Universitat Oberta de Catalunya

OpenNebula y Hadoop: Cloud Computing con herramientas Open Source

[Memoria del proyecto]

[Francisco Magaz Villaverde]

Consultor: Victor Carceler Hontoria

Junio 2012

I. Resumen

Cloud Computing (Computación en la nube) es un componente esencial en la llamada Web 2.0. Tanto los usuarios domésticos como las empresas se han acostumbrado a servicios y aplicaciones en la Red. Esto ha provocado que nuevas herramientas y estándares hayan aparecido para cumplir estas expectativas. Algunos de los aspectos clave de Cloud Computing están relacionados con la gestión de data-centers, reducción de costes, alta disponibilidad y seguridad y privacidad.

Como no podía ser de otra forma la comunidad de desarrolladores Open Source también está trabajando en este campo.

Hoy en día existen numerosas soluciones para implementar Cloud Computing pero para el desarrollo del presente proyecto vamos a centrarnos en proyectos Open Source. Entre las diversas opciones se ha escogido OpenNebula (www.opennebula.org) y Hadoop (http://hadoop.apache.org). No sólo son Open Source estos proyectos sino la tecnología sobre la que se despliegan como la virtualización (Xen) o el Sistema Operativo donde se ejecutan (GNU/Linux).

II. Índice de contenido

III. Índice de figuras

IV. Introducción

1. Descripción del proyecto

El presente proyecto lleva por título "**OpenNebula y Hadoop: Cloud Computing con herramientas Open Source**". Se divide en tres bloques principales. El primer bloque consiste en una introducción y descripción de lo que es el Cloud Computing, los diferentes modelos que existen así como las herramientas Open Source disponibles actualmente en el mercado. En este primer bloque se lleva a cabo un estudio de las ventajas e inconvenientes que aporta este nuevo paradigma tecnológico, así como los servicios que puede prestar. A continuación, en el segundo bloque se estudia en profundidad las herramientas **OpenNebula** dentro del modelo IaaS (*Infraestructure as a Service*) y **Hadoop** dentro del modelo PaaS (*Platform as a Service*). Y para finalizar, como parte práctica del proyecto, se lleva a cabo la instalación, integración, configuración y puesta en marcha de una plataforma Cloud Computing utilizando OpenNebula y Hadoop con el objetivo de aplicar los conceptos teóricos en una solución real dentro de un entorno de laboratorio que puede ser extrapolable a una instalación real.

2. Justificación de la tecnología elegida

Debido a la variedad de herramientas Open Source para la implementación de Cloud Computing se ha elegido dos en concreto para su puesta en marcha de forma práctica.

OpenNebula dispone de versiones para varias distribuciones de GNU/Linux y tiene soporte para hipervisores de virtualización como Xen, KVM y VMware. Esto nos permite flexibilidad a la hora de elegir que tecnología utilizar para implantar OpenNebula.

En cuanto al modelo PaaS, Hadoop está inspirado en el framework MapReduce de Google para computación paralela, y actualmente es utilizado por firmas como Yahoo y Facebook.

3. Objetivos del proyecto

El objetivo principal del proyecto es una introducción a las tecnologías Open Source de Cloud Computing. Empezando por una descripción teórica de las características comunes de toda herramienta de Cloud Computing y las diferentes opciones existen actualmente en el mercado. Se continua con estudio más en profundidad de las soluciones OpenNebula y Hadoop, poniendo el énfasis en las ventajas que aporta a las organizaciones que se deciden por utilizar dichas herramientas. Una vez realizado dicho estudio inicial se concluye con la instalación práctica de una maqueta de laboratorio utilizando OpenNebula y Hadoop.

Los resultados obtenidos se distribuyen entre los siguientes apartados que conforman el índice general del proyecto:

- Estado del arte del Cloud Computing. ¿Qué es Cloud Computing? y diferentes herramientas Open Source disponibles en el mercado.

- Estudio de OpenNebula como ejemplo del modelo IaaS (*Infraestructure as a Service*).

- Estudio de Hadoop como ejemplo del modelo PaaS (*Platform as a Service*).

- Aplicación práctica: Instalación e integración de OpenNebula y Hadoop. Cada máquina virtual controlada con OpenNebula será un nodo Hadoop.

4. Planificación

A partir de una estimación inicial se han identificado las siguientes tareas principales y su duración así como los entregables del proyecto.

4.1 Tareas principales del proyecto

Definición del proyecto

- Objetivo: Definir los objetivos y el alcance del proyecto.

- Duración: 8 días.

- Producto: Índice del proyecto junto con los objetivos principales.

Investigación inicial

- Objetivo: Llevar a cabo una fase de investigación de conceptos teóricos a partir de documentos y estudios previos acerca del tema del proyecto.

- Duración: 8 días.

- Producto: Bibliografía básica para desarrollar tanto la parte teórica como práctica del proyecto.

Estado del arte del Cloud Computing

- Objetivo: Descripción de Cloud Computing en cuanto a su historia, características, tecnologías involucradas y ventajas que aporta al mundo de la computación.

- Duración: 8 días.

- Producto: Primera parte del proyecto dedicada al estudio y estado del arte del Cloud Computing.

Estudio de OpenNebula

- Objetivo: Profundización en las herramientas Open Source para el Cloud Computing utilizando como ejemplo OpenNebula.

- Duración: 12 días.

- Producto: Segunda parte del proyecto dedicada a la profundización en dos herramientas Open Source.

Estudio de Hadoop

- Objetivo: Profundización en las herramientas Open Source para el Cloud Computing utilizando como ejemplo Hadoop.

- Duración: 8 días.

- Producto: Segunda parte del proyecto dedicada a la profundización en dos herramientas Open Source.

Integración OpenNebula y Hadoop

- Objetivo: Diseño, instalación e integración de las herramientas OpenNebula y Hadoop en una maqueta de laboratorio.

- Duración: 22 días.

- Producto: Tercera parte del proyecto en la que se aplican los conocimientos teóricos adquiridas en las capítulos previos del proyecto.

4.2 Entregables

PEC1: Plan de trabajo

Se define el alcance y objetivos del proyecto, una primera aproximación de las tareas en las que se divide y una planificación temporal de estas tareas.

PEC2

En esta segunda entrega se incluye el índice completo del proyecto, el capítulo dedicado al estado del arte del Cloud Computing así como el estudio de la herramienta OpenNebula.

PEC3

En la entrega previa al final del proyecto se incluye el estudio de la herramienta Hadoop y toda la documentación técnica generada en la instalación e integración de OpenNebula y Hadoop en una plataforma de laboratorio.

Memoria final

Esta última entrega se compone del proyecto final e incluye toda la documentación generada en las entregas previas además de las correcciones y revisiones detectadas previas al cierre del proyecto.

5. Calendario de trabajo

En el siguiente diagrama de Gantt se muestra la planificación temporal que podrá ir variando a lo largo del desarrollo del proyecto. Cada entregable se marca como un hito en el diagrama.

Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	Redacción de la memoria del proyecto	67 días	jue 01/03/12	vie 01/06/12	
2	**PEC1: Plan de trabajo**	**8 días**	**jue 01/03/12**	**lun 12/03/12**	
3	Definición del proyecto	8 días	jue 01/03/12	lun 12/03/12	
4	Investigación inicial	8 días	jue 01/03/12	lun 12/03/12	
5	Entrega PEC1	0 horas	lun 12/03/12	lun 12/03/12	
6	**PEC2: Entrega memoria**	**20 días**	**mar 13/03/12**	**lun 09/04/12**	
7	Índice del proyecto	1 día	mar 13/03/12	mar 13/03/12	
8	Estudio tecnologías Cloud Computing	8 días	mar 13/03/12	jue 22/03/12	
9	Estudio Opennebula	12 días	vie 23/03/12	lun 09/04/12	
10	Entrega PEC2	0 horas	lun 09/04/12	lun 09/04/12	
11	**PEC3: Entrega memoria**	**30 días**	**mar 10/04/12**	**lun 21/05/12**	
12	Estudio Hadoop	8 días	mar 10/04/12	jue 19/04/12	
13	**Integración Opennebula y Hadoop**	**22 días**	**vie 20/04/12**	**lun 21/05/12**	
14	Instalación Opennebula	5 días	vie 20/04/12	jue 26/04/12	
15	Instalación hipervisor Xen	3 días	vie 27/04/12	mar 01/05/12	
16	Instalación Hadoop	10 días	mié 02/05/12	mar 15/05/12	
17	Integración global de la solución	2 días	mié 16/05/12	jue 17/05/12	
18	Pruebas y conclusiones	2 días	vie 18/05/12	lun 21/05/12	
19	Entrega PEC3	0 horas	lun 21/05/12	lun 21/05/12	
20	**Entrega final**	**16 días**	**mar 22/05/12**	**mar 12/06/12**	
21	Corrección de errores y revisión	10 días	mar 22/05/12	dom 03/06/12	
22	Presentación Powerpoint	3 días	lun 04/06/12	mié 06/06/12	
23	Imprevistos	5 días	mié 06/06/12	mar 12/06/12	
24	Fin proyecto	0 horas	mar 12/06/12	mar 12/06/12	

Figura 1: Diagrama de Gantt del Proyecto

V. Cloud Computing

1. ¿Qué es Cloud Computing?

En los últimos años el término Cloud Computing se ha puesto de moda en el mundo de las nuevas tecnologías. Se asocia tanto a servicios de almacenamiento como a diversas aplicaciones publicadas por diferentes proveedores (Google, Amazon, etc.)

Como inicio del proyecto, el primer paso es definir claramente que es Cloud Computing así como sus características. En la literatura actual podemos encontrar diversas definiciones de lo que se entiende por Cloud Computing.

TIA[1], define Cloud Computing como el uso de diferentes recursos (servidores, datos, aplicaciones) en Internet.

Una definición más completa y exhaustiva es la expuesta por el NIST[2]:

Lo define como el modelo que permite el acceso ubicuo, práctico y bajo demanda, a través de la red, a un *pool* común de recursos configurables de computación (redes, servidores, almacenamiento, aplicaciones, servicios, etc.) que pueden ser rápidamente puestos a disposición del usuario sin necesidad de una gran complejidad en la gestión con el proveedor del servicio.

Este modelo está compuesto de cinco características esenciales, tres modelos de servicio y cuatro modelos de despliegue que veremos más adelante.

En resumen, podríamos definir Cloud Computing como el nuevo paradigma computacional que pretende el uso compartido de recursos (procesamiento, almacenamiento, servicios) a través de Internet (también redes privadas) de la forma más transparente posible para el usuario. El Cloud Computing es un paso más en la computación distribuida en contraposición a la tradicional arquitectura cliente-servidor.

Desde un punto de visa menos académico, podemos decir que esta tecnología permite a los consumidores y empresas acceder tanto a sus datos como a servicios desde cualquier lugar sin necesidad de instalar ninguna aplicación en su dispositivo de conexión (PC, Tablet, teléfono móvil).

---

[1] TIA: La Asociación de la Industria de Telecomunicaciones es una organización encargada de desarrollar estándares y políticas en el ámbito de las Tecnologías de la Información y las Comunicaciones (TIC). Forman parte de ella alrededor de 400 compañías.

[2] NIST: El Instituto Nacional de Normas y Tecnología es la agencia de la Administración de Tecnología del Departamento de Comercio de los Estados Unidos encargada de promover la innovación y la competencia industrial en Estados Unidos mediante avances en metrología, normas y tecnología.

2. Historia

Este paradigma aparece como consecuencia de que el usuario no dispone de suficiente capacidad de procesamiento, almacenamiento y servicios para satisfacer sus necesidades de negocio. Como solución se propone “alquilar” estos recursos desde cualquier lugar a través de Internet a un proveedor basándose en las necesidades concretas del momento.

Cloud Computing ha sido una evolución de la computación en Grid y la computación distribuida. El problema de las tecnologías existentes dentro de estos campos es que son fuertemente acoplados.

Uno de los precursores fue la empresa Amazon cuando se dio cuenta que los equipos de sus data-centers no se utilizaban más allá del 10%. Pensó que una buena solución sería alquilarlas para que fueran utilizados por clientes externos. Así nació Amazon Web Service en 2006 que permitía alquilar sus equipos para realizar procesamiento distribuido. A partir de aquí nacieron otros productos como Amazon EC2 y Amazon S3.

En 2008 aparecen las primeras soluciones Open Source como Eucalyptus o OpenNebula.

3. Características de la tecnología

- Servicio bajo demanda: El cliente paga solo por lo que usa. No tiene que comprar hardware para cubrir su máxima capacidad. Puede obtener más procesamiento o almacenamiento de forma automática sin necesidad de desplegar más equipos en su organización. Esto supone una reducción de los costes.

- Elasticidad: La nube se encarga de gestionar el uso de CPU, recursos de red o almacenamiento. Si hay pocos usuarios Cloud Computing se adapta a esta situación al igual que si aumenta el número de usuarios concurrentes.

- *Pool* de recursos: Para el cliente es como si dispusiese de recursos limitados

- Acceso por red: Supone un mecanismo estándar para la heterogeneidad actual de dispositivos de acceso (PC, Tablet, teléfonos móviles). Lo servicios en la nube permite abstraerse de todo esto.

- Recursos compartidos: Al compartir los recursos con otros clientes los gastos fijos de una infraestructura TIC (electricidad, espacio en edificios) se reparte entre todos los clientes con lo que el coste de esta solución es mucho menor.

4. Ventajas

Podemos destacar las siguientes ventajas principales de Cloud Computing:

- Escalabilidad: Permite añadir nuevos componentes a la infraestructura de forma transparente y sin que el servicio se vea afectado.

- Coste: Al contratar sólo el procesamiento y almacenamiento necesario no se están pagando los costes que supone tener equipos infrautilizados.

- Confiabilidad: Cloud Computing está diseñado sobre la base de la redundancia de la información y alta disponibilidad para dar un servicio sin caídas 24 h.

- Integración: Cloud Computing está preparado para integrarse con otras soluciones empresariales de las que ya disponga el cliente.

- Rapidez de despliegue: El cliente puede disponer de una infraestructura computacional (procesamiento y almacenamiento) funcionando al 100 % en cuestión de días, sin mal gastar meses y meses creando la suya propia. Además le proporciona un excelente banco de pruebas sin necesidad de invertir tiempo y dinero y software y hardware.

- Simplicidad: La mayoría de las aplicaciones se acceden a través de Internet por lo que el hardware necesario es mínimo así como las necesidades de almacenamiento local.

- Actualizaciones automáticas: Este proceso complejo y lleno de riesgos en las organizaciones se simplifica con Cloud Computing al ser solo necesario aplicar las actualizaciones en un solo punto de la infraestructura.

5. Inconvenientes

Entre los inconvenientes o desventajas de esta tecnología podemos destacar:

- Dependencia del proveedor: Al externalizar las aplicaciones y datos de la organización se depende totalmente de proveedor del servicio tanto por la necesidad de la disponibilidad de los datos como del tratamiento que hace de los datos cedidos

- Localización de los datos: El cliente debe poder acceder en todo momento a los datos independientemente de donde esté almacenado. Aunque es una de las ventajas de Cloud Computing también puede ser un riesgo.

- Protección de la información: Los datos de la organización están almacenados en sistemas externos y esto pude provocar fugas de información si el proveedor del servicio no presta especial atención a este aspecto. Además viajan a través de una red pública con lo que no se sabe porque sistema está pasando la información. Utilizar mecanismo como *HTTPS* puede degradar el servicio.

- Fiabilidad: Si hay una caída de los servidores del proveedor o de la red no podemos hacer nada, solo esperar a que el proveedor corrija la incidencia.

- Cuestiones legales: Acatamiento de las leyes de protección y seguridad de la información, independientemente de que ésta varíe según las normas propias del país donde se localicen los datos y aplicaciones del usuario.

- Cierre del proveedor: Una quiebra del proveedor pude suponer no sólo la indisponibilidad del servicio sino también una perdida de los datos. Es necesario un plan de contingencia para ese supuesto.

6. Modelos de servicio

Como hemos visto en la definición que da el NIST acerca de Cloud Computing, la divide en tres modelos de servicio.

De forma gráfica podemos ver los tres modelos de la siguiente manera:

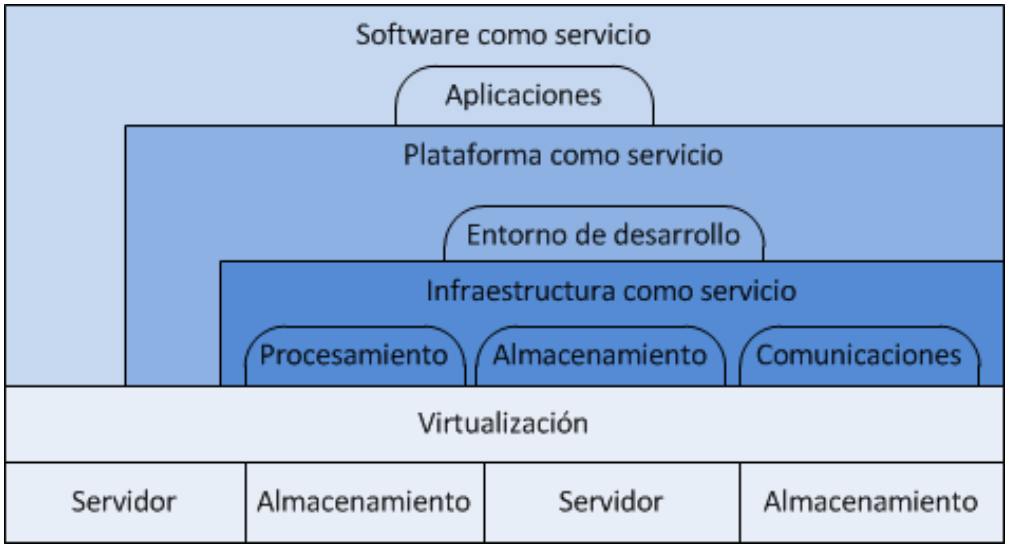

Figura 2: Modelos de servicio en la nube

6.1 Software como Servicio (SaaS)

En este modelo el proveedor instala y publica aplicaciones que serán utilizadas por los usuarios de la nube a través de clientes ligeros (normalmente cliente web o aplicaciones para dispositivos móviles). El usuario ni tiene acceso ni gestiona la infraestructura donde se ejecutan las aplicaciones.

Esto permite al cliente olvidarse de todas la gestión de actualización, tanto del software base como de las aplicaciones, siendo responsabilidad del proveedor.

Aplicaciones que se ajusten a este modelo son aplicaciones de tipo CRM, ERP o soluciones de BackOffice.

Entre las soluciones Open Source que encontramos dentro de este modelo están Phreebooks, Openl o Jaspersoft.

6.2 Plataforma como Servicio (PaaS)

PaaS consiste en una plataforma virtualizada que incluye Sistemas Operativos, aplicaciones específicas (por ejemplo motores de Bases Datos) y herramientas de desarrollo. Permite al usuario desplegar sus propias aplicaciones usando las librerías y herramientas que el proveedor del servicio pone a su disposición. El cliente no controla la infraestructura que hay por debajo, donde corren sus aplicaciones, pero tiene el control del despliegue de sus aplicaciones y la posibilidad de configurar el entorno de ejecución.

El desarrollo de software se lleva acabo usando software, así qué PaaS y SaaS están relacionados, pero PaaS incluye software específico para el desarrollo, y las actividades relacionadas como testeo y despliegue.

Además, con SaaS, la aplicación está controlada y desarrollada por el host, pero con PaaS es el cliente el que controla que aplicaciones se crean despliegan en la infraestructura del proveedor.

PaaS ofrece servicio de integración con Bases de Datos, servidores web, mecanismos de seguridad, almacenamiento, copias de seguridad y control de versión entre otros.

Entre las soluciones Open Source que encontramos dentro de este modelo están OpenShift, Cloud Foundry o Hadoop.

Más adelante se entrará en profundidad en Hadoop.

6.3 Infraestructura como Servicio (IaaS)

Este es el servicio más básico dentro de Cloud Computing y Consiste en proporcionar una infraestructura de computación. Como computación podemos entender procesamiento, almacenamiento y recursos de red. Ofrece la posibilidad, sobre dicha infraestructura, para desplegar aplicaciones y manejar recursos, todo ello gracias a la virtualización.

Nos permite abstraernos de las limitaciones puede imponer el hardware físico.

El cliente despliega sus Sistemas Operativos y aplicaciones en los recursos que Cloud Computing expone. Él es el responsable de administrar y actualizar los Sistemas Operativos y aplicaciones.

Entre las soluciones Open Source que encontramos dentro de este modelo están OpenStack, Eucalyptus, Nimbus u OpenNebula.

Más adelante se entrará en profundidad en OpenNebula.

7. Modelos de despliegue

En cuanto a los modelos de despliegue podemos identificar tres claramente diferenciados:

- Público: Las Aplicaciones, almacenamiento y otros recursos son proporcionados por el proveedor del servicio. Este servicio puede ser gratuito o por uso. El acceso a estos recursos se hace exclusivamente por la red pública Internet.

- Híbrido: Es la unión de una nube pública y otra privada. En este modelo se intenta aprovechar las ventajas de la nube pública en cuanto a reducción de costes combinándola con las ventajas en cuanto a seguridad de las nubes privadas.

- Privado: Toda la infraestructura, equipos redes, almacenamiento pertenece a la organización ya se propio o externalizado. El mayor inconveniente es que se pierden muchas ventajas del Cloud Computing como la reducción de costes o la simplificación de la gestión pero se mejora en la seguridad de los datos. Son una buena opción para organizaciones que requieren de una alta protección de sus datos.

8. Virtualización

Aunque no es el objetivo del Proyecto vamos a exponer una visión general de esta tecnología ya que es uno de los pilares del Cloud Computing.

La virtualización es la creación, a través de software, de una versión virtual de algún recurso tecnológico, como puede ser una plataforma de hardware, un sistema operativo, un dispositivo de almacenamiento u otros recursos de red.

Una capa software, hipervisor, abstrae los recursos de una computadora dividiéndolos en varios entornos de ejecución. Esta capa se encuentra entre el hardware (*host*) y el Sistema Operativo de la máquina virtual (*guest*).

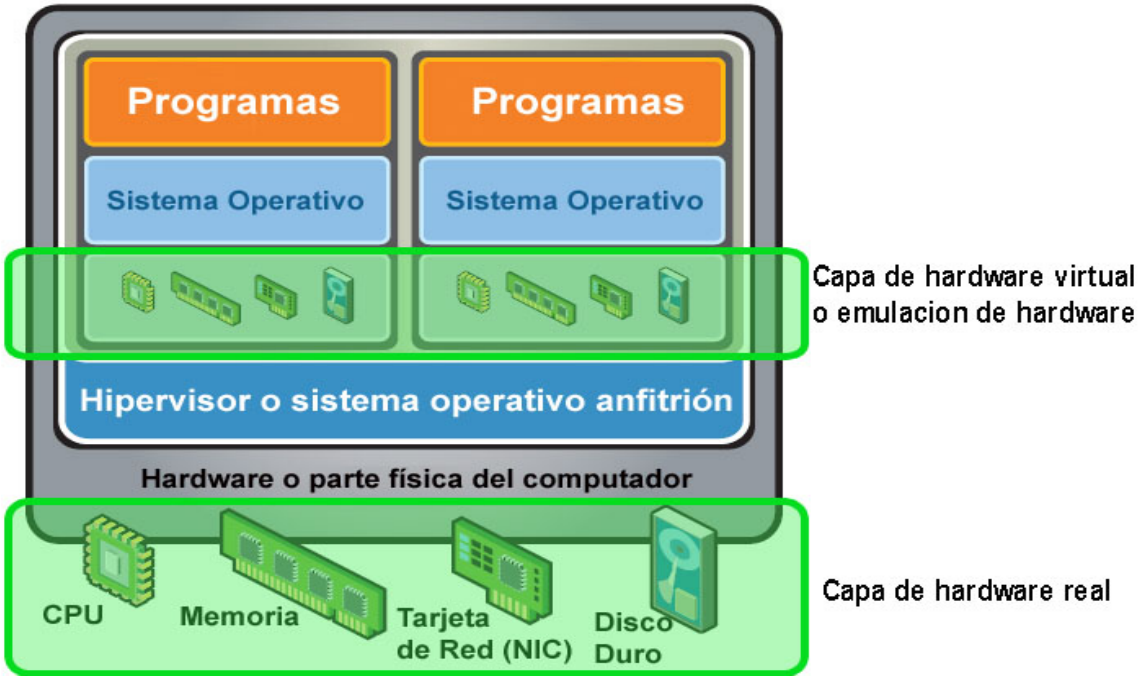

Figura 3: Virtualización completa o por hardware[3]

Aunque la virtualización no es una tecnología reciente, con la consolidación del Cloud Computing, la virtualización ha pasado a ser uno de los componentes fundamentales, especialmente en lo que se denomina infraestructura de nube privada.

Existen tres tipos de virtualización:

- Completa: El hipervisor simula un hardware en el que es posible ejecutar un sistema operativo *guest* sin modificar, es decir, como si se estuviese ejecutando en el hardware nativo. Podemos destacar el hipervisor Xen, VirtualBox (ambos siempre que tengan soporte hardware como *Intel VT-x* o *AMD-V*), KVM o VMWare WorkStation.

[3] Fuente: http://www.vmlogia.com/tiposdev.aspx

- Parcial: Acepta compartir recursos y alojar procesos pero no permite instancias separadas de Sistemas Operativos *guest*.

- Por S.O.: Se crean entornos virtuales en un único servidor físico. El Sistema Operativo sabe que se está ejecutando en un entorno virtual.

8.1 Ventajas

- Ahorro de costes: Ejecutar varias máquinas virtuales en un mismo hardware, sin necesidad de comprar más equipos.

- Ahorro energético: El gasto energético que suponen varios servidores se puede reducir a uno sólo. En grandes data-centers este aspecto es más importante.

- Incorporación rápida de nuevos recursos. Instalar un nuevo servidor es más sencillo que uno físico.

- Mantenimiento: Se centraliza en un solo punto y se simplifica.

- Balanceo de carga: Se puede distribuir el uso de CPU, almacenamiento o recursos de red entre las distintas máquinas virtuales desplegadas.

8.2 Desventajas

- Rendimiento: Un máquina virtual nunca va a ofrecer el mismo rendimiento que una máquina física.

- Hardware: No se puede utilizar hardware que no esté soportado por el hipervisor.

- Disponibilidad: Un fallo en la máquina física afecta a varias máquinas y servicios de la infraestructura.

8.3 Virtualización Open Source

De entre las soluciones de virtualización Open Source que existen actualmente en el mercado podemos destacar:

- Xen

- KVM

- VirtualBox

- OpenVZ

9. Cloud Computing y Open Source

Modelo	Nombre	Descripción
IaaS	OpenNebula	Proyecto iniciado en la Universidad complutense de Madrid.
	OpenStack	Se basa en el proyecto Nebula de la NASA y RackSpace. Surge de su necesidad de manejar grandes cantidades de datos.
	Eucalyptus	Se inició como un proyecto de investigación en la Universidad de California,
	Nimbus	Plataforma enfocada a la comunidad científica.
PaaS	OpenShift	Producto desarrollado por Red Hat.
	Cloud Foundry	Desarrollado por VMWare bajo licencia Apache.
	Hadoop	Producto desarrollado por Apache. Yahoo es el mayor contribuyente.
SaaS	OpenBravo	ERP destinado a pequeñas y medianas empresas
	Phreebooks[4]	Solución de contabilidad y ERP basada en Web.
	Openi[5]	Proporciona a los usuarios visualizaciones de datos OLAP y bases de datos relacionales.
	Jaspersoft[6]	Solución de Business Intelligence.

[4] http://www.phreesoft.com
[5] http://openi.org/product/
[6] http://www.jaspersoft.com/

VI. IaaS: OpenNebula

1. Introducción e Historia

Como ejemplo de una herramienta Open Source que cubra el modelo de servicio *IaaS* hemos escogido OpenNebula.

OpenNebula es una solución Open Source (bajo licencia Apache v2) que permite implementar fácilmente infraestructuras Cloud Computing privadas (también híbridas) según el modelo *IaaS*. Su parte principal consiste en software que permite desplegar máquinas virtuales sobre un *pool* de máquinas físicas. Además está diseñado para integrarse con otras soluciones de almacenamiento y de red. Así, maneja tanto las transferencia de máquinas virtuales como la configuración de la red, el almacenamiento y su gestión.

OpenNebula fue inicialmente desarrollado por la Universidad Complutense de Madrid en 2008. Más adelante el número de participantes ha ido creciendo y más organizaciones se han unido al desarrollo del proyecto. Algunos destacados contribuidores son IBM, Suse o AT&T.

Proporciona una plataforma de Cloud Computing escalable, segura y rápida de desplegar y permite manejar y construir infraestructuras publicas, privadas e híbridas siguiendo el modelo *IaaS*. Es una solución se ajusta muy bien a los data-centers actuales.

Los siete pilares básicos de OpenNebula son:

- Almacenamiento: Permite almacenar las imágenes de discos virtuales en repositorios desde donde serán usadas para desplegar rápidamente máquinas virtuales o compartidas con otros usuarios. Estas imágenes de disco pueden ser tanto de Sistemas Operativos o de datos.

- Repositorio de plantillas: Es donde se almacenan plantillas de máquinas virtuales, con sus características para ser instanciadas más tarde en el hipervisor elegido.

- Redes virtuales: Soporta el manejo de redes virtuales que interconectará las diferentes máquinas virtuales, pudiendo definir una IP concreta o rangos de IP para cada red.

- Manejo de máquinas virtuales: Una vez que se ha desplegado una instancia de una plantilla en un hipervisor (*host*) se puede controlar todo su ciclo de vida como el arranque, parada, clonación y apagado.

- Clústeres: Son *pools* de *host* que comparten almacenamiento y redes virtuales. Se utilizan para el balanceo de carga y alta disponibilidad y rendimiento.

- Usuarios y grupos: Soporta la definición de usuarios y grupos para el acceso a los recursos así como mecanismo de autenticación. También implementa ACL para la asignación permisos.

- API: Proporciona interfaces de comunicación con las diferentes funcionalidades ofrecidas a través desde herramientas de línea de comando o través del GUI Web Sunstone.
  Además proporciona interfaces para interactuar con otras infraestructuras Cloud Computing públicas como OCCI[7] y EC2[8] con lo que permite el despliegue de nubes híbridas.

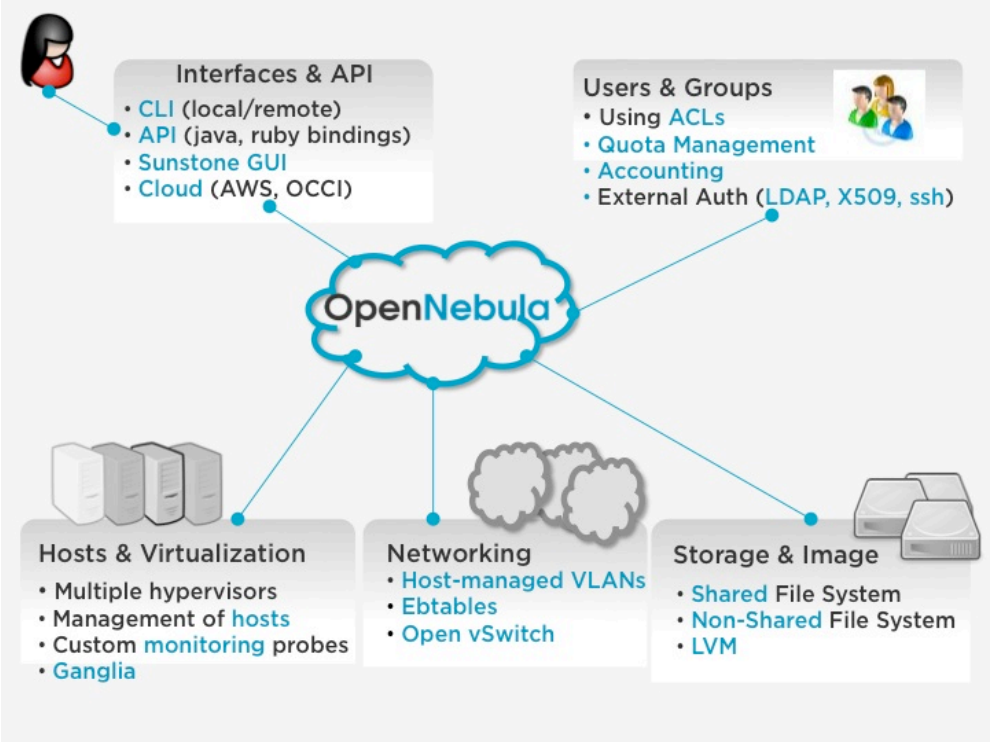

Figura 4: Componentes de OpenNebula[9]

Destaca principalmente por su flexibilidad para adaptarse a diferentes escenarios. En parte, esta flexibilidad la consigue gracias a que permite la utilización de diferentes soluciones de virtualización o hipervisores como Xen, KVM o VMWare. Estas tecnologías de virtualización son el corazón de OpenNebula.

---

[7] Open Cloud Computing Interface: Conjunto de especificaciones para crear una interface común entre diferentes plataformas de Cloud Computing dentro del modelo *IaaS*.

[8] Amazon Elastic Cloud Computing: Servicio web de Amazon que permite alquilar máquinas virtuales para que los usuarios ejecuten sus aplicaciones.

[9] Fuente: http://opennebula.org/documentation:archives:rel3.0:intro

2. Gestión de máquinas virtuales

Actualmente OpenNebula permite el control de tres tipos diferentes de hipervisores, Xen, KVM y VMWare, a través de sus respectivos drivers. Cada hipervisor registrado en el sistema se le denomina *host* o anfitrión. Así podemos tener varios *host* (máquinas físicas) y dentro de estas correr varias máquinas virtuales.

Una máquina virtual en OpenNebula consiste de las siguientes partes:

- Capacidad de procesamiento en términos de uso de CPU.
- Un conjunto de dispositivos de red asociados a una red virtual.
- Un conjunto de imágenes de disco.
- Un fichero de recuperación donde se almacena la memoria de la máquina virtual que se está ejecutando e información del hipervisor.

Toda esta información, más características específicas del Sistema Operativo, se almacenan en una plantilla que procesará OpenNebula para crear la máquina virtual.

Desde OpenNebula se puede gestionar el ciclo de vida de estas máquinas. Los estados posibles de un máquina virtual lo podemos ver como una máquina de estados representado por el siguiente grafo:

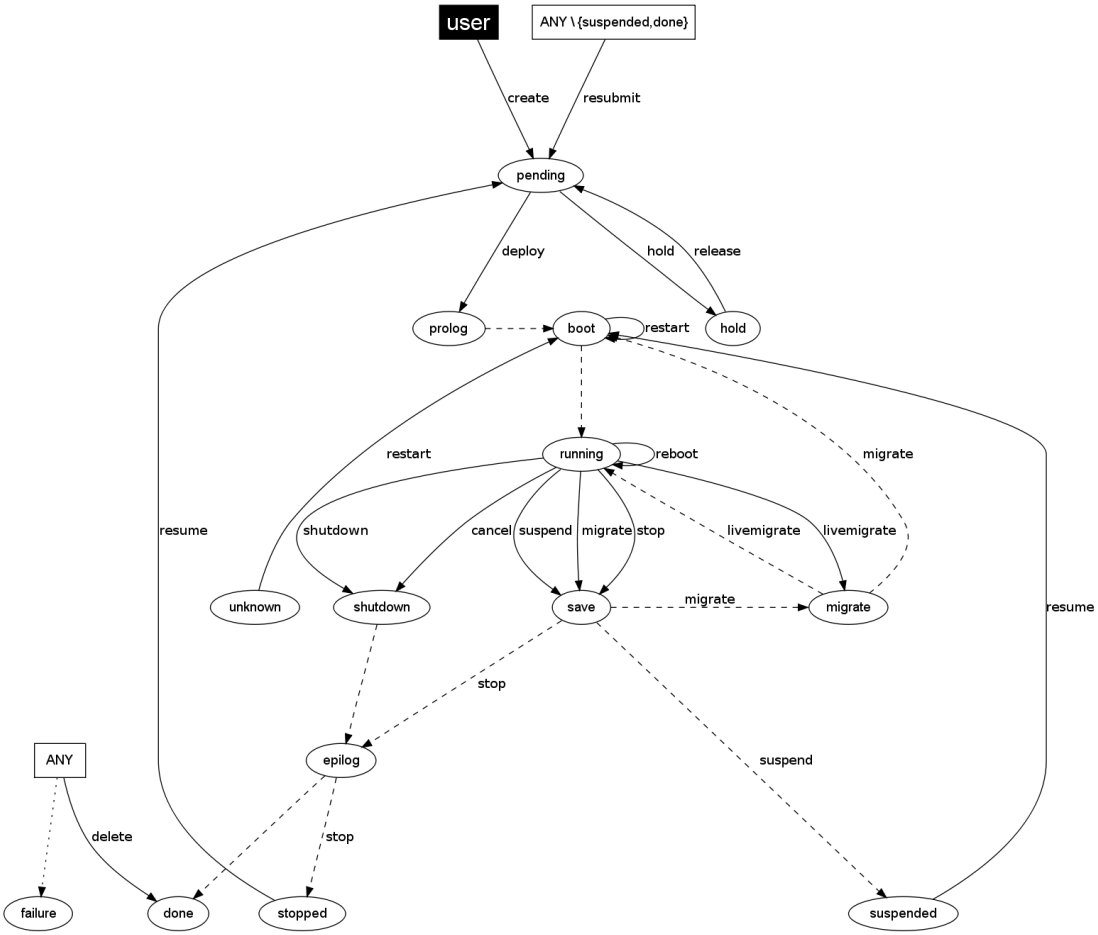

Figura 5: Estados de una máquina virtual dentro de OpenNebula[10]

Con esta gestión de máquinas virtuales OpenNebula permite abstraerse de los detalles de los diferentes hipervisores y verlos todos como máquinas iguales en las que podemos ejecutar y mover máquinas virtuales entre *host*. Esto es una gran ventaja a la hora de disponer por ejemplo de balance de carga.

Las operaciones para modificar el estado de un máquina virtual se hace a través del comando `onevm`:

- `create`: Añade una nueva máquina virtual a partir de una plantilla al pool de maquinas virtuales.
- `deploy`: Inicia una máquina virtual del pool en un Host especifico.
- `shutdown`: Envía una señal ACPI para apagar de forma limpia la máquina virtual.

[10] Fuente: http://opennebula.org/documentation:rel3.4:vm_guide_2

- `livemigrate`: Transfiere una máquina virtual de un host a otro sin apagar la máquina.
- `migrate`: Para la máquina virtual y la inicia en otro host.
- `hold`: Pone la máquina virtual en estado *hold*. En este estado no se despliega la máquina.
- `release`: Libera la máquina en estado *hold* y queda pendiente de desplegar en un host.
- `cancel`: Apaga inmediatamente la máquina virtual sin enviarle la señal ACPI.
- `suspend`: Se suspende la máquina en el host donde se está ejecutando.
- `resume`: Reinicia la ejecución de la máquina virtual que había sido previamente suspendida.
- `saveas`: Almacena la imagen de la máquina virtual en un nuevo fichero imagen.
- `delete`: Elimina la máquina virtual del host.
- `restart`: Fuerza al host a reiniciar la máquina virtual.
- `reboot`: Envía una señal ACPI para reiniciar de forma limpia la máquina virtual.
- `resubmit`: Devuelve la máquina virtual al estado *pending*. En este estado está disponible para ser desplegada de nuevo.
- `list`: Lista las máquinas virtuales disponibles.
- `show`: Muestra información de una máquina virtual especifica.
- `top`: Muestra información de las máquinas virtuales.

Todas estas operaciones se pueden realizar tanto desde línea de comandos como desde el GUI Sunstone.

3. Plantillas

Como hemos visto en el apartado anterior OpenNebula utiliza plantillas para definir una máquina virtual antes de desplegarla en el *host*. En estas plantillas se especifican las características que va a tener dicha máquina virtual.

Estas plantillas son ficheros en texto plano que se almacenan en un repositorio.

La sintaxis del ficheros puede ser pares de clave=valor o desde la versión 3.4 ficheros XML.

En estos ficheros se guarda la siguiente información:

- Nombre identificativo de la máquina virtual.

- Porcentaje de la CPU que se va utilizar dentro del hipervisor.

- Número de CPUs virtuales dentro del hipervisor.
- Parámetros del sistema operativo y el arranque como arquitectura del kernel, imágenes de arranque, parámetros para el *boot* o el dispositivo desde donde arrancar.

```
OS = [ KERNEL     = /vmlinuz,
       INITRD     = /initrd.img,
       ROOT       = sda1,
       KERNEL_CMD = "ro xencons=tty console=tty1"]
```

- Imágenes de disco a utilizar por la máquina virtual. Se pueden definir imágenes que existen en el repositorio de imágenes o una nueva que se generará automáticamente.

```
# Imagen del SO con ID 2 en el repositorio. Mapeada como sda
DISK = [ IMAGE_ID  = 2 ]
# Imagen limpia no existente en repositorio. Mapeada como sdb

DISK = [ TYPE   = fs,
         SIZE   = 4096,
         FORMAT = ext3,
         SAVE   = yes,
         TARGET = sdb ]
```

- Red virtual a utilizar de las que están definidas en el repositorio de Redes Virtuales.

4. Almacenamiento

OpenNebula dispone de un repositorio de imágenes de las máquinas virtuales desplegadas o disponibles para ser desplegadas en el futuro.

Este repositorio puede ser implementado de dos formas principalmente:

- Local: Las imágenes se almacenan localmente en el controlador de OpenNebula y los *host* que corren los hipervisores. Para desplegar las imágenes se utiliza SSH (SSH Transfer Driver). En esta operación se copia cada imagen bajo demanda desde el almacenamiento local del controlador de OpenNebula al almacenamiento local del *host* elegido.
  Esta operación puede ser costosa ya que es posible que haya que mover imágenes muy grandes a través de la red, lo que ralentiza el proceso de despliegue.
  Otro inconveniente es que no permite la migración en vivo (sin apagar la máquina) entre diferentes *host*.

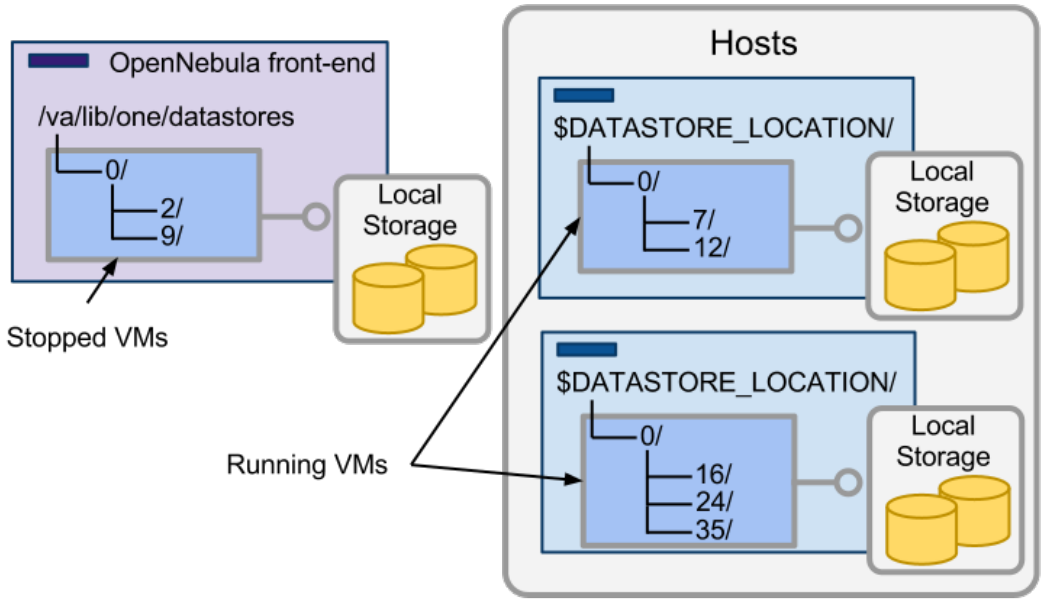

Figura 6: Almacenamiento local en OpenNebula[11]

[11] Fuente: http://opennebula.org/documentation:rel3.4:vm_guide_2

- Compartido: Se utiliza un repositorio común tanto para el controlador de OpenNebula como para los *host*. Se puede utilizar un sistema de ficheros distribuido para mejorar el rendimiento.
  La principal ventaja es que no hay transferencia de imágenes por la red, el despliegue es rápido y se pueden migrar maquinas virtuales entre *host* de manera casi inmediata, facilitando nuevamente el balanceo de carga.
  El principal inconveniente puede aparecer si las máquinas virtuales hacen un uso intensivo del repositorio común, pudiendo degradar el rendimiento. Para solucionar este problema se pueden utilizar diferentes servidores de ficheros

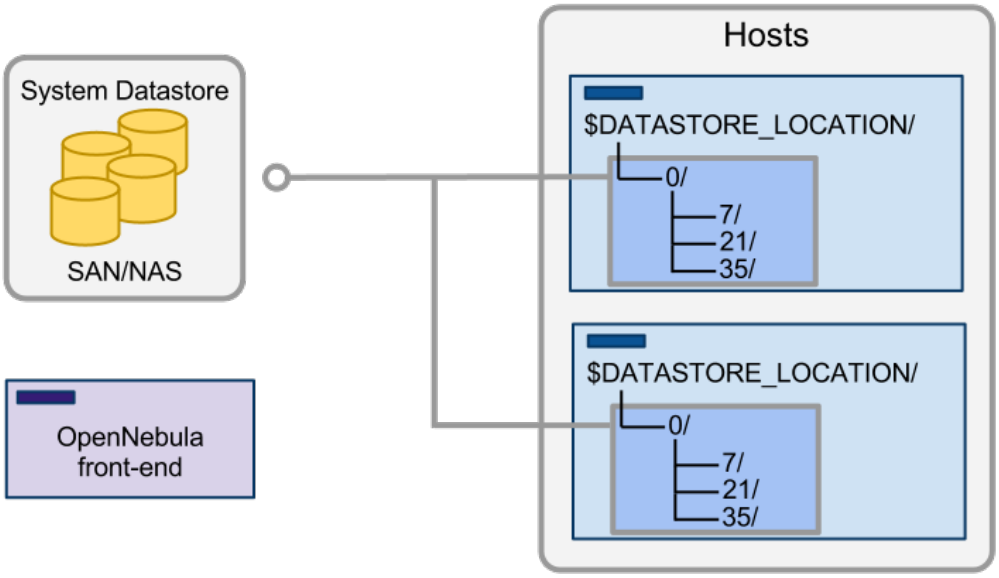

Figura 7: Almacenamiento compartido en OpenNebula[12]

Mediante el comando `oneimage` o Sunstone se realiza la gestión de las imágenes

[12] Fuente: http://opennebula.org/documentation:rel3.4:vm_guide_2

5. Redes privadas

OpenNebula permite la declaración de Redes Virtuales para disponer de nuestro propio direccionamiento dente de la infraestructura y aislar el tráfico de red.

Podemos definir dos tipos de redes virtuales:

- Una lista de IP: Se define una lista de IP que se podrán asignar a las máquinas virtuales de la Red Virtual

```
NAME    = "Blue LAN"
TYPE    = FIXED
  # Utilizará el interface vbr1 del host
BRIDGE  = vbr1
LEASES  = [IP=130.10.0.1]
LEASES  = [IP=130.10.0.2, MAC=50:20:20:20:20:21]
LEASES  = [IP=130.10.0.3]
LEASES  = [IP=130.10.0.4]
GATEWAY = 130.10.0.1
DNS     = 130.10.0.1
```

- Un rango IP: Se define una red (por ejemplo de tipo C) y se asignarán esas IPs a las máquinas virtuales.

```
NAME    = "Red LAN"
TYPE    = RANGED
 # Utilizará el interface vbr0 del host
BRIDGE  = vbr0
NETWORK_SIZE    = C
NETWORK_ADDRESS = 192.168.0.0
GATEWAY = 192.168.0.1
DNS     = 192.168.0.1
```

Así, cuando definamos una máquina virtual a través de un plantilla podemos indicarle que red virtual debe utilizar y asignarle una IP de las disponibles.

Mediante el comando `onenet` o Sunstone se realiza la gestión de las redes virtuales.

6. Gestión de usuarios

Otros de los subsistemas de OpenNebula es la gestión de usuarios. Existen cuatro tipos o roles de usuario con sus respectivos privilegios:

- Administradores: Pertenecen al grupo *oneadmin* puede realizar cualquier operación.
- Usuario normal: Pueden utilizar casi todas las funcionalidades de OpenNebula.
- Usuario public: Puede realizar operaciones básicas.
- Usuario service: Se utiliza para interactuar con otros servicios como EC2 o el GUI Sunstone a través de APIs.

El acceso a los recursos sigue la política típica de Unix en la que el usuario que crea un recurso es su propietario. Este usuario puede dar permisos a otros usuarios para que utilicen los recursos creados por él.

7. OpenNebula en la actualidad

Actualmente OpenNebula es utilizado por grandes empresas como Telefónica I+D o China Mobile o los centros de computación FermiLab y CERN.

Existen también paquetes precompilados para las distribuciones Ubuntu, Debian y OpenSuse lo que facilita la instalación y configuración de la herramienta.

En 2010 los autores de OpenNebula crearon la empresa C12G Labs para dar soporte comercial y servicios a empresas.

La última versión estable liberada ha sido la 3.4.

VII. PaaS: Hadoop

1. *Introducción e Historia*

Como ejemplo de una herramienta Open Source que cubra el modelo de servicio *PaaS* hemos escogido Hadoop de Apache[13]. Al igual que OpenNebula se publica bajo licencia Apache v2.

Apache Hadoop es un *framework* que permite el tratamiento distribuido de grandes cantidades de datos (del orden de peta bytes) y trabajar con miles de máquinas de forma distribuida. Se inspiró en los documentos sobre MapReduce y Google File System publicados por Google. Está desarrollado en Java y se ejecuta dentro de la JVM.

Actualmente está soportado por Google, Yahoo e IBM entre otros. También existen empresas como Cloudera (http://www.cloudera.com/) que ofrecen soluciones empresariales Open Source basadas en Hadoop.

Las características principales de Hadoop son:

- Económico: Está diseñado para ejecutarse en equipos de bajo coste formando clústeres. Estos clústeres pueden llevarnos a pensar en miles de nodos de procesamiento disponibles para el procesado de información.

- Escalable: Si se necesita más poder de procesamiento o capacidad de almacenamiento solo hay que añadir más nodos al clúster de forma sencilla.

- Eficiente: Hadoop distribuye los datos y los procesa en paralelo en los nodos donde los datos se encuentran localizados.

- Confiable: Es capaz de mantener múltiples copias de los datos y automáticamente hacer un re-despliegue de las tareas.

El aspecto clave de Hadoop es que en lugar de mover los datos hacia donde se hace el procesamiento, Hadoop mueve el procesamiento (Tasks) a donde están los datos. Como veremos más adelante Hadoop necesita tener conocimiento de la arquitectura del clúster para poder elegir el nodo más cercano o el que está en el mismo rack para enviar la tarea de procesamiento MapReduce.

---

[13] Apache Software Foundation: Es una comunidad descentralizada de desarrolladores que trabajan cada uno en sus propios proyectos de código abierto. Los proyectos Apache se caracterizan por un modelo de desarrollo basado en el consenso y la colaboración y en una licencia de software abierta y pragmática

2. Arquitectura

El diseño de Hadoop se divide en dos partes principales:

Por un lado la implementación de MapReduce que se encarga del procesamiento de la información de forma distribuida.

Por otro lado está el sistema de ficheros distribuido Hadoop Distributed File System (HDFS) que se encarga de almacenar todos los datos repartiéndolos entre cada nodo de la red Hadoop.

A continuación vamos a ver una visión general de la arquitectura de Hadoop:

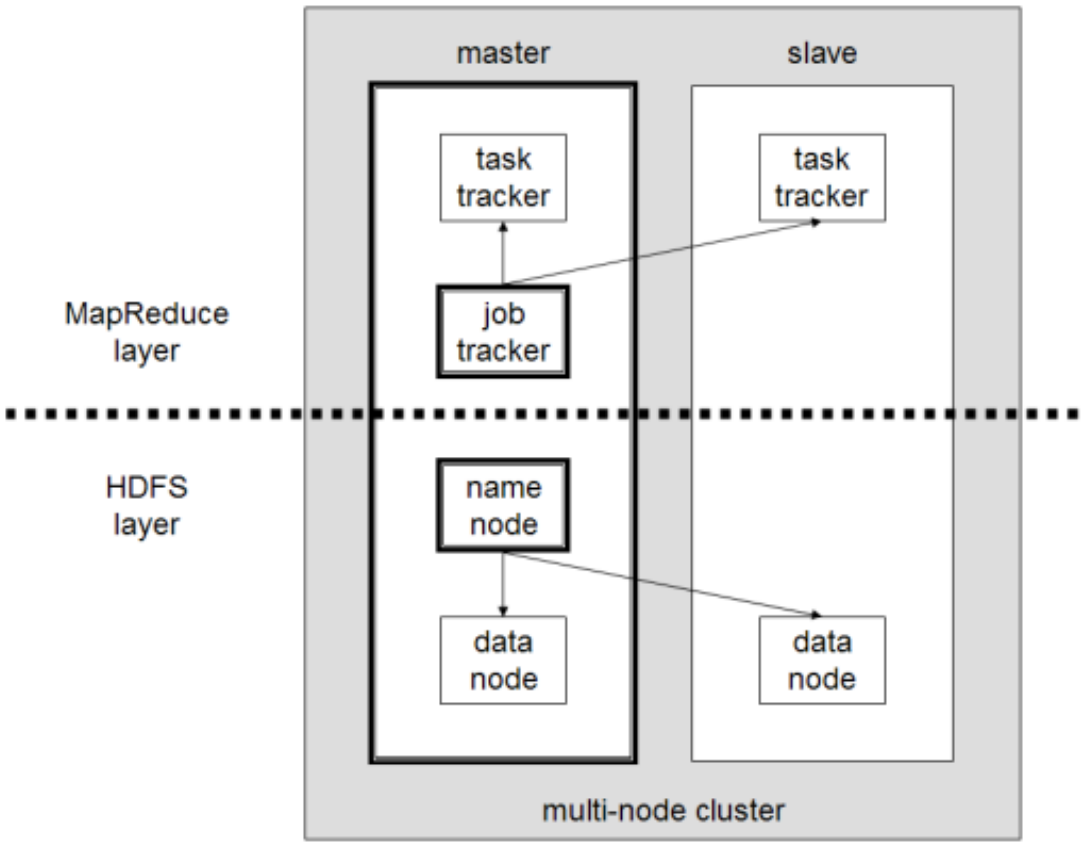

Figura 8: Clúster multi-nodo en Hadoop[14]

[14] Fuente: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

Como vemos en la imagen 7 existen cuatro tipos de nodos diferentes en Hadoop. Los nodos NameNode y DataNode actúan en la capa HDFS y los nodos JobTracker y TaskTracker en la capa MapReduce:

- **NameNode**: Es el maestro o *master* de todos los DataNode. Es el responsable de manejar el namespace del sistema de ficheros y controlar los accesos de los clientes externos. Esta información la almacena en un fichero llamado *FsImage* con otro fichero, *EditLog*, que almacena las transacciones de HDFS en el sistema de ficheros local del nodo. Además esta información se replica para protegerlo contra posibles perdidas o corrupciones del sistema de ficheros.
  Los metadatos contenidos en el NameNode indican en cual o cuales DataNodes está almacenado cada objeto y procesa las operaciones sobre el sistema de ficheros determinando que DataNode es el responsable de ejecutarla. Cada bloque de un objeto se guarda en diferentes DataNodes para proporcionar redundancia.

  Normalmente se ejecuta en una máquina separada del resto de nodos, pero no tiene porque ser obligatoriamente así.

  Haciendo una abstracción, podemos decir que asume el rol de una tabla de ficheros en un sistema de ficheros tradicional (como FAT, MFT o la tabla de inodos en FAT32, NTFS y ext3/ext4 respectivamente) o de un DNS en Internet. Almacena la información a cerca de donde se encuentra los objetos en HDFS.

- **DataNode**: Este tipo de nodos actúa como un almacenamiento de bloques para HDFS. Un clúster de Hadoop puede contener cientos o miles de DataNodes. Estos DataNodes responde a las peticiones de lectura y escritura de los clientes y hacen replicas de los bloques que le envía el NameNode.

  Cada DataNode envía periódicamente paquetes al NameNode con información de sus bloques para que éste pueda validar la consistencia de la información con otros DataNodes.

  Normalmente todos los DataNodes están en un rack conectados a un switch común para aumentar el ancho de banda y el rendimiento del clúster.

- **JobTracker**: Es el servicio de Hadoop que se encarga de las tareas MapReduce que solicitan los clientes. Estas tareas las ejecutan los TaskTracker. Normalmente el JobTracker escogerá el nodo que tenga los datos o en su defecto el que esté en el mismo rack.
  El JobTracker monitoriza la ejecución de las tareas en los TaskTracker para saber si se ha realizado con éxito, o si ha fallado encargársela a otro TaskTracker.

- **TaskTracker**: Este nodo acepta tareas Map o Reduce que le envía el JobTracker. Se configura con número determinado de tareas que pude ejecutar en paralelo, por lo tanto, si tiene *slots* libres será un nodo elegible por el JobTracker para ejecutar una tarea.

  Un TaskTracker se comunica directamente con los DataNode (como otro cliente cualquiera) una vez que el NameNode le ha dado su localización.

Hadoop es un framework totalmente escalable. Podemos configurar un pequeño clúster donde el nodo master contenga JobTracker, TaskTracker, NameNode, y DataNode y un nodo esclavo que actué como DataNode y TaskTracker. También es posible tener solo DataNodes si queremos prescindir del procesamiento distribuido y solo queremos almacenamiento. Siempre es obligatorio tener al menos un NameNode.

En clústeres más grandes, el sistema de ficheros HDFS se maneja a través de un servidor dedicado que ejecuta el NameNode almacenando los metadatos y otro NameNode secundario que se encarga de hacer instantáneas de las estructuras de los NameNode para prevenir la perdida de datos. Otro nodo del clúster ejecutará el JobTracker y le añadiríamos más nodos que ejecutasen los DataNodes y TaskTracker.

Esta escalabilidad es posible por los tres modos de despliegue que soporta Hadoop:

- Standalone: Hadoop está configurado por defecto para trabajar en este modo no distribuido. Tanto el NameNode, el JobTracker como los DataNode y los TaskTrackers se ejecutan en la misma máquina.
- Pseudo-distribuido: Cada servicio de Hadoop se ejecuta en diferentes procesos Java pero en la misma máquina.
- Distribuido: Cada servicio de Hadoop se ejecuta en diferentes nodos de un clúster. Un nodo puede actuar de NameNode, otro de JobTracker y el restos de nodos actúan como DataNode y TaskTracker o los dos a la vez.

Hadoop también permite utilizar otros sistemas de ficheros que no sean HDFS. Si se escoge esta opción los nodos NameNode y los DataNodes son reemplazados por sus equivalentes específicos del sistema de ficheros.

Hadoop puede trabajar con otros sistemas de ficheros distribuidos que puedan ser montados por el Sistema Operativo utilizando el método de acceso file://URL. El inconveniente es que se pierde el conociendo de la localidad de los datos. Para recudir el tráfico de red Hadoop necesita conocer que servidores están más cerca de los datos para ejecutar el procesamiento. Esta información se la proporciona HDFS.

Ejemplo de un clúster Hadoop:

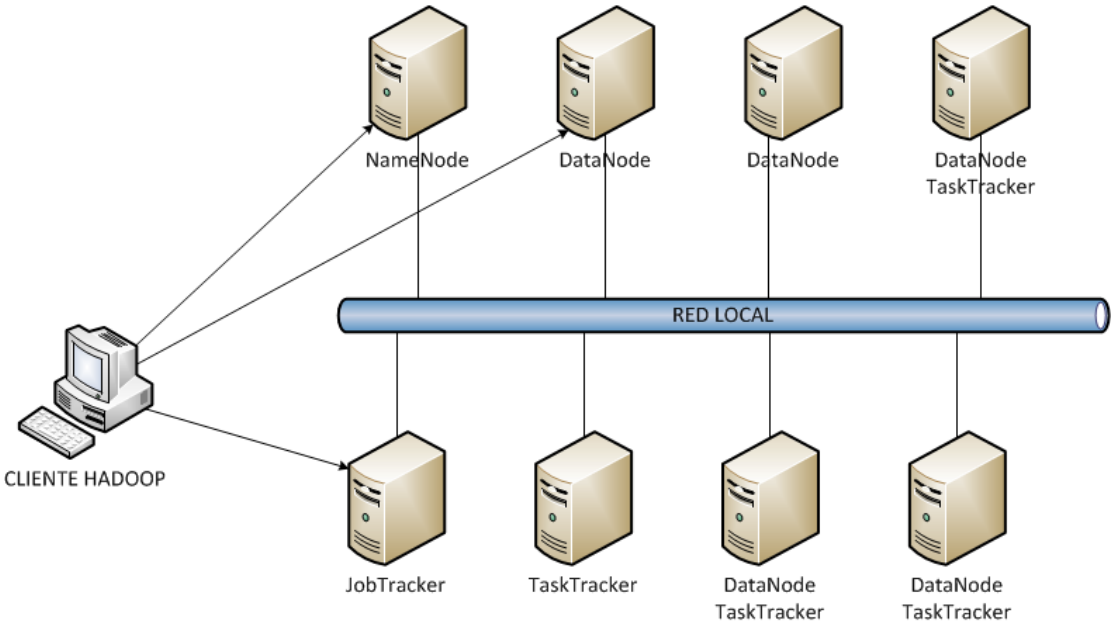

Figura 9: Ejemplo de un clúster en Hadoop

A continuación vamos a ver más en profundidad las dos capas principales de Hadoop: MapReduce y HDFS.

3. MapReduce

MapReduce es un framework desarrollado por Google que soporta procesamiento distribuido sobre grandes cantidades de datos en un clúster. Es la combinación de dos procesos llamados Map y Reduce.

El algoritmo Map and Reduce se basa en el clásico paradigma computacional *divide y vencerás*. Desde un punto de vista abstracto este algoritmo divide el problema en problemas más sencillos, los resuelve y al final une todos los resultados intermedios devolviéndolos al peticionario. Este tipo de algoritmos encajan perfectamente en arquitecturas distribuidas en las que cada nodo del clúster ejecuta una parte del problema a resolver.

MapReduce se basa en la programación funcional, en concreto en las funciones Map y Reduce de los lenguajes funcionales. Consiste en dos operaciones que pueden tener muchas instancias. La función Map toma un conjunto de datos y los transforma en un par clave/valor. La función Reduce toma esta lista de claves/valores y reduce la lista basándose en su clave, obteniendo como resultado un par clave/valor para cada clave.

Una aplicación MapReduce contiene al menos tres partes diferenciadas: una función Map, una función Reduce y una función principal que lleva el control de las trabajos y la entrada/salida del proceso.

Desde un punto de vista más formal:

Las funciones Map y Reduce están definidas respecto a datos estructurados en pares clave/valor. La función Map toma un de estos pares del dominio de entrada y devuelve una lista de pares en otro dominio:

$$Map(k1,v1) \rightarrow list(k2,v2)$$

La función Map se aplica en paralelo a cada par en el conjunto de entrada de datos. Con esto obtenemos una lista de pares en cada llamada.

Después de esto, se recogen todos los pares con la misma clave de todas las listas generadas creando un grupo para cada una de las claves generadas.

La función Reduce se aplica a continuación en paralelo a cada resultado devuelto por la función Map, produciendo un conjunto de valores, todos en el mismo dominio:

$$Reduce(k2, list\ (v2)) \rightarrow list(v3)$$

Normalmente cada llamada a Reduce produce uno o ningún valor v3, aunque es posible que devuelva más de un valor. El resultado de todas las llamadas re recogen en una lista que será el resultado del algoritmo.

En resumen, MapReduce transforma una lista de clave/valor en una lista de valores. Este comportamiento es diferente a la programación funcional típica que acepta una lista arbitraria de valores y devuelve un solo valor que combina todos los valores que devuelve la función Map.

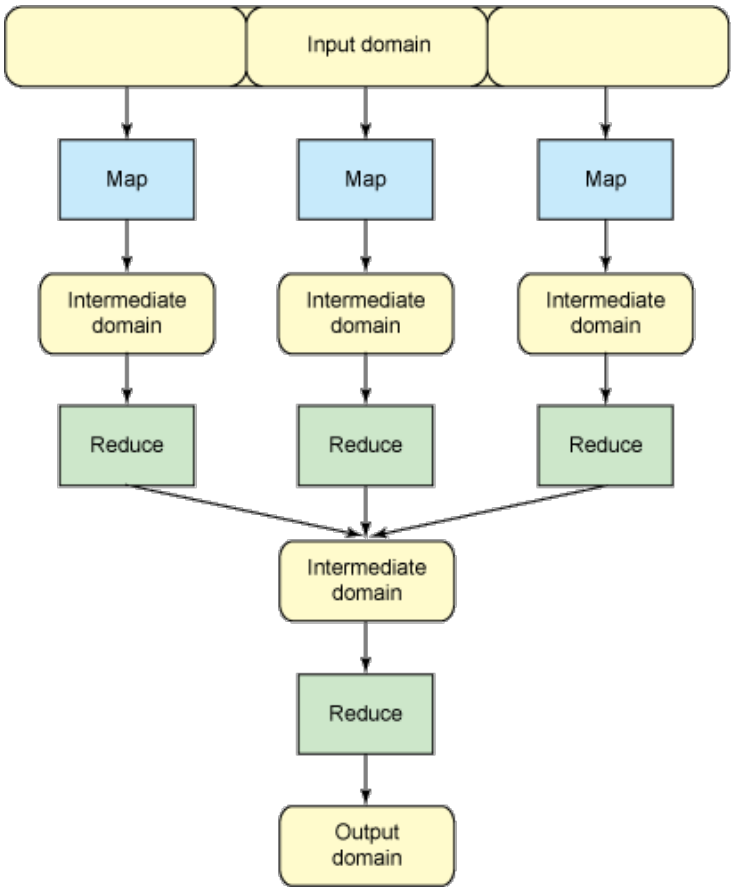

Figura 10: Diagrama de flujo de MapReduce[15]

Vamos a ilustrar esta operaciones con ejemplo sencillo:

Pongamos que el dominio de entrada al algoritmo es la siguiente lista de palabras: one small step for man, one giant leap for mankind. Ejecutando la función Map sobre el dominio nos devuelve la siguiente lista de clave/valor:

```
(one, 1)  (small, 1)  (step, 1)  (for, 1)  (man, 1)
(one, 1)  (giant, 1)  (leap, 1)  (for, 1)  (mankind, 1)
```

Si aplicamos la función Reduce a esta lista de claves/valores, obtenemos el siguiente conjunto de claves/valores:

[15] Fuente: http://www.ibm.com/developerworks/linux/library/l-hadoop/

```
(one, 2)    (small, 1)  (step, 1)(for, 2)  (man, 1)
(giant, 1)  (leap, 1)   (mankind, 1)
```

El resultado del proceso es el número de apariciones de una palabra. Si en lugar de tener un dominio de entrada tuviésemos dos, podríamos ejecutar la función MapReduce a los dos, y con el resultado aplicarle otra vez la función Reduce, con lo que tendríamos el mismo resultado. Así podríamos continuar dividiendo el proceso en parte hasta que resulten tareas triviales de realizar en paralelo.

Como visión global nos quedamos que MapReduce funciona de la siguiente forma en Hadoop:

- Un cliente realiza una petición MapReduce al servicio JobTracker. Este designa un TaskTracker disponible lo más cerca posible de los datos para realizar la operación MapReduce.
- Se divide las tareas en otras más pequeñas y se eligen los TaskTrackers que realizaran el proceso.
- El TaskTracker realiza la tarea y devuelve la respuesta al nodo que le envió la tarea. Cada TaskTracker puede realizar una tarea Map o Reduce.
- En el proceso Reduce se toma la respuesta de las sub-tareas, se combinan y se devuelve el resultado del proceso global.

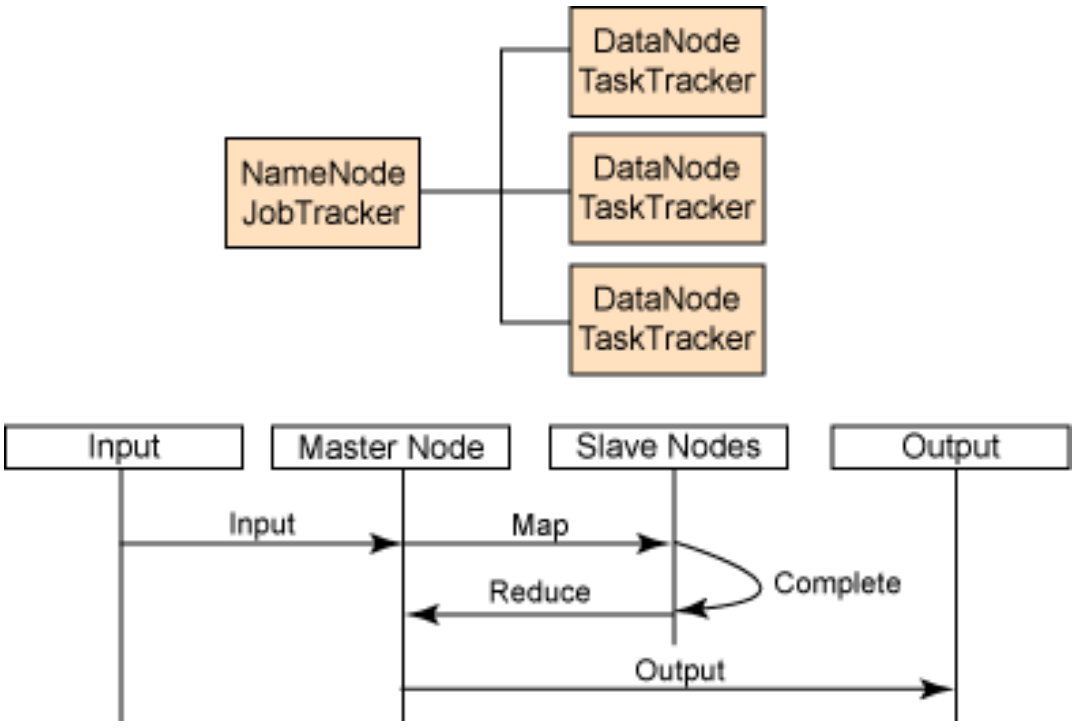

Figura 11: Flujo de datos de MapReduce[16]

[16] Fuente: http://www.ibm.com/developerworks/aix/library/au-cloud_apache/

En la implementación de Hadoop del algoritmo cada tarea se ejecuta en una instancia separada de la JVM para evitar que si una tarea falla, se caiga el nodo TaskTracker y el resto de tareas que está ejecutando.

Además se envía un paquete de control al JobTracker cada varios minutos para vigilar el estado de la tarea. La información del JobTracker y de los TaskTracker se puede ver a través de un explorador web ya que un pequeño servidor web en cada nodo publica esta información.

Para la gestión del orden de ejecución de las tareas, por defecto, Hadoop utiliza listas FIFO pero se pueden utilizar también colas con prioridad.

Actualmente existen implementaciones de MapReduce escritas en C++, Java, Python y otros lenguajes.

Ahora la pregunta es: ¿Cuándo se puede utilizar MapReduce?

Si se pueden reescribir las aplicaciones como un algoritmo Map y Reduce es posible. Así que no todos problema es solucionable mediante está técnica pero si es muy apropiado en problemas en los que hay que trabajar con grandes volúmenes de datos que se pueden dividir y trabajar en paralelo.

En el Anexo I del presente proyecto puede consultarse un ejemplo típico de utilización de Hadoop para contar palabras (muy útil para la indexación de archivos) escrito en Java.

4. Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) es el principal sistema de almacenamiento utilizado por Hadoop. HDFS crea múltiples replicas de los bloques de datos y los distribuye entre los nodos de un clúster.

HDFS no está limitado al uso conjunto con trabajos MapReduce. Existen otras aplicaciones como la base de datos HBase o el sistema Warehose Hive Data, ambos productos de Apache.

Es un sistema de ficheros distribuido con similitudes con otros sistemas de ficheros pero con algunas diferencias. HDFS altamente tolerante a fallos y está diseñado para ejecutarse en hardware de bajo coste. Proporciona un alto rendimiento en el acceso a grandes volúmenes de datos. Además no implementa algunos de los requerimientos POSIX para permite el acceso en streaming a los datos.

Desde la perspectiva del usuario final, HFS se muestra como un sistema de ficheros tradicional. Se pueden llevar a cabo operaciones CRUD[17] en un directorio concreto, siendo la responsabilidad de ejecutar estas operaciones los nodos NameNode y DataNode.

Características principales de HDFS.

- Recuperación antes fallos de hardware: Una instancia de HDFS puede consistir en miles de maquinas, cada una de ellas almacenado una parte de los datos. Cada componente de estas maquinas tiene una probabilidad de fallar dejando inutilizado el nodo. El objetivo de HDFS es proporcionar una rápida y automática recuperación de estos fallos.

- Acceso en streaming: Las aplicaciones que se ejecutan sobre HDFS necesitan el acceso en streaming a sus datos. Así, HFDS ha sido diseñado enfocado más hacia procesos batch que para un uso interactivo por parte del usuario, es decir, se prioriza el rendimiento en el acceso a los datos sobre la latencia.

- Grandes volúmenes de datos: Un fichero típico en HDFS puede tener un tamaño de giga bytes o tera bytes. HDFS proporciona gran ancho de banda y pude escalarse a miles de nodos en un solo clúster. Puede llega a soportar millones de archivos en una sola instancia.

- Coherencia simple: HDFS está optimizado según el modelo *write-once-read-many access*. Esta suposición simplifica los mecanismos de control de la coherencia de los datos y mejora el rendimiento de

[17] Create, read, delete, update. Acrónimo de las cuatros operaciones básicas que se pueden aplicar a almacenamiento persistente (ficheros, bases de datos, etc.)

acceso. De todas formas, en el futuro se espera dar soporte a la escritura de ficheros en modo *appending-writes*.

- Mover la computación donde están los datos: HDFS se basa en el principio que es más barato mover la computación donde están los datos que mover los datos en sí, cuyo volumen será normalmente mucho mayor. Esto minimiza el tráfico de red y mejora el rendimiento general del clúster.

- Portabilidad: HDFS ha sido diseñado para se fácilmente portable entre diferentes plataformas, en parte gracias a que se ha sido implementado en Java, que puede ejecutarles en numerosas arquitecturas.

A continuación vamos a explicar como funciona HDFS.

HDFS trabajo sobre una arquitectura master / esclavo.

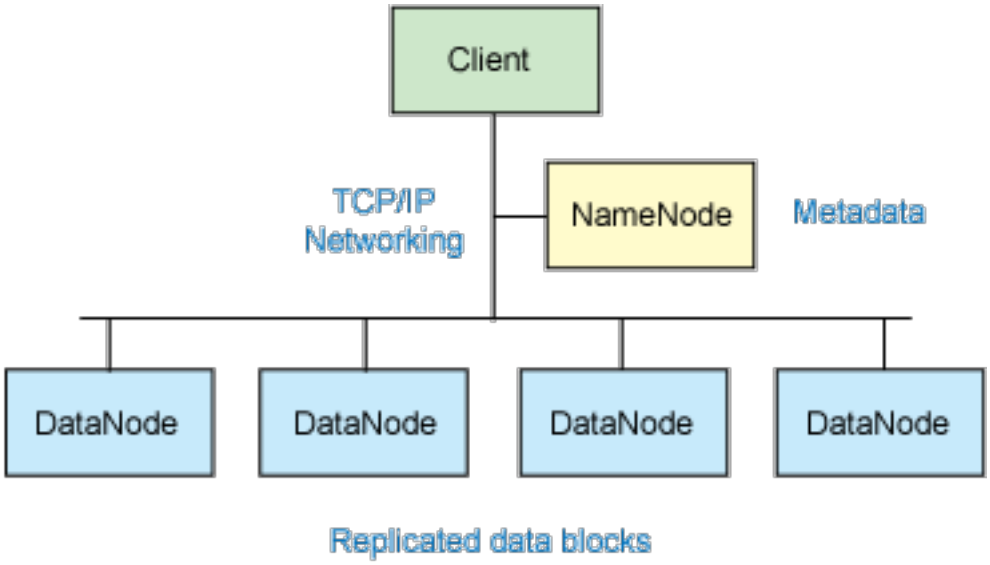

Figura 12: Clúster HDFS[18]

Un clúster HDFS consiste en un nodo NameNode que trabaja como master y que es el encargado de manejar el namespace del sistema de ficheros y controlar los accesos a los ficheros.

Además existen diversos DataNodes, normalmente uno por nodo del clúster, que almacenan los bloques de datos. Un fichero almacenado en HDFS es dividido en fragmentos y repartido entre los DataNodes.

Los DataNodes envían los bloques de los ficheros a los clientes que lo solicitan, además de crear, borrar y replicar bloques datos siguiendo las instrucciones del NameNode.

[18] Fuente: http://www.ibm.com/developerworks/linux/library/l-hadoop/

Cuando un cliente hace una petición el NameNode escoge el DataNode más cercano que almacena el bloque para mejor el rendimiento. Seguidamente, el NameNode responde con el identificador de bloque y la dirección IP del DataNode que almacena el fichero.

Una vez realizad la conexión entre el cliente el DataNode las transacciones de I/O (los datos en sí) no pasan a través del NameNode sino que se hace punto a punto entre el cliente y el DataNode.

Todas las protocolos de comunicaciones corren sobre la pila TCP/IP. Los clientes abren una conexión TCP a un puerto del NameNode y utilizan el protocolo ClientProtocol. Los DataNodes se comunican con el NameNode usando el protocolo DataNodeProtocol. Por diseño el NameNode nunca inicia una comunicación, sino que sólo responde a peticiones de los clientes o de los DataNodes.

Entre el NameNode y los DataNodes se intercambian continuamente mensajes, llamados *heartbeats*, para monitorizar el estado del sistema de ficheros y el progreso de las operaciones en curso.

4.1 Almacenamiento de datos

HDFS ha sido diseñado para soportar ficheros muy grandes. El tamaño de bloque típico en HDFS es 64 MB. Así, cada fichero es divide en bloques de 64 MB y si es posible se copia cada bloque en DataNodes diferentes.

A la hora de escribir un fichero, este se almacena inicialmente en la caché del cliente. Una vez que el fichero temporal llega al tamaño de bloque (64 MB), el cliente envía un mensaje al NameNode que lo inserta en el sistema de ficheros y le envía al cliente a que DataNode debe enviarlo. Cuando acaba la escritura y se cierra el fichero el NameNode realiza un *commit* de la operación.

Para acceder a los ficheros almacenados en HDFS hay varias formas:

- FS Shell: Permite al usuario acceder al sistema de ficheros a través de una Shell similar a bash. Es un aplicación de línea comandos que se ejecuta con Hadoop

```
$ bin/hadoop fs -ls
```

- DFSAdmin: Es un conjunto de comandos para administrar HDFS y sólo puede ejecutarlas el administrador

- Browser interface: Normalmente HDFS implementa un servidor web que permite acceder al namespace del sistema de ficheros a través de una conexión http a un puerto determinado. De esta forma se puede acceder al contenido de los ficheros a través de un explorador Web.

4.2 Replicación y tolerancia a fallos

Una de los objetivos de HDFS es ser un sistema de ficheros altamente confiable.

Para conseguir esto se utilizan dos técnicas. Copiar cada parte del fichero en diferentes nodos y replicar esa información varias veces. Un cliente puede especificar cuantas copias de un fichero quiere mantener, llamado factor de replicación, y el NameNode almacena esta información.

Como se ha explicado anteriormente cuando se escribe en un fichero se empieza a escribir en un fichero local del cliente. Si el factor de replica es 3, cuando el cliente realiza la petición de escritura al NameNode, éste le envía una lista DataNodes donde escribir las replicas de los datos. El cliente le envía los datos al primer DataNode de las lista. El primer DataNode va escribiendo fragmentos de 4 KB en su repositorio local y enviando estos fragmentos al siguiente DataNode de la lista hasta el ultimo DataNode. Así, a través del mecanismo *pipeline*, cada DataNode va recibiendo fragmentos del nodo anterior, escribiéndoles en su repositorio y reenviándolos al siguiente.

Respecto a la tolerancia a fallos, existen tres tipos de fallos en HDFS:

- Fallo del NameNode: Los ficheros FsImage y EditLog, almacenados en el sistema de ficheros local del NameNode, son las estructuras principales. Si se corrompen estos archivos HDFS puede dejar de funcionar. Para evitar esto, se pude configurar el NameNode para mantener varias copias de estos ficheros. Cada cambio en FsImage y EditLog se reproduce de manera síncrona en todas sus copias. Aunque puede parecer que esto puede degradar el rendimiento de HDFS, no tiene porque ser así ya que las aplicaciones que corren en HDFS realizan un acceso intensivo a los datos (almacenados en los DataNodes) pero no a los metadatos.

- Fallo del DataNode: Cuando un DataNode envía un bloque es posible que esté corrupto, quizá por un fallo en el dispositivo de almacenamiento del DataNode. El cliente de HDFS implementa un mecanismo de integridad que verifica el *checksum* del bloque con el que se ha almacenado en el namespace en un fichero oculto cuando se escribió. Si esta verificación no es exitosa, el cliente puede solicitar el bloque de otro DataNode que contenga una replica.

- Fallos de red: Cada DataNode envía mensajes al NameNode periódicamente. Si el NameNode deja de recibir estos mensajes de un DataNode, lo marcará como no disponible y no le enviará más peticiones de los clientes. Como sus datos no están disponibles es posible que el factor de replicación de un fichero esté por debajo de su factor de replica. Si esto es así el NameNode iniciará la réplica de dicho fichero a otros DataNode.

5. Inconvenientes

Algunos autores apuntan a un excesivo intercambio de mensajes entre los nodos de Hadoop (réplicas, sincronizaciones, actualizaciones de metadatos, localización) que pueden ralentizar la red. Una solución posible es utilizar una red separada para Hadoop y otra para el trafico de los datos.

Por otra parte, como se ha comentado anteriormente, no todos los problemas se pueden resolver o traducir a problemas MapReduce. Aún así se pude utilizar Hadoop como sistema de ficheros distribuido por medio HDFS.

6. Hadoop en la actualidad

Hadoop se puede utilizar en teoría para casi cualquier tipo de trabajo batch, mejor que ha trabajos en tiempo real, ya que son más fáciles de dividir y ejecutar en paralelo. Entre lo campos actuales a aplicación se encuentran:

- Análisis de logs
- Análisis de mercado
- Machine learning y data mining
- Procesamiento de imágenes
- Procesamiento de mensajes XML
- Web crawling
- Indexación de textos

Actualmente Hadoop es un framework muy extendido en el ámbito empresarial, sobre todo en compañías que manejan grandes volúmenes de datos. Entre las que podemos descarta las siguientes empresas:

Yahoo: La aplicación Yahoo! Search Webmap está implementado con Hadoop sobre un clúster de mas de 10.000 nodos Linux y la información que produce es la utilizada por el buscador de Yahoo.

Facebook: Tiene ha día de hoy el mayor clúster Hadoop del mundo que almacena hasta 30 peta bytes de información.

Amazon A9: Se utiliza para la generar índices de búsqueda de los productos ofertados en el portal. Disponen de varios clústeres de entre 1 y 100 nodos cada uno.

The New York Times: Utiliza Hadoop y EC2 (Amazon Elastic Compute Cloud) para convertir 4 Tera bytes de imágenes TIFF en imágenes PNG de 800 K para ser mostradas en la Web en 36 horas.

Además existen compañías cuyo negocio es principal es Hadoop, como Cloudera, que comercializa CDH (Cloudera's Distribution including Apache Hadoop), que da soporte en la configuración y despliegue de clústeres Hadoop. Además proporciona servicios de consultoría y formación en estas tecnología. Todo el software que distribuyen es Open Source.

VIII. Aplicación práctica: Despliegue de un Clúster Hadoop utilizando OpenNebula con hipervisor Xen

1. Introducción

Después de haber visto en profundidad diferentes tecnologías y soluciones Open Source dentro de Cloud Computing, el último paso dentro del presente proyecto es realizar un despliegue, en un entorno de laboratorio, de OpenNebula y Hadoop. Simularemos el despliegue de un clúster Hadoop, en nuestro caso de cuatro nodos, aunque en el mundo real puede estar formado por miles de nodos.

Como se ha visto OpenNebula puede trabajar con tres hipervisores diferentes, y nos hemos decantado por Xen ya que es uno de los hipervisores más extendidos del mercado, existiendo una comunidad muy activa que ofrece soporte.

El objetivo final de la presenta parte práctica es desplegar la siguiente infraestructura:

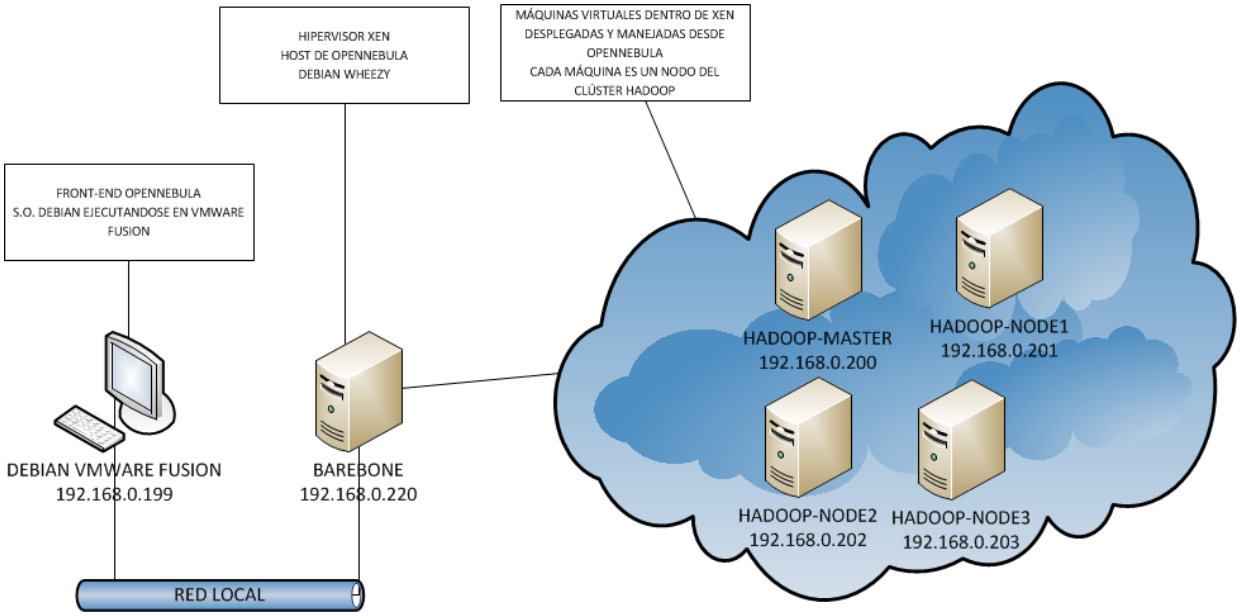

Figura 13: Visión general de la maqueta a desplegar

2. Material

- Ordenador Barebone. Este equipo ejecutará el hipervisor Xen donde se desplegarán desde OpenNebula cuatro máquinas virtuales, cada una de ellas realizando la función de un nodo Hadoop (una de cada tipo explicado anteriormente):
  - Hardware:
    - Procesador Intel Core i3 con VT-x a 3,10 GHz.
    - 8GB de memoria RAM.
    - 1 TB de disco duro.
    - Un Interfaz de red.
  - Software:
    - Debian Wheezy
    - Xen 4.1
- Mac-Mini: Una máquina virtual Debian corriendo en VMWare Fusion hará el rol de front-end de OpenNebula. Desde aquí se llevará acabo el despliegue y control del clúster Hadoop.
  - Procesador Intel Core 2 Duo. (1 procesador para la máquina virtual).
  - 2 GB de memoria RAM (512 MB para la máquina virtual).
  - 350 GB de disco duro (15 GB para la máquina virtual).
  - Un Interfaz de red.

3. Instalación hipervisor Xen

Empezamos instalado el hipervisor Xen. (dom0) Se han seguido los siguientes pasos:

1. Creamos las siguientes particiones en el disco de 1 TB
   - 100 Mb para la partición /boot
   - 3 GB para la partición /
   - 1 GB para la partición /var
   - 930 MB para la partición /var/lib/one (Donde se guardarán las máquinas virtuales)

2. Realizamos una instalación mínima del sistema Debian.

3. Una vez finalizada la instalación, instalamos los siguientes paquetes:
   - linux-image-xen-amd64: Kernel Linux compilado para ejecutarse como dom0 dentro de Xen.
   - xen-hypervisor-4.1-amd64: Hipervisor para la arquitectura amd64.
   - xen-tools: Herramientas de Xen.
   - nano: Editor de textos.
   - sudo: Permite ejecución de comandos como `root`.
   - openssh-server: Servidor SSH para acceso remoto.

4. Modificamos el archivo de configuración `/etc/xen/xend-config.sxp`.

5. Descomentamos la línea `(network-script network-bridge)` para habilitar *virtual network bridge*.

6. Modificamos el fichero `/boot/grub/grub.cfg` del gestor de arranque GRUB para que arranque por defecto con el kernel con el hipervisor.

7. Creamos el usuario `oneadmin`, que será con el que interactuará el front-end de OpenNebula.

8. Instalamos el paquete .deb `opennebula-node` versión 3.2. Este paquete hace que el usuario `oneadmin` pueda utilizar sudo, que su carpeta $HOME sea `/var/lib/one` y crea su par de claves pública/privada RSA para las conexiones SSH.

Después de estos pasos ya tenemos configurado nuestro hipervisor Xen. El nombre de la máquina será `xenserver` y su IP 192.168.0.220.

4. Instalación OpenNebula e integración con Xen.

A continuación instalaremos sobre una máquina virtual un sistema Debian que asumirá el rol de front-end de OpenNebula 3.2. Desde aquí desplegaremos las máquinas virtuales sobre el hipervisor Xen.

Los pasos llevados a cabo son los siguientes:

1. Instalamos los paquetes `opennebula`, `opennebula-common` y `openneubla-tools` y `opennebula-sunstone`. (versión 3.2.1).

2. Copiamos la clave publica del usuario oneadmin al hipervisor (`xenserver`) y viceversa, para que se puedan establecer conexiones SSH sin introducir contraseñas (esto es necesario para la transferencia de imágenes de disco y para guardar cambios).

```
opennebula:/$ ssh-copy-id -i ~/.ssh/id_rsa.pub oneadmin@xenserver
xenserver:/$  ssh-copy-id -i ~/.ssh/id_rsa.pub oneadmin@opennebula
```

3. Por defecto, OpenNebula viene configurado por defecto para trabajar con el hipervisor KVM. Para que acepte hipervisores Xen tenemos que modificar el archivo `/etc/one/oned.conf` y añadir el *Information Driver* y *Virtualization Driver*:

```
IM_MAD = [
        name       = "im_xen",
        executable = "one_im_ssh",
        arguments  = "xen" ]

VM_MAD = [
        name       = "vmm_xen",
        executable = "one_vmm_exec",
        arguments  = "xen",
        default    = "vmm_exec/vmm_exec_xen.conf",
        type       = "xen" ]
```

4. Antes de dar por finalizada la instalación de OpenNebula hay dos pequeños cambios que hay que realizar para que funcione correctamente con el hipervisor Xen 4.1.

    a. En el archivo `/var/lib/one/remotes/vmm/xen/xenrc` cambiar la línea

    ```
    export XM_CREDITS="sudo $XM_PATH sched-cred"
    ```

    por esta otra:

    ```
    export XM_CREDITS="sudo $XM_PATH sched-credit"
    ```

b. Al archivo `/var/lib/one/remotes/vmm/xen/poll` aplicar el parche http://dev.opennebula.org/attachments/377/one_xen_4.0_poll.patch.

(Los scripts que se encuentran la carpeta *remotes* se copian a cada host registrado en OpenNebula, y se utilizan para gestionar y obtener información de las máquinas que se han desplegado a través de OpenNebula).

Una vez que hemos finalizado de instalar OpenNebula vamos a empezar con su configuración. Las siguientes operaciones se pueden hacer tanto a través de línea de comandos como a través del GUI Sunstonte. Nos hemos decantado en esta ocasión por Sunstonte ya que es más fácil de visualizar el resultado de las operaciones.

1. Accedemos a la aplicación a través de la dirección http://localhost:9869 con el usuario `oneadmin` (La contraseña de este usuario de OpenNebula, no confundir con el usuario del sistema `oneadmin`, se almacena en `/var/lib/one/.one/one_auth`).

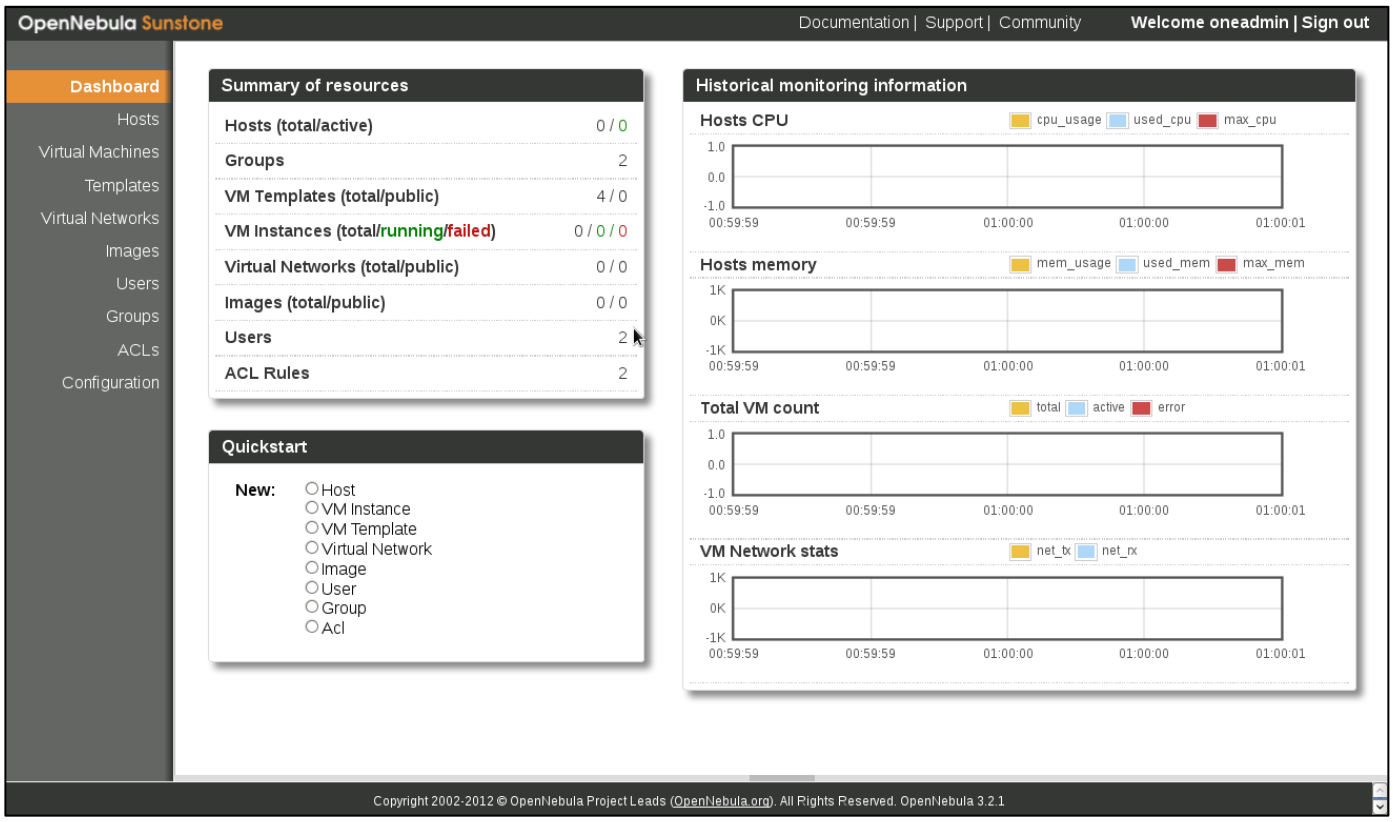

Figura 14: Interfaz gráfico de administración de OpenNebula (Sunstone)

Desde está aplicación podemos realizar todas los operaciones que permite OpenNebula, empezando por la creación de un *host*, que será nuesto siguiente paso.

2. Hacemos click en el menú Hosts y pulsamos el botón +New. En el campo *Name* indicamos el nombre o la IP del hipervisor que queremos añadir (en nuestro caso `xenserver`). Además indicamos que *Information Driver* y *Virtualization Driver*, y el modo de transferencia, SSH.

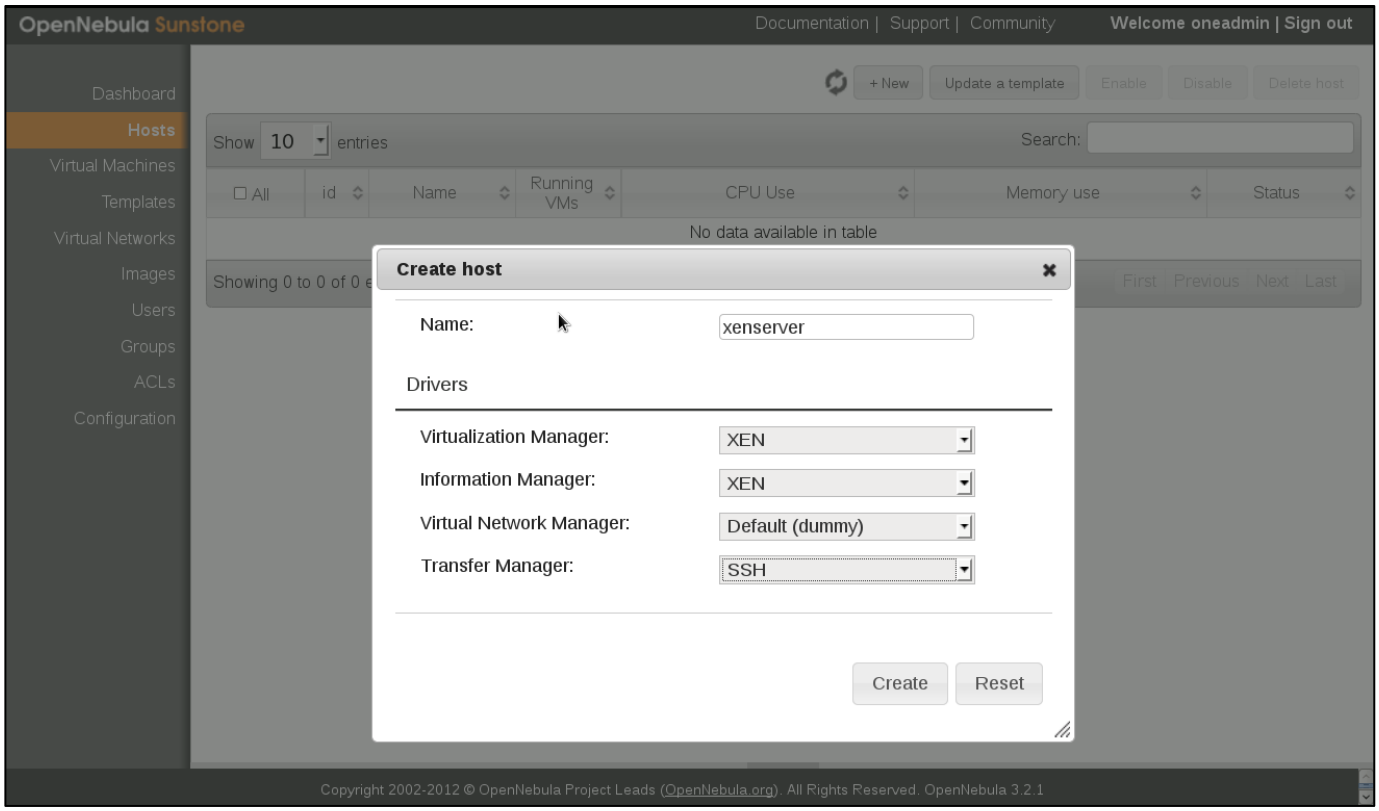

Figura 15: Creación de un *host* con Sunstone

Una vez pulsamos Create, se ha añadido el host a OpenNebula, que lo monitorizará constantemente, mostrando información del uso de la CPU, uso de memoria o máquinas virtuales que se están ejecutando actualmente.

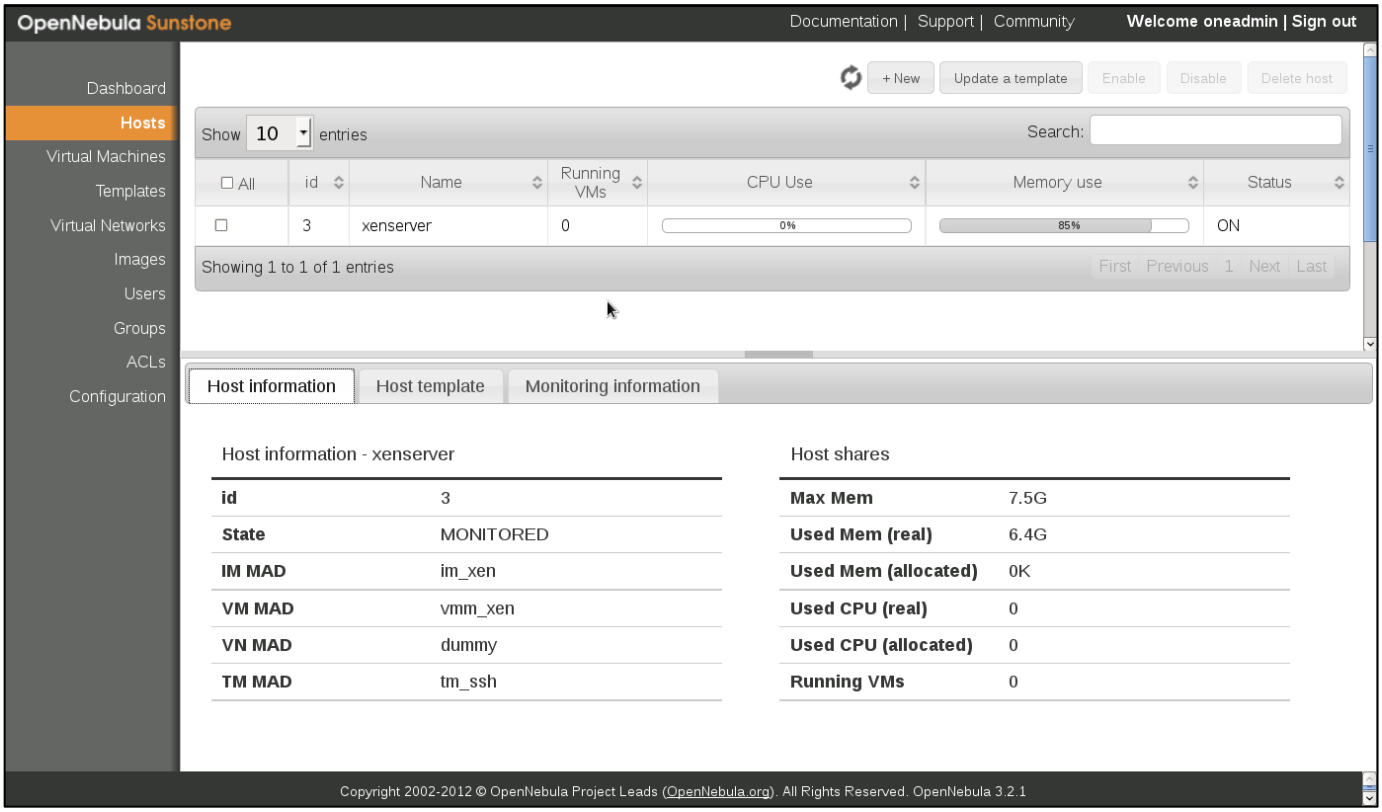

Figura 16: Información de un *host* en Sunstone

3. El siguiente paso es crear las imágenes que contendrán el Sistema Operativo de cada nodo del clúster. Como hemos visto al principio desplegaremos un nodo master que funcionará como NameNode y JobTracker y otros 3 nodos que funcionarán como DataNodes y TaskTrackers simultáneamente.
   Para crear estas máquinas utilizaremos el comando `xen-create-image` que viene incluido en xen-tools. Generaremos una imagen para cada nodo con los siguientes comandos:

```
xen-create-image --hostname=hadoop-master --size=1.5Gb --
memory=1Gb --arch=amd64 --ip=192.168.0.200 --gw=192.168.0.1 --
netmask=255.255.255.0 --dist=squeeze --role=dev --dir=/xen-
image/ --noswap

xen-create-image --hostname=hadoop-node1 --size=1.5Gb --
memory=1Gb --arch=amd64 --ip=192.168.0.201 --gw=192.168.0.1 --
netmask=255.255.255.0 --dist=squeeze --role=dev --dir=/xen-
image/ --noswap
```

```
xen-create-image --hostname=hadoop-node2 --size=1.5Gb --
memory=1Gb --arch=amd64 --ip=192.168.0.202 --gw=192.168.0.1 --
netmask=255.255.255.0 --dist=squeeze --role=dev --dir=/xen-
image/ --noswap

xen-create-image --hostname=hadoop-node3 --size=1.5Gb --
memory=1Gb --arch=amd64 --ip=192.168.0.203 --gw=192.168.0.1 --
netmask=255.255.255.0 --dist=squeeze --role=dev --dir=/xen-
image/ --noswap
```

4. Con esto hemos conseguido cuatro imágenes con un Debian Squeeze plenamente funcionales que correrán como máquinas virtuales en el host Xen.

Seguidamente vamos a instalar Hadoop en cada imagen de disco.

5. Instalación Hadoop

La instalación de Hadoop la vamos a realizar a través de una *jaula chroot*[19], antes de desplegarlos.

1. Montamos la imagen hadoop-master.img a través del dispositivo *loop*.

```
mount -o loop hadoop-master.img /mnt/loop

mount -o bind /proc /mnt/loop/proc

mount -o bind /sys /mnt/loop/sys

mount -o bind /dev /mnt/loop/dev
```

2. Ejecutamos chroot sobre el directorio /mnt/loop.Las operaciones que hagamos a partir de ahora se aplicarán a la imagen de disco `hadoop-master.img`.

3. Instalamos el paquete .deb http://archive.cloudera.com/one-click-install/squeeze/cdh3-repository_1.0_all.deb. Se añade el repositorio Debian de Cloudera para instalar Hadoop.

4. Instalamos la máquina virtual de Java:

```
sudo apt-get install sun-java6-jdk
```

5. Instalamos los siguientes paquetes

[19] Se invoca un proceso cambiando el directorio raíz del sistema.

```
sudo apt-get install hadoop-0.20 hadoop-0.20-namenode

sudo apt-get install hadoop-0.20 hadoop-0.20-jobtracker
```

Para el caso de las imágenes hadoop-nodeX en lugar de los paquetes anteriores hay que instalar los dos siguientes.

```
sudo apt-get install hadoop-0.20 hadoop-0.20-datanode

sudo apt-get install hadoop-0.20 hadoop-0.20-tasktracker
```

6. Creamos la carpeta `/hdfs` donde se almacenará toda la información referente la sistema de ficheros HDFS y las tareas MapReduce.

7. Una vez instalado el software vamos configurar el clúster. Los archivos que se van a modificar a continuación se copiaran a todos los nodos del clúster, por lo que sólo hay modificarlos una vez y luego distribuirlos entre los diferentes nodos.

    a. En el archivo `/etc/hadoop/conf/core-site.xml` se especifica la dirección del NameNode a través de la propiedad `fs.default.name`:

```
<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

  <name>fs.default.name</name>

  <value>hdfs://hadoop-master:54310</value>

</property>

</configuration>
```

    b. En el archivo `/etc/hadoop/conf/mapred-site.xml` se configura el servicio JobTracker:

```
<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

  <name>mapred.job.tracker</name>
```

```
  <value>hadoop-master:54311</value>

  <description>Dirección del servicio
JobTracker</description>

</property>

<property>

  <name>mapred.local.dir</name>

  <value>/hdfs/mapred/local</value>

  <description>Directorio donde el TaskTracker almacena
datos temporales durante los procesos Map y
Reduce</description>

</property>

<property>

  <name>mapred.system.dir</name>

  <value>/mapred/system</value>

  <description>Directorio dentro de HDFS donde se lamacenan
ficheros de sistema de MapReduce</description>

</property>

</configuration>
```

c. En el archivo `/etc/hadoop/conf/hdfs-site.xml` se configuran los directorios que utilizará HDFS:

```
<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

  <name>dfs.name.dir</name>

  <value>/hdfs/nn</value>

  <description>Directorio donde el NameNode almacena los
metadatos y los logs del sistema de ficheros</description>

</property>

<property>

  <name>dfs.data.dir</name>

  <value>/hdfs/dn</value>
```

```
  <description>Directorio donde los DataNodes almacenan los
bloques de datos</description>

</property>

<property>

  <name>dfs.replication</name>

  <value>1</value>

  <description>Número de réplicas de cada fichero en el
sistema de ficheros HDFS </description>

</property>

</configuration>
```

d. En el archivo `/etc/hadoop/conf/masters` se específica que nodos van a trabajar como SencondaryNameNode en nuestro caso el propio `hadoop-master`.

e. En el archivo `/etc/hadoop/conf/slaves` se específica que nodos van a ejecutar los servicios DataNode y TaskTracker. Esto no es estrictamente necesario, ya que cuando se arranca un DataNode el se conecta directamente al NameNode, pero a través de este archivo, cuando se arranca el servicio NameNode intenta levantar todos los DataNodes especificados:

```
hadoop-node1

hadoop-node2

hadoop-node3
```

8. Creamos los directorios que hemos especificado en los ficheros de configuración anteriores y establecemos sus permisos de acceso:

```
hdfs-site.xml (dfs.name.dir)

sudo mkdir –p /hdfs/nn

sudo chown -R hdfs:hadoop /hdfs/nn


hdfs-site.xml (dfs.data.dir)

sudo mkdir –p /hdfs/dn

sudo chown -R hdfs:hadoop /hdfs/dn
```

```
mapred-site.xml (mapred.local.dir)

sudo mkdir -p /hdfs/mapred/local

sudo chown -R mapred:hadoop /hdfs/mapred/local
```

Después de todas estas operaciones salimos de la *jaula chroot* y disponemos de cuatro imágenes de disco con los servicios Hadoop instalados y listas para ser desplegada en un hipervisor.

6. Despliegue de las imágenes

Volviendo a OpenNebula, el siguiente paso es registrar las imágenes creadas en el apartado anterior para luego poder desplegarlas en el hipervisor Xen.

1. Desde el menú Images, pulsamos en +New e introducimos los datos de la imagen:

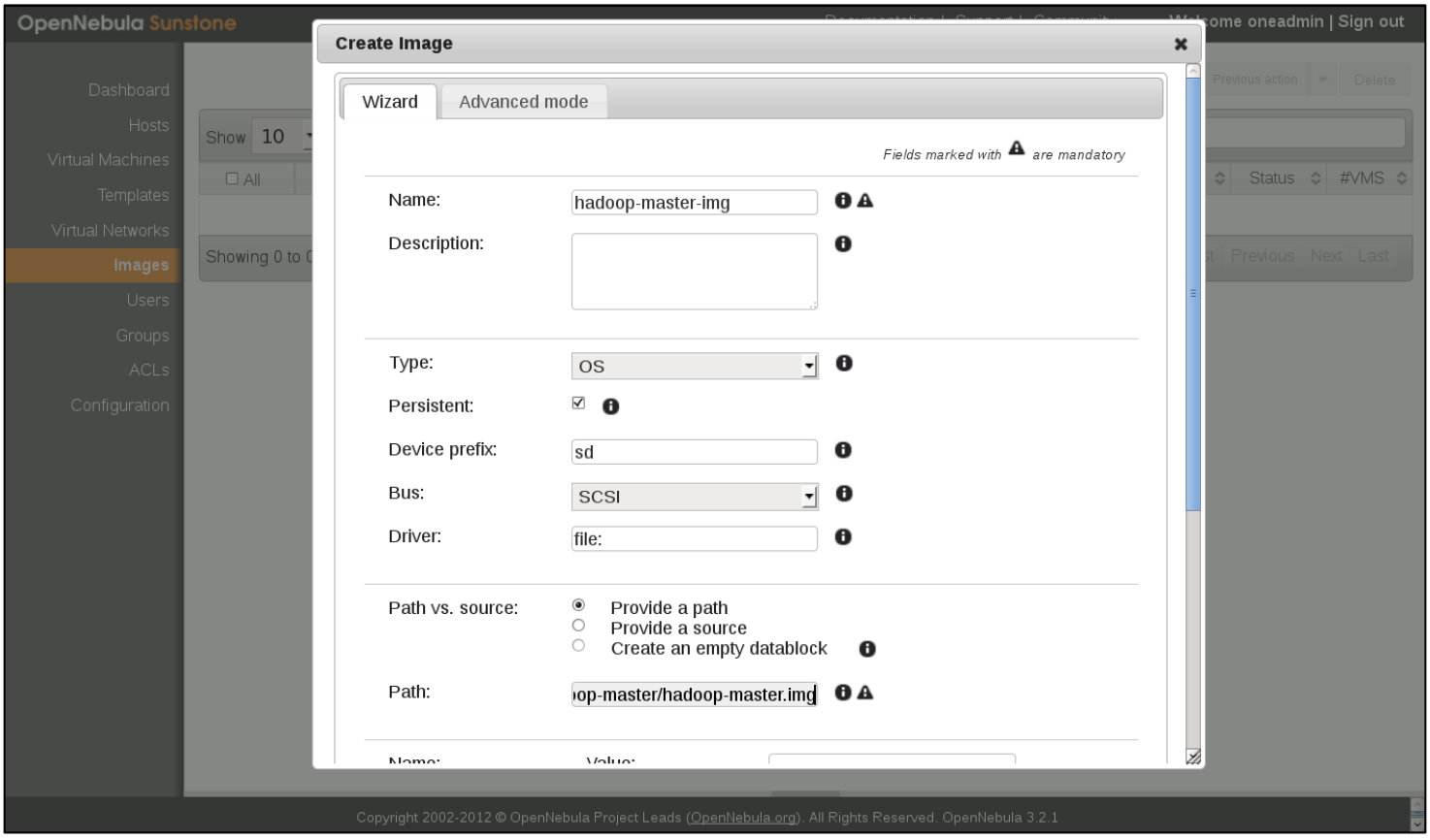

Figura 17: Creación de una imagen en Sunstone

2. La imagen se ha copiado al repositorio local de OpenNebula, en nuestro caso /var/lib/one/images. Realizamos el mismo proceso para las otras tres imágenes restantes.

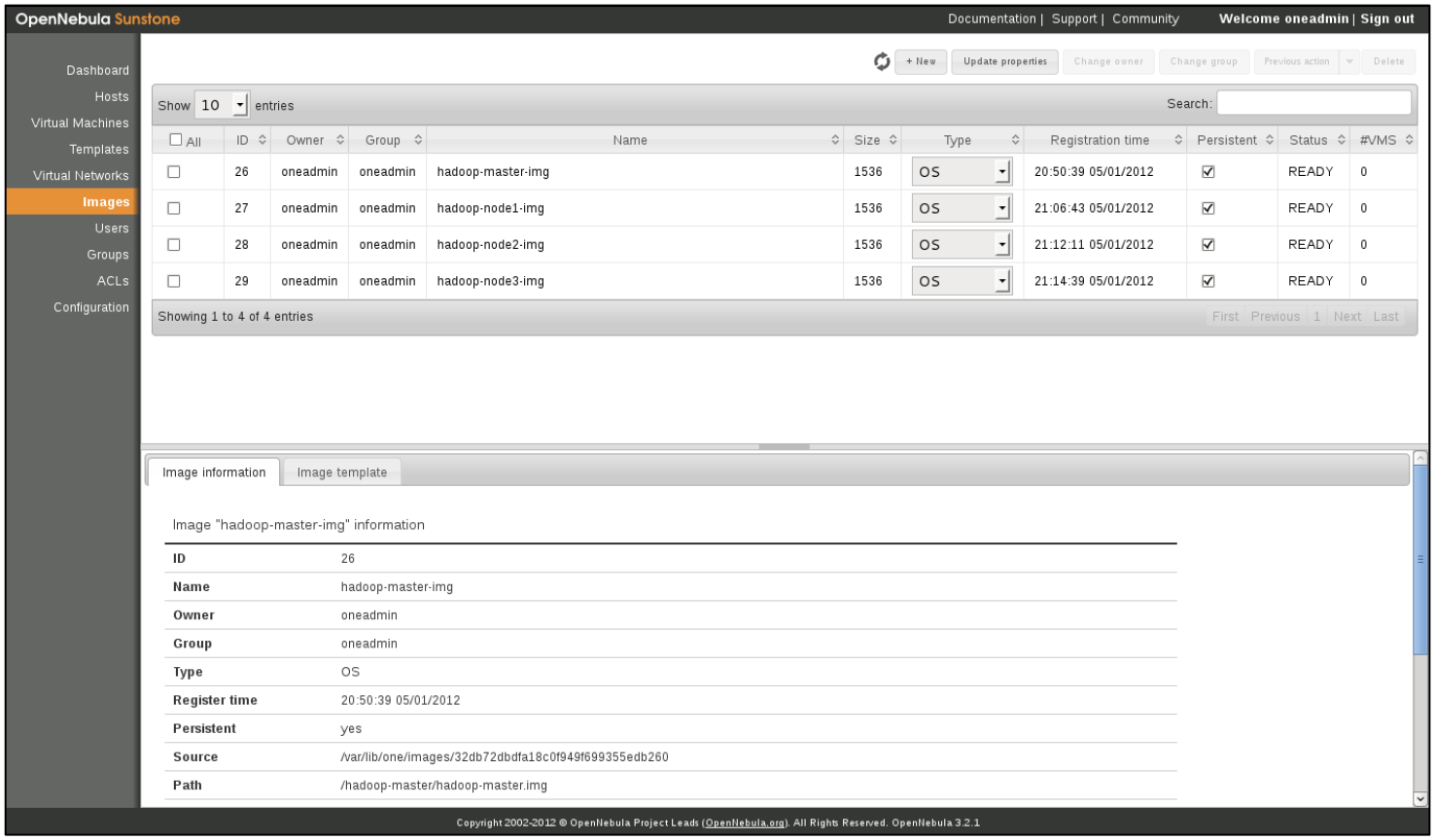

Figura 18: Información de las imágenes creadas en Sunstone

3. Creamos una plantilla para cada máquina virtual que vamos a desplegar. Pulsamos en el menú Templates y después en +New. Desde esta pantalla configuramos las opciones del arranque dentro del hipervisor, discos y memoria a utilizar, configuración de red, etc.

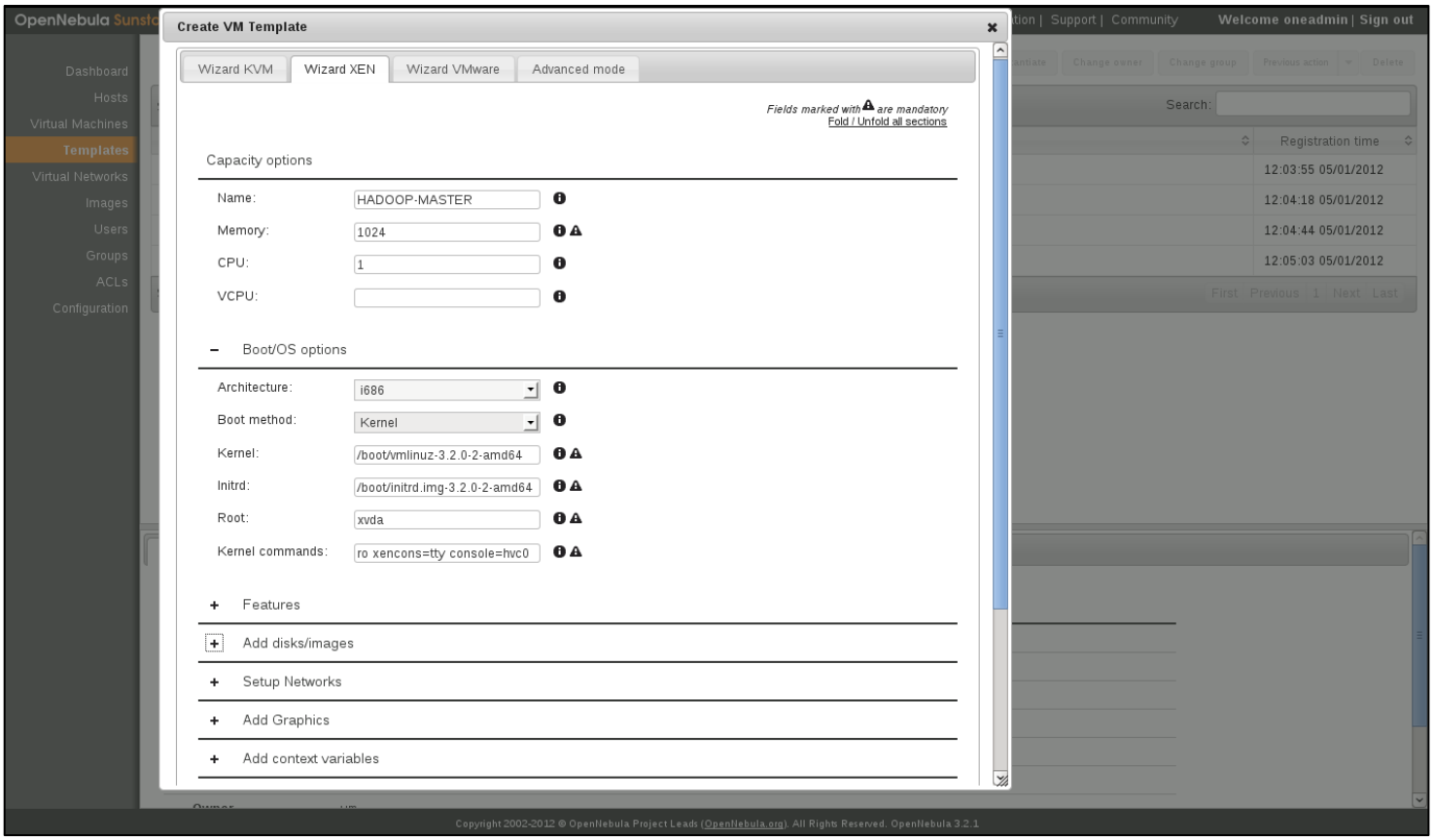

Figura 19: Creación de una plantilla en Sunstone

4. Una vez realizado este proceso disponemos de cuatro plantillas de máquinas virtuales.

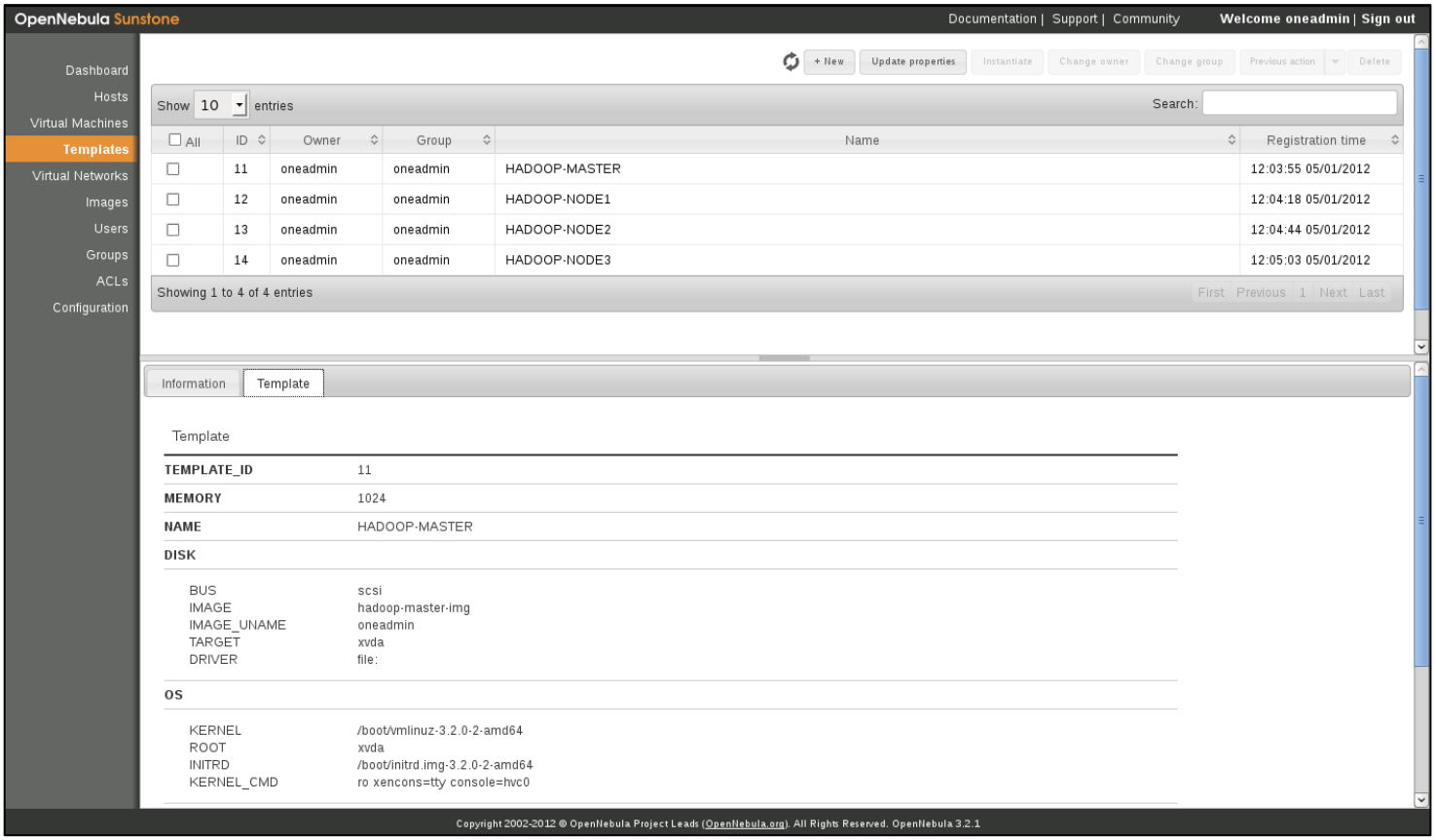

Figura 20: Información de las plantillas creadas en Sunstone

5. El paso final es desplegar cada máquina virtual, en base a una plantilla, en el host elegido. En nuestro caso en el equipo xenserver. Desde el menú Virtual Machine pulsamos en en +New desde donde le damos un nombre a la máquina virtual y la plantilla que utilizaremos de base. Una vez se ha transferido la imagen al host, se arranca automáticamente la máquina virtual.

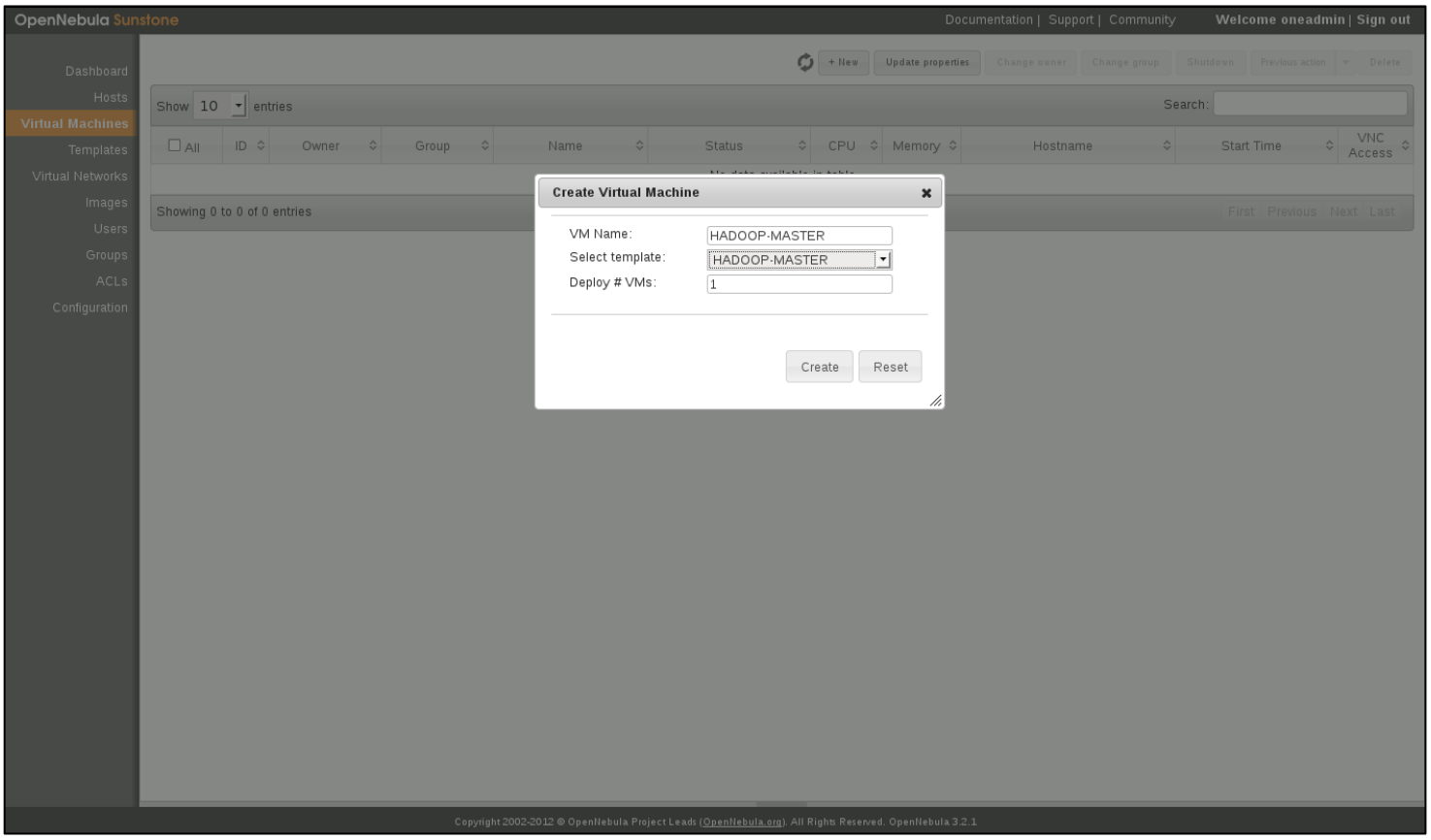

Figura 21: Creación de una máquina virtual en Sunstone

6. Después de realizar la misma operación con las tres plantillas restantes ya tenemos nuestro clúster hadoop ejecutándose sobre un hipervisor. Xen. Ahora, desde SunStone podemos monitorizar el estado de las máquinas virtuales.

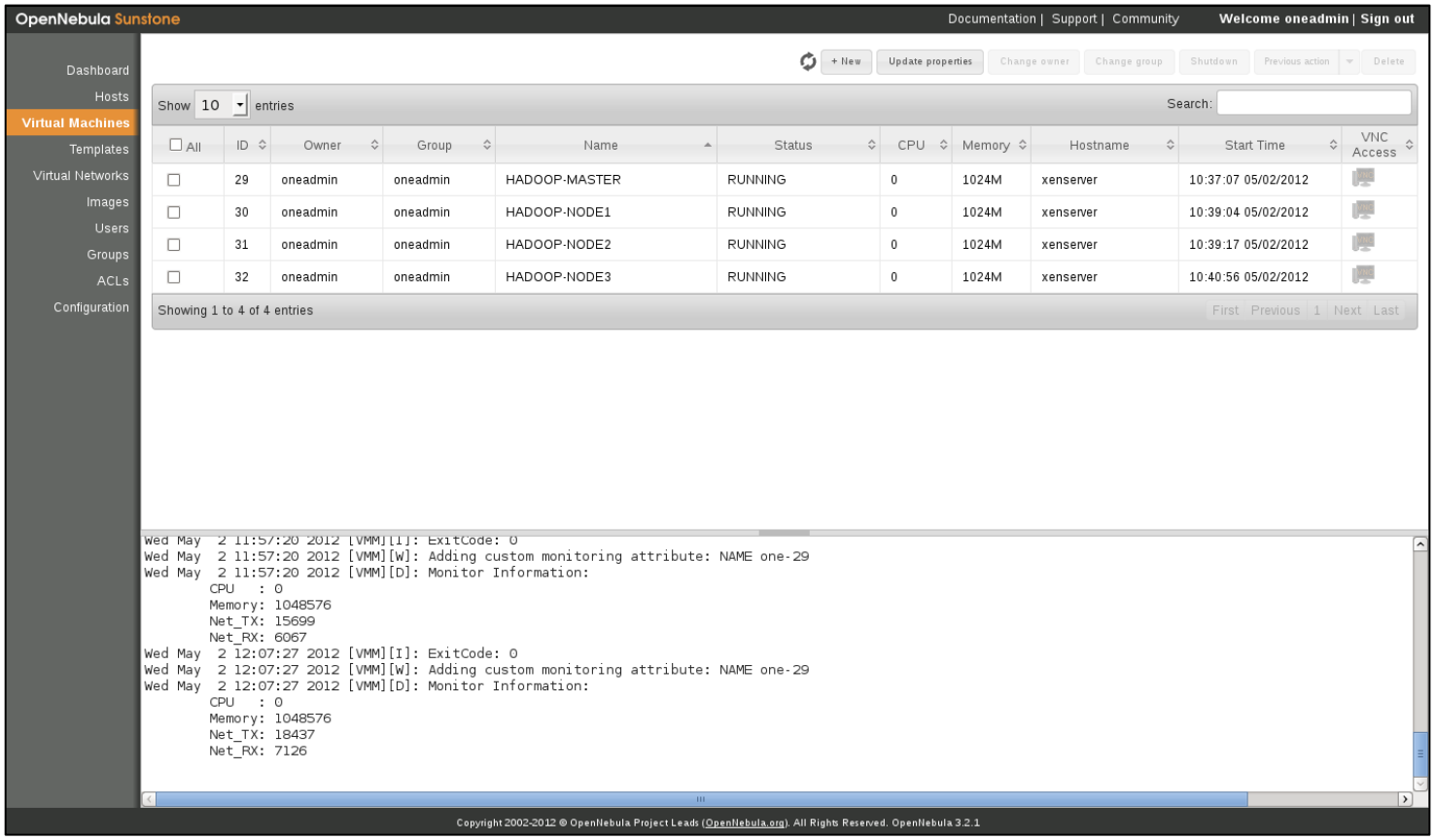

Figura 22: Información de las máquinas virtuales creadas en Sunstone

7. Ajustes finales de Hadoop

Una vez tenemos los nodos ejecutándose en el hipervesor es necesario realizar unos ajustes finales para que el sistema Hadoop empieze a estar operativo:

1. Debemos formatear el sistema de ficheros HDFS para comentzar a utilizarlo. Para ello ejecutamos el siguiente comando desde el nodo hadoop-master, que es el nodo donde va a correr el servico NameNode.

```
hadoop namenode -format
```

2. Iniciamos el servicio NameNode en la máquina `hadoop-master`. Arrancará el servicio DataNode en los nodos que se definieron en el archivo conf/slaves

```
sudo service hadoop-0.20-namenode start
```

3. Creamos los directorios `/tmp`, que establece la variable hadoop.tmp.dir y `/mapred/system` que establece la variable mapred.system.dir.

```
sudo -u hdfs hadoop fs -mkdir /tmp

sudo -u hdfs hadoop fs -chmod -R 1777 /tmp


sudo -u hdfs hadoop fs -mkdir /mapred/system

sudo -u hdfs hadoop fs -chown mapred:hadoop /mapred/system
```

4. Ahora podemos arrancar también el servcio JobTracker en la máquina `hadoop-master`. Arrancará el servicio TaskTracker en los nodos que se definieron en el archivo conf/slaves

```
sudo service hadoop-0.20-jobtracker start
```

Una vez que están todos los nodos funcionando podemos acceder a la información de cada nodo a través de las siguientes direcciones:

http://hadoop-master:50070/: Información del NameNode y del sistema de ficheros HDFS.

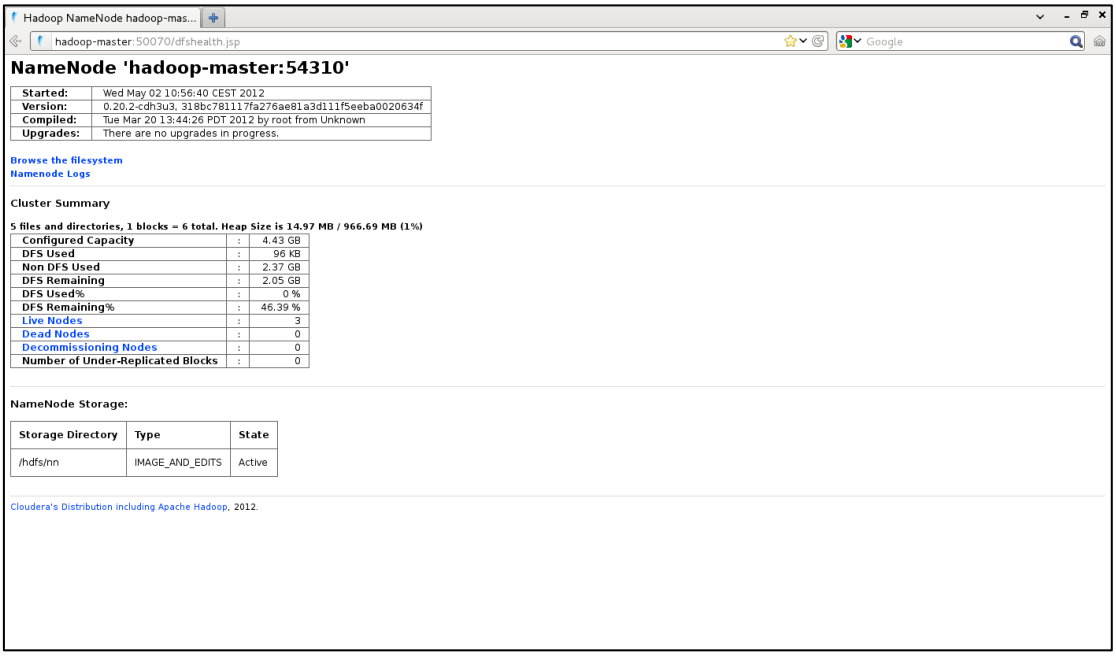

Figura 23: Información del NameNode del clúster y de HDFS

Se puede acceder a la información asociada a cada DataNode del clúster:

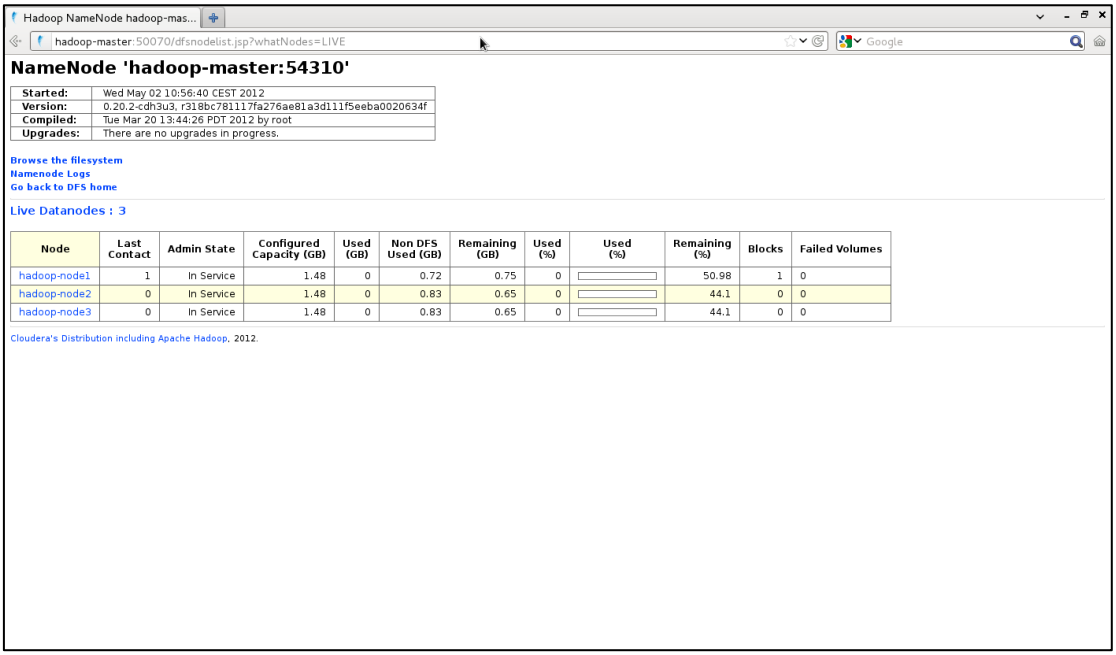

Figura 24: Información de un DataNode del clúster

Es posible navegar por el sistema de ficheros HDFS (aunque hay otras herramientas más amigables para realizar esta tarea).

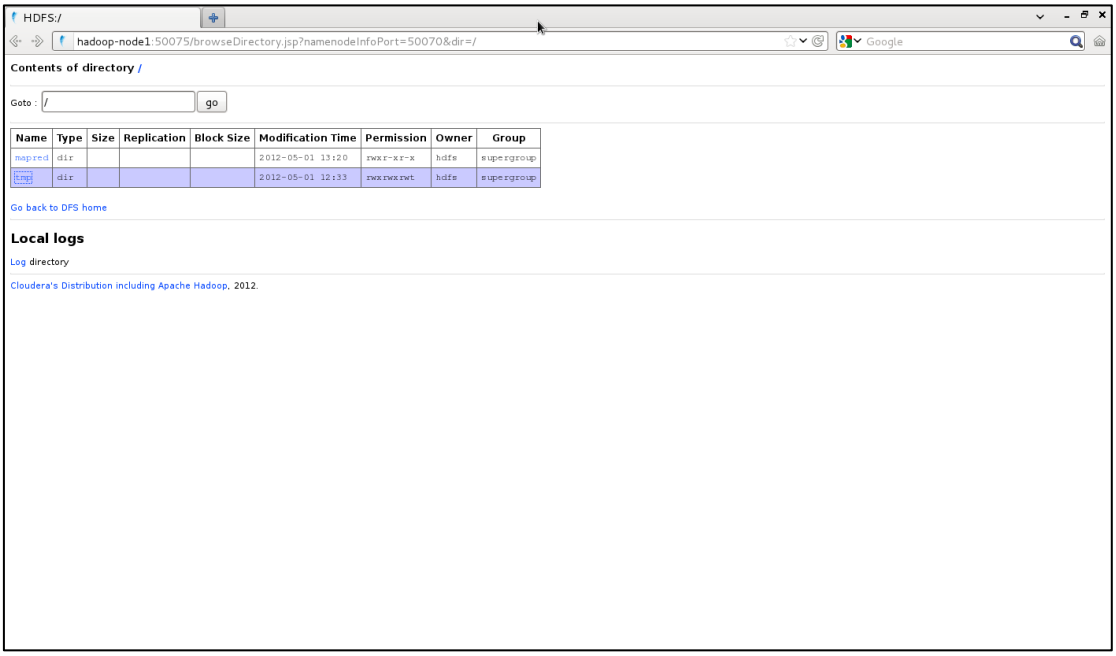

Figura 25: Listado del contenido del sistema de ficheros HDFS del clúster

http://hadoop-master:50030 Información del JobTracker y de los trabajos Map y Reduce.

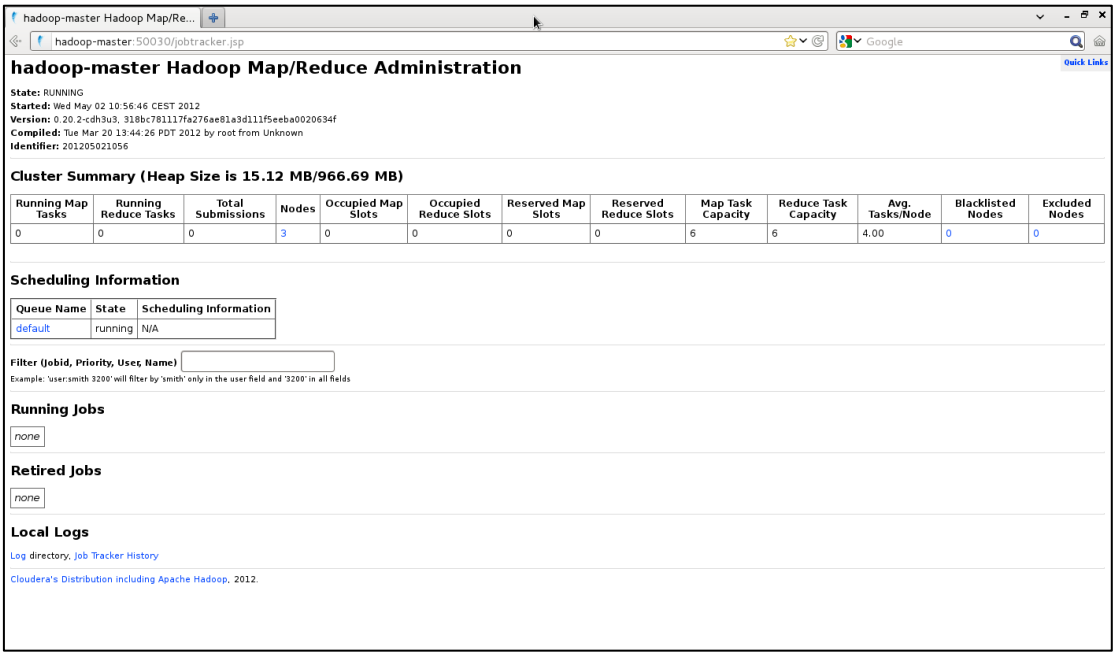

Figura 26: Información del JobTracker del clúster y los trabajos MapReduce

Se puede acceder a la información asociada a cada TaskTracker del clúster:

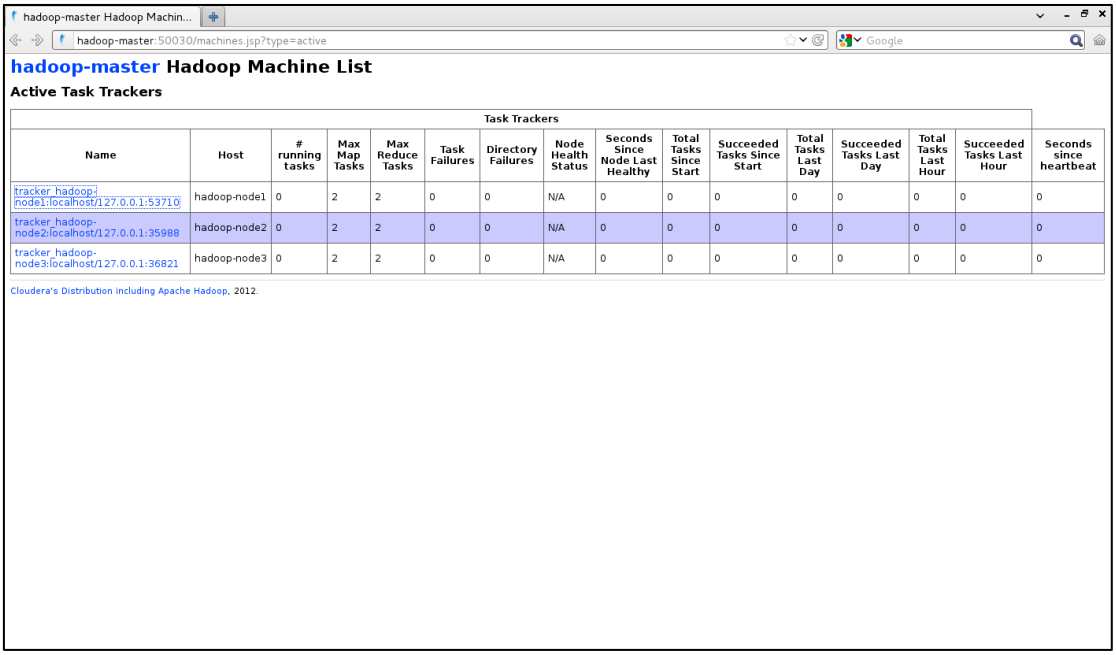

Figura 27: Información de un TaskTracker del clúster

Se puede acceder a la información de cada trabajo MapReduce que se ejecuta dentro de cada TaskTracker del clúster:

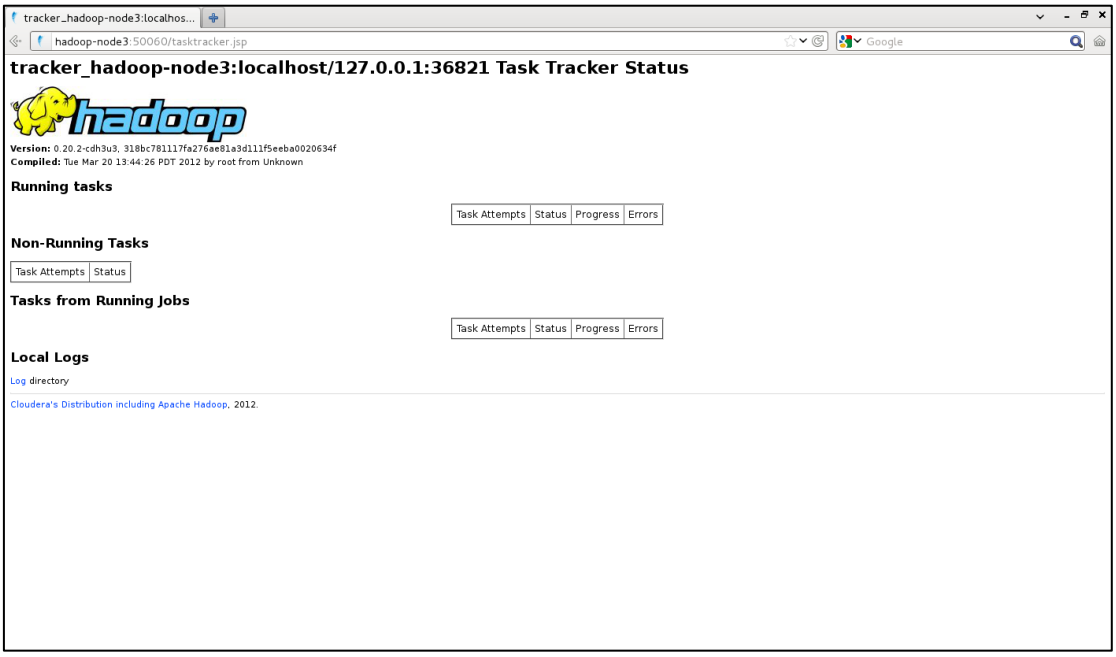

Figura 28: Información de un trabajo MapReduce

8. Pruebas

Ahora que están funcionando los cuatros nodos del clúster vamos a ejecutar una aplicación real que nos permitirá ver en funcionamiento tanto el sistema de ficheros HDFS.

La prueba que vamos a llevar a cabo es la ejecución de una aplicación que cuenta palabas dentro de ficheros. Como datos de prueba nos descargamos los siguientes libros electrónicos en formato ASCII:

- Metamorphosis de Franz Kafka
- Ulysses de James Joyce
- Alice's Adventures in Wonderland de Lewis Carroll
- A Tale of Two Cities de Charles Dickens
- Adventures of Huckleberry Finn de Mark Twain

Creamos un directorio en HDFS

```
sudo -u hdfs fs mkdir -p /libros
```

Vamos copiar estos archivos al sistema de ficheros HDFS con el siguiente comando:

```
sudo –u hdfs fs -copyFromLocal *.txt /libros
```

Podemos ver el contenido del directorio `/libros` con el siguiente comando.

```
sudo –u hdfs fs -ls /libros

Found 5 items

-rw-r--r--   1 hdfs supergroup     792927 2012-05-02 13:18 /libros/A
tale of Two Cities.txt

-rw-r--r--   1 hdfs supergroup     597587 2012-05-02 13:18
/libros/Adventures of Huckleberry Finn.txt

-rw-r--r--   1 hdfs supergroup     167517 2012-05-02 13:18
/libros/Alices.txt

-rw-r--r--   1 hdfs supergroup     141398 2012-05-02 13:18
/libros/Metamorphosis.txt

-rw-r--r--   1 hdfs supergroup    1573150 2012-05-02 13:18
/libros/Ulysses.txt
```

Lanzamos la operación MapReduce:

```
sudo -u hdfs hadoop jar hadoop-*-examples.jar wordcount /libros /out
```

Desde la dirección http://hadoop-master:50030 podemos ver la información del trabajo.

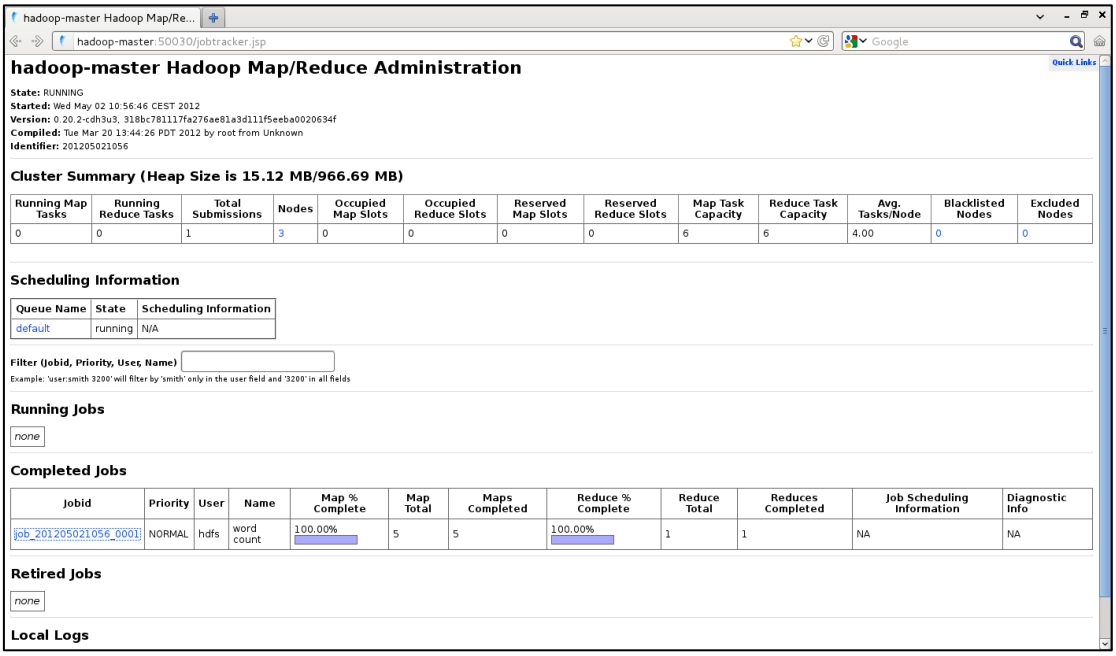

Figura 29: Información del trabajo *wordcount*

Y para cada trabajo las estadísticas de su ejecución:

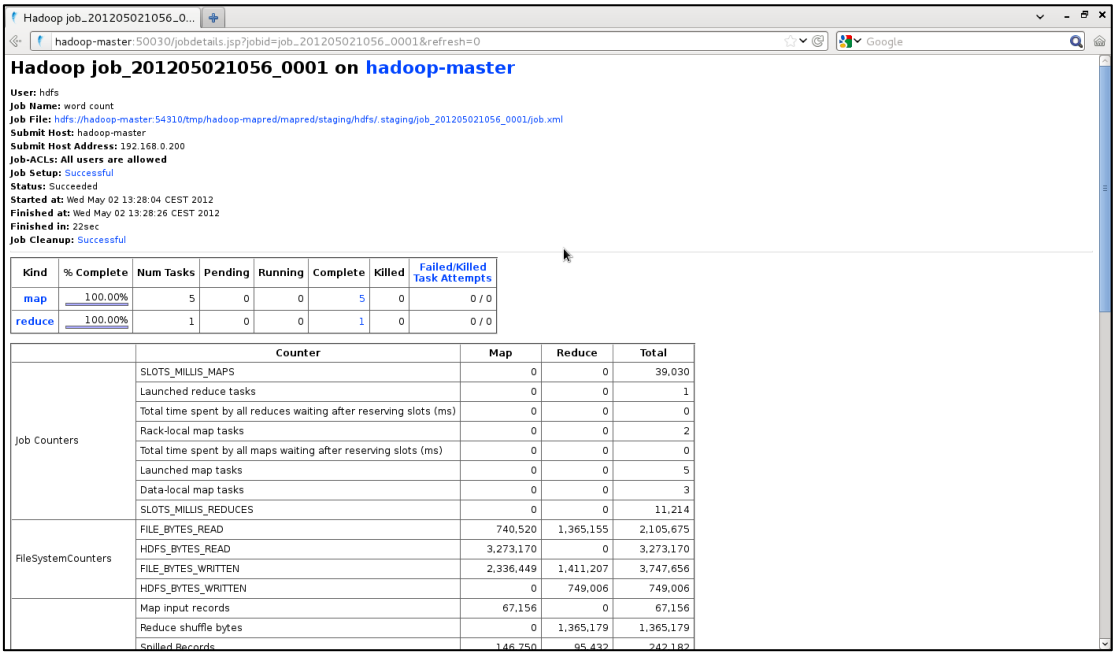

Figura 30: Estadísticas de ejecución del trabajo *wordcount*

9. Posibles mejoras

Vamos a señalar algunas posibles mejoras que se podrían haber implementado en la aplicación práctica pero que no se han llevado a cabo por falta de tiempo pero que facilitarían la gestión de la infraestructura en el mundo real:

- Utilización de redes virtuales
- Uso de la contextualización.

IX. Conclusiones

Cloud Computing no es solo una tendencia o moda actual sino que es una realidad. El usuario utiliza todos estos servicios en la nube diariamente y ya se está habituando a tener sus datos accesibles a través de las aplicaciones en cualquier lugar e independientemente del dispositivo de acceso que utilice.

Cloud Computing se encuentra en una fase relativamente temprana de desarrollo, pero cada vez más madura ya que cuenta con el apoyo de grandes empresas como Amazon, Google o Microsoft, lo que permite decir que su crecimiento es ya imparable.

Desde el punto de vista empresarial, uno de los grandes beneficiarios del Cloud Computing son las Pymes ya que por un coste mucho menor pueden disponer de una infraestructura de proceso y almacenamiento que sería casi imposible de pagar si estas Pymes tuvieran que implantarlas cada una individualmente.

En cambio, algunas grandes empresas son reacias a ceder sus datos a terceros, por lo que prefieren utilizar nubes privadas (como las que proporciona OpenNebula). Este tipo de nubes permiten mantener el control de los datos pero como contrapartida no se puede aprovechar todas las ventajas del Cloud Computing como trasladar toda la complejidad de la gestión de infraestructuras a un tercero especializado en ese campo.

De los tres modelos de servicio que hemos visto, los proveedores de servicio pueden trabajar en uno o varios de ellos a la vez.

Hemos visto en el Capítulo X que es posible desplegar y dar servicio a terceros mediante una infraestructura Cloud Computing utilizando herramientas Open Source. Al integrar los dos modelos de servicio (IaaS y PaaS) en una sola solución podemos prestar servicios en las dos capas similares a los que pude proporcionar un gran proveedor pero a una escala más pequeña.

Por otra parte, y como se ha dicho anteriormente, no todas las empresas están dispuesta a ceder a sus datos. Una razón puede ser que el país donde radica el proveedor no tiene la obligación de cumplir con LOPD por lo que la empresa buscaría un proveedor local para que le proporcione servicios en la nube según la normativa española.

Uniendo estas dos ideas, la posibilidad de desplegar infraestructuras Cloud Computing con herramientas Open Source y la necesidad de algunas empresas de almacenar sus datos en empresas que cumplan la LOPD, se puede abrir un mercado para empresas locales que, utilizando herramientas Open Source (no hay que pagar licencias y existen abundante documentación pública), presten estos servicios en la nube.

En lo referente a la implantación práctica de un clúster Hadoop utilizando OpenNebula me he encontrado con algún problema debido a la falta de una infraestructura similar al mundo real. No se han podido hacer pruebas, por ejemplo, de escalabilidad o mejoras de rendimiento al añadir nuevos nodos ya que las limitaciones hardware de la infraestructura de laboratorio no lo hacían factible.

Finalmente señalar que el desarrollo del presente proyecto me ha servido para aplicar los conocimientos adquiridos durante mis estudios de Ingeniería Informática tanto los referentes a Redes de Computadores como Computación Distribuida o Gestión de Proyectos entre otros.

X. Anexo I: Código fuente del ejemplo *wordcount* de Hadoop

```
package org.myorg;
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }
public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
```

```
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
 }
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);}}
```

Utilizando la API de Hadoop siempre tenemos que definir una clase que implemente los interfaces Mapper y Reducer, que serán las encargadas de realizar las funciones Map y Reduce del algoritmo. Para ejecutar este programa solo habría que empaquetar la clase en un archivo .jar y realizar la siguiente llamada:

```
hadoop jar wordcount.jar
```

XI. Bibliografía

Cloud Computing

http://www.tiaonline.org/standards/TIA_Cloud_Computing_White_Paper.pdf

http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

http://www.ibm.com/developerworks/library/ar-archman10/

http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031

http://en.wikipedia.org/wiki/Cloud_computing

http://en.wikipedia.org/wiki/Virtualization

http://www.ibm.com/developerworks/linux/library/l-linuxvirt/

OpenNebula

http://www.opennebula.org/documentation

http://www.c12g.com/

http://occi-wg.org/

Hadoop

http://www.ibm.com/developerworks/aix/library/au-cloud_apache/

http://www.ibm.com/developerworks/linux/library/l-hadoop/

http://en.wikipedia.org/wiki/MapReduce

http://hadoop.apache.org/common/docs/current/hdfs_design.html

http://wiki.apache.org/hadoop/PoweredBy

Aplicación Práctica

http://wiki.stocksy.co.uk/wiki/Xen_on_Debian_Squeeze_dom0

https://ccp.cloudera.com/display/CDHDOC/CDH3+Installation

http://archive.cloudera.com/cdh/3/hadoop/cluster_setup.html#Configuration

http://www.mapr.com/

http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/