cifrado en función de la forma en que el texto es tratado:

- **Cifrado en flujo.** El algoritmo de cifrado se aplica a los elementos de información de manera individual (carácter, bit, etc.). De este modo se implementa un cifrado continuo de los datos. Ejemplos de este tipo de criptosistemas son A5, RC4 y SEAL.
- **Cifrado en bloque.** El algoritmo de cifrado se aplica a elementos denominados bloques, de forma colectiva (grupo de caracteres, número de bytes, etc.). De esta forma se realiza un cifrado discreto de los datos. Ejemplos de este tipo de criptosistemas son AES, IDEA y RSA.

En la tabla 4.1 se puede observar un resumen de las principales ventajas y desventajas de las metodologías de cifrado en bloque y en flujo.

	Ventajas	Desventajas
En Bloque	Alta difusión de los elementos en el criptograma. Es imposible introducir bloques extraños sin detectarlo.	Baja velocidad de cifrado al tener que leer el bloque completo. Propenso a errores de cifra. Un solo error se propaga por todo el bloque.
En flujo	Alta velocidad de cifrado al no tener en cuenta otros elementos. Resistente a errores ya que cifra cada elemento de forma independiente.	Baja difusión de los elementos en el criptograma. Es vulnerable tanto en cuanto los elementos pueden ser alterados por separado.

Tabla 4.1 - Cifrado en Bloque vs Cifrado en Flujo

1.1.2. Criptosistemas según el tipo de clave utilizada

Existen dos tipos de criptosistemas a la hora de realizar el cifrado según el uso de las claves, los criptosistemas de clave secreta y los criptosistemas de clave pública.

a) Criptosistemas simétricos o de clave secreta

La gran mayoría de estos sistemas se apoyan en los conceptos de confusión y difusión. La confusión consiste en el ocultamiento de la relación entre el texto claro, el texto cifrado y la clave. Un buen mecanismo de confusión dificulta la extracción y las relaciones estadísticas existentes entre los anteriores elementos. Por otro lado, la difusión reparte la influencia de cada bit del mensaje original a lo largo del mensaje cifrado.

Estos criptosistemas están caracterizados por utilizar una clave compartida entre emisor y receptor para cifrar y descifrar el mensaje, de modo que la seguridad reside en mantener la clave en secreto. Claros ejemplos de los criptosistemas de clave secreta son DES y AES. En la figura 4.3 podemos ver con mayor claridad su funcionamiento básico, donde un emisor cifra el texto de entrada mediante el uso de una clave secreta compartida por ambos usuarios. Posteriormente, el texto cifrado viaja hasta el receptor y este recupera el texto original mediante el uso de la misma clave secreta.

Figura 4.3 - Criptosistema simétrico

b) Criptosistemas asimétricos o de clave pública

La gran mayoría de estos sistemas apoyan su fortaleza en problemas matemáticos irresolubles como lo es, por ejemplo, la factorización de enteros grandes. Utilizan una clave compuesta por una clave pública y otra privada, de forma que lo que con una se cifra con la otra se descifra.

Este tipo de sistemas permiten implementar tanto el cifrado (garantizando confidencialidad e integridad) como la firma (proporcionando autenticidad, integridad y no repudio). Así pues, según los objetivos que se tengan, se utiliza la clave pública o la privada para cifrar.

Los mayores representantes de este tipo de criptosistemas son RSA y ElGamal. En la figura 4.4 puede verse un ejemplo esquematizado del funcionamiento de los criptosistemas de clave pública para la operación de cifrado.

Figura 4.4 - Criptosistema asimétrico

Así en la tabla 4.2 realizamos una comparativa entre los criptosistemas de clave publica y clave privada

	Clave pública	Clave privada
Gestión de Claves	Sólo es necesario memorizar la clave privada del emisor y la pública del receptor.	Hay que memorizar un número muy alto de claves (N^2)
Longitud y Espacio de Claves	La clave es del orden de miles de bits.	La clave es del orden de centenares de bits.
Vida de una Clave	La duración de la clave suele ser larga.	La duración de la clave es muy corta, normalmente caduca al finalizar la sesión.
Autenticación	Al existir una clave pública y otra privada se puede autenticar el mensaje y al emisor.	Solo es posible autenticar el mensaje.
Velocidad de Cifra	La velocidad de cifra es muy lenta.	La velocidad de cifra es muy alta.
Uso	Son utilizados sobre todo para intercambios de claves y firma digital.	Son algoritmos utilizados sobre todo para cifrado.

Tabla 4.2–Criptosistema asimétrico vs simétrico

1.2. Funciones hash

Una de las herramientas para evitar ataques informáticos es la criptografía, y en concreto el uso de la firma digital, ya que gracias a ella es posible certificar tres de las características fundamentales que debe cumplir una comunicación segura (no repudio, autenticación e integridad). Para trabajar con la firma digital es necesario utilizar algoritmos de cifrado asimétrico con el consecuente retardo que estos conllevan. Esto puede ser solucionado gracias al uso de funciones resumen o hash.

Una función hash es una función encargada de resumir una cadena binaria conocida como ***preimagen***, dando como resultado una huella o imagen de la misma. Dicha huella suele ser sustancialmente más pequeña a la información original y es capaz de identificarla inequívocamente con altas tasas de probabilidad. Así, si se introducen diferentes informaciones a la misma función hash, la probabilidad de que devuelva el mismo resultado es ínfima. En la figura 4.5 se muestra un ejemplo gráfico de la funcionalidad ofrecida por las funciones hash.

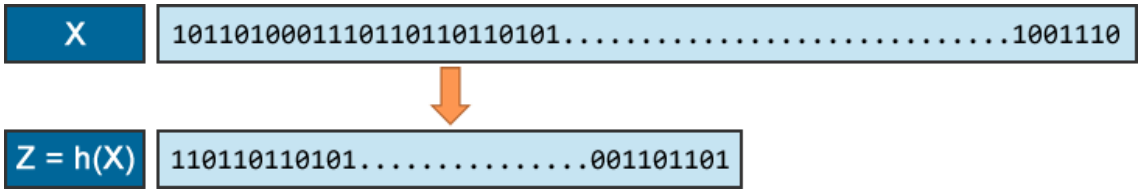

Figura 4.5 - Función HASH

La funciones hash son de gran utilidad ya que a la hora de firmar un archivo no es necesario realizar costosas operaciones de cifrado con la totalidad del mismo. En este caso lo único que hace falta es calcular su firma digital y es ésta la que se firma. Gracias a esta estrategia se ahorra ingentes cantidades de tiempo garantizando la misma calidad y seguridad.

1.2.1. Descripción de las funciones hash

Un mensaje puede presentar las características de autenticación, no repudio e integridad con tan sólo codificar la huella o hash con la llave privada del emisor. La información generada en este proceso se denomina firma digital del mensaje y sólo puede ser generada por el emisor. Cualquier entidad puede acceder a la clave pública correspondiente, y por lo tanto estar en condiciones de decodificar y verificar la firma digital.

Las funciones hash se basan en la idea de utilizar la compresión de datos, dando como resultado bloques destino de longitud *n* a partir de bloques origen de longitud *m*. Dichas compresiones se encadenan de forma iterativa haciendo que los bloques en el paso *i* dependan del bloque origen *i* del mensaje, y del bloque destino del paso *i-1*, tal y como representa la figura 4.6.

Figura 4.6 – Funcionamiento iterativo de las funciones Hash

Además, se suele incluir información acerca de la longitud total del mensaje en alguno de los bloques, ya sea al principio o al final del mismo, con lo cual se reducen las posibilidades de que dos mensajes con diferente longitud y contenido generen el mismo valor *hash*.

1.2.2. Características de las funciones hash

Toda función o algoritmo hash debe cumplir una serie de características sin las cuales no se puede verificar su seguridad, resultando en tal caso obsoleta e inservible:

- **Unidireccionalidad**. Dada la firma digital debe ser computacionalmente imposible recuperar el mensaje original.
- **Compresión**. La firma digital debe ser de longitud fija, independientemente de la longitud del mensaje original.
- **Facilidad de cálculo.** Dado el mensaje original, debe ser fácil calcular la firma digital de este.
- **Difusión**. La firma digital debe ser una función compleja de todos los bits del mensaje, de tal forma que si se modifica un solo bit, el hash resultante deberá cambiar al menos la mitad de sus bits aproximadamente.
- **Resistente a colisiones.** Existen dos tipos:
  - **Colisión Simple**. Dado el mensaje m_0, debe ser computacionalmente imposible obtener otro mensaje m_1, tal que el hash de m_0 sea idéntico al de m_1.
  - **Colisión fuerte**. Será computacionalmente complicado encontrar un par (m_0, m_1) de forma que el hash de m_0 sea idéntico al de m_1.

Observando estas características se puede deducir que un algoritmo o función hash no es un algoritmo de encriptación propiamente dicho, aun así gracias a sus inestimables propiedades se ha hecho un hueco de relevancia en el mundo de la criptografía, y en particular, de la seguridad informática.

1.2.3. Paradoja del cumpleaños

Vistas las anteriores características nos surge inevitablemente la siguiente pregunta: ¿cuál debe ser la longitud del valor hash?, Ante esta pregunta irrumpe la conocida paradoja del cumpleaños.

Esta paradoja se basa en la siguiente situación: dado un grupo de personas en una habitación, ¿cuál es la cantidad mínima de personas que debe haber en el grupo para que el cumpleaños de una de ellas sea el mismo día que el mío con más del 50% de probabilidad? Solamente 23 personas.

Este ejemplo permite ver que aunque resulte muy difícil dado un mensaje m_0 calcular otro m_1 tal que el hash de m_0 sea el mismo que el de m_1, es considerablemente

menos costoso generar muchos valores aleatoriamente, y posteriormente buscar entre ellos una pareja de mensajes cualquiera (m_0, m_1), tal que el hash de m_0 sea idéntico al de m_1.

En el caso de una firma de 64 bits, se necesitan 2^{64} mensajes consecutivos dado un m_0 para obtener otro m_1 cuyo hash sea idéntico, pero según esta paradoja, bastaría con generar aproximadamente 2^{32} mensajes aleatorios para que aparecieran dos cuyo hash coincida. Podemos observar que si para el primer caso el calculo computacional es muy elevado, la segunda cantidad representa aproximadamente su raíz cuadrada.

Para hacerse una idea el primer ataque nos llevaría 600.000 años con una computadora que generara un millón de mensajes por segundo, mientras que el segundo necesitaría apenas una hora. Hoy por hoy, se recomienda emplear hash de al menos 128 bits, siendo 160 bits el valor más usado.

1.3. Algoritmos hash

Los algoritmos hash, debido a su gran relevancia, han sido objeto de estudio durante las últimas décadas, dando lugar a una gran variedad de soluciones. Los algoritmos o funciones hash más importantes son:

- **MD5**. Ideado por Ron Rivest en 1992, proviene de las mejoras de MD4 y MD2 (1990), siendo más lento, aunque esto lo suple con un mayor nivel de seguridad. Su resumen es de 128 bits.
- **SHA-1**. Creado por el NIST (*National Institute of Standards and Technology*) en 1994. Proviene de SHA y es similar a MD5 pero su resumen es de tamaño superior: 160 bits. Existen otras propuestas o mejoras como lo son SHA-256 y SHA-512.
- **LM**. Creado por Microsoft, no es exactamente una función hash puesto que utiliza el algoritmo criptográfico simétrico DES para realizar la función resumen, la cual tiene un tamaño fijo de 128 bits. Existen versiones mejoradas del mismo como lo es NTLM.
- **NTLM.** Creado por Microsoft, utiliza otras funciones hash como MD4, además del DES para generar la respuesta. La función resumen tiene un tamaño fijo de 128 bits.
- **RIPEMD-160.** Creado por la Comunidad Europea RACE en 1992. Tiene un resumen de tamaño 160 bits.
- **N-Hash**. Creado por *Nippon Telephone and Telegraph* en 1990. Tiene un resumen de 128 bits.
- **Snefru.** Creado por Ralph Merkle en 1990. Sus resúmenes pueden variar de tamaño entre 128 y 256 bits. Cabe destacar que ha sido criptoanalizado y es muy lento.

- **Tiger**. Creado por Ross Anderson y Eli Biham en 1996. Genera resúmenes de hasta 192 bits. Este algoritmo se encuentra optimizado para máquinas de 64 bits (Alpha).
- **Panama.** Creado por John Daemen y Craig Clapp en 1998. Genera resúmenes de 256 bits de longitud. Puede trabajar en modo función hash o como cifrador de flujo.
- **Haval.** Creado por Yuliang Zheng, Josef Pieprzyk y Jennifer Seberry en 1992. Admite quince configuraciones diferentes, llegando a tener resúmenes de hasta 256 bits de tamaño.

En la tabla 4.3 se muestra una comparativa con las cinco funciones hash de mayor relevancia en la actualidad.

	MD5	**SHA1**	**RIPMD -160**	**LM**	**NTLM / MD4**
Tamaño de Resumen	128 bits	160 bits	160 bits	128 bits	128 bits
Tamaño de Bloque	512 bits	512 bits	512 bits	112 bits	512 bits
Número de Pasos	64	80	160	19	48
Tamaño de Mensaje	∞	∞	∞	14 bytes	∞
Fuerza de la Preimagen	2^{128}	2^{160}	2^{160}	2^{128}	2^{128}
Fuerza Contra Ataque de Colisión	2^{64}	2^{80}	2^{80}	2^{64}	2^{64}

Tabla 4.3–Comparativa de las funciones hash de mayor relevancia

1.4. Criptoanálisis de las funciones hash

En el año 2004 salieron a la luz las primeras noticias sobre la ruptura de la función hash MD5, y desde ese momento la comunidad criptológica se ha cuestionado la seguridad que ofrecen los algoritmos hash a nuestros esquemas de cifrado.

El criptoanálisis es la ciencia encargada de buscar las vulnerabilidades de los criptosistemas. En el caso de las funciones hash, la seguridad o fiabilidad está soportada sobre los pilares de las matemáticas más que sobre los de la informática o la telemática, de tal modo que las debilidades tienen más implicaciones matemáticas que computacionales. A continuación se muestran los principales tipos de ataques que se pueden realizar sobre una función hash.

- **Ataque del cumpleaños**. Es un ataque por Fuerza Bruta basado en las implicaciones matemáticas heredadas de la conocida paradoja del cumpleaños. Básicamente este tipo de ataque consiste en generar cadenas aleatorias *M'*, realizar *Hash(M')* y compararlas con *Hash(M)* original. Si coinciden se habrá encontrado la solución siendo $M = M'$, en caso contrario se continúa generando nuevas cadenas.

  En el caso de SHA1 (160 bits) este ataque necesitaría generar 2^{80} mensajes para hallar la solución. En la figura 4.7 se muestra como procede este tipo de ataque.

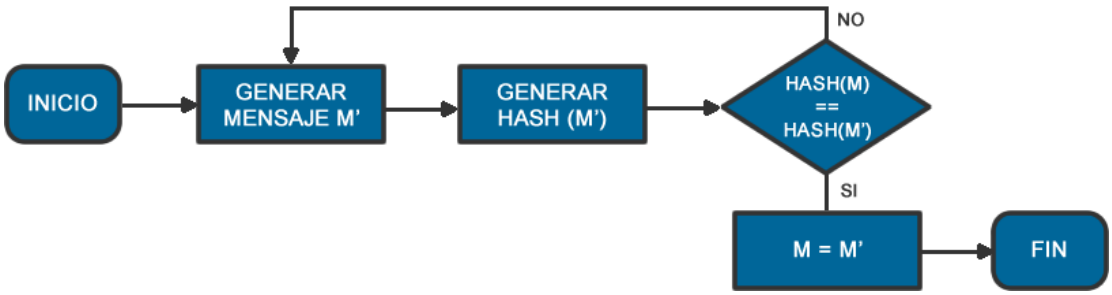

Figura 4.7- Ataque del Cumpleaños

- **Ataque Wang-Yin-Yu o Ataque chino**. Es un ataque por Fuerza Bruta simplificado, en el cual gracias a ciertos conocimientos previos al mismo, permite reducir el número de alternativas del hash. Ilustraremos este ataque con el siguiente ejemplo.

  Imagine que quiere realizar la venta de un inmueble, para ello redacta un contrato de venta *A*, el cual contiene tanto el texto con los datos pertinentes, como imágenes de la vivienda.

  Al mismo tiempo genera un contrato alternativo A' con las mismas fotos pero modificando el precio de venta a su favor. Por las propiedades vistas con anterioridad sería prácticamente imposible que Hash(A)=Hash(A'). Ahora bien puede generar múltiples versiones de A y A' modificando tan sólo unos pocos bits de las imágenes (modifica sólo las imágenes para que no llame tanto la atención la trampa realizada) hasta conseguir que Hash(A) y Hash(A') sean idénticos.

  Ahora solo tendría que presentar al comprador el documento A, él aceptará las condiciones y lo firmará electrónicamente. Luego cambia A por su variante A' y

ya tiene un contrato legal de compraventa firmado con unas condiciones distintas de lo establecido.

Con este tipo de ataque, para romper un algoritmo hash SHA1, es posible pasar de generar 2^{80} mensajes a tan sólo 2^{69}. En la figura 4.8 se muestra de forma gráfica este tipo de ataque.

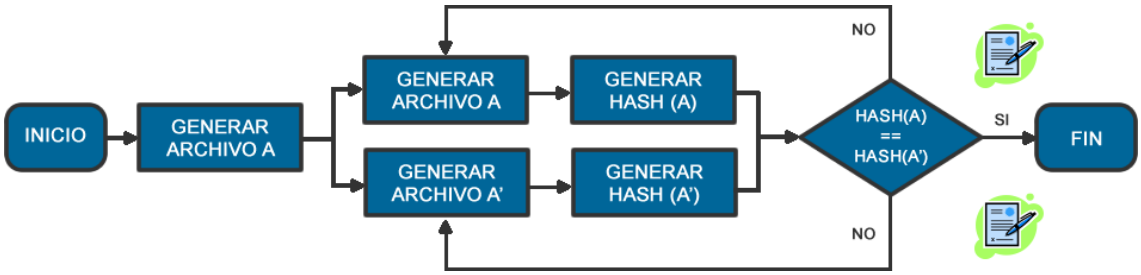

Figura 4.8 - Ataque Chino

- **Ataque por extensión de longitud.** Este ataque se basa en que dado un hash *Hash(m)*, en el que se conoce *Hash(m)* pero no el mensaje *m*, se puede generar nuevas huellas "válidas" que incluyan a la anterior aunque difieran de ella. Esto se realiza concatenando *Hash(m) + m'* para posteriormente realizar el hash a la totalidad dando como resultado *Hash(Hash(m) + m')*. En la figura 4.9 podemos visualizar este tipo de ataque.

Figura 4.9 - Ataque por extensión de longitud

- **Rainbowtables.** Aunque veremos esta técnica con más detalle más adelante, en resumen consiste en generar cadenas de hash, para ir guardando el primer y el último valor de las mismas en un archivo con formato de tabla. Posteriormente las tablas generadas son ordenadas y, por último, utilizadas para la búsqueda de la solución que rompa el hash gracias a una reconstrucción de las mismas. En la figura 4.10 se muestra la secuencia de pasos en la que consiste este tipo de ataque.

Figura 4.10 - Ataque con tabla Rainbow

1.5. Aplicaciones de las funciones hash

Hasta el momento se ha definido lo que es una función hash, cuáles son sus propiedades, tipos e incluso vulnerabilidades, pero apenas se vislumbra todo su

potencial. A continuación se enumeran algunas de las aplicaciones de mayor relevancia de este tipo de funciones.

- **Verificación de contraseñas**. Los procesos de verificación (*Login+Password*) utilizan comúnmente este tipo de algoritmos. Su modo de utilización está basado en que cuando un usuario accede a su ordenador introduce su *login* y su *password*. Evidentemente estos datos no pueden quedar en texto plano, puesto que serían vulnerables a ataques externos.

  Por este motivo suelen ser codificados mediante el uso de funciones hash, así en el caso de que un usuario malicioso logre acceder a nuestro archivo de registros, no conseguirá la contraseña. De este modo cuando introducimos nuestro *login* y nuestro *password* el sistema operativo genera su hash correspondiente y lo compara con los que tiene almacenados. Si coinciden los valores, significa que el *login* y *pasword* son legítimos, en caso contrario son falsos. Algunos ejemplos de este tipo de aplicaciones son:

  - Funciones hash LM y NTLM para la administración de claves de Windows.
  - Uso de SHA1 para la administración de claves en GNU/Linux.
  - Uso de MD5 para verificación de usuario en conversaciones VoIP.
  - Identificación de usuarios en sitios web mediante uso de MD5 y SHA1.

- **Comprobación de la integridad de mensajes y ficheros**. Gracias al uso de funciones hash se puede verificar la integridad de un mensaje, fichero o código. Lo único que necesita es realizar la función hash sobre el mismo y entregar la huella resultante junto con el fichero al receptor. El receptor sólo necesita generar su propio hash para posteriormente compararlo con el recibido. Si son idénticos significa que el mensaje o archivo no fue modificado, en caso contrario, se detecta una violación de la integridad. Algunos ejemplos de la utilización de este tipo de aplicaciones se encuentra en:

  - Utilización de MD5 para comprobar la integridad del software GNU/Linux.
  - Uso de algoritmos como MD5 y SHA1 para comprobar la integridad mensajes.
  - Comunicaciones seguras tipo TLS usan funciones hash como MD5 y SHA1.

- **Generación de números pseudoaleatorios.** Las funciones hash también son utilizadas por ciertas aplicaciones en la generación de números pseudoaleatorios añadiendo así nuevos grados de aleatoriedad. La idea es utilizar una cadena que ya exista para realizarle la función hash sobre la misma, a partir del hash generar el número pseudoaleatorio.

- **Firma digital**. Las funciones hash son de gran utilidad a la hora de realizar una firma digital, ya que gracias a ellas los algoritmos de cifrado asimétrico no necesitan cifrar con su clave privada la totalidad del mensaje o archivo, sino sólo la huella del mismo, disminuyendo y descartando cálculos que retardarían el

conjunto de la operación. Gracias al uso de la firma digital podemos otorgar a la comunicación las propiedades de: no repudio, integridad y autenticidad. Un claro ejemplo de esta situación lo encontramos en los certificados x.509, los cuales usan MD5 para la generación de su firma digital.

2. Entornos de trabajo

Una vez vistos de forma detallada los conceptos criptografía y criptoanálisis, así como los diferentes tipos de algoritmos hash más representativos y sus características, el resto de este capítulo se centrará en la implementación y obtención de resultados para diferentes aplicaciones basadas en fuerza bruta y generación de tablas *Rainbow*, las cuales se han implementado mediante diferentes metodologías de programación en dos entornos de trabajo diferentes, sobre los que obtendremos importantes conclusiones para futuras líneas de trabajo.

Para llevar a cabo estas aplicaciones de criptoanálisis de funciones hash se ha decidido utilizar dos escenarios distintos, por un lado el reciente uso de procesadores gráficos para aplicaciones de propósito general, conocidos como GPGPU, permitiendo el uso de superordenadores de alto rendimiento a un bajo costo. Y como escenario alternativo, continuamos trabajando con la metodología cloud computing, que como hemos visto en el capítulo anterior nos ofrece grandes ventajas, adaptándose a nuestras necesidades y con un bajo coste.

2.1. GPGPU

Para este primer entorno de trabajo utilizamos un servidor Tesla S1070 que nos permite obtener un rendimiento de hasta 4 Teraflops. Este servidor ofrece la posibilidad de disponer de una supercomputadora de alto rendimiento para resolver cualquier problema de forma rápida, representando éste un punto de inflexión para el criptoanálisis de funciones hash.

Todos los resultados que se muestran a continuación usando esta tecnología han sido obtenidos a partir de un servidor MX DUAL AZServer Xenon con NVIDIA Tesla S1070. Este servidor cuenta con dos procesadores Dual / Quad Core Intel Xenon 2,66 GHz, 8 módulos de 2 GB de SDRAM y dos discos duros SATA de 1 TB en RAID 1. Además, el equipo dispone de cuatro tarjetas Tesla Procesador T10 con un total de 960 núcleos de 1,44 GHz cada una.

2.2. Elastic Cluster

En este apartado volvemos a hacer uso de un clúster desplegado en una infraestructura cloud, por lo que utilizamos el equipamiento descrito en el capítulo anterior y que a continuación recoge en la tabla 4.4.

Modelo	CPU	Memoria	Disco Duro	Unidades
Dell R210	4 x INTEL XEON X340 - 2,40 GHz	4GB	250 GB HDD	5
Dell R200	4 x INTEL XEON X310 - 3,13 GHz	4GB	250 GB HDD	1
HP DL120 G5	2 x INTEL XEON E3110 - 3,00 GHz	4GB	250 GB HDD	3

Tabla 4.4 – Equipo físico del Elastic Cluster

En estas pruebas hacemos uso de la totalidad del equipamiento disponible, consiguiendo así contar con un Clúster de Máquinas Virtuales de 28 procesadores, ya que utilizaremos un servidor HP DL120 G5 como Cloud Controller, y utilizaremos el resto de equipos como Node Controller. Una vez creada la infraestructura cloud, es posible desplegar 14 instancias *m1.large* de 2 procesadores, 512 MB de RAM y 5 GB de disco duro cada una.

Es conveniente recordar que para la configuración de los sistemas se ha utilizado la distribución Ubuntu Maverick Meerkat 10.10 con Ubuntu Enterprise Cloud en los servidores físicos, así como Ubuntu Lucid Lync 10.04 en las instancias de las máquinas virtuales y Mpich2 1.2.1 para el clúster virtual.

3. Implementación y resultados

En este apartado nos vamos a centrar en las aplicaciones que han sido utilizadas para obtener los resultados que nos permitirán comparar el rendimiento del criptoanálisis de funciones hash en los dos escenarios mencionados. Para ello, el software empleado está basado en las dos técnicas más utilizadas para buscar las vulnerabilidades de los criptosistemas: los ataques de fuerza bruta y la generación de tablas *Rainbow*.

3.1. Fuerza Bruta

La Fuerza Bruta es una de las técnicas criptoanalíticas hash más extendidas y comúnmente conocidas. Básicamente, este tipo de criptoanálisis está basado en el conocido ataque del cumpleaños ya mencionado en este capítulo, y que consiste en que dado un hash *m*, hay que encontrar un texto M' cuyo valor de Hash(M') sea igual al hash original (*m*). En el caso de SHA1 (160 bits) este ataque necesitaría generar 2^{80} comprobaciones para obtener la solución.

En nuestro estudio se han utilizado dos programas de fuerza bruta, en uno de ellos se analiza el rendimiento en la generación de hash para distintos algoritmos, mientras que en el segundo analizamos los tiempos para la devolución del valor de una cadena de texto a partir de un hash MD5 dado. Del mismo modo que se ha utilizado para el algoritmo MD5, la adaptación del programa para utilizar cualquier algoritmo hash es rápida y sencilla, ya que nos hemos basado en las funciones hash de las librerías *Openssl*.

3.1.1. Generación de hash.

Nos encontramos ante un problema altamente paralelizable, en el cual a partir de una cadena de texto generamos su hash un número elevado de veces según el algoritmo criptográfico seleccionado. Para este programa, escrito en C y con el que continuamos el trabajo de proyectos anteriores, utilizamos dos metodologías de programación paralela, MPI para nuestro *Elastic Cluster* y CUDA para nuestro entorno *GPGPU*, y que comparamos con la versión secuencial de la aplicación. Dichas metodologías de programación paralelas, son objetivo de nuestro análisis de rendimiento, aprovechando la optimización de las funciones hash de las librerías Openssl, de las cuales utilizamos los principales algoritmos: MD5, SHA1 y LM.

Antes de comenzar a analizar los resultados obtenidos, vamos a introducir los detalles fundamentales de cada una de estas metodologías de programación paralela.

a) Elastic Cluster - MPI

La programación basada en MPI es una metodología paralela, en la cual se persigue mejorar el rendimiento de los programas mediante el uso de varios procesadores de forma concurrente, así, a cada procesador le es asignado una línea de ejecución diferente.

Por definición MPI (*Message Passing Interface*) es una interfaz estándar para la implementación de aplicaciones paralelas basadas en el paso de mensajes. Dicho estándar no exige una determinada implementación, gracias a ello, MPI brinda ciertas libertades, como por ejemplo el hecho de que el programador tenga cierto grado de independencia con respecto a la aplicación paralela. Esto se consigue debido a que el programador tan sólo necesita conocer una colección de funciones gracias a las cuales puede diseñar la aplicación, sin tener que conocer el hardware sobre el que subyace, ni la forma en la que se han implementado las funciones empleadas.

Para mejor entendimiento de este sistema en la figura 4.11 se muestra la situación de la interfaz propuesta por MPI en un sistema informático. Como se puede observar MPI se encuentra justo en la linde entre la aplicación paralela propiamente dicha y el software de paso de mensajes perteneciente al sistema operativo.

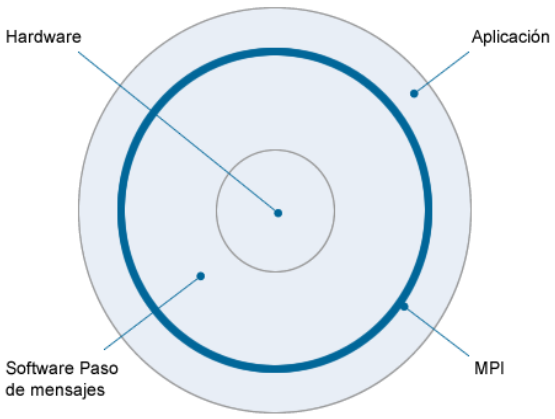

Figura 4.11 - Situación de MPI en el proceso de programación de aplicaciones paralelas

Gracias a este modelo, MPI está soportado por cinco grandes pilares, indispensables para cualquier modelo de programación paralela que se precie, y que a continuación citamos.

El primer pilar, la **portabilidad**, se basa en conseguir que MPI pueda ser ejecutado en una gran variedad de máquinas. El hecho de saber que existen implementaciones de MPI eficientes para una amplia variedad de sistemas, nos da cierto grado de flexibilidad para el desarrollo del código, la búsqueda de errores y la elección de la plataforma a utilizar.

El segundo pilar, la **heterogeneidad**, está basado en la capacidad aportada por MPI para ejecutarse en sistemas heterogéneos de manera transparente. De este modo el usuario no tiene que preocuparse de si el código está enviando mensajes entre procesadores de la misma o distinta arquitectura.

El tercer pilar, el **rendimiento**, ya que MPI fue diseñado de manera que permite implementaciones eficientes. Este rendimiento se conjuga con los dos pilares anteriores de forma que MPI aporta implementaciones eficientes independientemente del sistema operativo y de la arquitectura.

El cuarto pilar, la **escalabilidad**, este objetivo es intrínseco al procesamiento paralelo, ya que todo paralelismo se basa en la escalabilidad tanto del hardware como del software. De este modo los programas MPI deben mantener tener un alto nivel de capacidad para responder a cargas de trabajo crecientes. Así su nivel de rendimiento no mengua aunque se incremente el número de procesos a ejecutar.

El quinto y último pilar, la **formalidad**, este valor se centra en la validez de MPI como estándar. Y lo cumple al definir el comportamiento de las implementaciones de manera concisa e inequívoca. Esta característica libera al programador de tener que preocuparse de aquellos problemas que puedan aparecer por inestabilidades del sistema.

Así los creadores de MPI diseñaron una herramienta capaz de:

- Diseñar una interfaz para la programación de aplicaciones que pueda ser implementada en la mayoría de las plataformas.
- Permitir que las implementaciones puedan ser utilizadas en entornos heterogéneos.
- Permitir una comunicación eficiente y fiable.
- Proporcionar una semántica para la interfaz independiente del lenguaje.
- Permitir la utilización de hilos.
- Proporcionar extensiones que añadan más flexibilidad.

Con todo esto se concluye que MPI está especialmente diseñado para potenciar las aplicaciones paralelas. Para ello siempre sigue los mismos pasos básicos:

1. Lanzar en paralelo *n* copias del mismo código en forma de procesos independientes.
2. Realizar las operaciones designadas en el código en cada uno de sus procesos. Estos procesos no están sincronizados instrucción a instrucción, así que, si se desea realizar una sincronización es necesario hacerlo de forma explícita mediante el paso de mensajes. Para ello consta de funciones de comunicación punto a punto (involucran sólo a dos procesos), y de funciones colectivas (involucran a múltiples procesos).
3. Finalizar todos los procesos designados en el programa.

Tal y como se detalla en el primer capitulo, existen múltiples implementaciones libres de MPI tales como MPICH, OpenMPI o LAM/MPI. Para nuestro proyecto se ha utilizado MPICH, concretamente la versión MPICH2 1.2.1. MPICH2 es toda una nueva implementación de MPI, diseñada para soportar investigaciones basadas en las funcionalidades de alto rendimiento de las implementaciones de MPI-1 y MPI-2. Además de las características presentes en MPICH, MPICH2 incluye soporte de comunicación para operaciones colectivas y procesos dinámicos, así como funcionalidad expandida MPI-IO, además de soportar clústeres basados en un sólo procesador.

b) GPGPU - CUDA

La programación basada en CUDA (*Compute Unified Device Architecture*) es una metodología paralela que tiene por objetivo facilitar el desarrollo de aplicaciones que puedan ejecutarse en los multiprocesadores de las tarjetas gráficas (GPUs), sin tener que preocuparse por las características de cada modelo de tarjeta. Para ello está formada por un compilador y un conjunto de herramientas de desarrollo creadas por nVidia, permitiendo a los programadores usar una variación del lenguaje de programación C, para codificar algoritmos en GPUs mediante el uso de controladores nVidia.

CUDA intenta explotar las ventajas de las GPUs frente a las CPUs de propósito general, utilizando el paralelismo que ofrecen sus múltiples núcleos, los cuales permiten el lanzamiento de un altísimo número de hilos simultáneos. Por ello, si una aplicación está diseñada utilizando numerosos hilos que realizan tareas independientes (que es lo que hacen las GPUs al procesar gráficos, su tarea natural), una GPU puede ofrecer un gran rendimiento en campos que podrían ir desde la biología computacional a la criptografía como es nuestro caso. Las características de CUDA son:

- Es un lenguaje de programación C modificado para ser usado bajo GPU.
- No requiere ningún conocimiento previo de APIs gráficas o de programación en GPU.
- Tiene un acceso nativo a las instrucciones y a la memoria.
- Es fácil empezar a obtener beneficios reales de rendimiento.
- Fue diseñado y desarrollado por nVidia para sus tarjetas gráficas (GeForce 8xxx/9xxx/Tesla/Quadro), por tanto es necesario poseer una de ellas para poder utilizarlo.

- Es estable.
- Es gratuito.
- Está documentado y apoyado por la compañía nVidia.
- CUDA es multiplataforma siendo soportado por Windows, GNU/Linux y Mac.

El modelo de programación de CUDA está diseñado para crear aplicaciones que escalen de forma transparente su paralelismo. Para ello se basará en los modelos de ejecución y de memoria.

Modelo de ejecución

CUDA, al ser un modelo de programación paralela basado en GPU, no depende de un único soporte, como ocurriera con los modelos tradicionales basados en CPU. CUDA necesita de la cooperación de la CPU o host con la tarjeta gráfica o device. De esta forma el device opera como un coprocesador del host.

El kernel o núcleo es ejecutado en el device en forma de threads o hilos, gracias a una llamada de la función desde el host con el siguiente formato:

```
funcionDevice<<<dimGrid, dimBlock>>>(variables…);
```

Dicha ejecución se genera de forma ordenada tanto espacialmente como temporalmente, siendo el host (CPU) y el dispositivo (GPU) quienes administran sus propias memorias. Así durante la ejecución de un programa CUDA se intercalan y superponen ejecuciones de código en la CPU y en la GPU.

Por ello la CPU se suele encargar de:

- La inicialización de los datos.
- Las transferencias de datos a la memoria de la GPU.
- Del lanzamiento de los kernels
- De recoger sus resultados, mientras que la GPU realiza el grueso de los cálculos.

En el nivel de abstracción más bajo de CUDA, se encuentra la unidad básica de ejecución, los *threads* o hebras, los cuales son una serie de instrucciones de código que se ejecutan de forma secuencial. Cada hebra está identificada por un identificador único, al cual se accede mediante la variable ***threadIdx***. Las hebras en grupos de 32, forman ***warps***, de manera que todos las hebras pertenecientes al mismo ***warp*** se ejecutan en paralelo. Ya en un nivel de abstracción intermedio se encuentran los ***bloques***. Al igual que ocurría con los *warps*, los *bloques* están compuestos por *threads*, pero en esta ocasión no tiene porque ser en grupos de 32, sino que puede ser cualquier valor hasta un máximo de 512. Aun así es conveniente, por motivos de eficiencia, que los bloques estén compuestos por un número de hebras múltiplo de 32.

Todo esto puede ser observado con mayor claridad en la figura 4.12, donde se muestra un ejemplo de como interactúa el código entre el host y los *device* en un programa escrito en C.

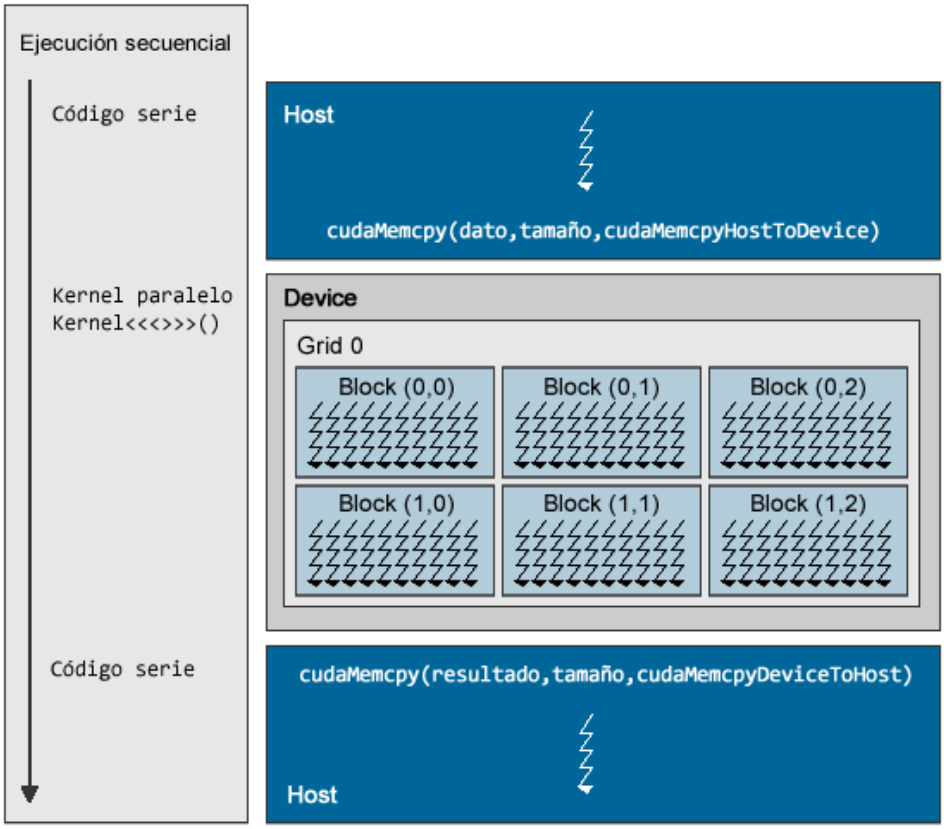

Figura 4.12 – Estructura de un programa CUDA

Al igual que ocurre con los *thread*, los bloques están identificados por un identificador único, al cual se accede mediante la variable ***blockIdx***. Cada bloque con su identificador es asignado a un multiprocesador, y este es el encargado de dividir los threads del bloque en warps para su posterior ejecución. Por el diseño, los multiprocesadores pueden tener un máximo de 8 bloques activos por multiprocesador.

Ya en último nivel de abstracción se encuentran los ***grid*** o **redes**. Un *grid* está compuesto por un conjunto de bloques de hebras. Cada *grid* puede contener hasta un máximo de 65535 bloques. En la figura 4.13 se muestra un gráfico con el modelo de ejecución típico de una arquitectura CUDA.

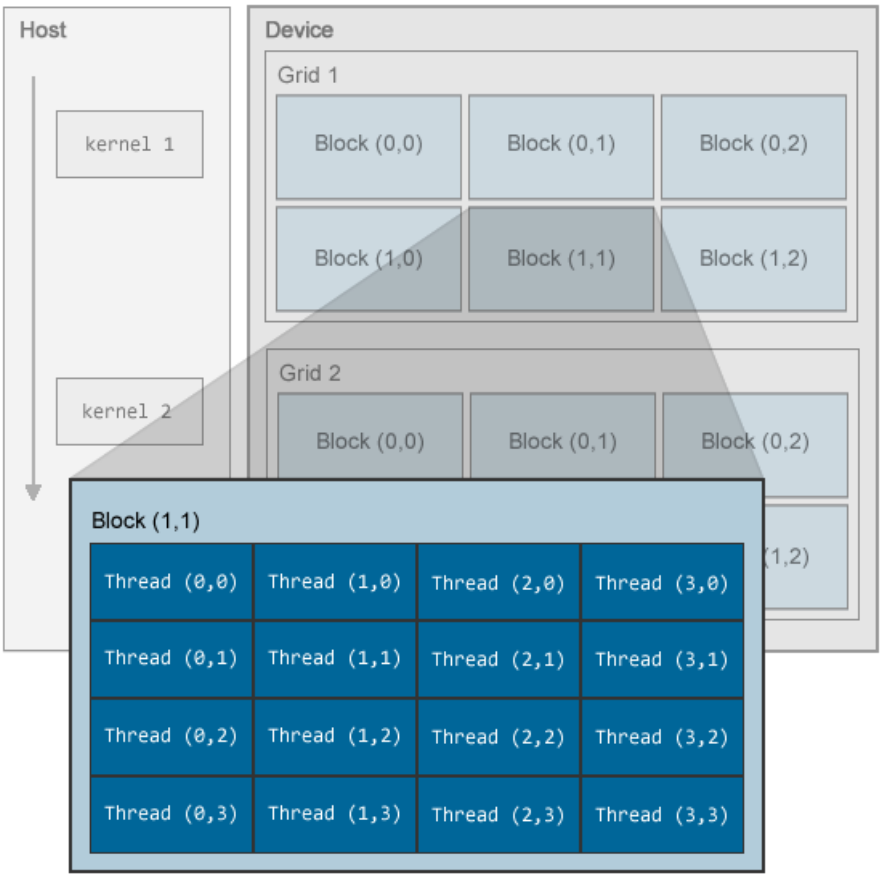

Figura 4.13 - Modelo ejecución CUDA

Modelo de memoria

La arquitectura de memoria CUDA está dividida en diversos niveles lo cual no hace más que redundar en su modularización ya comentada. Las memorias más cercanas a la unidad de proceso son los registros siendo solamente accesibles a nivel de thread. Justo después se encuentra la memoria compartida accesible por todos los threads pertenecientes al mismo bloque. Esta memoria es la que permite comunicar los distintos threads pertenecientes al bloque. Posteriormente están las memorias global y local.

La primera de ellas es accesible tanto por el host como por el device, y es usada como medio de transmisión de información entre la CPU y la GPU. Mientras que la local es sólo accesible por la hebra, y está situada en el mismo espacio de memorias que la global y consecuentemente es tan lenta como ella. En adición, existen otros dos espacios de memoria más, que son de sólo lectura para el device y accesibles por todos los hilos. Son la memoria de constantes y la de texturas. Si necesitamos modificar los valores de estas memorias es necesario hacerlo desde el host. En la figura 4.14 se muestra la jerarquía de memorias de CUDA.

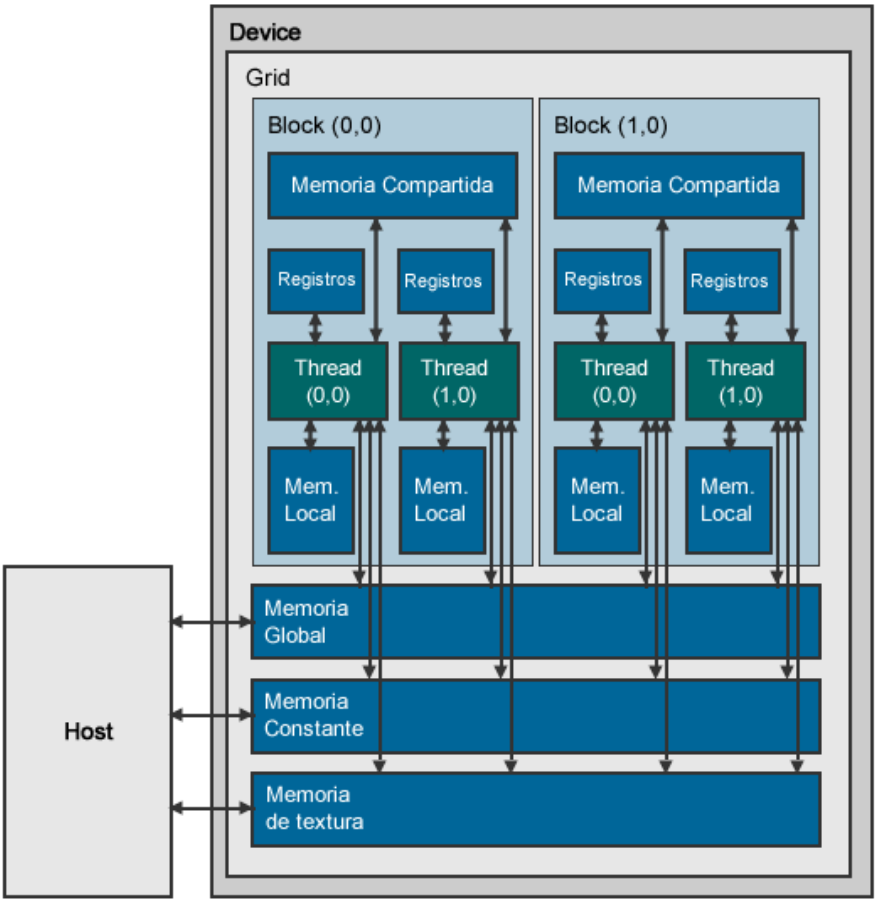

Figura 4.14 - Jerarquía de memorias en CUDA

El tiempo de acceso varía según el tipo de memoria, siendo la memoria que se encuentra más cercana a la unidad de proceso la más rápida.

En la tabla 4.5 se puede ver la relación de las memorias CUDA con las operaciones que se realizan sobre cada una y sus correspondientes tiempos de acceso.

MEMORIA	Acceso	Operación Host	Operación Device	Tiempo Acceso (ciclos)
Registros	Hebra	Ninguna	Read/Write	1
Local	Hebra	Ninguna	Read/Write	400 - 600
Compartida	Bloque	Ninguna	Read/Write	1
Global	Device+Host	Read/Write	Read/Write	400 - 600
Constantes	Device+Host	Read/Write	Read	10 - 100
Texturas	Device+Host	Read/Write	Read	10 - 100

Tabla 4.5 - Relación de Memoria Arquitectura CUDA

Con todo lo visto, es inevitable preguntarse cuándo es interesante usar cada tipo de memoria, ya que dependiendo de la elección, el rendimiento de la aplicación es más o menos óptimo. Para ello existen unas directrices generales:

- **Lecturas por parte del device sin estructura**. En este caso la memoria de constantes es la más oportuna aportando un rendimiento óptimo.
- **Lecturas por parte del device en el que hay un vector estructurado**. En este caso la memoria de texturas es la más oportuna mejorando el rendimiento.
- **Lecturas y escrituras por parte de las hebras de un mismo bloque dentro de un bloque.** En este caso la memoria compartida es la mejor optimizando los rendimientos.
- **Lecturas y escrituras de entradas y resultados tanto por parte del host como del device.** En este caso la memoria global es la más oportuna aportando un buen rendimiento óptimo.

c) Resultados

Una vez analizadas las características principales de las tecnologías de multiprocesamiento que vamos a utilizar en los escenarios mencionados anteriormente, podemos comenzar a mostrar los resultados obtenidos en las pruebas realizadas, de las que obtendremos conclusiones.

Como hemos indicado previamente, nos encontramos ante un problema altamente paralelizable, donde a partir de una cadena de texto generamos su hash un número elevado de veces según el algoritmo criptográfico seleccionado. En nuestra prueba hemos alcanzado los mil millones de hash, para así obtener tiempos válidos ya que con valores pequeños los resultados obtenidos no eran lo suficientemente representativos de la generación de hash, sino más bien de las comunicaciones de nodos y hebras.

```
int num_hash = 1000000000;

for(i=0; i<num_hash; i++){
    calcular_hash(cadena_texto, tamaño_cadena, hash);
}
```

Así en la figura 4.15 se muestra el rendimiento obtenido a partir de los resultados ofrecidos por nuestro *Elastic Clúster* con MPI y el servidor Tesla con CUDA para los algoritmos MD5, SHA1 y LM recogidos en la tablas 4.6, 4.7 y 4.8, y comparados con la ejecución secuencial y Threads empleadas en proyectos anteriores para 1.000.000.000 hashes.

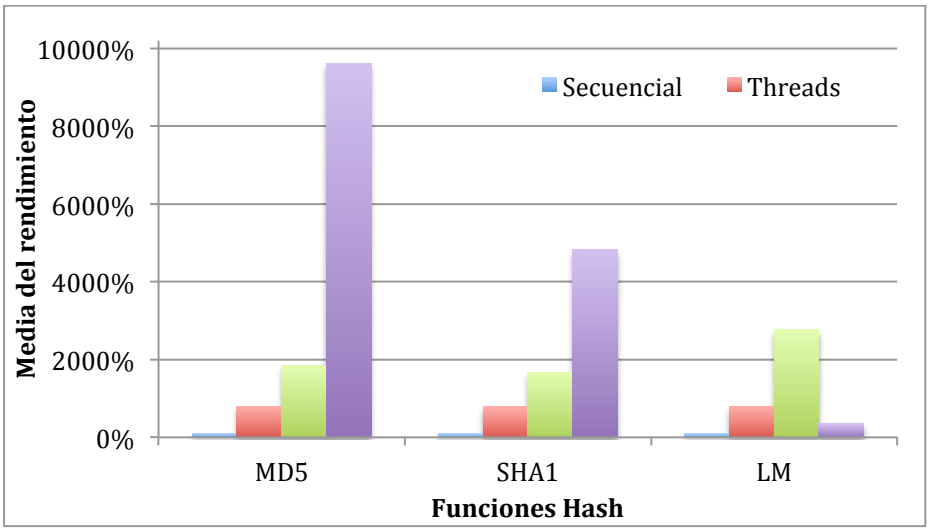

Figura 4.15 - Rendimiento en la generación de hash mediante fuerza bruta

MD5	Secuencial	Threads	Cloud	CUDA
Tiempo (s)	208,429	26,07	11,23	2,168
Hash/seg	4.797.797	38.358.266	89.047.195	461.254.613
%	**100,0%**	**799,5%**	**1856,0%**	**9613,9%**

Tabla 4.6 - Resultados generación hash MD5 mediante fuerza bruta

SHA1	Secuencial	Threads	Cloud	CUDA
Tiempo (s)	233,225	29,46	14,03	4,835
Hash/seg	4.287.705	33.944.331	71.275.837	206.825.233
%	**100,0%**	**791,7%**	**1662,3%**	**4823,7%**

Tabla 4.7 - Resultados generación hash SHA1 mediante fuerza bruta

LM	Secuencial	Threads	Cloud	CUDA
Tiempo (s)	421,356	52,743	15,24	120,76
Hash/seg	2.373.290	18.959.862	65.616.798	8.281.196
%	**100,0%**	**798,9%**	**2764,8%**	**348,9%**

Tabla 4.8 - Resultados generación hash LM mediante fuerza bruta

Si analizamos los resultados anteriores, podemos observar que las dos implementaciones de este proyecto mejoran en gran medida el rendimiento obtenido en otros casos, siendo CUDA el que presenta el mejor rendimiento llegando hasta los 461 millones de hash por segundo para el caso de la función MD5. La solución cloud, aunque para MD5 y SHA1 no alcanza los resultados de CUDA, sí que ofrece un rendimiento

mucho mejor que las versiones implementadas en proyectos anteriores, llegando a ofrecer una media de rendimiento del 1856% para MD5 y 1662% para SHA1 con respecto a la implementación secuencial del programa de generación de hashes. Para la función LM, utiliza el algoritmo de cifrado DES, mucho más complejo y al que se adapta peor CUDA.

Hay que tener en cuenta que tanto para la implementación cloud basada en MPI como la versión CUDA, es necesario un tiempo para iniciar el sistema, por lo que se penaliza de forma importante la generación de un número pequeño de hashes con estas dos metodologías, sin embargo para un número elevado de hashes queda demostrado que éstas son las mejores técnicas.

3.1.2. Obtención del valor a partir del hash.

Como hemos visto en el apartado anterior, las versiones CUDA y MPI utilizadas son las técnicas de multiprocesamiento que mejores resultados han obtenido, cada una en un escenario de trabajo distinto, en un servidor Tesla y en clúster cloud respectivamente.

El siguiente paso consiste en mejorar el ataque de fuerza bruta, por lo que se ha implementado un programa paralelizado para ambas metodologías, denominado "rompe_hash", que a partir de un hash dado, devuelva la cadena que lo originó, así como el tiempo que ha necesitado para obtener dicha cadena.

Para obtener unos resultados válidos y así realizar la comparativa entre el clúster físico y el clúster en el cloud, el objetivo de esta implementación será la búsqueda de un hash MD5 en un espacio de búsqueda amplio, por lo que hemos decidido trabajar con cadenas de caracteres de 5 y 6 caracteres y un alfabeto de 37 valores, por lo que en el caso de contar con 6 caracteres tenemos 2.565.725 combinaciones posibles (37^6). El alfabeto utilizado para nuestro proyecto es el siguiente:

```
charset[37]={'0','1','2','3','4','5','6','7','8','9','a','b','c','d
','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','
u','v','w','x','y','z',' '};
```

Al igual que en el apartado anterior, esta aplicación es altamente paralelizable, por lo que detallamos a continuación como se ha implementado tanto para nuestro *Elastic Clúster* con MPI, como para la versión CUDA ejecutada en el servidor Tesla.

A pesar de que esta prueba está orientada a la ruptura de hashes MD5, su aplicación a otras funciones como SHA1 es sencilla y rápida, ya que sólo hay que adaptar los tamaños de las estructuras de datos utilizadas de acuerdo a la nueva función y utilizar la función correspondiente de *Openssl*.

a) Elastic Clúster - MPI

Como podemos ver en la figura 4.16, el programa de fuerza bruta envía a cada nodo del cluster el hash que se desea "romper". A partir de este momento, cada nodo cuenta con su espacio de búsqueda, y comprueba si cada uno de los valores hash generados corresponde al hash deseado. En el instante en que un nodo encuentre el valor correcto, el proceso debe terminar y devolver la cadena que ha originado ese hash, además del tiempo empleado para ello.

Para poder repartir el trabajo entre todos los nodos, cuando se inicia el programa se calcula el identificador de cada nodo teniendo en cuenta el número de procesadores disponibles a partir de la función *MPI_Comm_rank()*. De esta forma, a partir del identificador, los nodos del cluster se reparten el trabajo de forma proporcional.

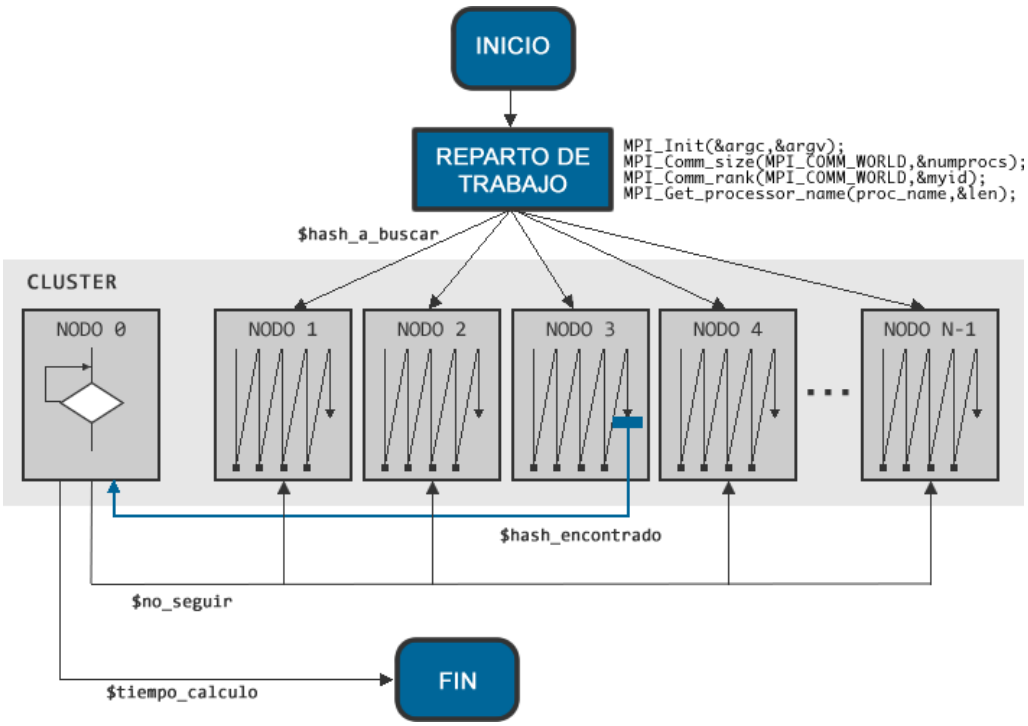

Figura 4.16 - Programa Rompe_Hash MPI

A partir de este identificador se calcula el valor de inicio y final que debe procesar cada nodo. Para cada valor de semilla se calcula la cadena de texto que este valor representa y se calcula el hash de dicha cadena en MD5. Si el hash calculado no coincide con el que buscamos se procesa el siguiente valor de semilla. Si por el contrario, si coincide con el que buscamos, finaliza la ejecución.

Es necesario distinguir el trabajo que deben desempeñar los nodos de nuestro clúster virtual, siendo el pseudocódigo que debe ejecutar cada nodo el siguiente.

```
seguir = 1000;

función buscar(hash, palabra, id_nodo, seguir){
    calcular_espacio_busqueda(id_nodo, v_inicio[], v_fin[]);
    mientras(hash_no_encontrado && no_fin)
    {
        generarCadena(v_inicio, semilla);
        buscarHash(semilla, hash, palabra);
        si(seguir == 0)
              comprobar_estado()
        seguir--;
    }
}
```

Como se puede observar, para evitar que un nodo tenga que esperar a recorrer todo su espacio de búsqueda para enviar o recibir un mensaje de estado (ha finalizado o no, debe continuar o no), se ha implementado que cada 1000 iteraciones de búsqueda, éste compruebe el estado del programa general indicando si lo ha encontrado o por el contrario indique si no lo ha encontrado, y si ha llegado al final de su espacio de búsqueda o por el contrario no lo ha alcanzado, pero debe seguir buscando, a menos que el nodo de control le indique que otro nodo lo ha hecho, por lo que no tiene sentido continuar con la búsqueda.

Para poder sincronizar toda la ejecución, el nodo 0 (nodo de control) es el encargado de recibir las comunicaciones del resto de nodos y actuar en concordancia. Este nodo de control espera a que uno de los nodos envíe un mensaje de que ha encontrado el hash deseado, o bien, si nadie lo encuentra, espera a que todos los nodos envíen sus respectivos mensajes de que no han encontrado el hash indicado. Éste es su pseudocódigo

```
if(id_nodo == 0){
    mientras(hash_no_encontrado && no_fin){
        recibir_mensaje(id_nodo,estado);
        if(estado == hash_encontrado){
              enviar_a_todos(no_seguir);
              mostrar_resultados();
              fin;
        }
        else_if(estado == hash_no_encontrado){
              nodos_finalizado ++;
              if(nodo_finalizados = total_nodos){
                    mostrar_resultados();
                    fin;
              }
        }
    }
}
```

Cuando un nodo envía un mensaje, indicando que ha encontrado el hash deseado, el nodo de control envía al resto de nodos del cluster un mensaje de fin de ejecución, como refleja el pseudocódigo anterior, así como la figura anterior 4.16.

b) GPGPU - CUDA

En este caso, como podemos ver en la Figura 4.17, el programa de fuerza bruta envía a cada *device* el hash que desea romper. En ese momento, cada hilo GPU cuenta con su espacio de búsqueda y comprueba si el valor hash generado para cada palabra corresponde al hash que se quiere "romper". En el momento que un hilo GPU encuentra el valor correcto todo el proceso termina.

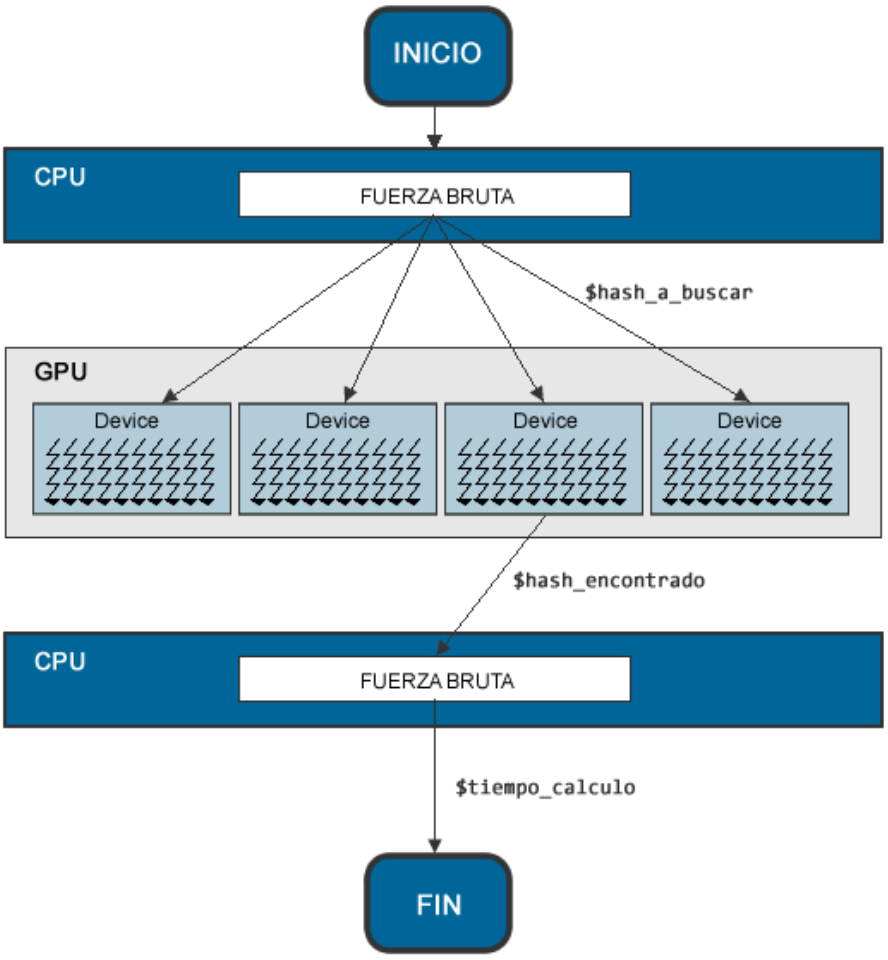

Figura 4.17 - Programa Rompe_Hash CUDA

Como era de esperar, el funcionamiento es idéntico al utilizado con MPI, sólo hay que tener en cuenta ahora las características hardware de los procesadores gráficos.

Una de las ventajas que ofrecen los procesadores gráficos es que disponen de un alto paralelismo de datos a través de la ejecución simultánea de múltiples hebras. Nuestro procesador Tesla T10 cuenta con 30 multiprocesadores, cada uno capaz de ejecutar 32 hebras. Por lo que, como cuenta con cuatro devices, puede ejecutar de forma simultánea hasta 3840 hebras.

En contraposición, con MPI sólo se puede ejecutar una tarea en cada core. Debido a que nuestro clúster virtual cuenta con 14 máquinas virtuales de 2 cores cada una, podemos ejecutar de forma simultánea 28 tareas.

Para repartir el trabajo entre todos los hilos, cuando se inicia un hilo, a éste se le asigna un identificador (`id_tread`), que se calcula teniendo en cuenta el número de *devices*, el número de bloques y el número de hebras por bloque. De esta forma, todos los hilos compartirán el trabajo de una forma proporcional a su identificador.

```
int id_hebra =  device * NUM_BLOQUES * NUM_HEBRAS_BLOQUE +
                blockIdx.x * blockDim.x + threadIdx.x;
```

Cada hilo calcula los valores de inicio y fin que debe procesar en función de su identificador. Así, con cada valor que procesa el hilo se obtiene su cadena de texto correspondiente con el *charset* de 37 caracteres. Para cada cadena o semilla se calcula su hash MD5. Si el hash generado coincide con el hash que estamos buscando, ya hemos encontrado la cadena deseada y el proceso debe terminar. En caso de que no coincida se calcula el siguiente valor, cadena y hash a procesar hasta que llegue al valor de fin de proceso para ese hilo.

```
obtenerMascara(mascara_inicio, mascara_fin, id_hebra);

función_buscar(hash, palabra, id_nodo, seguir){
    mientras(hash_no_encontrado && no_fin_trabajo)
    {
        generarCadena(mascara_inicio,semilla);
        buscarHash(semilla,hash_buscar,palabra);
        continuar=comprobarFin(mascara_inicio,mascara_fin);
              if(continuar==0)//No ha llegado al final
                    renovarMascara(mascara_inicio);
    }
}
```

Se ha utilizado una variable global, *`hash_encontrado`*, que se utiliza para la sincronización de todas los hilos. De esta forma, cuando un hilo encuentra la cadena de texto cuyo hash coincide con el deseado, la variable toma el valor 1, y en este momento, el resto de hilos detectan que ya se ha encontrado el hash y se detienen.

```
función_ buscarHash(semilla,hash_buscar,palabra {
     calcula_Hash(semilla,hash_salida);
     si(hash_salida == hash_buscar)
       hash_encontrado = 1;
}
```

Para obtener el mejor rendimiento del sistema, el programa ha sido ejecutado variando el número de bloques y el número de hebras que un bloque tiene, para obtener los valores más eficientes que deben tomar estas variables. Así, se ha ido incrementando de 32 a 512, que es el número máximo de hebras por bloque. Si utilizamos más de 512 hebras se requiere aumentar el número de bloques.

La figura 4.18 muestra el número de comprobaciones por segundo que el programa ha realizado variando el número de bloques y el número de hebras. Como se puede observar, para esta aplicación trabajando con cadenas de 5 caracteres, el máximo rendimiento de un device casi alcanza los 6 millones de comprobaciones por segundo, que se obtiene usando 4 bloques. Como nuestro sistema Tesla tiene 4 devices, el máximo rendimiento está próximo a los 24 millones de comprobaciones por segundo.

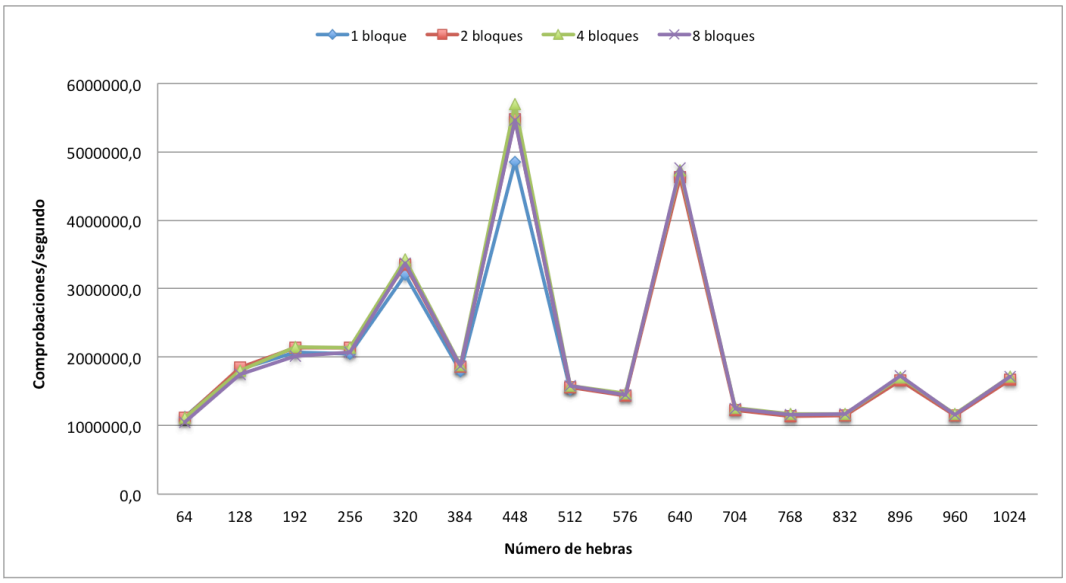

Figura 4.18 - Rendimiento de un device

Por lo tanto utilizamos este valor de bloques y hebras por bloque para nuestra configuración del entorno GPGPU, el cual es comparado con el *elastic clúster*.

La ventaja de utilizar GPU para realizar ataques de fuerza bruta es el gran paralelismo de datos que tenemos, ya que cada hebra tiene un espacio de búsqueda muy pequeño. En nuestro caso, podemos romper contraseñas débiles, ya que estamos hablando de un alfabeto alfanumérico de 37 caracteres, en pocos segundos.

c) Resultados

Una vez detallado el funcionamiento de las dos versiones multiprocesadas del programa "rompe_hash", se ha realizado para ambos casos diferentes ejecuciones utilizando las mismas cadenas de texto, teniendo en cuenta el mejor, el peor e intermedio de los casos que se nos puede presentar, para así obtener valores representativos. Las

cadenas utilizadas en ambos escenarios, tanto para 5 como 6 caracteres, son presentadas en las tablas 4.9 y 4.10 respectivamente.

Mejor	Casos intermedios								Peor
00000	3r46b	6abcd	9bcz4	dm13x	h1234	mwni3	tzwxy	x4ge2	*****

Tabla 4.9 - Cadenas 5 caracteres utilizadas en "rompe_hash"

Mejor	Casos intermedios								Peor
000000	3r46b2	6abcda	9bcz4s	dm13x5	h12345	mwni3g	tzwxyv	x4ge2a	******

Tabla 4.10 - Cadenas 6 caracteres utilizadas en "rompe_hash"

En el peor de los casos se ha escogido la cadena formada por asteriscos, pero sería válida cualquier cadena que no esté definida en el alfabeto de caracteres utilizado, ya que así requiere a todos los nodos o hebras en ejecución recorrer por completo su espacio de búsqueda y finalizar sin que ningún proceso encuentre solución.

Destacado esto, los resultados obtenidos en los dos entornos de trabajo utilizados hasta el momento tanto para cadenas de 5 caracteres como para cadenas de 6 caracteres se muestran en la tabla 4.11.

	5 caracteres		6 caracteres	
	Cloud	**CUDA**	**Cloud**	**CUDA**
Tiempo promedio	4,96 s.	3,64 s.	194,44 s	126,40 s.
Tiempo mínimo	0,05 s	0,71 s	0,06 s .	1,13 s.
Tiempo máximo	9,83 s	10,94 s	409,4 s.	405,48 s.
Desviación típica	0,78	0,91	0,79	1.02

Tabla 4.11 - Rendimiento búsqueda hash MD5. Cloud vs CUDA

En este caso, el tiempo mínimo corresponde al mejor de los casos, es decir la primera cadena que puede encontrar el primer nodo o hebra, mientras que el tiempo máximo corresponde al peor de los casos, es decir, que ha finalizado el proceso sin que ningún nodo o hebra haya encontrado con éxito la cadena.

De los resultados obtenidos hay que destacar que con esta aplicación, es posible "romper" una contraseña alfanumérica cifrada de cinco caracteres en menos de cuatro segundos, y alrededor de dos minutos para contraseñas de seis caracteres.

Como se puede observar, en el mejor de los casos, la solución cloud ofrece un rendimiento mejor que CUDA, esto es debido a que en MPI las comunicaciones penalizan

mucho menos que en CUDA. En el peor de los casos, las dos propuestas ofrecen resultados similares.

Sin embargo, es CUDA la solución que mejor se adapta en los diferentes casos planteados, ya que presenta unos tiempos más estables como demuestra la desviación típica, que para CUDA ofrece unos resultados más próximos a 1.

3.2. Tablas Rainbow.

La utilización de Tablas *Rainbow* permite obtener, de una forma mucho más rápida que con ataques de fuerza bruta, el valor de un determinado hash, por lo que es uno de los tipos de ataques más utilizados en la actualidad.

Estas tablas se generan previamente para obtener una contraseña con una determinada longitud y conjunto de caracteres. El problema y dificultad para obtener las contraseñas reside en disponer de las tablas suficientemente grandes para romper cualquier tipo de contraseña. Para entender mejor el problema, en la tabla 4.12 se pueden ver las magnitudes, para diferentes conjuntos de caracteres, el tamaño en disco que ocupa y el tiempo de ejecución necesario en generar una tabla con el algoritmo hash MD5 de clave comprendida entre 1 y 7 caracteres con una probabilidad de éxito de 99,99 %.

Alfabeto de caracteres	Tamaño en disco	Tiempo de ejecución
0123456789	859.375 Kb	7,357 segundos
ABCDEFGHIJKLMNOPQRSTUVWXYZ	610,35 Mb	14,86 horas
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789	5,96 Gb	6,19 días
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 !@#$%^&*()-_+=	59,6 Gb	61,9 días
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 !@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/	521,54 Gb	541 días

Tabla 4.12 – Comparativa generación de Tablas Rainbow MD5

Lógicamente, cuanto mayor sea el conjunto de caracteres mayor es la probabilidad de que la contraseña pertenezca a dicho conjunto y por lo tanto de encontrar la solución.

Para la utilización de las Tablas Rainbow en nuestro proyecto se ha utilizado el proyecto *RainbowCrack*. El proceso de ruptura de un hash consta de 3 fases de las cuales las dos primeras sólo necesitan realizarse una vez, mientras que la última es realizada tantas veces como hashes se deseen romper.

1. **Generación de la Tabla Rainbow.** En esta fase se crea la Tabla Raibow propiamente dicha guardándola en un fichero de texto con el formato *.rt*.
2. **Ordenación de Tabla Rainbow.** En esta fase se ordena la tabla guardada en el fichero de texto. Esto es vital para la eficiencia de las Tablas Rainbow y por tanto obligatorio para el buen funcionamiento del algoritmo.
3. **Ruptura de los hash.** En esta fase se buscan los hash deseados en la tabla. En el caso de ser encontrados nos devuelve el resultado de la clave original.

Gracias a la combinación de las dos primeras fases se genera una tabla *Rainbow* ya ordenada, siendo la tercera la que combina el hash de entrada con la tabla para producir como solución la clave original.

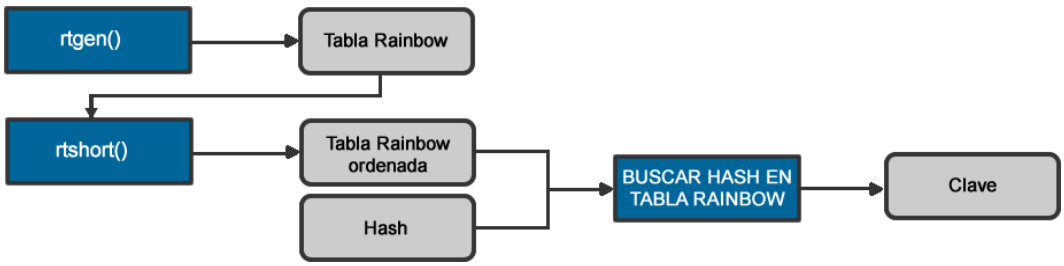

Figura 4.19 - Obtención de la clave con tablas Rainbow

3.2.1. Estructura y funcionamiento

Las Tablas Rainbow tienen su predecesor en las conocidas Cadenas de Hash, inventadas por Hellman Martin. Estas cadenas se basan en la utilización de dos funciones básicas: función reductora y función de hash.

- **Función Reductora (R).** Es una función encargada de convertir un hash a una cadena de menor tamaño.
- **Función Hash (H).** Es una función encargada de generar los hash para cada texto plano.

Así, mediante el uso combinado de ambas funciones se generan Cadenas de Hash de longitud *n* como las de la figura 4.20.

Figura 4.20 - Generación de cadenas Hash

Intuitivamente se puede pensar que para hacer una tabla de tamaño *n* x *m*, hay que repetir el proceso m veces. Posteriormente son guardados el último y el primer valor de cada cadena en un archivo como se puede observar en las figuras 4.21 y 4.22.

Este último paso se realiza porque guardar el total de los hash generados es inviable. Gracias al uso de las funciones reductoras es fácil la reconstrucción de la cadena a partir del primer valor hasta el último.

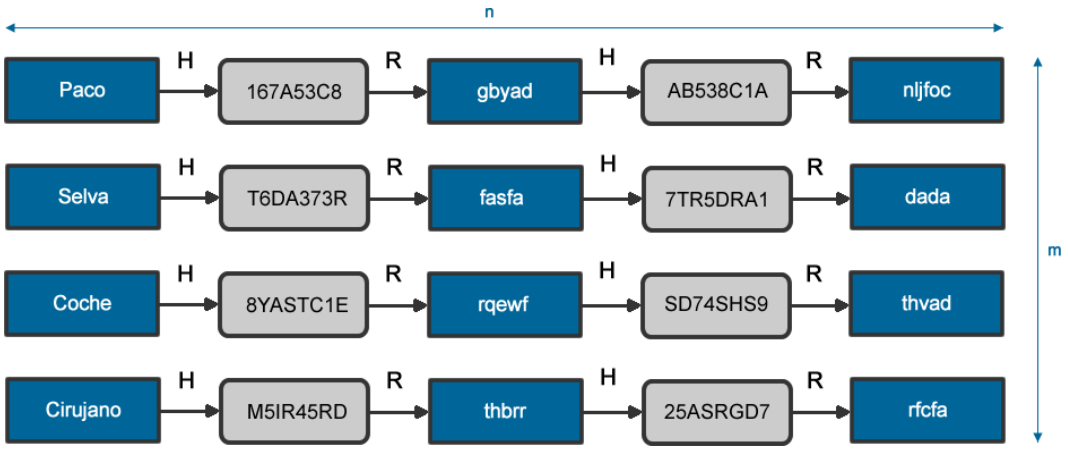

Figura 4.21 - Generación tablas Hash

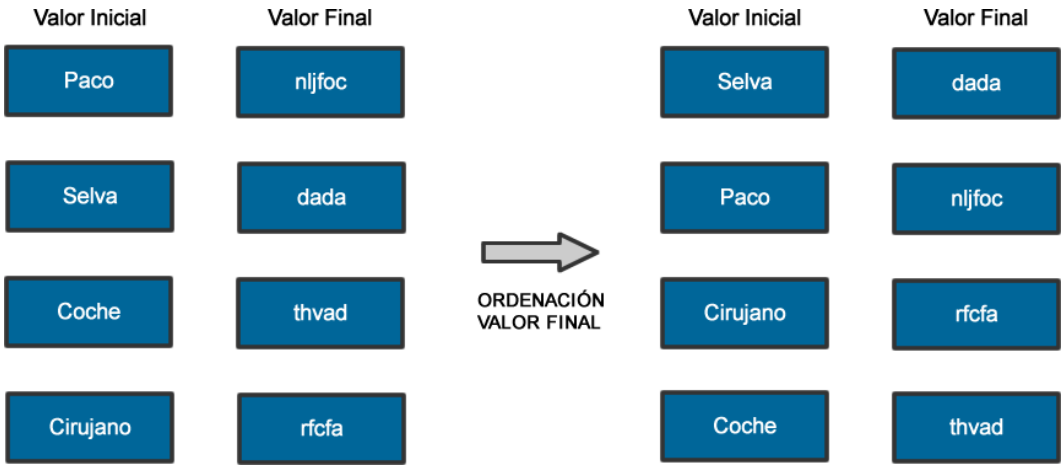

Figura 4.22 – Ordenación de la tabla por el valor final

La ordenación de la tabla por el valor final es imprescindible ya que gracias a ella se pueden realizar búsquedas binarias que optimicen enormemente el trabajo de ruptura de hash. Es importante recordar que todo este proceso es necesario realizarlo una única vez, de forma que la misma tabla puede ser usada para romper múltiples hash.

Una vez tenemos generada y ordenada la tabla rainbow, para romper un hash tenemos que realizar las mismas operaciones con el hash dado para ir creando las cadenas de hash, hasta encontrar un valor final que coincida con alguno de los valores finales de la tabla. Por ejemplo dado el hash “8YASTC1E”, obtenemos el valor final “thvad”, que en nuestra tabla ordenada se corresponde con “Coche”.

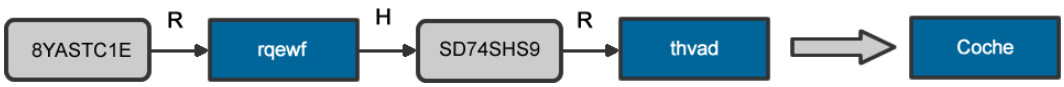

Figura 4.23 - Búsqueda de un hash en un tabla Rainbow

3.2.2. Implementación

Para la implementación del algoritmo de generación de tablas rainbow nos hemos basado en las implementaciones del paquete *rainbowcrack 1.2* del proyecto Rainbow Crack.

Básicamente, una tabla *Rainbow* está compuesta por un conjunto de vectores de datos. La generación de esta tabla a partir de la función *rtGen* en forma secuencial viene representada en la figura 4.24.

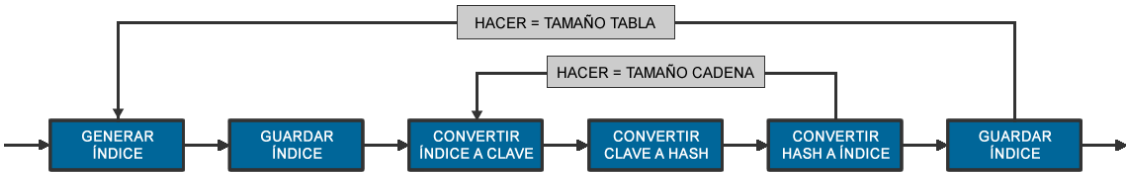

Figura 4.24 - Generación de Tablas Rainbow mediante rtGen Secuencial

Las distintas vueltas producidas por el bucle externo son totalmente independientes apareciendo solo dependencias dentro de la generación de la propia cadena. Este dato nos hace reflexionar sobre la ingente cantidad de oportunidades que nos brinda el algoritmo para ser implementado y optimizado mediante el uso de tecnologías paralelas como las vistas en el presente proyecto.

En la figura 4.25, se muestra una relación porcentual del coste computacional asociado a cada uno de los métodos prioritarios del algoritmo rtGen, que son:

- **IndexToPlain**. Método encargado de convertir un índice numérico a texto plano.
- **PlainToHash**. Método encargado de calcular el hash de un texto plano.
- **HashToIndex**. Método encargado de convertir un hash a un valor tipo numérico. Hace las funciones de una función reductora.
- **Otros**. Se refiere a las múltiples operaciones E/S y comprobaciones realizadas por *rtGen*.

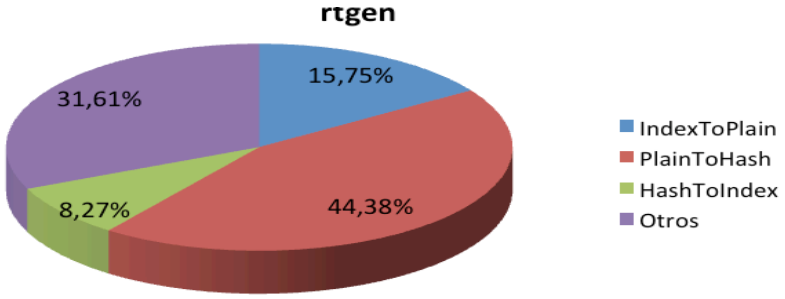

Figura 4.25 – Relación del coste computacional del algoritmo rtGen.

Una vez comprendido mejor el funcionamiento interno de *rtGen*, el objetivo de la paralelización reside en multiprocesar el bucle interno del programa compuesto por *IndexToPLain, PlainToHash y HashToIndex*.

a) Elastic Cluster - MPI

La utilización de múltiples procesos para la generación de tablas rainbow encuentra su fundamento en el alto nivel de paralelismo existente en todo el proceso de su generación.

De tal modo, es posible aprovechar todo el potencial de nuestro clúster virtual con un mínimo desaprovechamiento de recursos, optimizando así todo el proceso. El funcionamiento básico de *rtGen MPI* consiste en la realización de los siguientes pasos en cada uno de los catorce nodos del cluster, que recordamos cuenta cada uno con dos procesadores (disponiendo en total de 28 procesadores):

```
MPI::Init(argc, argv);
num_procs = MPI::COMM_WORLD.Get_size();
id_proc = MPI::COMM_WORLD.Get_rank();

for(i=0, i<tam_tabla; i++)
  nIndexInicio[i]= num_aleatorio; // Valores iniciales

tam_tabla_proc = tam_tabla / num_procs; // Reparto el trabajo

función_calcular_valores(nIndexInicio, id_proc, num_procs){
  for (i = 0; i < tam_tabla_proc; i++){
    for (posicion = 0; posicion < tam_cadena - 1; posicion++){
      IndexToPlain(); // Convierto índice a texto
      PlainToHash();  // Convierto texto a hash
      HashToIndex();  // Convierto hash a indice
 }
}
for (i = 0; i < tam_tabla; i++){
      fwrite(&nIndexInicio[i], file);
      fwrite(&nIndexFin[i], file);
}
```

Cuya representación viene dada en la figuras 4.26 y 4.27.

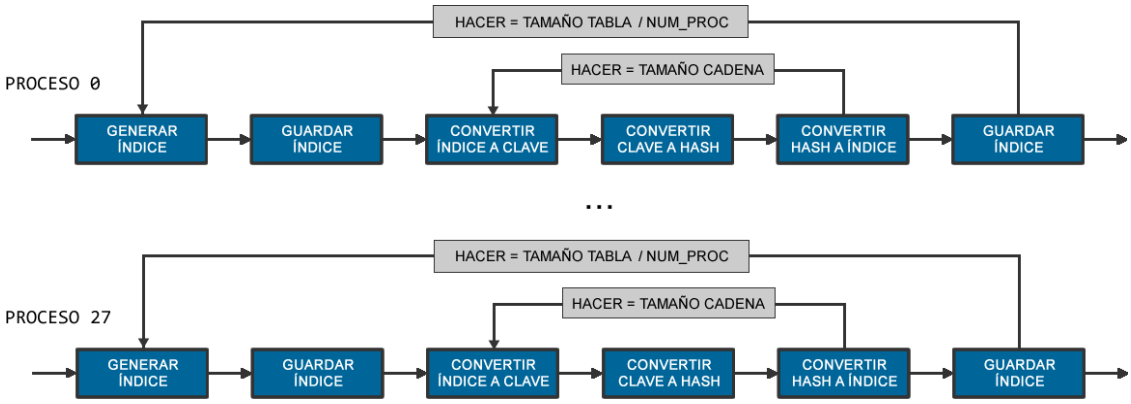

Figura 4.26 – Proceso Generación tablas rainbow mediante rtGen MPI

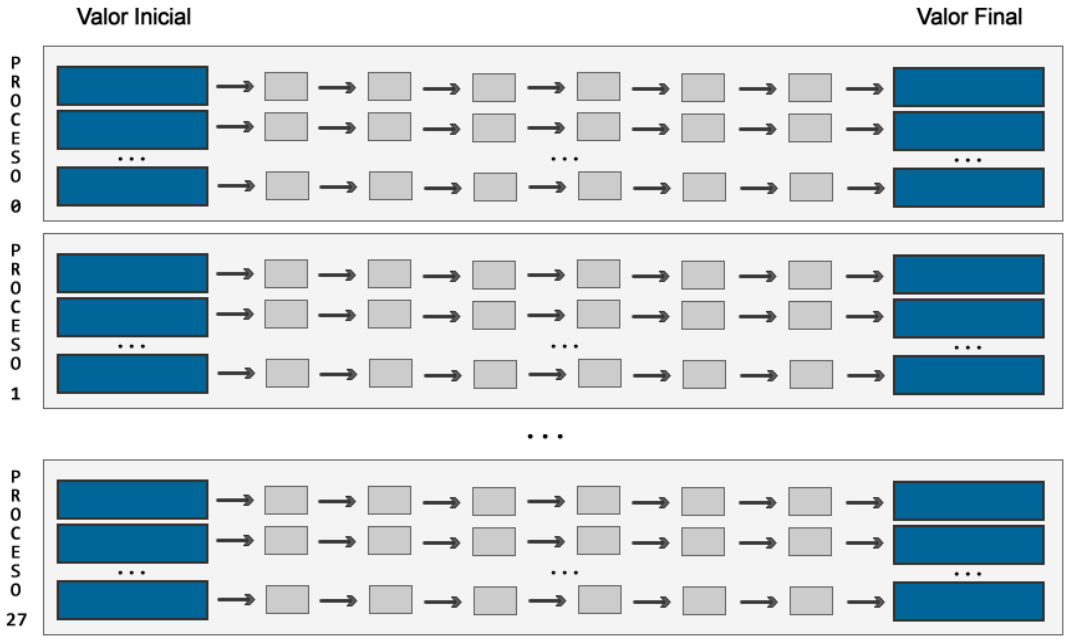

Figura 4.27 - Esquema generación tablas rainbow mediante rtGen MPI

b) GPGPU - CUDA

La implementación del algoritmo *rtGen* CUDA presentada en este proyecto también está basada en el paquete *rainbowcrack* 1.2 del proyecto Rainbow Crack. Para la realización de dicha implementación ha sido necesario realizar importantes cambios estructurales al programa que se adecúen al funcionamiento del lenguaje CUDA.

El funcionamiento básico de *rtGen* CUDA divide su ejecución en dos partes. La primera tiene lugar en el host donde se define un vector de cadenas de caracteres del tamaño de la tabla y se generan los índices aleatorios, y la segunda para los *devices* en forma de hebras de GPU, que reciben el vector con los índices y son las encargadas de la generación de las cadenas de hash. Una vez que se hayan generado los nuevos índices se envían al host que escribirá en el fichero de la tabla los valores iniciales y finales.

Así, por parte del host, se realizan los siguiente pasos:

```
nIndexInicio[tam_tabla], nIndexFin[tam_tabla];
FILE* file = fopen(szFileName, "a+b");

//Genero los valores iniciales
for(i=0, i<tam_tabla; i++)
  nIndexInicio[i]= num_aleatorio;

//Copio el vector al device
cudaMemcpy(nIndexInicio, tam_tabla, cudaMemcpyHostToDevice);

//Calculo la tabla en el device
calcular_valores <<<num_bloq,num_hebras_bloq>>>( nIndexInicio);

//Recibo el resultado del device
cudaMemcpy(nIndexFin, tam_tabla, cudaMemcpyDeviceToHost);

//Guardo los datos en el fichero
for (i = 0; i < tam_tabla; i++){
       fwrite(&nIndexInicio[i], file);
       fwrite(&nIndexFin[i], file);
}
```

Mientras que el funcionamiento para los *device,* una vez recibido el vector de índices, consiste en la realización de los siguientes pasos

```
int id_hebra = blockIdx.x * blockDim.x + threadIdx.x;
int posicion;

for(i= id_hebra; i < tam_tabla; i=i+num_bloq*num_hebras_bloq){
    for (posicion = 0; posicion < tam_cadena - 1; posicion++){
       IndexToPlain(); // Convierto índice a texto
       PlainToHash();  // Convierto texto a hash
       HashToIndex();  // Convierto hash a indice
    }
}
```

Y al igual que en el apartado anterior, el esquema general del algoritmo, tanto para host como *devices*, viene representado en la figura 4.28.

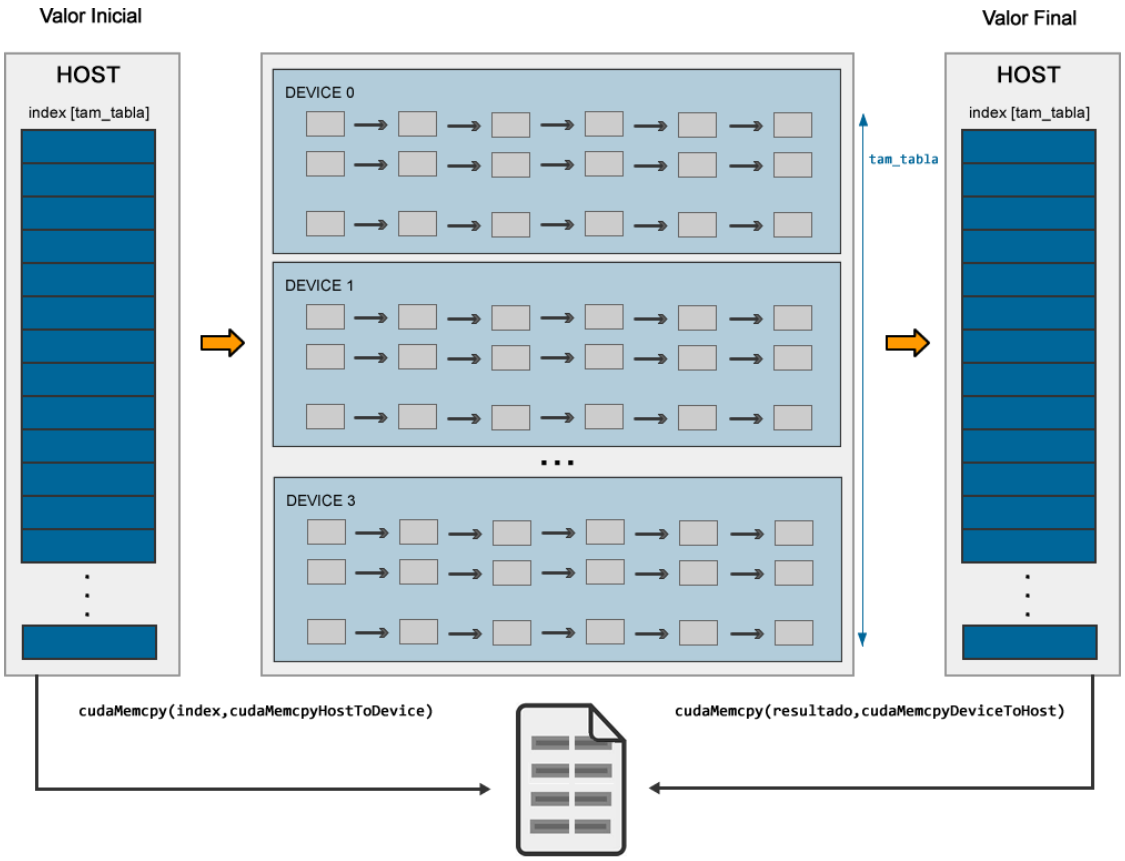

Figura 4.28 – Generación de tablas rainbow mediante rtGen CUDA

c) Resultados

Una vez vistas las diferentes implementaciones *rtGen*, se ha realizado una serie de pruebas en la generación de una tabla de 10.000.000 cadenas, con tamaño de cadena 1.000 (10.000.000 x 1.000), de las que hemos obtenido los resultados mostrados en la figura 4.29 y tablas 4.13, 4.14 y 4.15 siguientes, donde se han comparado con los resultados de proyectos anteriores con respecto a la ejecución secuencial y con *threads*.

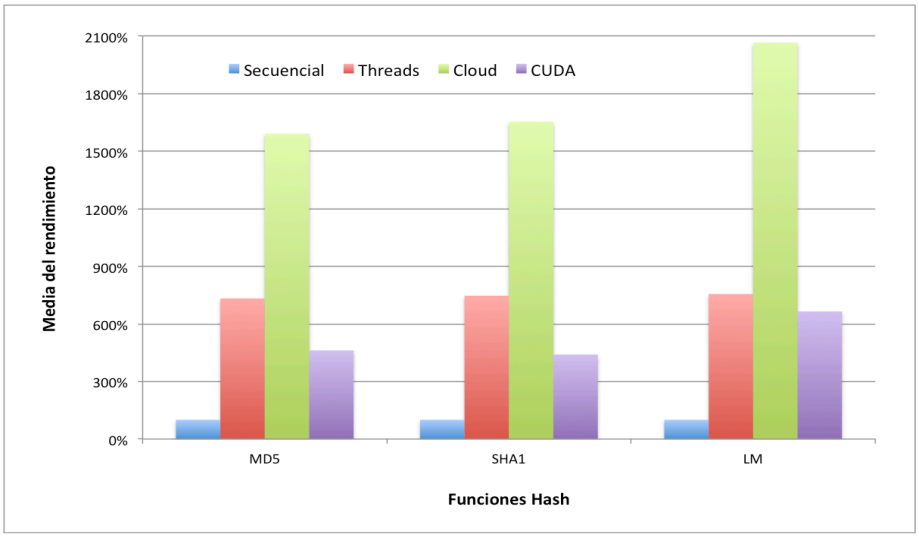

Figura 4.29 - Comparación del rendimiento en la generación de tablas rainbow

MD5	Secuencial	Threads	Cloud	CUDA
Tiempo (s)	2.822,64	385,32	177,392	611,44
Hash/seg	3.542.783	25.952.455	56.372.328	16.354.834
%	**100,0%**	**732,5%**	**1591,2%**	**461,6%**

Tabla 4.13 - Resultados generación tablas rainbow MD5

SHA1	Secuencial	Threads	Cloud	CUDA
Tiempo (s)	3.145,17	421,094	190,225	714,568
Hash/seg	3.179.478	23.747.667	52.569.326	13.994.455
%	**100,0%**	**746,9%**	**1653,4%**	**440,1%**

Tabla 4.14 - Resultados generación tablas rainbow SHA1

LM	Secuencial	Threads	Cloud	CUDA
Tiempo (s)	4.751,68	628,872	229,999	714,528
Hash/seg	2.104.519	15.901.487	43.478.450	13.995.248
%	**100,0%**	**755,6%**	**2066,0%**	**665,0%**

Tabla 4.15 - Resultados generación tablas rainbow LM

Si analizamos estos resultados podemos observar que sólo la implementación en el *Elastic Clúster* mejora en gran medida el rendimiento obtenido en proyectos anteriores, sobrepasando nuestro clúster virtual los 56 millones de hashes por segundo para el caso de la función MD5. En cambio, CUDA ofrece un rendimiento muy por debajo de la versión con hebras de CPU, salvo en la función LM, donde se acerca a los resultados obtenidos a la versión Threads.

Para la versión cloud el mejor rendimiento obtenido con respecto a la versión secuencial es para la función hash LM, en la cual se alcanza un rendimiento del 2066% y más de 43 millones de hashes por segundo.

Realizando un promedio de los resultados obtenidos para nuestros dos entornos de trabajo, el Elastic Cluster ofrece un rendimiento medio del 1770%, mientras que CUDA ofrece un rendimiento medio del 522%, ambos con respecto a la versión secuencial.

CUDA, al contrario que ocurría en las implementaciones de fuerza bruta, se presenta como la implementación menos eficiente debido a las limitaciones presentadas por este lenguaje en los casos en los que es tiene lugar multitud de comunicaciones.

CONCLUSIONES

La investigación con clústeres de alto rendimiento (HPC) mantiene una gran tradición por la integración de tendencias y nuevas tecnologías que permite obtener un mayor poder computacional en la resolución de problemas con elevada carga de computación y complejidad en temas relacionados con la ciencia e ingeniería.

Este trabajo se ha centrado en dos metodologías de trabajo que se han combinado para ofrecer una solución de altas prestaciones con un bajo coste, mejorando los puntos fuertes de cada una de ellas: tecnología clúster y cloud computing.

Este entorno de computación híbrido que se ha implementado para el desarrollo del proyecto es denominado *Elastic Cluster*, formando una infraestructura dinámica para la administración de servicios, que incluye servicios a nivel de clúster como la administración de cargas de trabajo, así como módulos inteligentes que actúan como puente entre los servicios a nivel de clúster y los servicios de administración de la infraestructura dinámica.

Este modelo es autoajustable, en cuanto recursos, ya que es posible adaptar la carga de trabajo presente y alcanzar así los requisitos de tiempo para las aplicaciones, ofreciendo una mejor disponibilidad y en definitiva un mejor rendimiento. A diferencia de los clústeres HPC tradicionales, en los que los nodos que trabajan son aprovisionados de forma estática, este clúster virtual permite reservar de forma dinámica recursos adicionales pertenecientes a clouds privados o públicos, presentando soporte para la integración con servicios de alto nivel y su asociación con otros clústeres.

Desafortunadamente, no todo son ventajas, el despliegue de una infraestructura cloud conlleva de forma inherente una pérdida de rendimiento. Para cuantificarla, se ha realizado una comparativa entre un clúster virtual y un clúster físico con el mismo equipamiento, aplicando una serie de *benchmarks* que analizan el rendimiento del sistema.

Tras realizar dichas pruebas y con los resultados obtenidos es posible concluir que el despliegue de clúster en un entorno cloud penaliza el rendimiento del sistema

entre un 8,07% y un 23,59% a lo sumo, dependiendo del grado de paralelización de la aplicación.

Pero esta pérdida puede ser asumida sin problemas teniendo en cuenta este grado de paralelización ya ésta tecnología resulta ideal para la puesta en marcha de clústeres de alto rendimiento de bajo coste, adaptables a las necesidades individuales de cada proyecto.

Tomada esta decisión, de ha decidido comprobar el rendimiento que el clúster virtual puede ofrecer ante un problema de elevada carga computacional, el criptoanálisis de funciones hash, y comparando los resultados con otro entorno de trabajo, estudiado en proyectos fin de carrera anteriores, el uso de procesadores gráficos para aplicaciones de propósito general, conocidos como GPGPU.

Las pruebas realizadas se han basado en las dos técnicas más utilizadas en la búsqueda de las vulnerabilidades de criptosistemas: los ataques de fuerza bruta y la generación de tablas *Rainbow*.

La primera prueba consiste en la generación de hashes a partir de una cadena de texto según el algoritmo criptográfico seleccionado, y con los resultados obtenidos podemos concluir que las dos implementaciones de este proyecto mejoran en gran medida el rendimiento obtenido en otros casos, siendo CUDA el que ofrece el mejor rendimiento llegando hasta los 461 millones de hash por segundo para el caso de la función MD5. La solución cloud, aunque para MD5 y SHA1 no alcanza los resultados de CUDA, sí que ofrece un rendimiento mucho mejor que las versiones implementadas en proyectos anteriores, llegando a ofrecer una media de rendimiento del 1856% para MD5 y 1662% para SHA1 con respecto a la implementación secuencial del programa de generación de hashes.

Hay que tener en cuenta que tanto para la implementación cloud basada en MPI como la versión CUDA, es necesario un tiempo para iniciar el sistema, por lo que se penaliza de forma importante la generación de un número pequeño de hashes con estas dos metodologías, sin embargo para un número elevado de hashes queda demostrado que éstas son las mejores técnicas.

El objetivo de la segunda prueba de rendimiento realizada es que dado un hash, devuelva la cadena que lo originó, así como el tiempo que ha necesitado para obtener dicha cadena.

Con la implementación de este proyecto y los resultados obtenidos se ha demostrado que es posible “romper” una contraseña alfanumérica cifrada de cinco caracteres en menos de cuatro segundos, y en el caso de seis caracteres, son necesarios unos dos minutos, siendo CUDA la solución que ofrece unos resultados más estables, aunque para el mejor de los casos propuestos para esta prueba la solución CLOUD es la que mejor resultados ofrece, ya que en MPI las comunicaciones penalizan mucho menos que en CUDA.

Y por último, la tercera prueba realizada para comparar el rendimiento del clúster virtual con la solución GPGPU, es la generación de tablas Rainbow, que representa la mayor parte de los costes pertenecientes al uso de este tipo de tablas. En este caso el multiprocesamiento vuelve a ser la solución en la mejora del rendimiento del sistema.

Realizando un promedio de los resultados obtenidos para los dos entornos de trabajo, el Elastic Cluster ofrece un rendimiento medio del 1770%, mientras que CUDA ofrece un rendimiento medio del 522%, ambos con respecto a la versión secuencial, mejorando en gran medida los resultados obtenidos en proyectos anteriores. Pudiendo concluir que CUDA, al contrario que ocurría en las implementaciones de fuerza bruta, se presenta como la implementación menos eficiente debido a las limitaciones presentadas por este lenguaje en los casos en los que es tiene lugar multitud de comunicaciones.

LÍNEAS DE TRABAJO FUTURO

El clúster virtual o *Elastic Cluster* que se ha abarcado en este proyecto proporciona una herramienta que ofrece un gran poder de procesamiento que puede ser empleada para resolver problemas que requieran de una elevada carga computacional.

Una posible línea de trabajo futuro que pueda aprovechar los resultados de este proyecto es la realización de un estudio y/o implementación de un sistema que permita la falsificación de certificados de comunicaciones seguras *https* (X.509), de esta forma se podrá interceptar una comunicación cifrada en tiempo real y monitorizarla para protegerla de ataques externos.

Un certificado de seguridad *https* está compuesto por una serie de datos que identifica el servidor, como pude ser emisor, nombre de dominio, etc., que son firmados por una autoridad de certificación. Se utiliza un algoritmo de firma digital para asegurar la integridad del certificado digital, que en el caso de los certificados X.509 es la función hash MD5.

Para auditar la comunicación cifrada es necesario que se genere un certificado en tiempo real cuya firma digital coincida con la del original. En este sentido, Arien Lenstra demostró la creación de dos certificados X.509 con diferentes llaves públicas y el mismo valor hash MD5. De esta forma, el objetivo es poder modificar un certificado original y encontrar una colisión que permita que el nuevo certificado tenga el mismo valor hash.

El problema de esta operación es que el sistema necesita ir calculando la firma digital de los certificados que va generando hasta conseguir uno en el que coincida la firma digital con el certificado original. Este problema aunque es fácil de realizar resulta computacionalmente inviable con equipos normales y márgenes de tiempo válidos.

Dejando esta posibilidad abierta, otra línea de trabajo futuro puede consistir en el estudio e implementación de un clúster de alto rendimiento basado en el paradigma cloud computing basándose en una arquitectura de nube híbrida con algún servicio de nube pública que ofrezca algún proveedor de servicios, como puede ser Amazon, que permitirá ampliar el poder computacional que en este proyecto está limitado al equipamiento físico disponible.

De esta forma, por un bajo coste, se puede aprovechar las soluciones implementadas en este trabajo, para realizar un estudio comparativo de rendimientos.

ANEXO I. PUESTA EN MARCHA DEL CLOUD

En este anexo se detallan los pasos seguidos para el despliegue de la arquitectura cloud del clúster virtual que ha sido utilizado a lo largo del proyecto.

La solución utilizada a la hora de poner en marcha la tecnología cloud computing es ***Ubuntu Enterprise Cloud*** (UEC). Recordamos que UEC esta basada en la solución de computación en nube Eucalyptus, una de las más importante para la comunidad open source.

A continuación se describe los pasos necesarios para desplegar UEC con una máquina sirviendo como controlador (cloud controller, cluster controller, storage controller, walrus) y varias máquinas como nodos (node controller).

I.1 REQUISITOS DEL SISTEMA

En la tabla I.1 puede ver los requisitos hardware mínimos y recomendados para la implementación de la nube de Ubuntu, tanto en el equipo controlador como en los nodos del clúster.

	Controlador de la Nube		**Nodos del Clúster**	
Hardware	**Mínimo**	**Recomendado**	**Mínimo**	**Recomendado**
CPU	1Ghz	2x2Ghz	Extensiones VT	VT, 64-bit, Multicore
Memoria	2Gb	4Gb	1Gb	4Gb
Disco Duro	40GB	200GB	40GB	100GB
Red	100Mbps	1000Mbps	100Mbps	1000Mbps

Tabla I.1 - Requisitos para la instalación de Ubuntu Enterprise Cloud

I.2 INSTALACIÓN DEL CONTROLADOR

En esta primera fase debe instalar los diferentes componentes a incluir en el equipo controlador del clúster. Comenzamos descargando la imagen ISO de Ubuntu Server desde su web oficial: *http://www.ubuntu.com/business/server/overview.* Para realizar la instalación debe seguir los siguientes pasos:

1. Tras seleccionar el idioma para el instalador, se muestra el menú de arranque (véase Figura I.1)

Figura I.1 – Pantalla de selección de instalación para Ubuntu Server.

2. Seleccione la opción ***Instalar Ubuntu Enterprise Cloud***. Como puede ver los primeros pasos de la instalación son los mismos que para *Ubuntu Server*: selección del idioma, distribución de teclado y nombre del equipo.

3. Después aparece la pantalla que vemos en la Figura I.2, en la cual se debe introducir la **dirección de red del *controlador*** en caso de que ya exista en la red y no haya sido detectado. Si está instalando por primera vez un controlador en la red, se deba dejar el campo en blanco y pulse el botón *Continuar*.

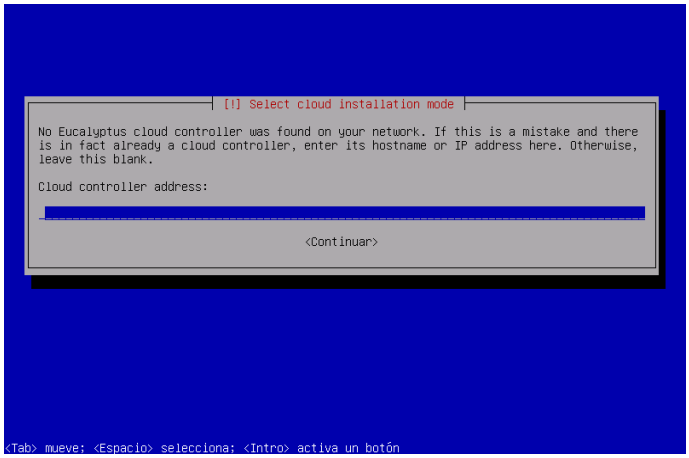

Figura I.2 - Dirección de red del Cloud Controller.

4. A continuación se tiene que seleccionar los componentes que desea instalar en el equipo. Al tratarse del servidor *controlador*, marque las casillas correspondientes a *Cloud Controller, Walrus Controller, Cluster Controler* y *Storage Controller*.

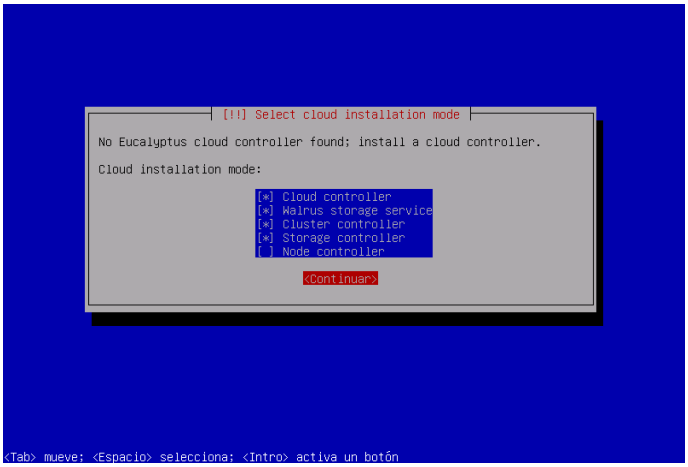

Figura I.3 - Selección de los componentes a instalar en el controlador.

5. En los siguientes pasos debe seleccionar la zona horaria de nuestra instalación y configurar el particionamiento del equipo. Cuando éste haya finalizado se instala el sistema base y **configura la cuenta del usuario administrador** (nombre de usuario y contraseña).

6. Tras configurar *aptitude* y las actualizaciones automáticas del sistema operativo comienza la instalación de programas. Es en este momento cuando se debe escribir el **nombre del clúster**, por ejemplo *cluster1*, y el **rango de direcciones públicas que pueden ser asignadas a las instancias de las máquinas virtuales** en la red local, que en el proyecto se han definido como *192.168.8.200-192.168.8.250* (figuras I.4 e I.5).

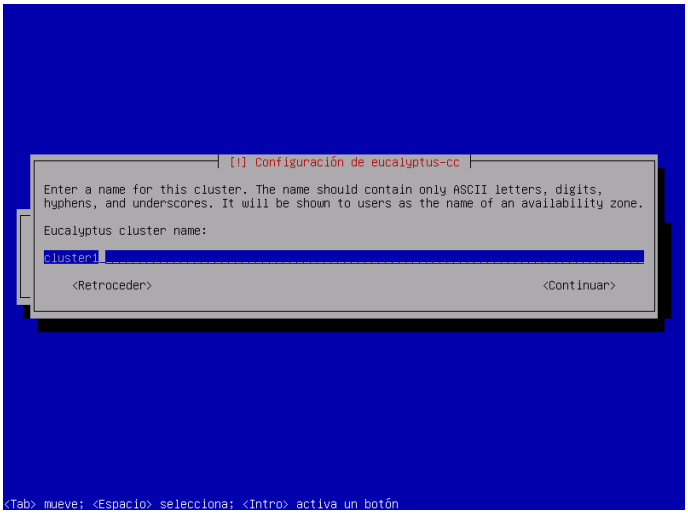

Figura I.4 - Introducción del nombre del clúster.

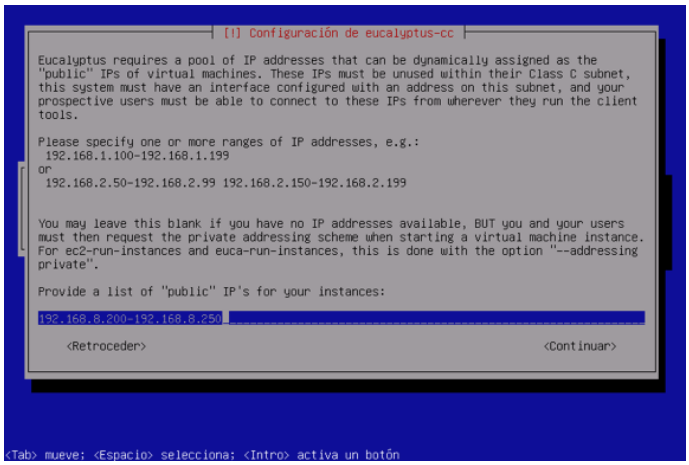

Figura I.5 - Introducción del rango de direcciones públicas en la red local para las instancias.

7. Se inicia la instalación de programas, el gestor de arranque *GRUB* y una vez finalizado el proceso de instalación se debe extraer la imagen de instalación e iniciar de nuevo el sistema con el usuario anteriormente definido.

Como primeras tareas es recomendable establecer la contraseña del usuario *root* mediante el comando *passwd* e instalar, si así lo cree conveniente, el entorno gráfico

```
$ sudo passwd root
$ sudo apt-get install Ubuntu-desktop
```

Para probar que el controlador ha sido instalado correctamente puede acceder a la interfaz web administrativa del mismo, escribiendo en el navegador la siguiente dirección:

```
$ https://<direccion_ip_cloud_controller>:8443/
```

NOTA: Si no conoce la dirección de red del controlador, puede ejecutar el comando `ifconfig`

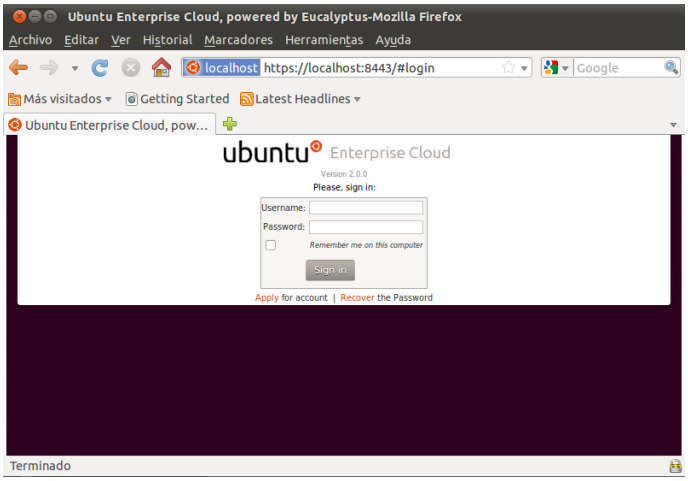

Figura I.6 - Acceso a la interfaz web administrativa de UEC.

Para iniciar sesión por primera vez debe hacerlo con el nombre de usuario *admin* y la contraseña *admin*.

I.3 INSTALACIÓN DE LOS NODOS

En la instalación de los nodos del clúster instale el componente *Node Controller* de forma muy sencilla. El proceso tiene lugar de forma similar a la instalación del controlador del clúster, partiendo del inicio de los nodos desde la imagen ISO de *Ubuntu Server* descargada de la página web oficial.

Tras seleccionar el idioma para el instalador, seleccione de nuevo *Instalar Ubuntu Enterprise Cloud* en el menú de bienvenida. Lo primero que solicita el proceso de instalación es el nombre para el equipo , en nuestro ejemplo lo nombramos nodo01.

El sistema detecta que ya se encuentra instalado el controlador del cloud en la red, y permite instalar el nodo, por lo que aparece la pantalla que se muestra en la Figura I.7.

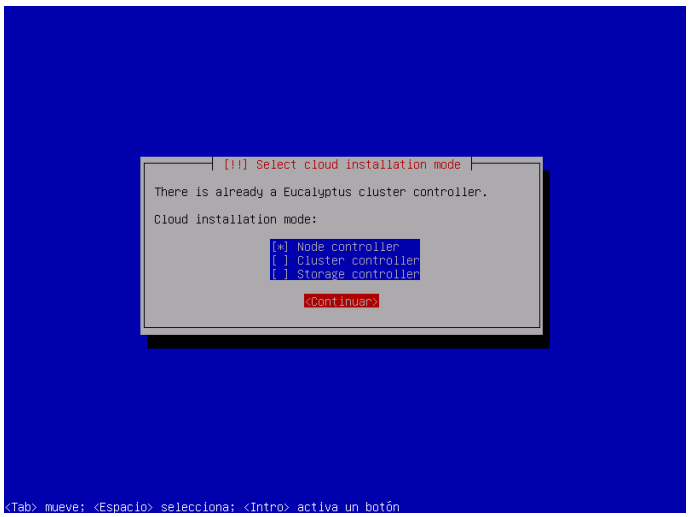

Figura I.7 - Selección del componente Node Controller para su instalación.

Marque la casilla *Node Controller* y confirme el esquema de particionamiento. Prosiga paso a paso la instalación hasta finalizarla. Al terminar, reinicie el sistema con el usuario administrador anteriormente definido en la instalación del controlador del cloud.

De igual forma que en el caso anterior, establezca la contraseña del usuario *root* con la utilidad *passwd* e instale el entorno gráfico, si lo considera oportuno:

```
$ sudo passwd root
$ sudo apt-get install Ubuntu-desktop
```

Es posible que durante la configuración de *Node Controller* el sistema muestre el mensaje de error que aparece en la figura I.8. Éste error se produce porque el sistema no dispone de aceleración para virtualización por *hardware* (el procesador no soporta ninguna de las tecnologías Intel-VT o AMD-V), cuyo soporte es un requisito indispensable en los nodos *Eucalyptus*. En este caso *UEC* no podrá tomar ventaja de estas tecnologías y podemos encontrar graves inconvenientes en el desarrollo de la nube e instanciación de las máquinas virtuales.

Figura I.8 - Advertencia acerca del soporte de aceleración hardware.

I.4 REGISTRO DE LOS NODOS EN EL SISTEMA

El registro de los nodos del clúster en el sistema *cloud* (con el controlador del clúster) **tiene lugar de forma automática** si la instalación del componente *Node Controller* se realiza al instalar *Ubuntu Enterprise Cloud* desde cero (a partir de la imagen ISO descargada de la web de *Ubuntu*), como es nuestro caso.

Si instala el componente *Node Controller en* una instalación previa, el registro del nodo debe realizarse de forma manual. Para registrar el nodo de forma manual es recomendable que visites la página:

https://help.ubuntu.com/community/UEC/CDInstall#STEP_4:_Register_the_Node.28s.29

I.5 CONFIGURACIÓN BÁSICA

Una vez realizada la instalación del controlador y de los distintos nodos que componen el clúster UEC debe realizar las primeras tareas de configuración de las comunicaciones del controlador y del clúster antes de poder comenzar a trabajar con la nube. Primero hay que obtener y utilizar las credenciales de acceso al servidor controlador de la nube, ya que es necesario para poder ejecutar las herramientas

administrativas de la nube. Después se muestra la interfaz web administrativa incluida con *UEC*.

I.5.1 OBTENCIÓN DE CREDENCIALES

El primer paso es obtener las credenciales de acceso al controlador de la nube (par de claves públicas y privadas) para los usuarios del mismo mediante herramientas de consola y gráficas.

Para realizar esta obtención de credenciales tenemos dos opciones: a través del navegador web en la interfaz administrativa de *UEC* o a través de la línea de comandos.

A TRAVÉS DEL NAVEGADOR DE LA INTERFAZ UEC:

Inicie el navegador web escribiendo en la barra de direcciones la dirección de red del controlador *cloud* a través del puerto *8443* mediante una comunicación segura (*https*):

```
https://<direccion_ip_cloud_controller>:8443/
```

De esta forma accede a la interfaz web administrativa incluida con *UEC*. Para iniciar sesión por primera vez utilice el usuario *admin y* contraseña *admin* (véase figura I.9).

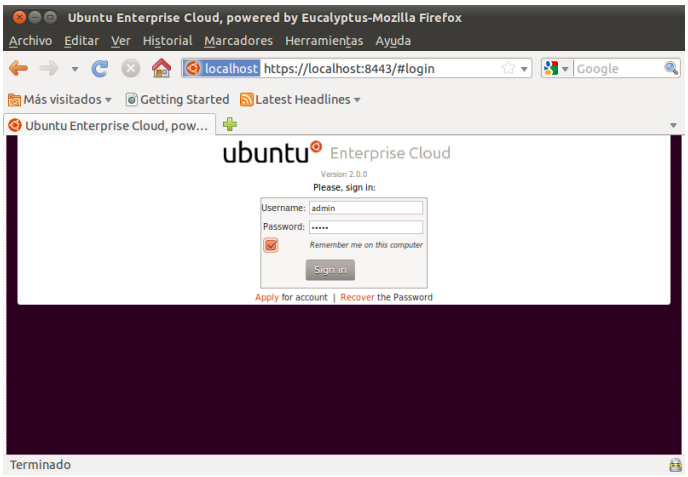

Figura I.9 - Inicio de sesión en la interfaz web de UEC.

Una vez se autentifique hay que editar la configuración de la cuenta *admin* y cambiar su contraseña de acceso. Entre la información que puede editar se incluye, por ejemplo, la dirección de correo electrónico, teléfono, o el nombre del líder del proyecto. Ver figura I.10.

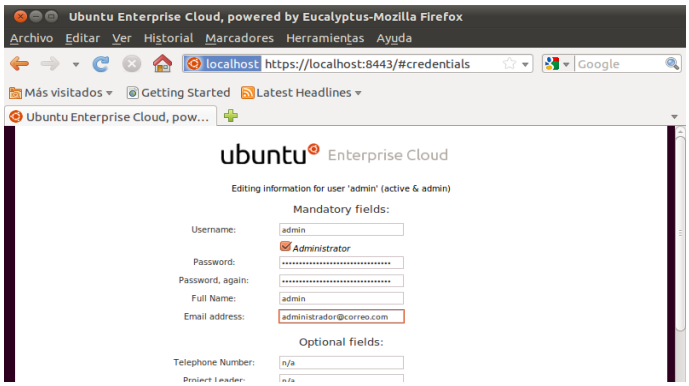

Figura I.10. Configuración de la cuenta admin.

Una vez completada la configuración de la cuenta *admin* seleccione la pestaña *Credentials* (*Credenciales*) situada en el menú superior de la pantalla y pulse el botón *Download Credentials* (en el área *Credentials ZIP-file*) para descargar las credenciales y así obtener los certificados de acceso.

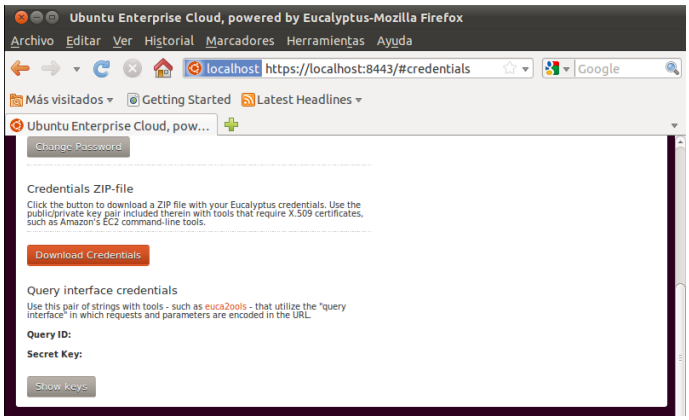

Figura I.11 - Descarga de credenciales.

Una vez descargadas las credenciales en un fichero *zip*, se deben copiar en el directorio oculto *~/.euca* y descomprimir ejecutando los comandos:

```
$ cp euca2-admin-x509.zip ~/.euca
$ cd ~/.euca
$ unzip euca2-admin-x509.zip
```

A TRAVÉS DE LÍNEA DE COMANDOS:

Otra posibilidad para la obtención de las credenciales es ejecutando los comandos *Eucalyptus* correspondientes en un terminal del equipo controlador. En primer lugar compruebe que existe el directorio oculto *~/.euca* y que dispone de los permisos necesarios (*700*), en caso contrario debe crearlos y establecer los permisos:

```
$ mkdir -p ~/.euca
$ chmod 700 ~/.euca
```

Después ejecute el comando *euca_conf* con la opción *--get-credentials* como usuario privilegiado y descargue las credenciales en un fichero *zip* que debe descomprimir:

```
$ cd ~/.euca
$ sudo euca_conf --get-credentials mycreds.zip
$ unzip mycreds.zip
```

Finalmente hay que crear el enlace simbólico y volver al directorio anterior:

```
$ ln -s ~/.euca/eucarc ~/.eucarc
$ cd -
```

I.5.2 USO DE CREDENCIALES E INSTALACIÓN DE EUCA2OOLS

Con los certificados o credenciales de acceso a *Ubuntu Enterprise Cloud* ya puede comenzar a utilizar las herramientas administrativas que permiten operar sobre la nube, para lo cual es necesario configurar la *API EC2* y las herramientas *AMI* del servidor usando los certificados X.509 instalados (credenciales descargadas en el paso anterior).

Para instalar *euca2ools*, las herramientas en línea de comandos utilizadas por los usuarios del *cloud* para su administración ejecute:

```
$ sudo apt-get install euca2ools
```

En este momento se valida que el clúster se ha instalado y configurado correctamente ejecutando el comando de *euca2ools euca-describe-availability-zones*, que nos permite mostrar en consola la disponibilidad de los recursos de un clúster y las posibilidades existentes para la instanciación de diferentes tipos de instancias (*libres* y *máximo*) en función del número de procesadores que requiere, cantidad en *megabytes* de memoria RAM y espacio en disco en *gigabytes*. Para que ello sea posible, antes cargamos los certificados/credenciales de acceso (esto es necesario hacerlo antes de cada sesión cliente para el uso de las herramientas administrativas de *UEC*):

```
$ . ~/.euca/eucarc
$ euca-describe-availability-zones verbose
```

En la figura I.12 puede observar un ejemplo de salida para el comando *euca-describe-availability-zones*.

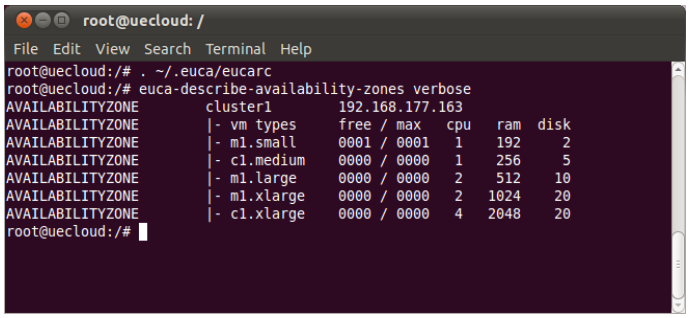

Figura I.12 - Comprobando la disponibilidad del clúster con euca-describe-availability-zones.

I.5.3 FICHEROS Y CONTROL DEL SERVICIO EUCALYPTUS

Es importante conocer la configuración básica del software *cloud computing Eucalyptus*, aunque vaya administrarse prácticamente por completo utilizando herramientas y servicios web. Por ejemplo, para iniciar, detener o reiniciar los servicios *eucaliptus* en el controlador de la nube (equipo *Cloud Controller, Cluster Controller, EBS Controller y Walrus Controller*) ejecute como usuario privilegiado:

```
$ sudo service eucalyptus [start|stop|restart]
```

Para hacerlo en cualquiera de los equipos *Node Controller* se opera de forma análoga, aunque con el comando *eucaliptus-nc*:

```
$ sudo service eucaliptus-nc [start|stop|restart]
```

Para conocer mejor el sistema es recomendable conocer la ubicación de los ficheros de configuración más importantes:

- **Ficheros de configuración.** */etc/eucalyptus*
- **Ficheros *log* o *de registro*.** */var/log/eucalyptus*
- **Base de datos.** */var/lib/eucalyptus/db*
- **Claves.** */var/lib/eucalyptus* y */var/lib/eucalyptus/.ssh*

I.5.4 INTERFAZ WEB ADMINISTRATIVA DE UEC

Como se ha citado en varias ocasiones con anterioridad *Ubuntu Enterprise Cloud* dispone de una interfaz web administrativa propia que es instalada al mismo tiempo y a través de la cual es posible realizar la configuración del clúster *cloud*.

Las pestañas de la interfaz web son:

CREDENTIALS

Esta pestaña, que ya hemos visto anteriormente, permite la configuración de la cuenta de usuario *admin* (características básicas y contraseña), descargar las credenciales de *Eucalyptus* en un fichero comprimido *ZIP* (par de claves públicas y privadas a usar en las herramientas que requieran utilizar certificados X.509, como las utilidades en línea de comandos *EC2* de *Amazon*) y mostrar las *credenciales de consulta vía URL* para aplicaciones externas (veremos en qué consiste a la hora de configurar *Hybridfox,* una herramienta web para la administración de *clouds Eucalyptus*).

IMAGES

Muestra un listado de las imágenes (además de su núcleo e imagen de disco *RAM*) actualmente instaladas en el controlador y que por tanto pueden ser instanciadas en máquinas virtuales para la nube. Además, para cada imagen permite ver su estado así como habilitarla o deshabilitarla. Ver figura I.13.

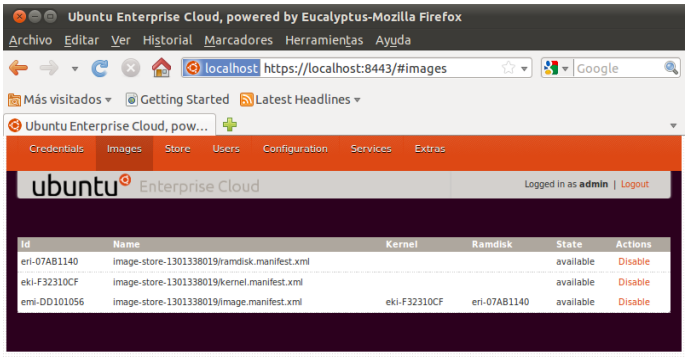

Figura I.13 - Pestaña *Images* de la interfaz web administrativa de UEC.

STORE

En la pestaña *Store* o *almacén* puede ver las diferentes imágenes disponibles para su instalación en el sistema. En un principio contiene un pequeño conjunto de las mismas, aunque es posible añadir más descargándolas y registrándolas en el sistema (para más información consulte la documentación oficial de *UEC*). Más adelante se verá cómo instalar imágenes utilizando las disponibles en el almacén.

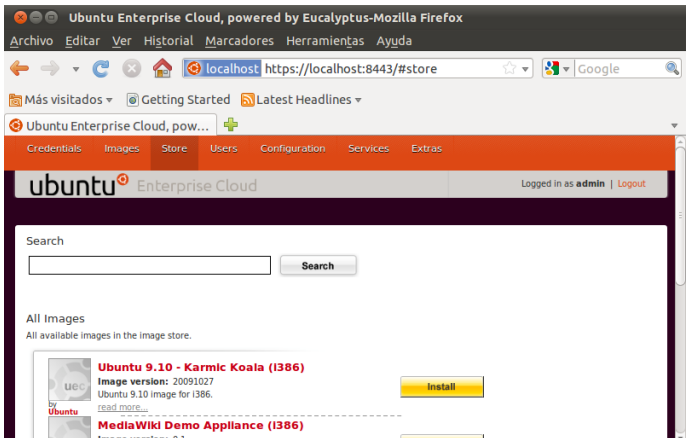

Figura I.14 - Pestaña *Store* de la interfaz web administrativa de UEC.

USERS

Permite administrar los diferentes usuarios que utilizan la nube. Inicialmente sólo existe el usuario *admin*. Para cada uno de ellos puede verse su identificador único, nombre de usuario, dirección de correo electrónico y nombre completo, estado de la cuenta y un enlace para proceder con la edición de sus propiedades (véase figura I.15). Si queremos añadir un nuevo usuario, pulsamos el botón *Add user*.

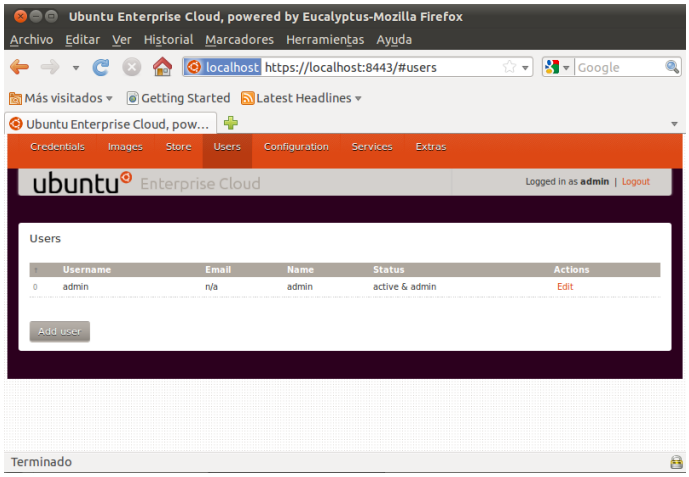

Figura I.15 - Pestaña *Users* de la interfaz web administrativa de UEC.

CONFIGURATION

Permite configurar los parámetros más importantes de la funcionalidad del equipo como *Cloud Controller, Walrus Controller* y *Cluster Controller*.

El primer apartado contiene la configuración general de la nube como la dirección de red del *cloud host*, su núcleo (*kernel*) y disco de inicio *RAM* (*ramdisk*) predeterminados.

Asimismo en esta primera se puede editar los parámetros relacionados con el servicio DNS, como el nombre del dominio, del servidor DNS y su dirección IP, tal y como muestra la figura I.16,

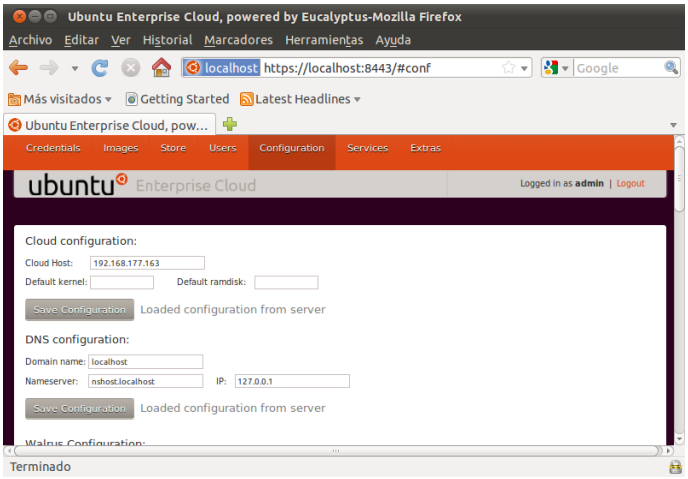

Figura I.16. Pestaña *Configuration* de la interfaz web administrativa de UEC.

Más abajo en esta pantalla, como representa la siguiente figura, puede ver y editar la configuración del clúster (o clústeres) definido para la nube así como añadir nuevos: nombre del clúster, dirección IP del Cluster Controller, la configuración de las direcciones de red públicas disponibles para ser asignadas a las instancias, y los parámetros relativos a la configuración del Storage Controller (como el tamaño máximo para un volumen, el espacio reservado para los mismos y su localización o el nombre de su interfaz). Todo esto queda representado en la figura I.17.

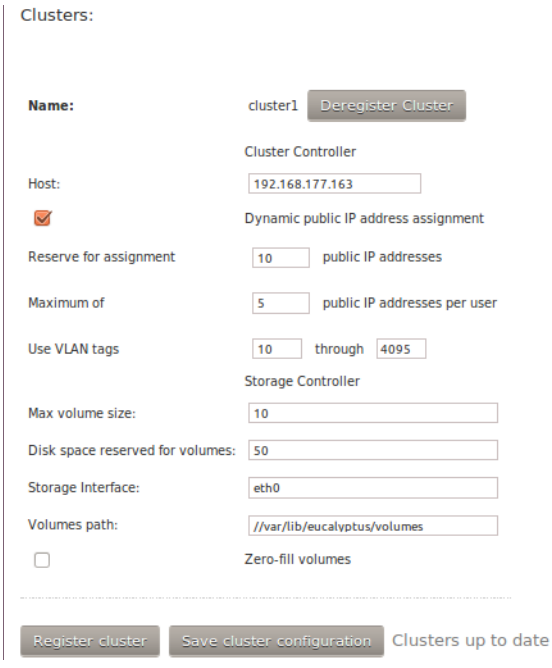

Figura I.17 - Pestaña *Configuration* (Clusters) de la interfaz web administrativa de UEC.

Para terminar, en la tercera sección de la página, es posible especificar los valores de configuración para los distintos tipos de máquinas virtuales o instancias a crear a partir de las imágenes registradas. Para cada uno de los tipos de máquinas virtuales (*m1.small, c1.medium, m1.large, m1.xlarge,* y *c1.xlarge*) puede escribir el número de CPUs, la cantidad de memoria RAM en *megabytes* y el tamaño de disco en *gigabytes*. En la figura I.18 se muestran la configuración utilizada en este proyecto.

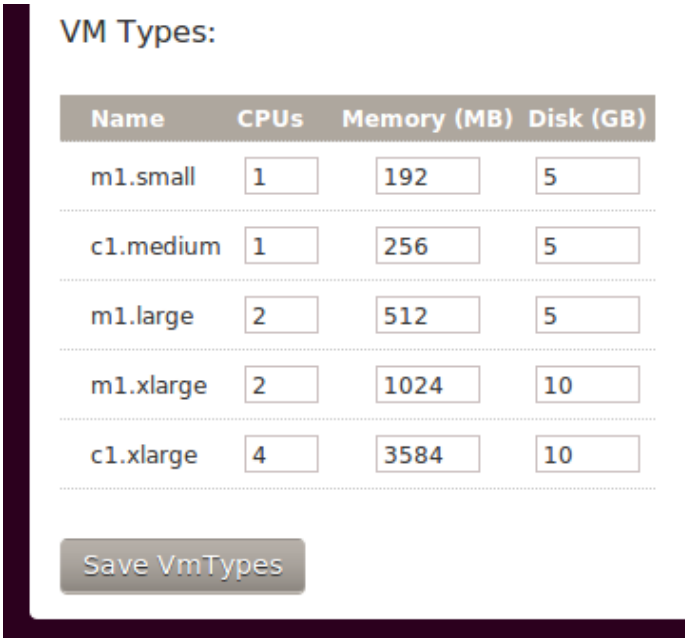

Figura I.18 - Pestaña *Configuration* (VM Types) de la interfaz web administrativa de UEC.

SERVICES

Ofrecen enlaces a diferentes páginas, herramientas y recursos que pueden resultar de gran interés para el correcto desarrollo de las actividades administrativas en nuestra nube, como la documentación *online*, listas de usuarios y páginas de la comunidad *Ubuntu Enterprise Cloud.*

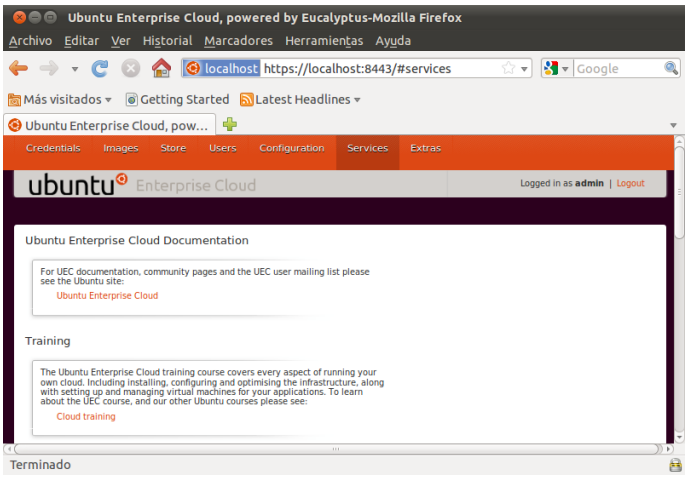

Figura I.19 - Pestaña *Services* de la interfaz web administrativa de UEC.

También proporciona enlaces relacionados con *Eucalyptus* (sitios oficiales para usuarios individuales y empresas), *Landscape* (servicio para administración de los sistemas *cloud* permitiendo gestionar paquetes o personalizar *scripts* de automatización y monitorización), *Rightscale* (interfaz administrativa para nubes *Eucalyptus* y *EC2*), o soporte comercial.

EXTRAS

Permite el acceso a la descarga de imágenes certificadas para *Eucalyptus* de las distribuciones Linux más utilizadas (*CentOS, Fedora, Debian* y *Ubuntu*) así como herramientas cliente compatibles con *Eucalyptus*, como *euca2ools*.

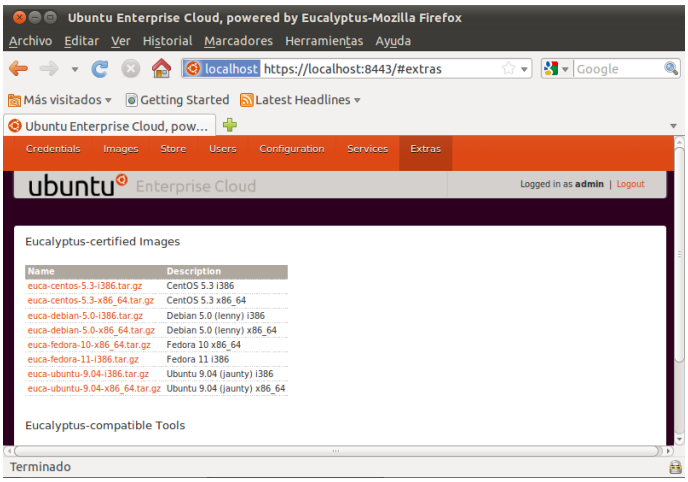

Figura I.20 - Pestaña *Extras* de la interfaz web administrativa de UEC.

I.6 INSTALACIÓN DE IMÁGENES DESDE EL ALMACÉN

Antes de poder instanciar una imagen es necesario instalarla en el sistema. La forma más sencilla para hacerlo es a través de la pestaña *Store* (*Almacén*) de la herramienta web para la administración de *UEC*. Para ello abrimos la interfaz web *UEC* a través de la dirección de red del controlador de la nube y el puerto 8443 de forma segura:

```
$ https://<direccion_ip_cloud_controller>:8443/
```

Tras autenticarnos de forma correcta seleccionamos la pestaña *Store*, localice entre las diferentes imágenes disponibles la que desea instalar y pulse el botón *Install* y se inicia la descarga de la imagen como muestra la figura I.21.

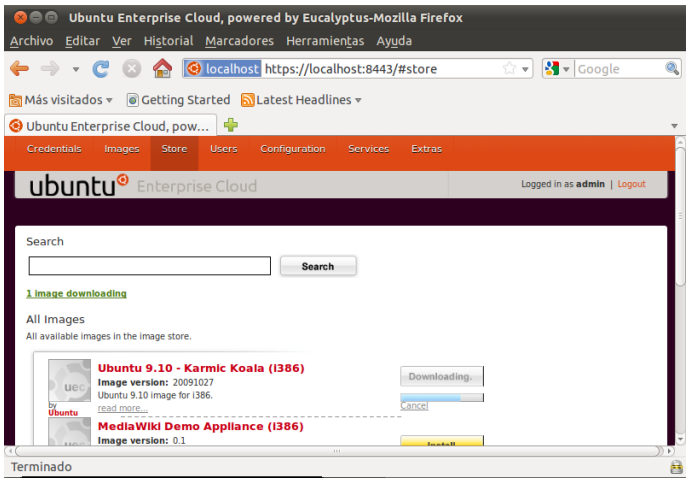

Figura I.21 - Descarga previa a la instalación de una imagen desde el almacén.

Una vez finalizada la descarga, la imagen se instala de forma automática, apareciendo junto a su nombre el estado *Installed* y un vínculo *How to run* que al pulsarlo muestra una ventana (figura I.22) con el comando *Eucalyptus* correspondiente para instanciarla en una máquina virtual a través del comando *euca-run-instances* añadiendo como parámetros nuestro par de claves de autenticación y el nombre de la imagen a instanciar (más adelante veremos un ejemplo concreto sobre cómo utilizar este comando).

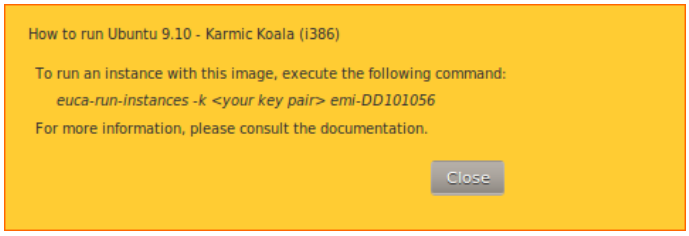

Figura I.22 - Ejemplo de ventana que *How to run* con el comando para instanciar una imagen.

Si consulta el listado de imágenes del sistema en la pestaña *Images* puede comprobar que efectivamente la nueva imagen esta *disponible* (véase la figura I.23).

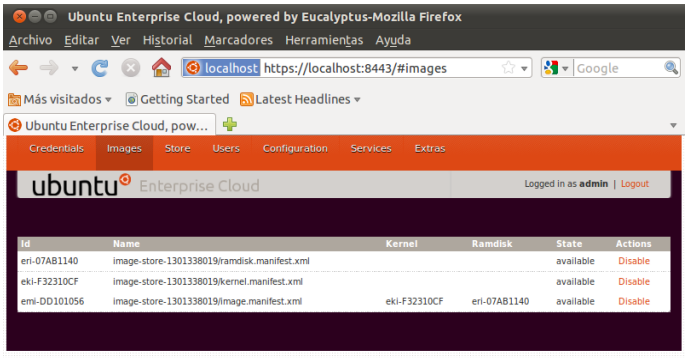

Figura I.23. Imagen recién creada disponible en la pestaña *Images*.

I.7 INSTANCIACIÓN, MONITORIZACIÓN Y CONEXIÓN A IMÁGENES

A partir de las imágenes instaladas en nuestro sistema *cloud computing UEC* puede instanciar nuevas máquinas virtuales a desplegar en los nodos del clúster, monitorizarlas y acceder a su consola.

I.7.1 DESDE LA LÍNEA DE COMANDOS

Una de las opciones habituales es instanciar las máquinas virtuales a través de una terminal en el servidor *Cloud Controller* utilizando la herramienta *euca2ools*. De forma previa debemos **crear un par de claves SSH que permiten autenticarnos en la instancia como usuario *root*** una vez que ésta haya sido iniciada (la clave privada creada es almacenada por lo que sólo es necesario hacerlo una vez). Esto puede realizarlo escribiendo un *script* a ejecutar con el siguiente contenido (si ya está creado el directorio *~/.euca* puede descartar la tercera línea):

```
. ~/.euca/eucarc
if [ ! -e ~/.euca/miclave.priv ]; then
      mkdir -p -m 700 ~/.euca
      touch ~/.euca/miclave.priv
      chmod 0600 ~/.euca/miclave.priv
      euca-add-keypair miclave > ~/.euca/miclave.priv
fi
```

Una vez asignado el nombre *miclave* para la clave, debe recordarlo para especificar las claves a usar en el momento de instanciación de la clave. Si desea ver un listado de las claves almacenadas en el sistema puede ejecutar el comando *euca-describe-keypairs*. En el *script* anterior se realiza la carga de las credenciales para posibilitar la ejecución de los comandos en la consola cliente; si no se hace esto, debe ejecutarla de todos modos antes de generar las claves.

A continuación **permita el acceso vía SSH**, ya que éste será el medio por el que se comunique con las instancias, ejecutando el comando *euca-authorize* y añadiendo la regla de permiso correspondiente:

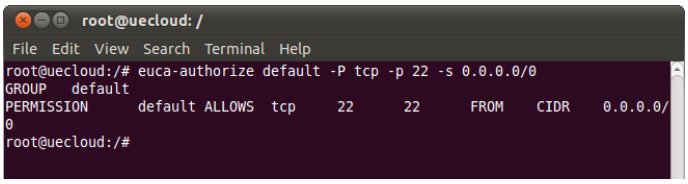

Figura I.24 - Salida del comando *euca-authorize* para permitir el acceso SSH a las instancias.

Realizado esto ya está preparado **para instanciar e iniciar la máquina virtual** con el comando *euca-run-instances*, tal y como se mencionó anteriormente. En este caso

facilite como parámetros de ejecución el identificador de la imagen a instanciar (*EMI*), la clave generada y el tipo de instancia (por ejemplo, *m1.small*):

```
$ euca-run-instances emi-DD101056 -k miclave -t m1.small
```

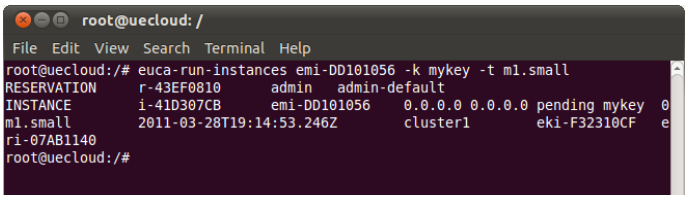

Figura I.25 - Salida del comando *euca-run-instances* para instanciar e iniciar una nueva máquina virtual.

Si desconoce el identificador *EMI* de la imagen puede consultar bien en la pestaña *Images* de la interfaz web administrativa de *UEC* o ejecutando el comando *euca-describe-images*. La primera vez que se ejecuta una instancia el sistema *cachea* la imagen a partir de la cual es creada; este proceso, por tanto, puede tomar algún tiempo dado que las imágenes suelen tener un tamaño del orden de *gigabytes*. En este momento, como muestra la figura anterior, la máquina virtual se encuentra en el estado ***Pending*** (*pendiente*).

Para **monitorizar** el estado de las instancias puede ejecutar el comando *euca-describe-instances*, mostrando además otra información sobre ellas como su nombre, imagen utilizada para crearla, dirección de red o tipo de instancia.

```
$ watch -n5 euca-describe-instances
```

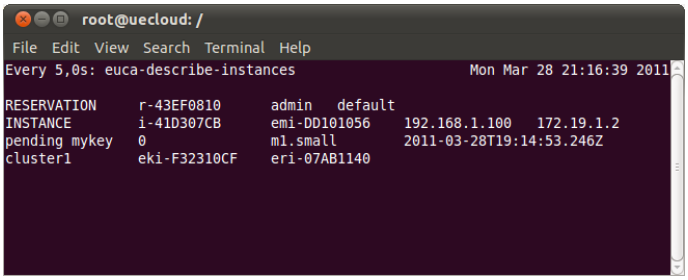

Figura I.26 - Salida del comando *euca-run-instances* para instanciar e iniciar una nueva máquina virtual.

Una vez que ha concluido por completo el inicio de la instancia pasa a encontrarse en estado ***Running*** (*en ejecución*). Ahora puede acceder a ella vía SSH obteniendo primero su dirección de red de la salida del comando *euca-describe-instances* (en el que debemos incluir como filtro el identificador *EMI* de la instancia) y utilizando el par de claves generadas anteriormente:

```
$ EMI=emi-DD101056
$ IPADDR=$(euca-describe-instances | grep $EMI | grep running | tail -n1 |
awk '{print $4}')
$ ssh -i ~/.euca/miclave.priv ubuntu@$IPADDR
```

Cuando termine de trabajar en la máquina virtual cierre la conexión SSH con el comando *exit* y termine la instancia (estado ***Terminated***) mediante *euca-terminate-instances*, previa recuperación del identificador de la instancia:

```
$ EMI=emi-DD101056
$ INSTANCEID=$(euca-describe-instances | grep $EMI | grep running | tail -n1
| awk '{print $2}')
$ euca-terminate-instances $INSTANCEID
```

En la Tabla *I.2* puede ver un resumen de los comandos básicos utilizados en la herramienta *euca2ools* para la administración y control de las máquinas virtuales del *cloud*:

Comando	Descripción	Ejemplo
euca-describe-instances	Muestra un listado con las instancias existentes en el sistema. Si queremos conocer información sobre una instancia concreta añadimos su identificador	`euca-describe-instances i-3DF4075E`
euca-run-instances	Permite desplegar nuevas instancias en función de una imagen registrada. Como mínimo debemos especificar la imagen utilizada y las claves de autenticación	`euca-run-instances emi-DD101056 -k miclave -t m1.small`
euca-terminate-instances	Se utiliza para apagar y terminar una instancia. Ésta deja de estar presente en el sistema cloud	`euca-terminate-instances i-3DF4075E`
euca-reboot-instance	Permite reiniciar instancias en ejecución	`euca-reboot-instances i-3DF4075E`

Tabla I.2 - Comandos básicos para la administración de máquinas virtuales con euca2ools

Para conocer las opciones completas disponibles para cada uno de los comandos utilice la opción *--help* o consulte su página *man*, por ejemplo:

```
$ euca-describe-instances –help
$ man euca-describe-instances
```

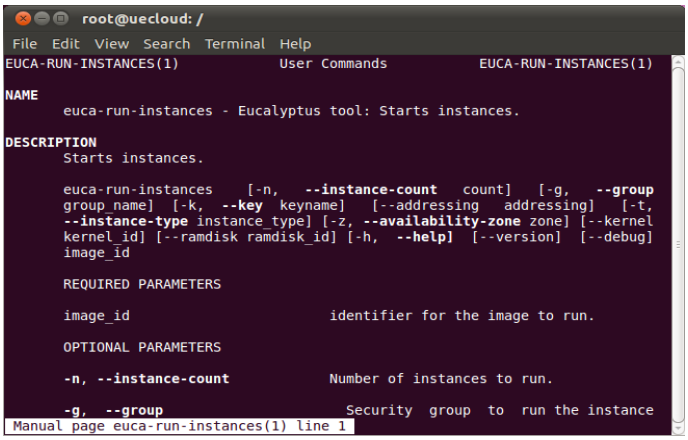

Figura I.27 - Página man del comando euca-run-instances.

I.7.2 DESDE LA INTERFAZ WEB DE HIBRIDFOX

Otra forma de instanciar imágenes creando máquinas virtuales, monitorizarlas y en definitiva gestionar de forma completa su ciclo de vida es mediante herramientas externas a *Ubuntu Enterprise Cloud*, normalmente interfaces web, como por ejemplo *Landscape* o **Hybridfox**, que presentamos a continuación.

Figura I.28. Logo de Hybridfox.

Hybridfox (http://code.google.com/p/hybridfox/), disponible desde la versión *1.6* de *Eucalyptus* como una escisión de *ElasticFox* –ésta para nubes *Amazon-*, es una versátil y sencilla herramienta web compatible con la administración de dos de las soluciones *cloud computing* más populares: *Eucalyptus* (para el desarrollo de nubes privadas, como es el caso de *Ubuntu Cloud*) o *Amazon EC2* (para la interacción con nubes públicas). Una de las grandes ventajas que presenta es la posibilidad de permutar entre las cuentas administrativas de un tipo y otro de nubes. Veamos a continuación cómo instalar y configurar de forma básica *Hybridfox*.

INSTALACIÓN

Hybridfox se distribuye como una *extensión* para el navegador *Mozilla Firefox* por lo que el proceso de instalación es muy sencillo. En primer lugar descargue la extensión desde el propio sitio web oficial del proyecto, seleccionando la versión correspondiente que desea instalar desde la pestaña *Downloads* (*descargas*).

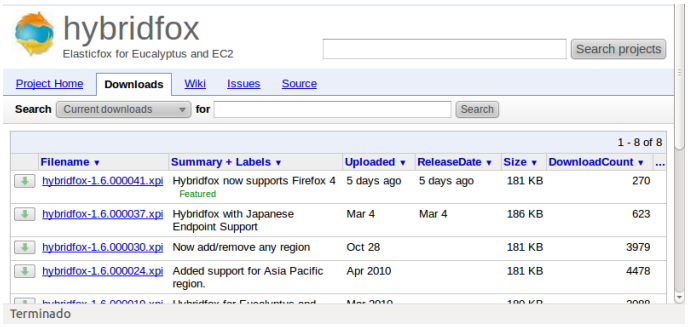

Figura 29. Descarga de Hybridfox en el sitio web del proyecto.

Una vez descargado el fichero se abre con *Firefox* y muestra una advertencia sobre el origen del fichero (ver la figura I.30) que debe aceptar pulsando el botón *Instalar ahora.*

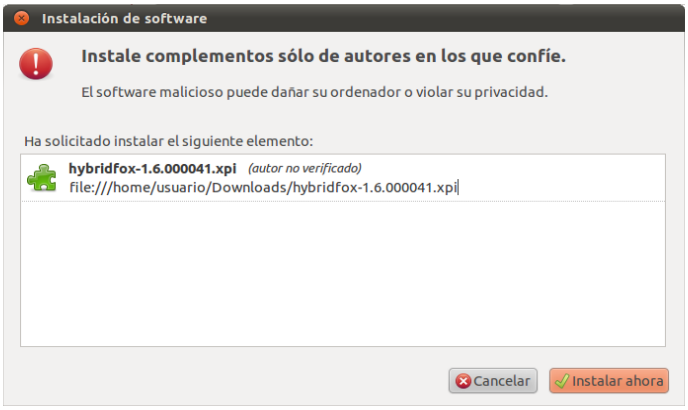

Figura I.30. Instalación de Hybridfox.

Al finalizar la instalación, es necesario reiniciar *Firefox* para aplicar los cambios realizados. Puede comprobar que *Hybridfox* se ha instalado con éxito al arrancar de nuevo al navegador, ya que aparece en pantalla la ventana *Complementos* con un mensaje al respecto.

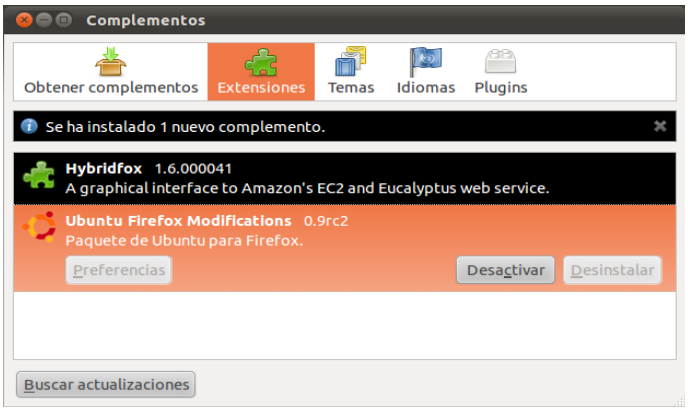

Figura I.31 - Confirmación de la instalación de Hybridfox.

A partir de ahora, para iniciar *Hybrifox* hay que seleccionar el elemento de la entrada correspondiente en el menú *Herramientas* de *Firefox*. *Hybridfox* aparece en una nueva pestaña y, tras realizar los pequeños pasos de configuración de la conexión con la nube a administrar, podemos empezar a trabajar con él. En la figura I.32 puede ver el estado mostrado por *Hybrifox* cuando se inicia por primera vez.

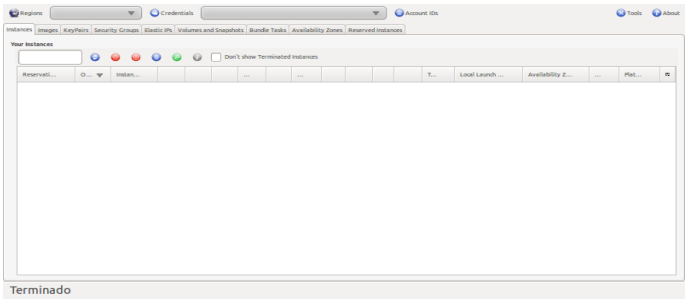

Figura I.32 - Estado inicial de Hybrifox (pestaña Instances).

CONFIGURACIÓN

Como se ha indicado anteriormente, antes de poder trabajar con *Hybrifox* es necesario configurar la comunicación con la nube a administrar (ya sea una nube privada *Eucalyptus* como una nube pública *Amazon EC2*).

Definición de una región

Antes de conectarnos a una nube para administrarla es necesario incluirla en la herramienta. Para *Hybridfox* una *región EC2* no es más que una nube con la que es compatible, como la instancia *Ubuntu Enterprise Cloud* a la cual tiene acceso. Para añadir una nueva región pulse el botón *Regions* disponible en la parte superior de la pantalla (véase figura I.33).

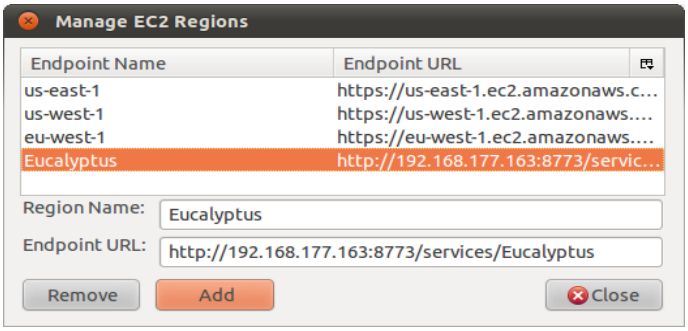

Figura I.33 - Ventana para añadir y eliminar Regiones a Hybridfox.

En la ventana *Manage EC2 Regions* escriba un nombre descriptivo para la región así como la URL *punto final*, que debe seguir el siguiente formato:

```
http://<direccion_ip_cloud>:8773/services/Eucalyptus
```

Completados los dos campos pulse el botón *Add*. Si desea editar o eliminar una región también puede hacerlo desde esta misma ventana seleccionándola del listado y cambiando el valor de los campos o pulsando el botón *Remove* (*eliminar*), respectivamente.

Definición de las credenciales

Con la región correspondiente a la nube incluida, debe especificar las credenciales de acceso a la misma. Obtenerlas es fácil a través de la interfaz administrativa web de *UEC*, como sabemos, accesible a través de la dirección:

```
https://<direccion_ip_cloud>:8773
```

Inicie sesión mediante el nombre de usuario y contraseña de la cuenta administrativa (recuerde que por defecto, tanto el nombre de usuario como la contraseña es *admin*) y acceda a la pestaña *Credentials* (*credenciales*). Como puede observar en la figura I.34, en la parte baja de la pestaña dispone de la sección *Query interface credentials* para la consulta de los valores de las credenciales de acceso para herramientas externas a *UEC*.

Figura I.34 – Consulta de credenciales en la pestaña Credentials de la interfaz UEC.

Para consultar estos valores (campos *Query ID* y *Secret ID*) y mostrarlos en pantalla pulse el botón *Show Keys* (*Hide Keys* cuando se encuentran disponibles las claves). Ahora abra *Hybridfox* en una ventana diferente a la que ventana en la que tengamos la interfaz *UEC* y pulse el botón *Credentials* (*credenciales*), en esta ocasión a la derecha del botón *Regions*, para abrir la ventana *Manage EC2 Credentials* (figura I.35).

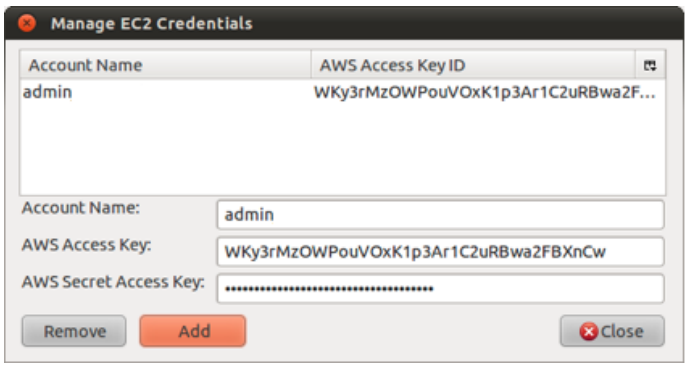

Figura I.35 - Ventana para añadir y eliminar Credenciales a Hybridfox.

En esta ventana añada las credenciales de acceso a la región cumplimentando los campos *Account Name* con un nombre para la cuenta, *AWS Access Key* con el valor *Query ID* y *AWS Secret Key* con *Secret Key*, proporcionados por la interfaz web administrativa *UEC*. Escritos todos los valores haga clic en el botón *Add* para añadir la cuenta y sus credenciales. De igual forma puede editar y eliminar las existentes seleccionándolas en el listado y pulsando el botón *Remove*.

Alcanzado este punto puede seleccionar una región y las credenciales de acceso a la misma en los cuadros desplegables habilitados para ello (figura I.36) y comenzar a trabajar en tareas administrativas de la nube con *Hybridfox*.

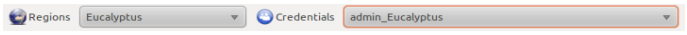

Figura I.36 - Selección de región y credenciales en Hybridfox.

UTILIZACIÓN

A continuación se va a realizar un breve recorrido por las diferentes pestañas en las que se distribuye la funcionalidad de *Hybridfox*. Como puede comprobar la interfaz de *Hybridfox* es bastante simple y sencilla, por lo que la mejor forma de conocer las posibilidades que ofrece es utilizándola.

Instances (instancias)

Esta primera pestaña es una de las más importantes ya que permite manipular, monitorizar y realizar cualquier tarea relacionada con las instancias desplegadas en la nube, incluyendo la modificación de su estado y la conexión con su consola.

En el listado de instancias puede ver para cada una la siguiente información: identificador de reserva, propietario, identificador único, *AMI* (referencia de la imagen utilizada para la instancia), *AKI* (*kernel* utilizado), *ARI* (disco *RAM de inicio*), el estado en el que se encuentra, DNS público y privado, clave, grupo de seguridad al que pertenece, tipo de máquina virtual, arquitectura, plataforma, y otros.

En la figura I.37 puede ver un ejemplo de múltiples instancias lanzadas por el usuario en estado de ejecución (*running*), donde cada una va adquiriendo una IP publica en función del rango indicado en la instalación de *Ubuntu Enterprise Cloud*. Del mismo modo cada instancia tiene asignada una IP privada, por defecto 172.19.1.*, la cual sólo es accesible dentro de la red privada donde hayamos desplegado nuestro clúster.

Desde esta pantalla puede reiniciar o terminar una instancia, mostrar la consola o conectarse a la misma. Además, si tenemos muchas instancias, puede realizar una búsqueda, tal y como muestra la figura I.37.

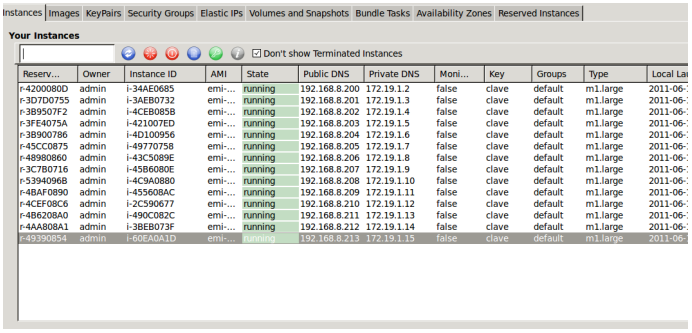

Figura I.37. Instancias en ejecución en Hybridfox.

Images (imágenes)

Permite listar y administrar las imágenes registradas en el sistema *cloud*. Puede ejecutar una búsqueda de las imágenes disponibles, crear y añadir nuevas, eliminarlas, etc.

Para cada una de las imágenes (identificadas por la inclusión de la cadena *emi* al comienzo de su identificador), además de disponer en la lista de acceso a su *kernel* e imagen de disco RAM de inicio, puede ver, entre otros datos, su identificador, el *manifiesto* asociado, el estado que presenta y la visibilidad de la misma, o la arquitectura.

Cada vez que se instale una imagen de la *Store* de la interfaz web de UEC, aparece en esta pestaña (véase la figura I.38).

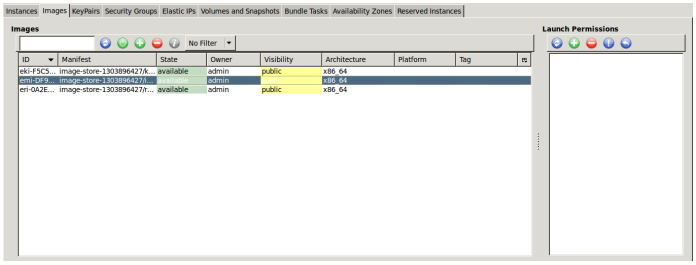

Figura I.38 - Hybridfox - Images.

Sin duda la tarea más importante es la creación y lanzamiento de nuevas instancias a partir de una de las imágenes listadas. Para hacerlo, seleccione la imagen correspondiente y pulse el botón derecho del ratón sobre ella para seleccionar la opción *Launch Instance(s) of this AMI,* apareciendo la ventana *Launch new Instance(s)* (véase la figura I.39).

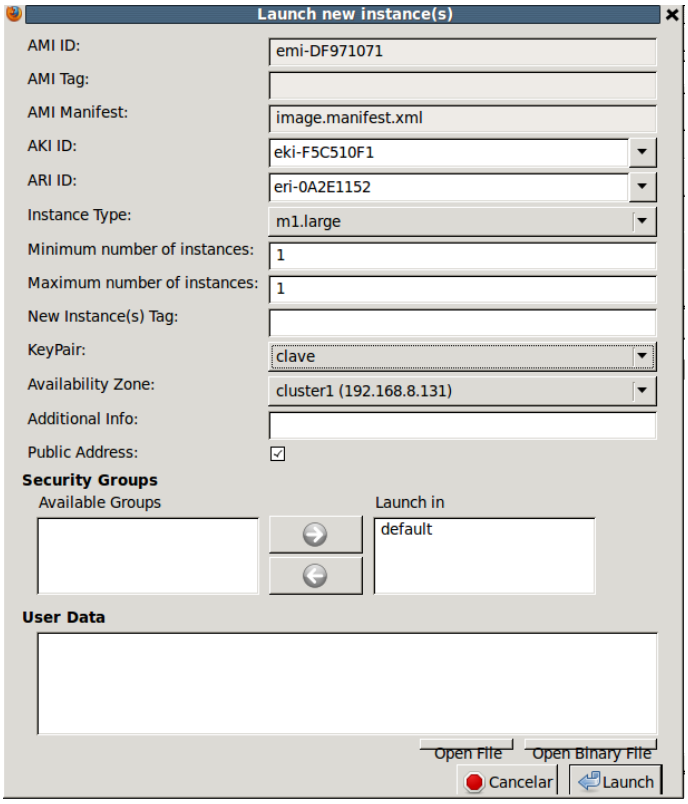

Figura I.39 - Lanzamiento de una nueva instancia.

Las diferentes opciones incluidas en esta ventana ayudan en la configuración completa de las nuevas instancias que desea desplegar en la nube, indicando por ejemplo la imagen del *kernel* a utilizar (*AKI ID*), el disco RAM de inicio (*ARI ID*), el tipo y número de instancias, el grupo de seguridad al que pertenece, los pares de claves a utilizar, etc. Seleccionadas todas las opciones deseadas pulse el botón *Launch* para proceder con la instanciación y lanzamiento; si la operación ha sido realizada con éxito, la(s) nueva(s) instancia(s) debe(n) aparecer listada en la pestaña *Instances*.

Aunque es posible configurarlo de otro modo, *Ubuntu Enterprise Cloud* realiza el despliegue de nuevas instancias basándose en el algoritmo *Round Robin* y la lista de los nodos del clúster que aportan sus recursos para la ejecución de las instancias (disponible en el fichero */var/lib/eucaliptus/nodes.list*).

KeyPairs (pares de claves)

Permite gestionar los diferentes pares de claves registrados en el sistema para el acceso y conexión con las instancias. Como se muestra en la figura I.40, es listado el nombre y la *huella* de cada una de ellas, y como tareas específicas es posible crear nuevos pares o eliminar los existentes.

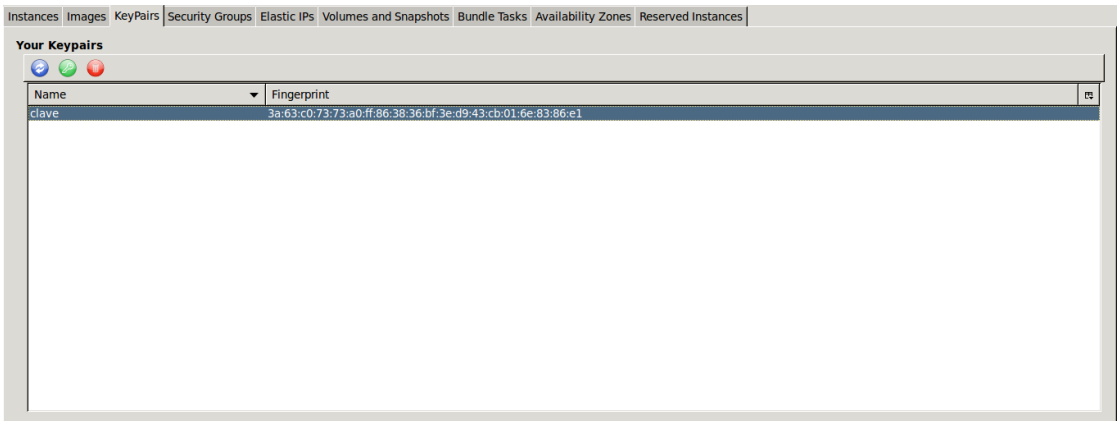

Figura 40. Hybridfox - KeyPairs.

Security Groups (grupos de seguridad)

Como su propio nombre indica, desde esta pestaña se permite definir nuevos grupos de seguridad para las instancias. Un grupo de seguridad permite asociar reglas de permiso o denegación para ciertas operaciones relacionadas con las instancias, como las comunicaciones vía SSH para abrir en ellas nuevas terminales.

Si desea añadir un grupo de seguridad, pulse el icono "añadir" e indique los detalles del protocolo que desea autorizar (véase la figura I.41).

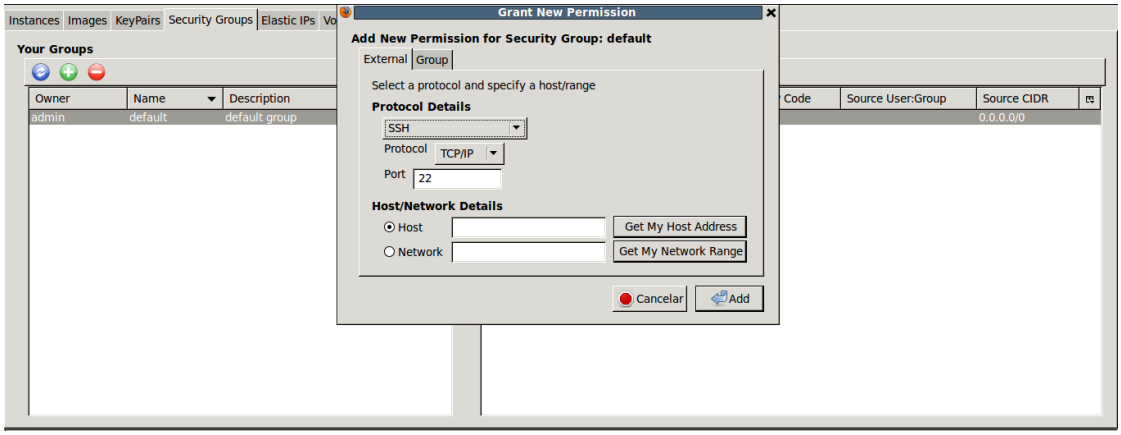

Figura 41. Hybridfox – Security Groups.

Elastic IPs

Permite la visualización del conjunto de direcciones de red completo utilizable por las instancias del clúster. Dicho de otro modo, las direcciones IP públicas libres disponibles para ser asignadas a nuevos adaptadores de red en las instancias y las ocupadas por conexiones en las ya creadas con anterioridad.

En esta misma pestaña dispone de controles y botones para añadir nuevas direcciones al conjunto, asignar una determinada dirección a una instancia o liberarla.

Volumes and Snapshots (volúmenes e imágenes), Bundle Tasks, Availability Zones, Reserved instances

Otras pestañas permiten realizar operaciones de administración avanzada *cloud computing* como la creación de volúmenes (directorios y espacio en disco que puede ser compartido por diferentes instancias), toma y recuperación de *snapshots* (imágenes tomadas de las instancias durante su ejecución), construcción de nuevas imágenes o la consulta de instancias *reservadas* y clústeres conectados a *Hybridfox* (*Availability Zones*).

CONEXIÓN

Una vez que ha concluido por completo el inicio de la instancia pasa a encontrarse en estado **Running** (*en ejecución*). Ahora puede acceder a ella vía SSH a través de su IP pública o privada, en función de la red en la que se encontre.

Para ello acceda mediante un terminal empleando la clave privada que *Hybridfox* ha generado, y solicitado donde guardar al registrar el par de claves en la pestaña KeyPairs. Además debe conectarse al directorio home del usuario *ubuntu* que crea por defecto *Hybridfox* en la instancia.

```
$ ssh -i ~/my_path/clave_privada ubuntu@172.19.1.2:home/
```

ANEXO II. PUESTA EN MARCHA DEL CLÚSTER

Una vez se encuentren desplegadas todas las instancias de la nube privada puede comenzar la configuración mediante MPI de la implementación de un clúster de alto rendimiento en un cloud.

Como se ha mencionado en el proyecto, se ha procedido a implantar un clúster tipo Beowulf, basado en la computación paralela utilizando librerías de paso de mensaje (MPI) para que los procesos se ejecuten en múltiples procesadores siguiendo el paradigma cliente/servidor.

Se ha utilizado la implementación MPICH de MPI, concretamente la versión MPICH2 1.2.1, optimizada para entornos homogéneos, lo que proporciona un mayor rendimiento en el paso de mensajes entre nodos.

II.1 CONFIGURACIÓN INICIAL

Para poder instalar MPICH2 es necesario resolver algunas dependencias básicas, entre ellas el compilador gcc. Para ello instale el paquete *build-essential* en cada uno de los nodos que formen el clúster.

```
$ sudo apt-get install build-essential
```

Una vez resueltas estas dependencias, es necesario, que el nodo *master* de nuestro clúster pueda comunicarse por ssh con el resto de nodos, pero es necesario que se autentifique sin tener que escribir la contraseña.

Además es conveniente disponer de un directorio donde todos los nodos puedan compartir ficheros NFS (Network File System). El nodo master desempeña la función de servidor y el resto de nodos de clientes.

Conexión SSH sin contraseña

SSH proporciona varios métodos de acceso sin contraseña; el que se describe aquí se basa en el uso de criptografía de clave pública para identificar a los usuarios. En este apartado explicamos como generar el par de claves (identidades) e instalarlas en los distintos equipos.

En el nodo master, como usuario root y desde el directorio home, genere las claves privada y pública del sistema RSA. Para ello ejecute:

```
$ cd ~
$ $ ssh-keygen -b 4096 -t rsa
```

Copie la clave pública en cada uno de los nodos del clúster desde el *master*

```
$ scp .ssh/id_rsa.pub ubuntu@ip_nodo:.ssh/nueva_clave
```

Ahora acceda a cada uno del resto de nodos y copie la clave pública del *master* en *authorized_keys*

```
$ ssh ubuntu@ip_nodo:home/
$ cd .ssh/
$ cat nueva_clave >> authorized_keys
```

Para evitar problemas, copie en el *master* la clave pública en *authorized_keys.*

```
$ cat .ssh/id_rsa.pub >> authorized_keys
```

Y establezca, en cada nodo, los permisos de acceso al fichero:

```
$ chmod 0700 $HOME/.ssh/
$ chmod 0600 $HOME/.ssh/authorized_keys
```

Para comprobar que ya no se solicita contraseña al nodo master, desde éste acceda al resto de nodos del clúster

```
$ ssh ubuntu@ip_nodo
```

La primera vez que el nodo master acceda a cada uno del resto de nodos se almacena su *host_key* en el fichero *$HOME/.ssh/know_hosts* y a partir de ese momento puede conectarse sin necesidad de contraseña.

Servidor NFS

NFS es un sistema de archivos distribuido para un entorno de red de área local. Posibilita que distintos sistemas conectados a una misma red accedan a ficheros remotos como si se tratara de locales. A continuación se detalla la configuración del servidor y de los clientes.

Configuración en el servidor (nodo *master*)

Los paquetes necesarios para el funcionamiento del servidor son *portmap, nfs-kernel-server y nfs-common*. Para la descarga de los mismos ejecutaremos los siguientes comandos como usuario *root*:

```
$ apt-get install portmap
$ apt-get install nfs-kernel-server
$ apt-get aptitude install nfs-common
```

Para la configuración de un servidor de NFS necesita editar tres ficheros: ***/etc/exports, /etc/hosts.deny*** y ***/etc/hosts.allow***.

El fichero */etc/exports* contiene una línea por directorio a compartir. La estructura de dicha línea es:

```
directorio equipo1(opcion11,opcion12) equipo2(opcion21,opcion22)
```

donde:

- **directorio**: Es el directorio a compartir.
- **equipox**: Clientes que tendrán acceso al directorio compartido. Estos equipos se podrán indicar por su IP o dirección DNS.
- **optionxx**: Son las opciones que nos permitirán tener acceso a esos directorios con determinados privilegios:
  - `ro|rw` : Con la opción ro el directorio será compartido de solo lectura. Esta opción está por defecto.y con la opción rw se permitirá tanto acceso de lectura como de escritura.
  - `sync|async` : sync es la opción recomendada, ya que se ha de respetar el protocolo NFS, es decir, no se responden a las peticiones antes de que los cambios realizados sean escritos al disco. Con la opción async se permite mejorar el rendimiento y agilizar el funcionamiento global, pero supone un riesgo de corrupción de archivos o del sistemas de ficheros en casos de caidas del servidor y/o errores de éste.
  - `root_squash|no_root_squash|all_squash` : root_squash indica que un cliente identificado como root tendrá acceso al directorio con privilegios de un usuario anónimo. Si seleccionamos la opción

no_root_squash evitaremos esto, y si indicamos all_squash, entonces aplicaremos esto último a todos los usuarios, no sólo root.

En este caso el fichero queda de la siguiente forma:

```
/home 172.19.1.0(rw,sync,root_squash)
```

Permitiendo a todos los equipos de la red 172.19.1.0 (red privada de nuestro clúster) acceder con permiso de lectura y escritura al directorio */home* del nodo *master.*

Ahora modifique los ficheros */etc/hosts.allow* y */etc/hosts.deny.*

En el fichero */etc/hosts.deny* indique todas las restricciones posibles para hacer mas seguro el sistema. Para ello deniegue el acceso a *portmap*, ya que si se deniega, aunque permita nfs, no puede compartir porque éste depende de *portmap*. Por lo que solo tendrá acceso a *portmap* para aquellos equipos que estén definidos en el fichero */etc/hosts.allow*.

```
portmap:ALL
```

En el fichero */etc/hosts.allow* debe indicar los equipos que tiene acceso al servicio *nfs* y *portmap*. Se pueden indicar hosts individuales o una red.

```
portmap:172.19.1.0/255.255.255.0
nfs:172.19.1.0/255.255.255.0
```

Una vez configurados los ficheros inicie el servicio *portmap* y *rpc.nsfd*:

```
$ /etc/init.d/nfs-common restart
$ /etc/init.d/nfs-kernel-server restart
$ /etc/init.d/portmap restart
```

Configuración en los clientes (resto de nodos del clúster)

En el cliente monte el directorio exportado por el servidor, para ello ejecute el siguiente comando en cada uno de los nodos:

```
$ mount -t nfs ip_nodo_master:/home /home
```

Con el comando anterior se ha montado el directorio */home* exportado por el nodo master en el directorio */home* de cada uno del resto de nodos.

Si desea que el directorio remoto se monte al arranque del cliente debe añadir la siguiente línea al fichero */etc/fstab*:

```
$ ip_nodo_master:/home /home nfs defaults,rw 0 0
```

Esta línea indica que se monte en el directorio /home el directorio remoto /home

II.2 INSTALACIÓN MPICH2

Tras realizar la configuración inicial del clúster, el siguiente paso es la instalación de MPICH2, la cual se detalla en este paso, y que debe realizar en cada uno de los nodos del clúster.

Si tiene alguna versión previa de MPICH instalada en algunos de los nodos se debe realizar una limpieza previa en cada uno de los nodos implicados:

```
$ sudo aptitude purge mpich-bin mpich-mpd-bin mpich-shmem-bin libmpich1.0-dev
```

Descargue MPICH2 de su página oficial:

```
$ wget http://www.mcs.anl.gov/research/projects/mpich2/downloads/tarballs/1.2.1/
  mpich2-1.2.1.tar.gz
```

Descomprima el fichero descargado y configure los parámetros de instalación de MPICH2

```
$ tar -zxvf mpich2-1.2.1.tar.gz
$ cd mpich2-1.2.1/
$ ./configure --prefix=/home/mpich2 --with-device=ch3:sock
```

Y proceda a su instalación:

```
$ make
$ sudo make install
```

Establezca las variables de entorno en cada nodo:

```
$ export PATH=/home/mpich2/bin:$PATH
$ export LD_LIBRARY_PATH="/home/mpich2/lib:$LD_LIBRARY_PATH"
```

Añada el directorio */home/mpich2/bin* a la variable *PATH*

```
$ sudo vi /etc/environment
```

Y prepare el fichero de autenticación:

```
$ echo secretword=palabra_secreta >> ~/.mpd.conf
$ chmod 600 ~/.mpd.conf
```

II.3 INICIALIZACIÓN

Una vez instalado MPICH2 en cada uno de los nodos del clúster, debe configurar el servidor para indicarle cuáles son los nodos disponibles para ejecutar programas MPICH a través de un fichero de máquinas.

Para ello edite el fichero *mpd.hosts*, que debe guardar en el directorio *home* del nodo *master.* Este fichero contiene las IPs de cada uno de los nodos del clúster.

```
172.19.1.2
172.19.1.3
172.19.1.4
172.19.1.5
172.19.1.6
172.19.1.7
172.19.1.8
172.19.1.9
172.19.1.10
172.19.1.11
172.19.1.12
172.19.1.13
172.19.1.14
172.19.1.15
```

En este punto ya están preparados todos los equipos (14 instancias en el proyecto) para iniciar el proceso *mpd* y poder ejecutar cualquier programa de forma paralela.

```
$ mpdboot -n 14 -f ~/mpd.hosts
```

Si no hay ningún problema puede realizar diferentes test para comprobar el correcto funcionamiento paralelo del clúster:

```
$ mpdtrace
$ mpdringtest
$ mpdringtest 100
$ mpiexec -l -n 30 hostname
```

Para ejecutar un ejemplo real MPI, durante la instalación de MPICH2 se instala una carpeta con diferentes programas de ejemplo que puede probar en el clúster.

```
$ cd mpich2-1.2.1/examples
$ make clean
$ make
```

Pudiendo ejecutar el programa cpi, que devuelve el error de calcular el número pi entre todos los nodos.

```
$ /home/mpich2/bin/mpiexec -n 14 ~/mpich2-1.2.1/examples/cpi
```

Cuyo resultado es:

```
Process 2 of 14 is on 172.19.1.8
Process 5 of 14 is on 172.19.1.3
Process 6 of 14 is on 172.19.1.2
Process 13 of 14 is on 172.19.1.15
Process 12 of 14 is on 172.19.1.14
Process 11 of 14 is on 172.19.1.10
Process 0 of 14 is on 172.19.1.4
Process 4 of 14 is on 172.19.1.6
Process 3 of 14 is on 172.19.1.5
Process 1 of 14 is on 172.19.1.7
Process 7 of 14 is on 172.19.1.11
Process 9 of 14 is on 172.19.1.12
Process 10 of 14 is on 172.19.1.13
Process 8 of 14 is on 172.19.1.9
pi is approximately 3.1415926544231274, Error is 0.0000000008333343
Wall clock time 0.0096124
```

BIBLIOGRAFÍA Y ENLACES

A continuación es listada la documentación y páginas Web que han servido de referencia bibliográfica complementaria durante el desarrollo del proyecto.

Bibliografía:

- Rafael Francisco Benedicto Tovar. *Aplicación de metodologías de paralelización para la generación de tablas rainbow mediante la utilización de servidores de altas prestaciones en gnu/Linux*. Universidad de Almería. 2010.
- J. Gómez, F. Gil, E. Villar, F. Méndez. *Administración avanzada de sistemas informáticos.* RA-MA Editorial 2010.
- Shuai Zhang, Shufen Zhang, Xuebin Chen, Xiuzhen Huo. *Cloud Computing Research and Development Trend*. Second International Conference on Future Networks. 2010.
- Ana Cristina Guerrero Alemán, Elisa Karina Mena Maldonado. *Implementación de un Prototipo de Cloud Computing de Modelo Privado para ofrecer Infraestructura como Servicio*. Escuela Politécnica Nacional Quito. Febrero 2011.
- Sun Microsystems. *Introduction to Cloud Computing Architecture*. White Paper 1st Edition. Junio 2009.
- Enric Pagès Montanera. 1366: *Gestión sostenible de clústers de recursos virtuales*. Universitat Autònoma de Barcelona. Septiembre 2009.
- L. Gillam. *Cloud Computing: Principles, Systems and Applications*. Springer, 2010.
- T. Velte, A. Velte, T. J. Velte, Robert C. Elsenpeter. *Cloud Computing: A Practical Approach.* McGraw Hill Professional, 2009.
- Tim Jones. *Virtual Linux, An Overview of virtualization methods, architectures, and implementations.* Consultant Engineer, Emulex. IBM, 2006.
- B. Furht, A. Escalante. *Handbook of Cloud Computing*. Springer, 2010.

- R. Buyya. *High Performance Cluster Computing: Architectures and Systems. Volumen 1*. Prentice-hall PRT. 1999.
- R. Buyya. *High Performance Cluster Computing: Programing and applications. Volumen 2*. Prentice-hall PRT. 1999.
- B. Sotomayor R. S. Montero I. M. Llorente and I. Foster. *Virtual infrastructure management in private and hybrid clouds*. IEEE Internet Computing, in press, 2009.
- Daniel Nurmi Rich Wolski Chris Grzegorczyk Graziano Obertelli Sunil Soman Lamia Youse_ Dmitrii Zagorodnov. *The eucalyptus open-source cloud-computing system*. 9th IEEE International Symposium on Cluster Computing and the Grid, Shanghai, China, 2008.
- P. Oechslin. *Making a Faster Cryptanalytic Time-Memory Trade-Off*. LASEC. 2003.
- M.G. Piattini, E. Del Peso. *Auditoría Informática. Un enfoque práctico*. 2ª Revisión ampliada y revisada. Ra-Ma. 2001.
- Manuel J. Lucena López. *Criptografía y Seguridad en Computadores* 4ª Edición. Escuela Politécnica Superior. Universidad de Jaén.
- Michal Rjasko. *Properties of Cryptographic Hash Functions.* University Bratislava. 2006.
- José Miguel Alonso. *Programación de aplicaciones paralelas con MPI*. Universidad del País Vasco. Enero 1997.
- Vicente F. Reyes, José Antonio Jiménez. *Procesamiento Paralelo en Redes Linux. Utilizando MPI*. Julio 2003.
- NVIDIA, "*CUDA Compute Unified Device Architecture Programming Guide v2.0*", Septiembre 2008.
- David Miraut Andrés. *Procesadores gráficos CUDA modelo de programación y jerarquía de memoria*. Universidad Rey Juan Carlos. 2009.
- Lahabar, S., Narayanan, P.J. *Singular value decomposition on GPU using CUDA. Parallel & Distributed Processing*. IEEE International Symposium. 2009.
- J. D. Owers, M. Houston, D. Luebre, S. Green, J. E. Stone and J. C. Philips, *GPU computing*, IEEE, 96(5), 879-899, 2008.
- X. Wang, H. Yu. *How to Break MD5 and Other Hash Functions*. Shandong University, Jinan 250100, China. 2004.
- Hassan Alnoon, Shaima Al Awadi. *Executing Parallelized Dictionary Attacks on CPUs and GPUs*. 2009.
- Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf Harvey J. Wasserman, Nicholas J. Wright. *Performance Analysis*

of High Performance Computing Applications on the Amazon Web Services Cloud. 2nd IEEE International Conference on Cloud Computing Technology and Science. 2010.

- Christian Vecchiola, Suraj Pandey, Rajkumar Buyya. *High Performance Cloud Computing: A View of Scientific Applications*. 10th International Symposium on Pervasive Systems, Algorithms, and Networks. 2009.
- Gabriel Mateescu, Wolfgang Gentzsch, Calvin J. Ribbens. *Hybrid Computing – Where HPC meets grid and Cloud Computing*. Elsevir B.V. 2010.
- D. H. Bailey et al. *The NAS Parallel Benchmarks*. International Journal of Supercomputer Applications, Vol 5, No.3 (Fall1991), pp. 63-73, 1996.
- J. Gómez et Al. Cryptanalysis of Hash Functions Using Advanced Multiprocessing, Advances in Soft Computing. DOI:10.1007/978-3-642-14883-5_29. 2010.
- S. Halevi, H. Krawczyk. *Strengthening Digital Signatures via Randomized Hashing*. CFRG. Mayo 2005.
- A. Lenstra, X. Wang, B. de Weger. *Colliding X.509 Certificates, Cryptology*. ePrint Archive Report 2005/067, 1 Mar 2005, revised 6 May 2005. Retrieved July 27, 2008.

Enlaces:

- *Cloud Computing: Las TI como servicio*. Artículo Network World España, 2008.
  http://www.networkworld.es/Cloud-Computing:-Las-TI-como-servicio/seccion-recursos/articulo-191003
- *Cloud Computing Services - A comparison.* Art. Torry Harris Business Solutions.
  http://www.thbs.com/pdfs/Comparison%20of%20Cloud%20computing%20services.pdf
- Jon Brodkin, *Private cloud networks are the future of corporate IT*. Artículo Network World, 2008.
  http://www.networkworld.com/news/2008/111208-private-cloud-networks.html
- Mark Baker. *Cluster Computing White Paper*. University of Portsmouth 2000.
  http://www.csa.syr.edu/~chapin/papers/pdf/ieee_tfcc_wp_clustercomputing.pdf
- Sitio Web oficial del hipervisor Xen
  http://www.xen.org/
- Sitio Web oficial de KVM, solución de virtualización a nivel del kernel
  http://www.linux-kvm.org/
- Eucalyptus – Página oficial
  http://www.eucalyptus.com/
- Ubuntu Enterprise Cloud – Página oficial
  http://www.ubuntu.com/business/cloud/overview

- Opennebula 2.0 – Página oficial
  http://www.opennebula.org/
- Enomaly´s Elastic Computing Platform – Página oficial
  http://www.enomaly.com/
- BitNami – Página oficial
  http://www.bitnami.org/
- OpenQRM – Página oficial
  http://www.openqrm-enterprise.com/
- CloudStack – Página oficial
  http://www.cloudstack.org/
- Openstack – Página oficial
  http://www.openstack.org/
- Web oficial de la tecnología Intel VT en Intel
  http://www.intel.com/technology/virtualization/
- Beowulf.org: The Beowulf Cluster Site
  http://www.beowulf.org/
- LVS Introduction - Load Balancing Server Cluster
  http://www.linuxvirtualserver.org/whatis.html
- OpenSSI (Single System Image) Clusters for Linux
  http://openssi.org/cgi-bin/view?page=openssi.html
- Grid Computing Info Centre (GRID Infoware)
  http://www.gridcomputing.com/
- Message Passing Interface Forum
  http://www.mpi-forum.org/
- MPICH2 *: High-performance and Widely Portable MPI*.
  http://www.mcs.anl.gov/research/projects/mpich2/
- Open MPI*: Open Source High Performance Computing.*
  http://www.open-mpi.org/
- LAM/MPI Parallel Computing.
  http://www.lam-mpi.org/
- Ubuntu Enterprise Cloud Community
  https://help.ubuntu.com/community/UEC
- Artículo configuración Ubuntu Enterprise Cloud 10.10
  http://elblogdepicodev.blogspot.com/2010/10/ubuntu-enterprise-cloud-1010.html
- *HybridFox (Fork of ElasticFox) configuration for UEC*
  https://help.ubuntu.com/community/UEC/ElasticFox
- Servidor de ficheros NFS. Configuración
  http://blackhold.nusepas.com/2010/03/servidor-de-ficheros-nfs/

- NAS Parallel Benchmarks- NAS Division - NASA
http://www.nas.nasa.gov/Resources/Software/npb.html
- Benchmark MPI mflops
http://www.generacio.com/cluster/mflops.c
- Amazon – Página oficial
http://aws.amazon.com/es/
- Amazon S3 – Página oficial
http://aws.amazon.com/es/s3/
- R. Rivest, *The MD5 Message-Digest Algorithm*. Network Working Group 1992.
http://www.ietf.org/rfc/rfc1321.txt
- D. Eastlake. US Secure Hash Algorithm 1 (SHA1). Network Working Group 2001.
http://www.ietf.org/rfc/rfc3174.txt
- Página oficial de la librería openSSl.
http://www.openssl.org/
- Sergio S. Mitos y leyendas: Las contraseñas en Windows III (LM y NTLM).
http://www.hispasec.com/unaaldia/3464.
- Página oficial de Nvidia – CUDA.
http://www.nvidia.es/object/cuda_home_new_es.html.
- Deyuan Qiu. *GPGPU: The Art of Acceleration*. Tutorial.
http://www2.inf.h-brs.de/~dqiu2s/docu/GPGPU_v0.2_qiu.pdf
- Página oficial del Proyecto RainbowCrack
http://project-rainbowcrack.com/
- How Rainbow Tables work
http://kestas.kuliukas.com/RainbowTables/
- Página oficial del proyecto Free Rainbow Tables.
http://www.freerainbowtables.com/faq/.
- La enciclopedia libre
http://es.wikipedia.org/
- Wiki sobre la Administración de Sistemas Operativos
http://www.adminso.es/wiki/