Universidad de Almería

Titulación de Ingeniero en Informática

PROCESAMIENTO DE GRANDES VOLÚMENES DE DATOS EN ENTORNOS CLOUD COMPUTING UTILIZANDO HADOOP MAPREDUCE

Autor:
Carlos Gómez Martínez

Directores:
Nicolás Padilla Soriano
Julio Gómez López

Almería, Abril de 2013

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

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

Las licencias para la mayoría del software y otros trabajos prácticos están destinadas a suprimir la libertad de compartir y modificar esos trabajos. Por el contrario, la Licencia Publica General GNU persigue garantizar su libertad para compartir y modificar todas las versiones de un programa y asegurar que permanecerá como software libre para todos sus usuarios.

Cuando hablamos de software libre, nos referimos a libertad, no a precio. Las Licencias Publicas Generales están destinadas a garantizar la libertad de distribuir copias de software libre (y cobrar por ello si quiere), a recibir el código fuente o poder conseguirlo si así lo desea, a modificar el software o usar parte el mismo en nuevos programas libres, y a saber que puede hacer estas cosas.

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

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

Quiero aprovechar estas primeras líneas para agradecer a mis directores Nicolás Padilla Soriano y Julio Gómez López su dedicación, su esfuerzo y todo el conocimiento que me han aportado.

Además, dado que con este documento cerraré un ciclo, quiero acordarme también de mis compañeros de la carrera, especialmente de Eugenio, Luismi y Esther, que son para mí algo mas que simples compañeros, gracias por todos estos años.

Por supuesto, no puedo dejar de lado a mis padres, que han sido la principal fuente de apoyo todos estos años, muchas gracias por vuestra paciencia. Junto con mis padres hay otra persona que aporta mucho a mi vida, gracias Ana.

Contenido

Introducción

La **mejora de las comunicaciones de red** y el hecho de que cada vez sea **más sencillo y menos costoso el acceso** a la misma, está generando un modo de vida en el que podemos conectarnos a Internet desde casi cualquier dispositivo y en cualquier lugar. Este incremento ha provocado que las conexiones de red cada vez soporten mayor tráfico de datos y con mayor velocidad.

Una consecuencia positiva de este **aumento de velocidad y de cantidad de tráfico soportado** es que comienzan a proliferar entornos de computación distribuida que utilizan la red para entablar sus comunicaciones. Este tipo de entornos cada vez alcanzan mayor rendimiento y a menor coste, dado que pueden integrarse servidores en distintas localizaciones físicas y de distintas características hardware en un sistema comunicado en red.

Otra de las secuelas positivas que se ha producido ha sido la **aparición del paradigma de computación *Cloud Computing***. Este tipo de sistemas combina en un sistema de computación único equipos que no tienen por qué estar físicamente en la misma ubicación e incluso no tienen que ser iguales a nivel hardware, integrando todos sus recursos como si de un solo sistema se tratase. Beneficiándose de las comunicaciones de red se crea un sistema en el que varios servidores trabajan de manera conjunta y aprovechan todos sus recursos poniéndolos a disposición del entorno global. Los sistemas *Cloud Computing* ofrecen a los usuarios a través de la red recursos, sistemas completos o servicios que de otra forma serían muy costosos de implantar tanto por costes hardware como de trabajo de configuración e implantación. Estos sistemas **permiten al usuario abstraerse del hardware y trabajar directamente sobre el entorno aprovechando las bondades de la virtualización de servidores**. La **idea principal** de estos entornos de servidores que forman un *Cloud*, es poder **desplegar instancias virtuales que permitan aprovechar al máximo los recursos**. Una de las principales ventajas es que no se trabaja directamente sobre los servidores y puede crear y eliminar instancias virtuales sin preocuparse sobre la compatibilidad o no del hardware y los drivers del sistema.

El objetivo principal de los entornos *Cloud Computing* es **dotar al usuario de servicios como infraestructuras informáticas completas, plataformas para el desarrollo de aplicaciones o software, de forma general**. Esta es la prestación de la empresa proveedora del *Cloud* a sus usuarios, siendo para estos últimos indiferente el hardware. Además, dicha empresa es la encargada del establecimiento y el mantenimiento de la infraestructura que da soporte a sus servicios. Se trata de una tendencia de computación cada vez más fiable, con una escalabilidad elástica capaz de atender exigentes cambios en la demanda sin que ello suponga incrementos en los costes de administración y gestión. Su introducción ha permitido definir un nuevo modelo por consumo de servicios adaptando nuevas formas para el cobro por recursos consumidos. Sin embargo, este entorno de desarrollo presenta algunas dudas sobre su

implantación, sobre todo debido a aspectos legales, de seguridad y de preocupación por la ubicación física de los datos.

La mejora en las comunicaciones y el aumento de usuarios de la red mencionados trae consigo que **cada vez sea mayor la cantidad de datos que viajan** por la misma y surge la **necesidad de crear entornos de computación capaces de almacenarlos y procesarlos**. En este proyecto se va a tratar de utilizar un sistema *Cloud Computing* para integrar en él un entorno de computación distribuida capaz de procesar grandes cantidades de datos alcanzando un rendimiento eficiente.

Recientemente se ha creado un **entorno de computación distribuida de alto rendimiento que tiene la capacidad de procesar grandes volúmenes de datos**. Ha sido desarrollado y es mantenido por la empresa ***Apache*** y su nombre es ***Hadoop***. Este entorno de computación permite **combinar en un mismo clúster servidores con distinto hardware para que trabajen de manera distribuida configurando en cada nodo dos partes bien diferenciadas, una dedicada al almacenamiento y otra al procesamiento de los datos**. El procesamiento de los datos en este marco de trabajo está **orientado al uso de programas escritos en el lenguaje de programación *MapReduce***. Es un lenguaje basado en *Java* y cuya principal característica es la de **procesar importantes conjuntos de datos de entrada dividiendo el trabajo global en tareas que son realizadas de manera distribuida** dentro del clúster *Apache Hadoop*.

El objetivo principal del proyecto es unir y verificar las ventajas que proporcionan ambos entornos, *Cloud Computing* y computación distribuida de grandes volúmenes de datos con *Apache Hadoop* y *MapReduce*. Para ello es necesario desplegar una nube que permita aprovechar al máximo los recursos físicos disponibles mediante la instanciación de máquinas virtuales, creando dentro de ella un entorno de procesamiento distribuido *Apache Hadoop MapReduce*. Estas instancias de máquinas virtuales desplegadas en el entorno Cloud son las que formarán un clúster *Hadoop*, con tantas máquinas como los recursos permitan. Este clúster sirve de base para la realización de pruebas con aplicaciones que trabajen sobre grandes volúmenes de datos para demostrar los beneficios derivados de la integración. **Todo el software que se va a emplear será de libre distribución, bajo licencia GPL** (*General Public License*). Una vez puesto en marcha el clúster de programación *MapReduce*, se va a proceder a comprobar la potencia del sistema con varios ejemplos a desarrollar donde se tratará con diferentes tipos de datos como pueden ser texto, imágenes, etc. La meta final es conocer el entorno de programación *Apache Hadoop MapReduce* y comprobar su rendimiento para el procesamiento de grandes volúmenes de datos integrado en un entorno *Cloud Computing*.

1. Introducción a *Cloud Computing*

Cuando hablamos de *Cloud Computing* o de Computación en la Nube, estamos hablando de un nuevo paradigma de computación en el que los usuarios ejecutan servicios alojados en la red. Estos usuarios no saben dónde se ejecutan esos servicios que utilizan, ellos simplemente acceden a esos recursos como un **servicio** (véase la *Figura 1.1*). La computación en la nube consiste en un nuevo concepto en la oferta de servicios a través de Internet en el que el alojamiento de los mismos es transparente al usuario.

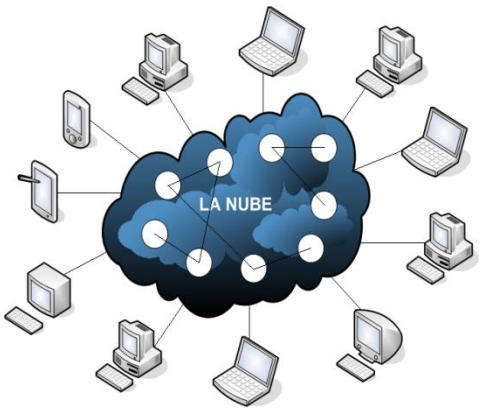


Figura 1.1. Esquema Cloud Computing

El concepto fundamental de la Computación en la Nube es el de ofrecer los recursos de un sistema de información sin que los usuarios tengan conocimientos avanzados sobre estos recursos, su configuración o su mantenimiento y administración. Este concepto permite que los usuarios puedan usar servicios y pagar por ello de una manera flexible y que se ajuste realmente a sus necesidades, pagando en función del consumo realizado.

La mayoría de las ventajas de *Cloud Computing* vienen dadas gracias al uso de la virtualización, que convierte al sistema en una infraestructura de servicios escalable, dinámica y automatizable. Una capacidad destacable de la computación en la nube es la de poder migrar servicios entre servidores y adaptarse a la demanda registrada por los usuarios de determinados servicios.

Los entornos *Cloud Computing* ofrecen a sus usuarios múltiples tipos de sistemas, pero una característica común de todos ellos es el ahorro en costes hardware y administración que aportan al cliente. Contratando un entorno de Computación en la Nube puede llegar a tener acceso a un sistema que, de otro modo, supondría costes muy elevados, tanto en adquirir los componentes hardware como en tiempo para configurarlos.

1.1. Tipos de Nube

Pueden clasificarse cinco tipos de nubes en función de quién las administra y utiliza:

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

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

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

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

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

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

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

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

En la *Figura 1.2* pueden ver de forma esquemática en qué consisten los distintos tipos de nubes.

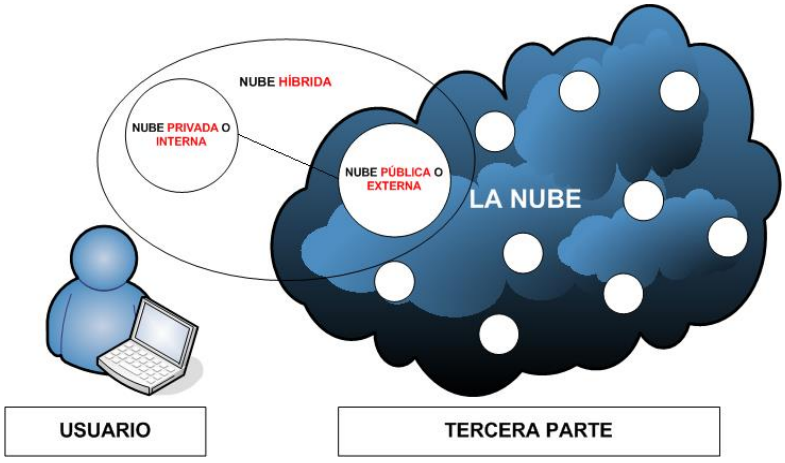


Figura 1.2. Tipos de Nube

1.2. Tipos de servicios ofrecidos por la nube.

Los servicios que se ofrecen mediante la aplicación de *Cloud Computing* dependen de las necesidades, negocios, objetivos,… de la organización en la que se desea implantar. Generalmente se clasifican en tres tipos:

- **Infraestructura como servicio** (*Infrastructure as a Service - IaaS*). El servicio ofrecido a los usuarios de la nube consiste en la infraestructura necesaria para utilizar el entorno de ejecución y software deseados. Fundamentalmente consta de recursos de **capacidad de computación, conexión y almacenamiento como servicio** (servidores, sistemas de almacenamiento local y compartido, conexiones de red, *routers*,…), por lo que en ocasiones también es denominado como *HaaS* (*Hardware as a Service*).
- **Plataforma como servicio** (*Platform as a Service - PaaS*). El servicio proporcionado por la nube es un entorno completo sobre el que puede trabajar con el software, aplicaciones, servicios, etc., que consideramos necesarios en la organización. Es decir, la abstracción de un ambiente de desarrollo junto a una serie de servicios es encapsulada y entregada al cliente/usuario, permitiendo llevar a cabo cada una de las fases de desarrollo y prueba del software tanto de forma genérica como especializada.

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

En la *Tabla 1.1. Tipos de servicios Cloud Computing* puede ver de forma resumida los tres tipos de servicios que se pueden ofrecer con C*loud Computing*, además de algunas soluciones ejemplo que permiten desplegar nubes para prestar esos servicios.

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

Tabla 1.1. Tipos de Servicios Cloud Computing

1.3. Ventajas y desventajas de *Cloud Computing*

A primera vista una tecnología como *Cloud Computing* modifica los modelos de negocio existentes aportando numerosas ventajas tanto a los proveedores de los servicios de la nube como a los usuarios de esos servicios. Por ejemplo, a los primeros permitiéndoles ofrecer un mayor número de servicios en la red, de forma estandarizada, rápida y eficiente en su despliegue, y a los segundos facilitándoles un acceso a servicios de forma inmediata, configurados y de forma transparente. Por el contrario, esta tecnología genera desconfianza y cierto escepticismo en algunos aspectos de seguridad y es además un modelo dependiente del estado de la red.

Las ventajas más importantes que reporta el modelo de computación en la nube son:

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

Por el contrario y, como hemos comentado antes, también posee desventajas, casi todas ellas derivadas de la seguridad y la desconfianza que puede generar, son las siguientes:

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

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

Si trasladamos las ventajas y desventajas a nuestro proyecto, apreciamos que son muchas más las prerrogativas que los inconvenientes que nos afectan. Es decir, si analizamos una a una las desventajas que directamente nos afectan, vemos que, al alojar nuestra nube en una red interna y privada (el laboratorio) el sentimiento de seguridad aumenta, ya que se reducen las posibles amenazas, así como es poco probable que disminuya la velocidad en la red o se produzcan sobrecargas ya que controlamos el número de usuarios de la conexión. Además, al tratarse de un tipo de nube privada o interna, tenemos en propiedad los servidores físicamente y sin que nadie más tenga acceso a ellos. Concluimos que en nuestro caso, este modelo de computación es excepcionalmente apropiado dadas las condiciones y los requisitos.

1.4. Estudio de las tecnologías *Cloud Computing*

Para el desarrollo del proyecto, es necesario disponer de una infraestructura eficiente y que nos proporcione el soporte que precisamos. Hemos de localizar un entorno *Cloud Computing* que se ajuste a las necesidades de nuestro marco de trabajo. A continuación, se va a presentar en la *Tabla 1.2* una enumeración de algunas de las distintas nubes que podemos encontrar y posteriormente, iremos comentando los aspectos más reseñables de cada una de ellas.

Nombre	Logotipo	Página Oficial
Abiquo	abiquo	http://www.abiquo.com/
Bitnami	BITNAMI	http://bitnami.org/
CA 3Tera AppLogic	ca technologies	http://www.ca.com/us/cloud-platform.aspx
Cloud.com CloudStack	cloudstack open source cloud computing	http://www.cloudstack.com/

Collectd		http://collectd.org/
ControlTier		http://controltier.org/wiki/Main_Page
Convirture ConVirt		http://www.convirture.com/index.php
Enomaly plataforma Elastic Computing (ECP)		http://src.enomaly.com/
Eucalyptus		http://www.eucalyptus.com/
Extility Flexiant		http://www.flexiant.com/2011/11/18/extility-1-5-2-release/
HP CloudSystem		http://www8.hp.com/us/en/business-solutions/solution.html?compURI=1079455#.UGwT_K416ZQ
IBM CloudBurst		http://www-01.ibm.com/software/tivoli/products/cloudburst/
Incontinuum CloudController		http://www.incontinuum.com/
Nimbula Director		http://nimbula.com/
OnApp		http://onapp.com/cloud/pricing/

OpenNebula		http://opennebula.org
OpenQRM		http://www.openqrm.com/
OpenStack		http://www.openstack.org/
Opscode Chef		http://wiki.opscode.com/display/chef/Home
Parallels Automation Cloud Infrastructure (CI)		http://www.parallels.com/es/products/iaas/
Puppet		http://reductivelabs.com/products/puppet/
StackOps		http://www.stackops.com/
VMware vCloud		http://www.vmware.com/products/datacenter-virtualization/vcloud-suite/overview.html
Xen plataforma de nube (XCP)		http://www.xen.org/products/cloudxen.html
Zenoss		http://community.zenoss.org/index.jspa

Tabla 1.2. Distribuciones *Cloud Computing*

En las distribuciones *Cloud Computing* mencionadas en la tabla anterior hay un gran repertorio de nubes que pueden ajustarse a prácticamente todos los requisitos que se precisen. En los siguientes apartados, vamos a comentar cada una de ellas, haciendo mayor hincapié en las que son más destacadas, bien sea por su popularidad o bien porque se ajusten más a lo que este proyecto requiere.

1.4.1. *Abiquo*

Abiquo fue diseñada para **evitar** la **dependencia** de cualquier **hipervisor**. Soporta los principales del mercado, incluso simultáneamente y permite la conversión de máquinas virtuales de un hipervisor a otro en cualquier combinación, eliminando la dependencia de estos proveedores de tecnología con tan solo una operación de "arrastrar y soltar". Los hipervisores soportados son *VMware ESX, ESXi, Microsoft Hyper-V, Citrix XenServer, Xen, KVM y Virtual Box.*

Abiquo es compatible con todos los motores de virtualización comunes. Diseñado para múltiples clientes y disponible en una edición de comunidad libre y una versión para empresas con diferentes características.

1.4.2. *Bitnami*

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

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

1.4.3. *CA 3Tera AppLogic*

Plataforma comercial de *3Tera* basado en *Xen*, diseñada para el hardware de los productos básicos sin la necesidad de una *SAN* debido a su solución integrada de almacenamiento distribuido. Interfaz Web, API y la medición de los recursos se incluyen en la solución llave en mano *AppLogic*.

1.4.4. *Cloud.com CloudStack*

Diseñado para soluciones *multi-tenant* con soporte para *XenServer, VM y VMware vSphere. CloudStack Cloud.com* soporta facturación/medición, interfaz Web, API basados en los estándares existentes y la creación de redes virtuales con la segmentación del tráfico de red en *VLANs*.

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

CloudStack puede desplegarse en uno o más servidores de gestión de tal forma que se conectan a una única base de datos *MySQL*. Opcionalmente, se pueden distribuir las peticiones Web mediante el empleo de gestores de balanceo de carga. Además, se puede realizar una copia de seguridad de la base de datos del servidor de gestión empleando la replicación *MySQL* en un sitio remoto. Como se puede observar en la *Figura 1.3*, existen varios servidores de gestión que reciben las peticiones que gestionan los distribuidores de carga. Además, la base de datos se encuentra replicada.

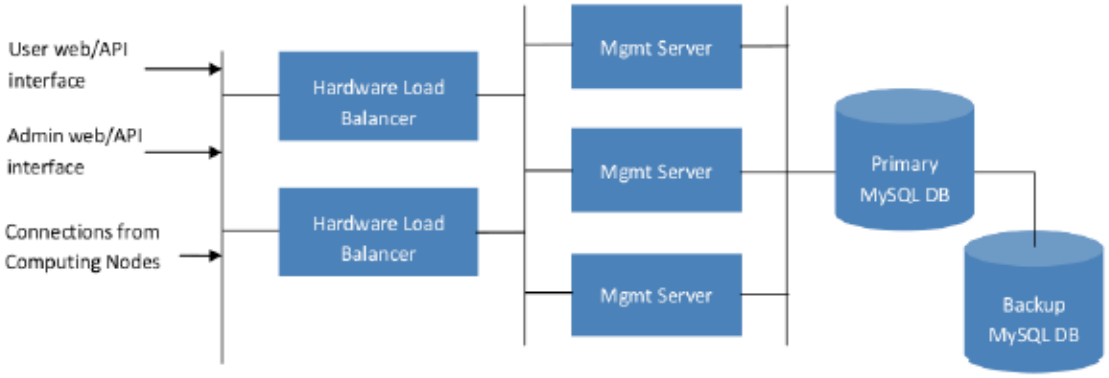


Figura 1.3. *CloudStack*

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

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

El administrador del sistema puede determinar:

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

1.4.5. *Convirture ConVirt*

Convirture es una empresa que trata de ayudar a las organizaciones a administrar las plataformas de virtualización de código abierto. Esta empresa basa todo su desarrollo *Cloud Computing* en *Ubuntu* y, de hecho, los usuarios de dicho sistema operativo pueden instalar el administrador del entorno igual que cualquier otro paquete en *Ubuntu*, siempre y cuando esté habilitado en el Repositorio de Asociados de Negocios.

La versión Beta de *ConVirt Enterprise Cloud* es una herramienta que se utiliza para la gestión de las infraestructuras de nube que tiene soporte para muchas plataformas, incluyendo *KVM*, *Xen*, *OpenStack*, *Amazon EC2* y *VMware*.

En la versión *ConVirt* 2.0, la novedad más destacada es que proporciona una herramienta que, de una manera sencilla, permite a los usuarios administrar varias instancias virtuales. Permite a los usuarios administrar los entornos virtuales basados tanto en *Xen* como en *KVM*. Además es una gran herramienta para los que operan en un entorno de nube privada. También ofrece herramientas para implementar y administrar los servicios de nube desde *Eucalyptus*, el cual es una implementación de las *APIs Amazon EC2*.

1.4.6. *Enomaly plataforma Elastic Computing (ECP)*

Diseñado para proveedores de servicios con la automatización en mente, con la capacidad de integrarse con los sistemas de facturación existentes a través de *Enomalys API*. Incluye interfaz de autoservicio web y el apoyo a los hipervisores más comunes, así como una función especial que permite a los proveedores *Enomaly* vender capacidad adicional a través de su plataforma *SpotCloud*.

Enomaly´s Elastic Computing Platform (ECP) permite crear infraestructuras C*loud Computing* para el despliegue de negocios y servicios tanto pequeños como medianos o grandes.

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

En la *Figura 1.4* puede ver de forma esquemática, extraída de la web oficial del proyecto *ECP* (http://src.enomaly.com/), la forma en que las nubes implementadas con esta solución operan a través de sus proveedores.

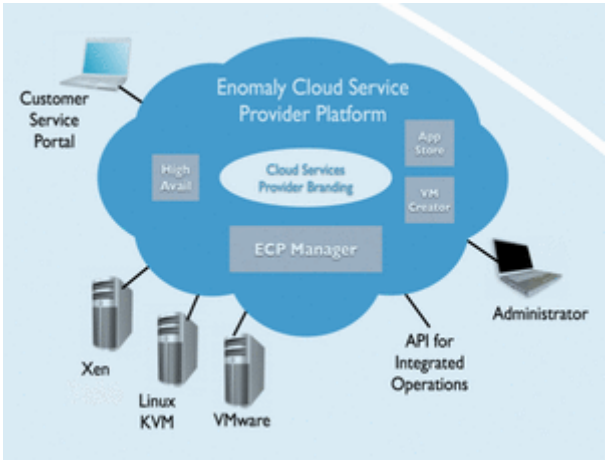


Figura 1.4. Diagrama ECP

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

1.4.7. *Eucalyptus*

Plataforma de nube abierta, disponible con soporte comercial, que se inició originalmente como un proyecto universitario, pero comercializado desde entonces. Diseñado para ser independiente del hipervisor y compatible con el utilizado *API EC2*.

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

Eucalyptus, en sus inicios fue la base *Cloud* de la distribución *Ubuntu Enterprise Cloud (UEC)* incluida en la edición servidor de *Ubuntu* y que **utiliza *KVM* como hipervisor**. Actualmente, la distribución *Cloud* de *Ubuntu* tiene dos vías de desarrollo y crecimiento. La primera de ellas es ***Ubuntu Orchestra*** que permite la creación de nubes privadas de una forma realmente sencilla, basándonos en el sistema operativo *Ubuntu 11.10*. La segunda vía de expansión es ***Ubuntu Juju*** actualmente en desarrollo y cuyo objetivo es crear un entorno de computación en la nube con un funcionamiento similar al *APT* (sistema de gestión de paquetes). La idea detrás de *Juju*

es que haya una comunidad de desarrolladores que paquetizen el despliegue de sus servicios mediante lo que llaman *charms* (hechizos).

Gracias a *UEC* puede desplegar infraestructuras de *Cloud Computing* privado o interno de una forma fácil y rápida.

Los distintos componentes que intervienen en su arquitectura son:

- ***Cloud Controller*** o controlador de la nube. Es el servidor principal y con el que el usuario interactúa para administrar la nube y todos sus componentes. Dispone de una interfaz compatible con la *API EC2* de *Amazon*, la interfaz web de *UEC* (accesible al realizar la instalación) y otras interfaces como *euca2ools* en línea de comandos (que debemos utilizar en ocasiones para crear nuevas máquinas virtuales o monitorizarlas) o *Hybridfox* vía Web (que veremos más adelante).
- ***Walrus Storage Controller*** *(WS3)* o controlador de almacenamiento *Walrus*. Compatible con protocolos de *Amazon*, permite almacenar las máquinas virtuales instanciadas además de otros datos. Normalmente se ejecuta en el mismo equipo que el *Cloud Controller*.
- ***Elastic Block Storage Controller*** *(EBS)* o controlador de almacenamiento en bloque elástico. Se instala en el mismo equipo que el *Cloud Controller* y se configura durante el proceso de instalación. Su funcionalidad principal es la de crear dispositivos de bloques susceptibles de ser montados por máquinas virtuales en ejecución, por ejemplo, para crear un sistema de ficheros. Entre otras muchas cosas permite también la creación de instantáneas de volúmenes que almacena en *WS3*.
- ***Clúster Controllers*** o controladores de Clúster. Como se aprecia en la *Figura 1.5*, operan entre el *Cloud Controller* y los *Node Controllers* o controladores de nodo. Se encargan de recibir las peticiones de parte del *Cloud Controller* para la asignación de las instancias de máquinas virtuales decidiendo, en función de su estado, en que *Node Controller* deben hacerlo. Además colabora con el *Cloud Controller* intercambiando información sobre los recursos que están siendo consumidos y controla el tráfico de las redes virtuales e instancias conectadas a ellas.
- ***Node Controllers*** o controladores de nodo. Se ejecutan en los equipos físicos que se encargan de alojar las instancias de las máquinas virtuales. Se encuentra en comunicación con el hipervisor y sistema operativo instalados en el nodo. Debe identificar los recursos utilizados por las máquinas virtuales así como los disponibles (memoria, espacio en disco, tipo de procesador y número de núcleos) en todo momento.

Puede verse en la *Figura 1.5* como los *Node Controllers* reciben órdenes del *Cloud Controller* para la manipulación de las instancias y responder a consultas de disponibilidad. Para instanciar una imagen primero verifica la autenticidad de la petición y después descarga la imagen del *WS3* (las cachea, por si debe instanciar varias veces una máquina virtual), crea la interfaz virtual de red y la inicia. Si la orden es detener dicha máquina virtual, se realizan las mismas acciones pero en orden inverso.

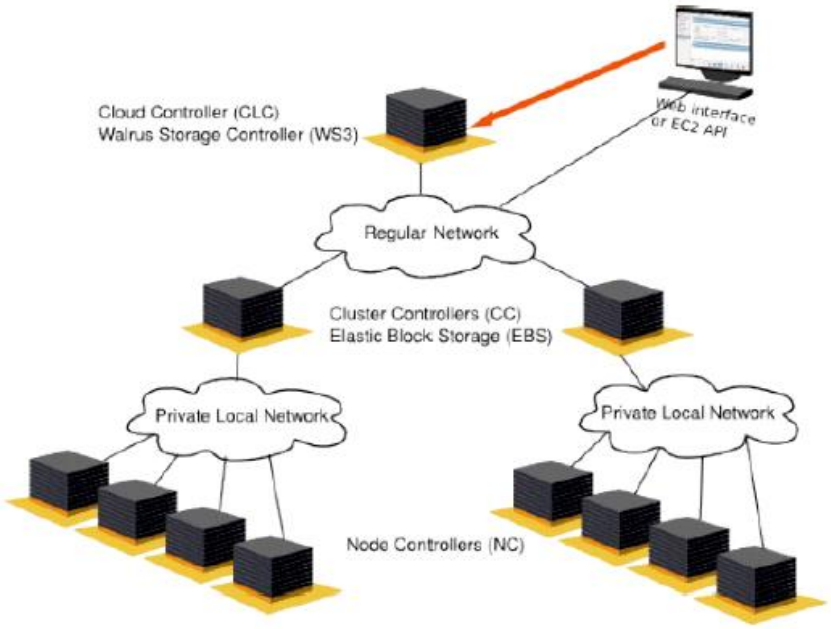


Figura 1.5. Diagrama *Eucalyptus*

En esta grafica la *'Regular Network'*, no tiene por qué significar Internet, sino una red que sea generalmente accesible por los usuarios del *Cloud*.

1.4.8. *Extility Flexiant*

Extility fue creado por el proveedor de la nube *Flexiant* para gestionar todos los centros de datos virtuales, y ofrece una solución llave en mano de nube dirigida a los proveedores de servicios, incluye una *API* completa, así como un panel de control basado en web con facturación integrada. Además ofrece un producto basado en una solución *SAN* llamado *Flexisan* que permite eliminar la necesidad de almacenamiento de la empresa.

1.4.9. *HP CloudSystem*

HP ***CloudSystem*** es la solución de ***HP*** para construir y gestionar servicios en la nube a través de entornos privados, públicos e híbridos que integra la gestión del sistema, los servidores, el almacenamiento, la red y la seguridad. La ventaja que supone este tipo de nube es que *HP* provee de todos los servicios al cliente desde el hardware hasta todo el entorno *Cloud Computing*, la configuración, el mantenimiento, etc. La desventaja principal son los costes.

Trabaja con los principales hipervisores, sistemas de almacenamiento, redes y servidores del mercado. Posee herramientas de monitorización y recuperación de desastres. Una de sus principales ventajas es que aseguran el soporte de su entorno y la posibilidad de adaptar su nube a posteriores necesidades, tanto hardware (añadiendo nuevos servidores), como software ya que su diseño modular permite incluir módulos con nuevas funcionalidades.

1.4.10. *IBM CloudBurst*

IBM CloudBurst es una oferta de nube privada que proporciona el hardware, software y los servicios necesarios, para establecer el entorno privado *Cloud Computing*. Integra el sistema de software de gestión de servicios con servidores, almacenamiento, y servicios *Quickstart* para habilitar una nube privada. Todo lo que necesita hacer es instalar sus aplicaciones y empezar a aprovechar los beneficios del *Cloud Computing*, como son la virtualización, flexibilidad, escalabilidad y el portal de auto servicio para proporcionar nuevos servicios.

CloudBurst de *IBM* proporciona una alternativa para la infraestructura tradicional de TI que busca mejorar la prestación de servicios y para transformar el centro de datos en una infraestructura dinámica. Está "Construido para rendir", basado en arquitectura y configuraciones necesarias para cargas de trabajo específicas. Permite al centro de datos acelerar la creación de servicios para una gran variedad de cargas de trabajo con un alto grado de flexibilidad, fiabilidad y optimización de recursos.

1.4.11. *Incontinuum CloudController*

Solución comercial de la compañía holandesa de software *InContinuum*, que automatiza y simplifica la gestión de un centro de datos virtual.

CloudController es una plataforma de gestión de aplicación basada en web que permite a las empresas corporativas, proveedores de *Virtual Data Center (VDC)* de servicios y clientes finales automatizar la configuración de pedidos, pagos de facturación, aprovisionamiento o devolución de cargo y el despliegue, la gestión, mantenimiento, soporte e información sobre todos los servicios de gestión prestados desde una plataforma de infraestructura de nube privada o pública.

1.4.12. *Nimbula Director*

Nimbula Director proporciona a empresas y proveedores de servicios una solución para construir de una forma sencilla y segura y rentable entornos *Cloud Computing* basados tanto en infraestructuras privadas, públicas o híbridas. Incluye el marco de administración Web, además de la *API*, y está disponible en una edición gratuita para implementaciones pequeñas.

1.4.13. *OnApp*

Es una solución dirigida especialmente a los proveedores de alojamiento. Permite la integración con soluciones de facturación existentes ampliamente utilizadas por los proveedores de servicios. Proporciona la *API* del usuario final y el panel de control del entorno. Su origen fue en el proveedor de servicios *VPS.NET*, pero hoy en día *OnApp* es una empresa independiente.

1.4.14. *OpenNebula*

Solución de código abierto que proporciona un conjunto de herramientas de gestión con la *API* completa y una interfaz Web sencilla, permitiendo a las organizaciones construir su plataforma en la nube para satisfacer sus propias necesidades.

Aunque se utiliza normalmente para crear nubes *IaaS* permite prestar prácticamente cualquier servicio; con *OpenNebula* puede implementar cualquier tipo de desarrollo *Cloud* (incluyendo nubes públicas e híbridas) y administrar su infraestructura virtual en un *data center* o clúster, o integrarla y combinarla con otras infraestructuras de nubes públicas –como *Amazon EC2*- o externas. Esto último permite un gran escalado de los entornos y servicios virtualizados. El desarrollo de nubes públicas es soportado gracias a la exposición de la funcionalidad de la nube con interfaces que permiten la administración tanto de las máquinas virtuales desplegadas en la infraestructura como del almacenamiento o conexiones de red.

La arquitectura presentada por *OpenNebula*, que puede ver en la *Figura 1.6*, está basada en la perspectiva clásica de un grupo de nodos que forma un clúster, actuando uno de ellos como equipo *frontend* (en el cual se encuentran instalados los drivers de virtualización) para la administración del resto, que ejecutan las instancias/imágenes de las máquinas virtuales creadas y desplegadas por el *frontend*. Lógicamente, es requisito indispensable que exista una conexión de red entre los nodos del clúster y el *frontend,* ya que las comunicaciones entre el *frontend* y el resto de nodos se realizan por *SSH*.

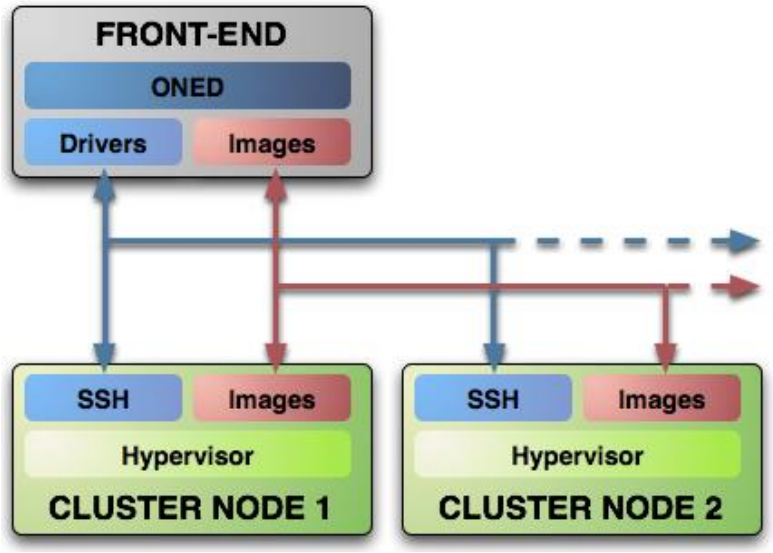


Figura 1.6. *OpenNebula*

Los componentes básicos de un sistema *Cloud Computing* con *OpenNebula* son los siguientes:

- ***Frontend*.** Ejecuta el servidor *OpenNebula* y los servicios del clúster.
- **Nodos.** Son equipos con un hipervisor instalado (*Xen*, *KVM*, o *VMWare*), encargados de proporcionar los recursos para la ejecución de las máquinas virtuales.
- **Repositorio de imágenes**. Se trata de un medio de almacenamiento, local en el equipo *frontend* o compartido a través de un sistema avanzado de almacenamiento (por ejemplo, una unidad *NAS*), para las imágenes o patrones base de las máquinas virtuales de la nube.

- **Servicio *OpenNebula***. Se trata del demonio principal que sirve el sistema. Administra todos los subsistemas del clúster (red, almacenamiento e hipervisores de los nodos) y las máquinas virtuales en su ciclo de vida.
- **Drivers**. Permite que el servidor *OpenNebula* interactúe con los subsistemas del clúster, por ejemplo un sistema de almacenamiento o un hipervisor.
- **Usuario *oneadmin*.** Es el usuario administrador de la nube privada, encargado de ejecutar cualquier operación relacionada con las máquinas virtuales, redes virtuales, nodos del clúster o usuarios.
- **Usuarios.** Utilizan *OpenNebula* para crear y administrar sus propias máquinas y redes virtuales.

Las interfaces de la infraestructura virtual creada con *OpenNebula* exponen tanto a usuarios como administradores funcionalidades de virtualización, *networking*, configuración de imágenes y recursos físicos, administración, monitorización y gestión de cuentas.

1.4.15. *OpenQRM*

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

1.4.16. *OpenStack*

Un proyecto liderado por la empresa ***Rackspace*** que ha decidido liberar el código de sus servicios *Cloud Files* y *Cloud Servers* bajo una licencia *Apache 2.0*. Es una innovadora solución *Cloud Computing* que permite la creación de infraestructuras virtuales de gran fiabilidad y escalabilidad en la red. Aunque es muy reciente su desarrollo, dispone de un gran soporte por parte de grandes compañías. La *NASA* también participa en el proyecto con "*Nova*", base de su plataforma "*Nebula*", ambos servicios desarrollados utilizando *Phython*. Otras empresas implicadas en el proyecto son *Citrix, Dell, AMD, Intel, Cloud.com, Puppet Labs* y *Zenoss*.

En la actualidad *OpenStack* engloba, además de otros muchos proyectos, tres proyectos fundamentales y que forman el corazón de *OpenStack*: ***OpenStack Compute, OpenStack Object Storage* y *OpenStack Image Service*.** El primero, soportado por el código *Nebula* que aporta la *NASA*, consiste en software dedicado al aprovisionamiento y administración de un gran número de servidores privados virtuales, mientras que el segundo, apoyado por la plataforma *Cloud Files* de *Rackspace*, consiste en software para la creación de almacenamiento redundante, escalable y de alta disponibilidad e integridad para terabytes o petabytes de datos utilizando clústeres de servidores. Finalmente, *Image Service* proporciona las funcionalidades de descubrimiento, registro y entrega de servicios para imágenes de disco virtuales.

OpenStack proporciona sus servicios disponibles a través de *APIs* compatibles con *Amazon EC2*/*S3* por lo que las herramientas cliente codificadas para *Amazon Web Services* pueden ser utilizadas de igual manera con *OpenStack*. Hay tres familias principales de servicios bajo *OpenStack*, cuya arquitectura simplificada puede apreciar en la *Figura 1.7.*

- ***Compute Infrastructure (Nova).*** Se trata de la capa *Cloud Computing* de la nube de *OpenStack*. *Nova* gestiona todas las actividades y procesos necesarios para dar soporte al ciclo de vida completo de las instancias (máquinas virtuales) en la nube *OpenStack*. Administra todos los recursos de computación, *networking*, autorización, así como las necesidades de escalabilidad de la nube. *Nova* es la plataforma administrativa; no proporciona capacidad de virtualización en sí, sin embargo, hace uso de *APIs libvirt* para interaccionar con los diferentes hipervisores que soporta (*Xen*, *XenServer*/*XCP*, *KVM*, *UML*, *VMware vSphere y Hyper-V*). *Nova* expone sus servicios a través de una *API* de servicios web compatible con *EC2* de *Amazon Web Services*.
  La infraestructura de computación *Nova* está formada por los siguientes componentes:
  - ***API Server: nova-api****.* Proporciona una interfaz de interacción para el mundo exterior con la nube. Es el único componente utilizado desde el exterior para administrar la infraestructura.
  - ***Message Queue: rabbit-mq server****.* El *Cloud Controller* de *OpenStack* se comunica con otros componentes como *Scheduler, Network Controller* o *Volume Controller* utilizando el protocolo *AMQP* (*Advanced Message Queue Protocol*).
  - ***Compute Workers: nova-compute****.* Los *compute workers* son los que lidian con el ciclo de vida de las instancias. Hay varios en una infraestructura *Cloud OpenStack*, reciben peticiones vía el componente *Message Queue* y realizan las operaciones.
  - ***Network Controller: nova-network****:* Se encarga de la configuración de red de los *hosts* anfitriones de las instancias. Entre sus operaciones destaca la asignación de direcciones de red, la configuración de *VLANS* para proyectos, la implementación de grupos de seguridad y la configuración de las redes para los nodos computacionales –*compute workers*.
  - ***Volume Worker: nova-volume****.* Los *volume workers* son utilizados para la administración de volúmenes basados en *LVM* para las instancias. Llevan a cabo operaciones relacionadas con volúmenes como la creación, eliminación, asignación de un volumen a una instancia, o eliminación de un volumen de una instancia.
  - ***Scheduler: nova-scheduler****.* Es el encargado de asignar las llamadas de *nova-api* a los diferentes componentes *OpenStack*. Ejecuta un demonio denominado *nova-schedule* para tomar servidores *compute/network/volumen* de una cola de recursos disponibles dependiendo del algoritmo de planificación en uso. Basa sus decisiones en diversos factores como la carga, la memoria, la distancia física de la zona de disponibilidad, arquitectura de la *CPU*, etc. En la actualidad implementa tres algoritmos básicos de planificación: *chance, availability zone,* y *simple*.

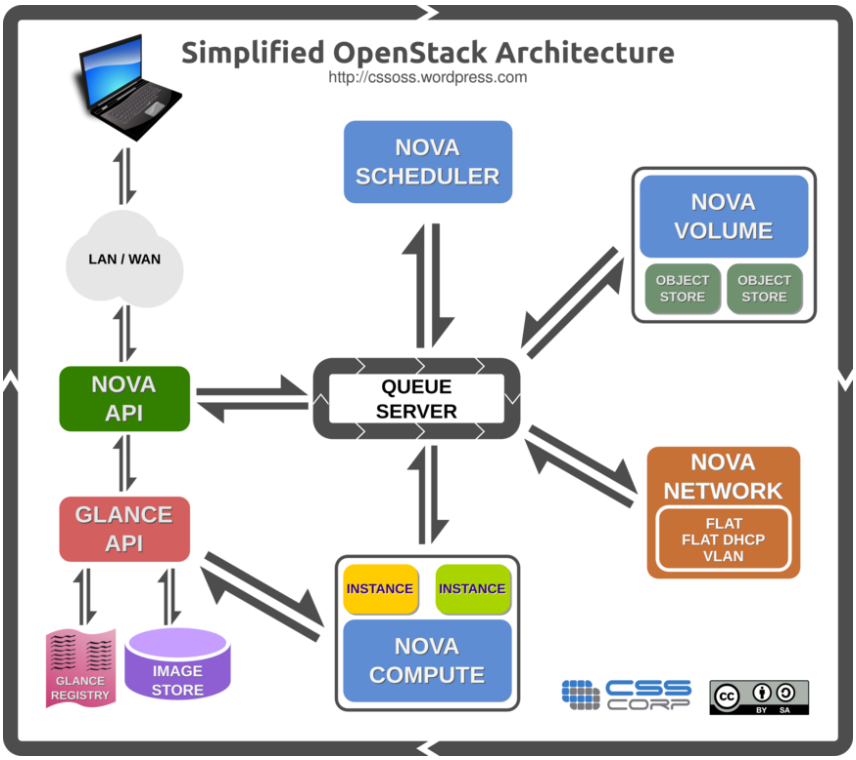


Figura 1.7. Arquitectura *OpenStack*

- ***Storage Infrastructure (Swift).*** Es un almacén de objetos que permite almacenar a través de hardware común un gran número de los mismos. Dispone de administración de redundancia y recuperación ante fallos integrada y características como el *backup* y archivado de datos, servicio de gráficos y videos. Es escalable a múltiples petabytes y a millones de objetos. ***Swift*** proporciona flexibilidad y elasticidad en el almacenamiento basado en cloud para aplicaciones Web. Almacena las imágenes de las instancias de la nube y puede trabajar como un contenedor de datos independiente.
- ***Imaging Service (Glance).*** Es un sistema para la consulta y recuperación de imágenes de máquinas virtuales. Puede ser configurado para utilizar cualquiera de los tres siguientes *backends* de almacenamiento: *OpenStack Object Store* para el almacenamiento de imágenes, almacenamiento *S3* de *AWS* directamente, o almacenamiento *S3* utilizando *OpenStack Object Store* como intermediario en el acceso de *S3*. Está formado por dos componentes: *glance-control* y *glance-registry*.

Teniendo en cuenta los componentes *OpenStack* que le permiten proporcionar los diferentes servicios citados (*Nova, Swift* y *Glance*) es posible desplegar distintas configuraciones de los mismos dependiendo del número de equipos físicos utilizados y por tanto de la cantidad de recursos hardware disponibles en la infraestructura. Así, es posible implementar las siguientes **tres arquitecturas básicas *OpenStack***:

- ***Single Node*:** (véase *Figura 1.8)* Todos los componentes de la solución *OpenStack* se instalan en un único nodo. Esto permite testear *OpenStack* incluso en una máquina virtual y utilizar el emulador *Qemu* en lugar de *KVM* como hipervisor. Es una buena elección para una primera aproximación a *OpenStack*. Tan solo es necesario un adaptador de red para poner en marcha el sistema.

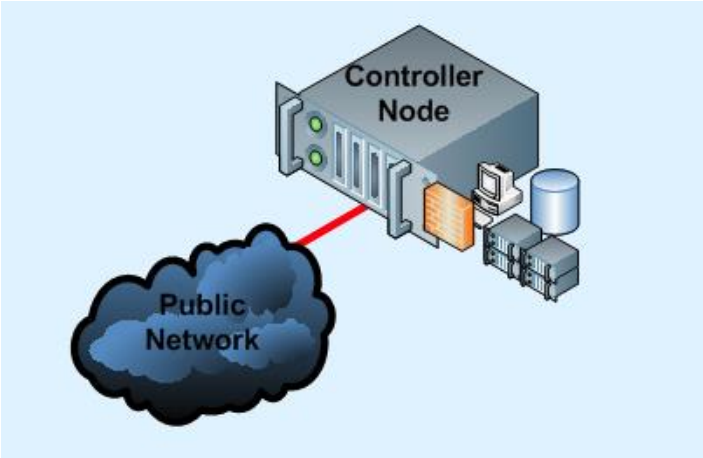


Figura 1.8. *OpenStack Single Node*

- ***Dual Node***. (véase la *Figura 1.9*)Es la mínima instalación recomendada para sistema en producción. Además de otros, todos los componentes *OpenStack* son ubicados en un *Controller Node*, pero el *Compute Node* se instala en un equipo diferente. Es posible ejecutar ambos nodos en un entorno virtualizado, aunque lo recomendable es que al menos el *Controller Node* corra sobre un servidor físico. Se pueden añadir tantos nodos de computación como queramos en cualquier momento.

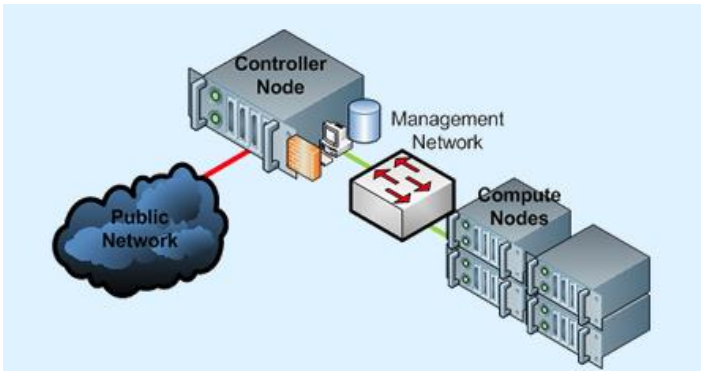


Figura 1.9. *OpenStack Dual Node*

- ***Multinode***. (véase la *Figura 1.9*) Se trata de la configuración estandarizada para producción. Los componentes *OpenStack* son distribuidos a lo largo de un conjunto de nodos, siendo recomendado disponer de al menos cuatro servidores físicos. Como ocurre con *Dual Node* es posible añadir cuantos nodos de computación queramos en un momento dado. Los nodos *Controller, Network, Compute* y *Volume* necesitan un servidor dedicado con requisitos muy específicos, por lo que debemos asegurarnos en primer lugar que nuestros equipos los cumplen completamente.

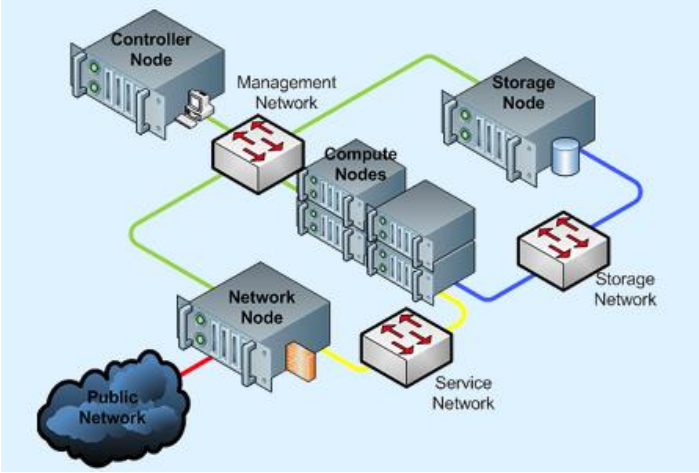


Figura 1.10. *OpenStack Multinode*

1.4.17. *Parallels Automation Cloud Infrastructure (CI)*

Parallels Automation for Cloud Infrastructure (PACI) es una completa solución para la entrega de *Cloud Infrastructure as a Service (IaaS). PACI* se despliega como módulo de servicio de *Parallels Automation* y permite a los proveedores de servicio ofrecer centros de datos virtuales con capacidades plenamente integradas de virtualización, automatización, facturación y autogestión de los clientes, así como una tienda online. Es modular y ampliable, por lo que puede añadir o eliminar servicios de acuerdo con sus necesidades particulares. Y es una solución de *Parallels*, el líder en materia de automatización y virtualización con un éxito probado en la entrega de servicios *Cloud* a más de 10 millones de empresas de todo el mundo.

1.4.18. *VMware vCloud*

El entorno *vCloud* de *VMware (VMware vCloud Inititiave)* pretende ayudar a las compañías a asegurar la calidad de servicio de cualquier aplicación que quieran ejecutar, internamente o como un servicio.

Esta iniciativa posibilita que las empresas y los proveedores de servicios ofrezcan un conjunto gradual de ofertas de servicios que van desde los servicios *Cloud* básicos a funciones más avanzadas gracias a los nuevos *Cloud vServices*. Dependiendo de la extensión de las funciones *Cloud* que exploten, los proveedores de servicios pueden ofrecer los siguientes servicios *Cloud VMware Ready*:

- Los servicios *Cloud* ***VMware Ready*** aprovechan los beneficios de *VMware Infrastructure* para ofrecer *Cloud* empresarial flexible, *hosting* y servicios gestionados. Los servicios *Cloud Ready* de *VMware* están disponibles a través de proveedores de servicios.
- Los servicios ***VMware Ready Optimized*** que utilizan la *Cloud vServices API*, la tecnología *vApp*, la especificación *OVF* y otras tecnologías de *VMware* para posibilitar una movilidad constante, el aprovisionamiento, la gestión y la seguridad de los servicios que se ejecutan en centros de datos instalados en el propio cliente o en *Clouds* externos.
- Los servicios *Cloud* ***VMware Ready Integrated*** hacen posible un enfoque común de la gestión integrada entre los entornos Cloud externos e internos como parte de *vCenter*, la línea de productos de gestión de *VMware*. Además, otras ofertas de servicios *Cloud* integrados ofrecen versiones optimizadas para *Cloud* de continuidad de negocio, creación de dispositivos virtuales y otras funciones clave. Los servicios *Cloud VMware Ready Integrated* están en desarrollo con los *partners*.

1.4.19. *Xen plataforma de nube (XCP)*

Xen Cloud Plataform *(XCP)* es una solución ***open-source*** derivada de ***Citrix XenServer 5.6.FP1*** totalmente funcional, de código abierto basada en el hipervisor *Xen*, con el objetivo de cubrir el aislamiento y las necesidades de seguridad de una implementación multi-tenant. ***XCP*** es una versión empaquetada de ***GNU/Linux CentOS*** *(kernel 2.6.32)* con ***Xen 3.4.2*** y ***XenAPI*** *(XAPI).*

Hay mutiltud de herramientas de administración, tanto del uso tradicional del hipervisor como *frontends* de gestión de *Clouds*, compatibles con XCP como son *Libvirt* y *virt-manager, xen-tools, Zentific, Convirt, OpenQRM, Xn-Suite, Xen Orchestra o Ganeti.*

1.4.20. *StackOps*

StackOps es una empresa española incubada por *Seedrocket,* que nace con el objetivo de **facilitar** el **acceso** a las tecnologías de ***Cloud Computing*** a empresas de cualquier tamaño. Permite desplegar una **nube *OpenStack*** de forma **sencilla** y **rápida**. Es una distribución *baremetal* basada en la distribución ***Ubuntu 10.04 Lucid Lynx*** y que permite la instalación y configuración o despliegue de cualquiera de las arquitecturas *OpenStack*, utilizando para guiarnos, sencillos e intuitivos asistentes para cada uno de los procesos involucrados. Gracias a la simplicidad de esta distribución, las empresas pueden abaratar en gran medida los costes de implantación de un entorno *Cloud Computing*.

Al tratarse de una distribución *OpenStack*, su uso está más que probado y garantizado y está diseñada para llegar al mayor número de usuarios posible gracias a su novedoso y sencillo proceso de instalación. Se puede decir que *StackOps* democratiza la tecnología *Cloud Computing* para empresas de todos los tamaños y sectores. Únicamente se necesita descargar la imagen (*ISO*) con la distribución e instalarlo y configurarlo en tantos servidores como se quiera.

Al estar basada en *OpenStack*, hereda las cualidades de esa plataforma, como son la fiabilidad y la escalabilidad en la red y además de trabajar con un entorno *Cloud Computing* que cuenta con un soporte muy amplio, como comentamos anteriormente en el *Apartado 1.4.16. OpenStack*, formado por importantes empresas.

1.5. Comparativa de las tecnologías *Cloud Computing*

Tras enumerar algunas de las distintas plataformas *Cloud Computing* que podemos encontrar, y antes de decidirnos por una de ellas para llevar a cabo nuestro proyecto, vamos a realizar una comparativa de las mismas, basándonos en ciertas características que son interesantes para el sistema que se va a poner en práctica.

Varias de las distribuciones estudiadas, pueden descartarse desde un primer momento para formar la base del proyecto porque están orientadas a sistemas de otro tipo. Son descartadas las nubes que ofrecen software o plataformas como servicio, dado que en nuestro caso pretendemos crear una infraestructura en un entorno *Cloud Computing*. Por tanto, descartamos plataformas *Cloud* como *BitNami*, *OnApp*, *Incontinuum CloudController* o *Extility Flexiant*. Además, para aumentar la criba, excluiremos de nuestra comparativa final nubes que no estén suficientemente asentadas, que se encuentren en desarrollo o que no cuenten con el soporte necesario (*Abiquo*, *Parallels Automation for Cloud Infrastructure* o *Nimbula Director*) además es preciso que éstas sean de libre distribución (*VMware vCloud*).

Finalmente, cuatro son las plataformas *Cloud Computing* que cumplen las características básicas para formar la base de nuestro proyecto. Son las siguientes:

- ***CloudStack.***
- ***Eucalyptus.***
- ***OpenNebula.***
- ***OpenStack.***

Para poder seleccionar una de ellas, vamos a enumerar una serie de características y, en función de estas, escogeremos la plataforma que más cumpla nuestros requisitos. Estas prerrogativas son:

- **Adaptabilidad**: a los distintos entornos hardware.
- **Escalabilidad**: posibilidad de variar el número de nodos de manera sencilla.
- **Topologías**: que permitan crear *Cluds* en los que los nodos se repartan las funciones de formas distintas.
- **Interfaz Web**: que aporte la funcionalidad necesaria para controlar la nube de una forma simple.
- **Instalación**: ha de ser lo mas sencilla posible.
- **Manejo de Instancias.**
- **Soporte.**

Vamos a describir cada una de las cuatro plataformas *Cloud Computing* centrándonos en los aspectos que consideramos fundamentales.

En primer lugar, la distribución *CloudStack* es una plataforma que cumple con gran parte de los requisitos que planteamos para nuestra elección, dado que posee una amplia comunidad de desarrollo que implica a algunas grandes empresas, como *Apache*. Esta comunidad ha desarrollado un entorno *Cloud Computing* adaptable a cualquier tipo de servidor, con una interfaz web sencilla y que permite gestionar la nube y manejar las instancias con facilidad. Por el contrario, esta plataforma está en cierto modo limitada en cuanto a las distintas topologías que se pueden configurar, dado que solo permite una arquitectura basada en PODS (nodos de computación) gestionados desde nodos controladores. En cuanto a la escalabilidad, permite añadir y eliminar PODS sin que el sistema se vea afectado.

Otra de las plataformas que podría ser elegida es *Eucalyptus*, ya que resulta ser una distribución con un amplio soporte (basada en *Ubuntu Enterprise Cloud*), sencilla de instalar y que se adapta a cualquier servidor. Además permite configurar distintos tipos de topologías en las que los nodos de computación pueden agruparse en grupos controlados por nodos de gestión. Esta nube permite configurar distintas interfaces web de control con lo que ofrece compatibilidad con controladores públicos como EC2 o S3 de *Amazon*. Sin embargo, estas interfaces web, no aportan una gestión de la nube tan sencilla y eficaz como las de otras plataformas.

La tercera de las nubes es *OpenNebula* que destaca por ser adaptable a cualquier entorno y por su escalabilidad, dado que una de sus características a reseñar resulta ser el hecho de que puede integrarse en otras nubes públicas como EC2 de *Amazon*. Aporta una interfaz web propia sencilla, pero a su vez no resulta muy eficaz en el manejo de instancias virtuales. No destaca por ser especialmente sencilla de instalar y solo permite un tipo de arquitectura clásica en la que un nodo *Frontend* controla el sistema y los nodos de cómputo realizan el trabajo.

Por último, la distribución *OpenStack* es una plataforma *Cloud Computing* que aunque es de reciente desarrollo, cuenta con apoyos de numerosas grandes empresas, lo que hace que este garantizado su buen funcionamiento. La interfaz web que controla el sistema aporta al usuario la capacidad de gestionar la nube de forma sencilla. Resulta muy destacable el manejo de instancias en esta nube ya que pueden crearse, modificarse y eliminarse instancias virtuales, creando patrones que sirven para crear nuevas máquinas virtuales. Esta plataforma permite crear nubes con distintas topologías que se adaptan a cualquier tipo de sistema, pudiendo añadir nuevos servidores al sistema o eliminando los que ya forman parte del mismo sin que la nube pierda integridad. Para terminar, hay que destacar uno de los aspectos fundamentales y es que la plataforma *OpenStack* cuenta con una distribución llamada *StackOps* que aporta una solución para configurar nubes *OpenStack* de forma fácil y rápida gracias a un asistente de instalación.

Teniendo en cuenta lo anteriormente comentado sobre cada una de las plataformas *Cloud Computing* que estamos comparando para decidirnos por la que va a formar parte del sistema, vamos a crear una tabla comparativa (ver *Tabla 1.3*) en la que se van a valorar las características que para nosotros son necesarias con los siguientes valores: bajo (B), medio (M), alto (A) o muy alto (MA). Así daremos un valor cuantitativo a la característica en cuestión para cada una de las nubes, valorando si es un aspecto destacable de la plataforma o no.

	CloudStack	Eucalyptus	OpenNebula	OpenStack
Adaptabilidad	MA	MA	MA	MA
Escalabilidad	MA	MA	MA	MA
Topologías	B	A	M	MA
Interfaz Web	A	M	A	MA
Instalación	M	A	M	MA
Manejo de Instancias	M	M	M	MA
Soporte	A	MA	M	MA

Tabla 1.3. Comparativa *Cloud Computing*

1.6. Elección de la Plataforma *Cloud Computing*

El objetivo principal en este capítulo ha sido dar a conocer la tecnología de computación en la nube, sus características, los tipos de servicios que ofrece y sus ventajas y desventajas. Una vez conocido el entorno *Cloud Computing*, realizamos una investigación para conocer las distintas opciones que podíamos desplegar para el desarrollo del proyecto.

Tras ello, pudimos comprobar que existen varias distribuciones muy extensamente implantadas como pueden ser *CloudStack*, *OpenNebula, Eucalyptus u OpenStack*. Tras realizar un listado de características principales que precisamos para nuestro sistema, hemos realizado una comparativa con estas cuatro plataformas.

En la *Tabla 1.3* hemos recogido la comparativa de estas distribuciones y en ella podemos ver que la plataforma con mayor calificación es ***OpenStack*** dado que es una de las que **cuenta con un mayor soporte** y desarrollo. Son **muchas** las **empresas implicadas en su crecimiento** y además ofrece **varias topologías de desarrollo** que la hacen muy flexible y adaptable a cualquier tipo de entorno.

Pero no es *OpenStack* por sí sola la solución *Cloud Computing* que nos parece más adecuada, sino que es la **distribución baremetal** basada en ella y llamada ***StackOps*** la que decididamente vamos a elegir como entorno *Cloud Computing*.

OpenStack es una distribución *Cloud Computing* **altamente adaptable**, capaz de desplegarse en **cualquier tipo de entorno o infraestructura**. A su vez posee un **muy alto nivel de escalabilidad**, pudiendo incluirse nuevos equipos a los clústeres que se despliegen en cualquier momento.

Llegamos a la conclusión de que **la mejor elección es la distribución *StackOps*** que permite desplegar y **configurar** de manera sencilla **una nube *OpenStack*** manteniendo todas sus ventajas, pero haciendo que el proceso de instalación sea algo mucho menos costoso.

En posteriores capítulos profundizaremos más en estas soluciones *Cloud Computing*, dando a conocer más a fondo sus características, sus distintas topologías de nube y sobre todo aprendiendo a desplegar y configurar el entorno.

2. Puesta en Marcha del *Cloud*

En el capítulo anterior se introdujo el paradigma de la computación ***Cloud Computing*** y se expusieron las distintas opciones que ofrece el mercado actualmente. Con todo ello, se eligió desplegar una nube ***OpenStack*** como base del proyecto y, en concreto se va a utilizar una distribución ***StackOps*** basada en ***Ubuntu*** que configura una nube *OpenStack* pero de una forma más sencilla, manteniendo todas sus ventajas.

A continuación, vamos a desplegar nuestra nube y a configurarla, teniendo en cuenta los recursos hardware de los que se disponen. Tras presentar esos recursos, comenzaremos con la instalación del **sistema operativo base** del entorno para cada servidor. Teniendo todos los servidores preparados, puede configurarse la nube, pero previamente se ha de elegir una **arquitectura** de nube para hacer que los servidores trabajen conjuntamente de manera organizada. Tras la configuración, pasaremos a mostrar la **interfaz Web** del entorno *Cloud Computing* y a desplegar nuestras primeras **instancias**.

2.1. Introducción

Previo a la descripción de los pasos de la instalación de la infraestructura de nube *StackOps*, vamos a hacer una breve presentación de los recursos hardware con los que contamos para ello.

Vamos a trabajar con seis servidores de la marca *Dell* modelos R210 (cinco servidores) y R200 (un servidor). Ambos modelos cuentan con cuatro núcleos de cómputo, cuatro gigabytes de memoria RAM y doscientos cincuenta gigabytes de disco duro. En la *Tabla 2.1* pueden apreciarse las características técnicas de estos servidores.

Modelo	CPU	Memoria	HDD	Uds
***Dell* R210** 	4x*Intel Xeon* X340 2,40 GHz	4GB	250GB	5
***Dell* R200** 	4x*Intel Xeon* X310 3,13 GHz	4GB	250GB	1

Tabla 2.1. Recursos Hardware

Sobre este conjunto de servidores vamos a formar un clúster para que trabajen de manera conjunta conformando lo que llamamos "nube". Así, de cara al usuario, el sistema aparece como un solo dispositivo al que se accede y sobre el que se realizan las operaciones de procesamiento. El conjunto cuenta con un servidor que actúa como **controlador del sistema** y a la vez como nodo de procesamiento. El resto de los servidores serán "esclavos" que trabajen bajo las órdenes del controlador, es decir, como **nodos de cómputo**.

Tanto la infraestructura *Cloud Computing OpenStack* como el entorno *StackOps*, basado en el anterior, precisan de un sistema operativo base sobre el que realizar una serie de pasos para completar la configuración. Por tanto, el primer paso es instalar sobre los servidores el sistema operativo de base. Para el caso de *OpenStack,* puede configurarse la nube sobre varias distribuciones *Linux*.

En condiciones normales, una vez preparados los servidores con el sistema operativo base, para desplegar una nube *OpenStack,* es necesario configurar los componentes de este entorno uno a uno. Lo que aporta la solución *StackOps* es que, siguiendo un instalador, podremos configurar el entono C*loud Computing* de una forma realmente sencilla. Además, esta solución instala y configura el sistema operativo base. Esta infraestructura tiene como base la distribución *Linux Ubuntu Server 10.04.*

Una vez que conocemos de forma general qué aporta la solución *StackOps* que la diferencia del resto, comenzamos a preparar el entorno, realizando en cada uno de nuestros servidores los pasos necesarios para la configuración definitiva y que a continuación son descritos.

2.2. Instalación del Sistema Operativo Base

Para comenzar la instalación, hemos de descargar el entorno *StackOps* de la Web oficial de la distribución: http://www.stackops.com (véase la *Figura 2.1*).

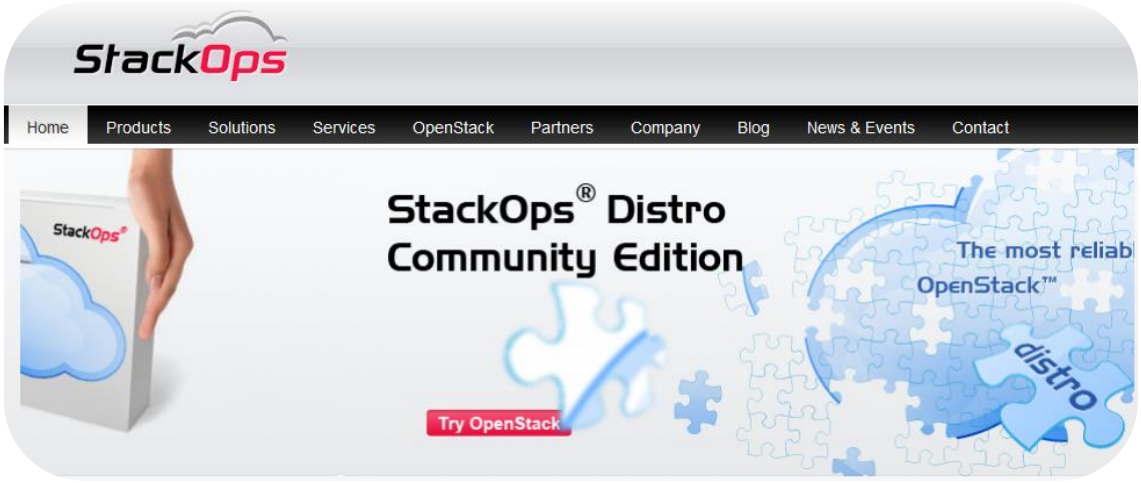


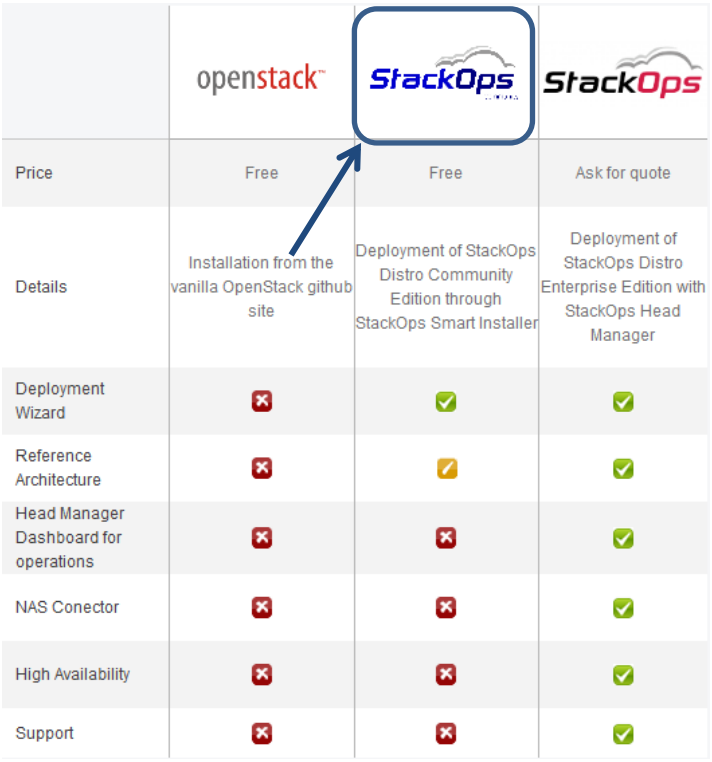
Figura 2.1. Web *StackOps*

En el apartado "*Products*" puede encontrar las distintas versiones del sistema que se ofrece (véase la *Figura 2.2*).

	openstack™	StackOps	StackOps
Price	Free	Free	Ask for quote
Details	Installation from the vanilla OpenStack github site	Deployment of StackOps Distro Community Edition through StackOps Smart Installer	Deployment of StackOps Distro Enterprise Edition with StackOps Head Manager
Deployment Wizard	✗	✓	✓
Reference Architecture	✗	✎	✓
Head Manager Dashboard for operations	✗	✗	✓
NAS Conector	✗	✗	✓
High Availability	✗	✗	✓
Support	✗	✗	✓

Figura 2.2. Versiones *StackOps*

En la *Figura 2.2* aparece una comparativa de las tres versiones que se ofrecen en la Web. Son la versión *OpenStack,* que implica la configuración del sistema sin ningún tipo de asistente, y dos versiones propiedad de la empresa *StackOps*. Una de ellas es gratuita y la otra no. La versión de licencia libre que cumple de forma más que suficiente con los requisitos que precisamos para nuestro entorno *Cloud Computing*, comentados en el apartado anterior. En nuestro caso hemos seleccionado "*StackOps Distro Community Edition"* (véase la *Figura 2.3*).



	openstack	StackOps	StackOps
Price	Free	Free	Ask for quote
Details	Installation from the vanilla OpenStack github site	Deployment of StackOps Distro Community Edition through StackOps Smart Installer	Deployment of StackOps Distro Enterprise Edition with StackOps Head Manager
Deployment Wizard	✗	✓	✓
Reference Architecture	✗	✓	✓
Head Manager Dashboard for operations	✗	✗	✓
NAS Conector	✗	✗	✓
High Availability	✗	✗	✓
Support	✗	✗	✓

Figura 2.3. Elección *StackOps*

A continuación, en la *Figura 2.4*, se muestra una breve una descripción y el link para poder descargar la distribución.

Figura 2.4. Distribución *StackOps*

A continuación (*Figura 2.5*) se despliega otra página en la que aparece la misma tabla que vemos en la *Figura 2.2*, pero esta vez con el enlace para la descarga.

Figura 2.5. Enlace Descarga *StackOps*

Al pulsar sobre el enlace "*Download now*" de la versión *StackOps Community* se abre la Web de la comunidad *StackOps*. Esta comunidad es la encargada de dar soporte a la distribución. Se trata de un soporte privado de la empresa (como en el caso de la distribución superior), sin embargo se encarga de que la distribución se mantenga estable ante cualquier cambio y que además vaya siendo actualizada periódicamente. Podemos ver en la *Figura 2.6* el enlace de descarga final.

Figura 2.6. Descarga *StackOps*

Llegados a este punto, puede acceder al *link* de descarga directa del sistema. Una vez tengamos el archivo de imagen que contiene el sistema, lo grabamos en algún dispositivo de almacenamiento y comenzamos la instalación en los servidores.

De esta forma, arrancamos el instalador desde el medio de almacenamiento en el servidor y comenzamos la instalación del sistema operativo (véase la *Figura 2.7*).

Figura 2.7. Inicio Instalación *StackOps*

Este primer paso es similar al primer paso de cualquier instalación de una distribución *Ubuntu*. Para continuar con la instalación, ejecute el paso señalado en la *Figura 2.7* "*Install Stackops node (US Keyboard)*", o bien el siguiente en el que se nos da la opción de seleccionar el idioma del teclado, como se prefiera. Si hemos decidido seleccionar idioma, aparecen un par de pasos para ello. Una vez seleccionados los idiomas de teclado y de instalación, comienza a cargar los componentes de la instalación. Durante toda la instalación vamos a mostrar con figuras, el procedimiento que se sigue.

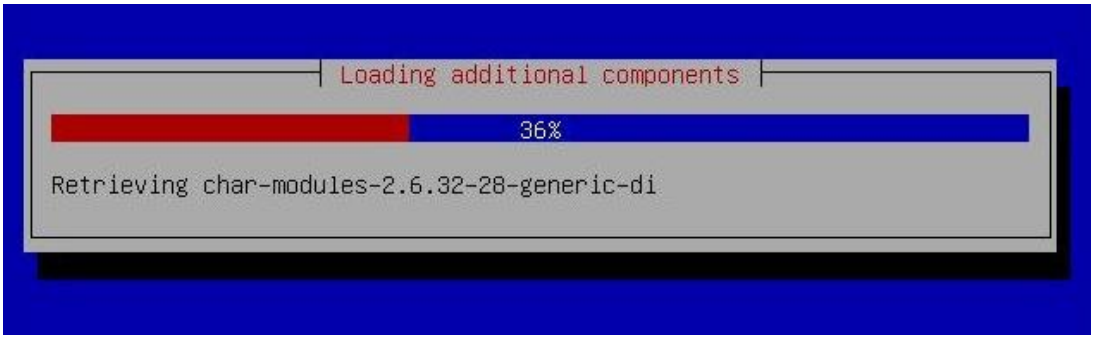


Figura 2.8. Cargando Componentes Instalación

Una vez cargados los componentes precisos, los siguientes pasos hacen referencia la configuración de la red de la nube. En el primero de ellos, debe indicar la dirección IP del servidor que estamos configurando (véase la *Figura 2.9*).

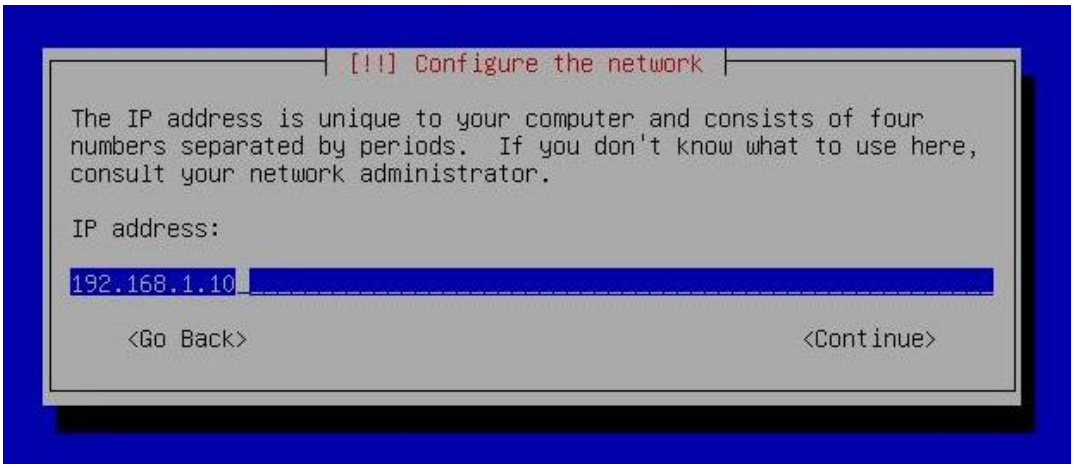


Figura 2.9. Configuración IP

A continuación hemos de indicar la dirección IP del *Gateway* de la red (véase la *Figura 2.10*).

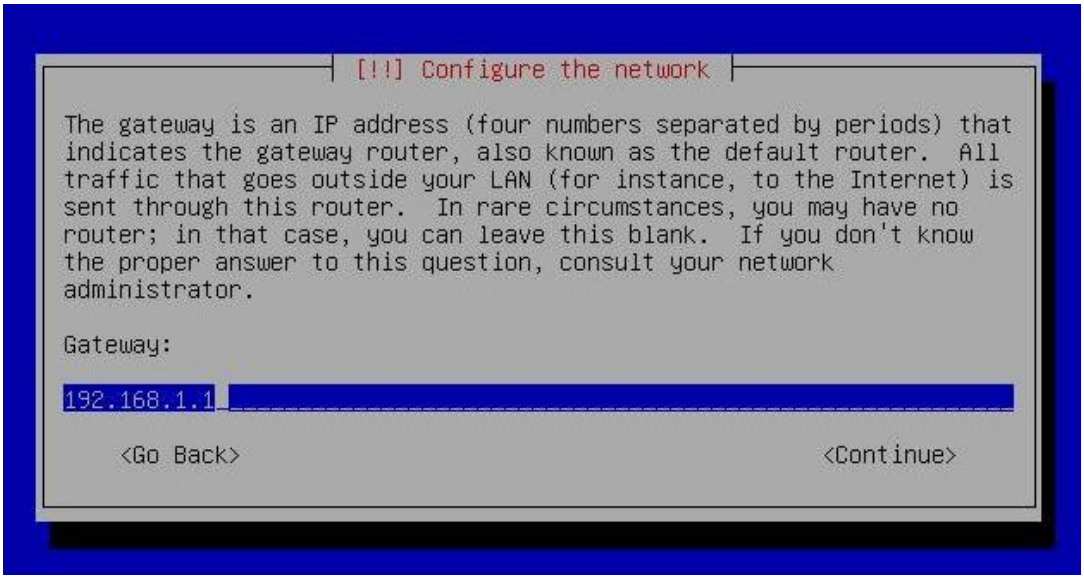


Figura 2.10. Configuración *Gateway*

Una vez indicados los datos que se nos piden para conformar la red, pasamos a la configuración de las particiones del disco. Vamos a seguir el método por defecto que utiliza todo el disco y configura LVM (*Logical Volume Manager)*. Por tanto, el primer paso es verificar esta opción (véase la *Figura 2.11*).

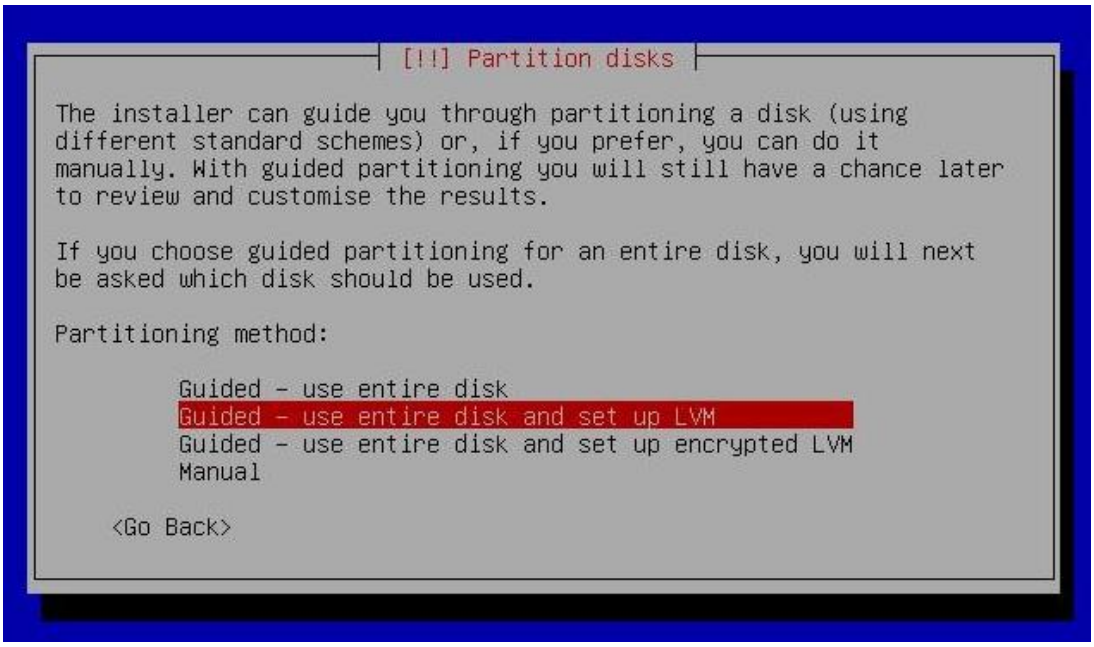


Figura 2.11. Método de Particionado de Discos

En posteriores apartados de la instalación, simplemente hemos de ir configurando distintas características. Estas son: seleccionar el disco duro en el que se va a alojar la partición, verificar que queremos escribir los cambios en el disco y configurar LVM.

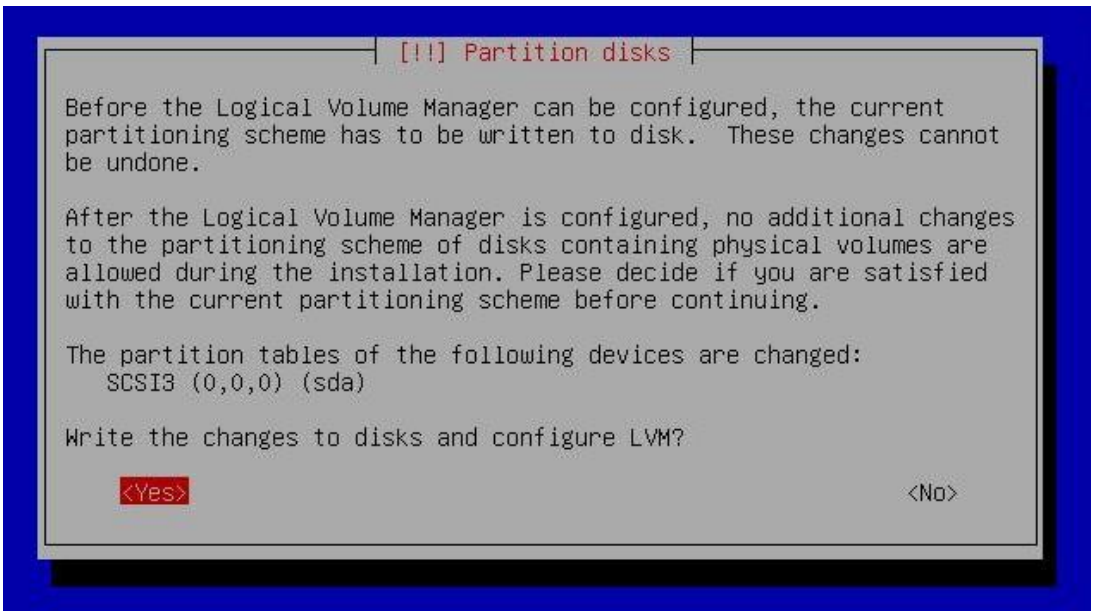


Figura 2.12. Elección LVM

Debemos confirmar que queremos utilizar la totalidad del disco y para finalizar nos presenta un resumen de todo lo anteriormente seleccionado para comprobar que son correctas todas las configuraciones realizadas.

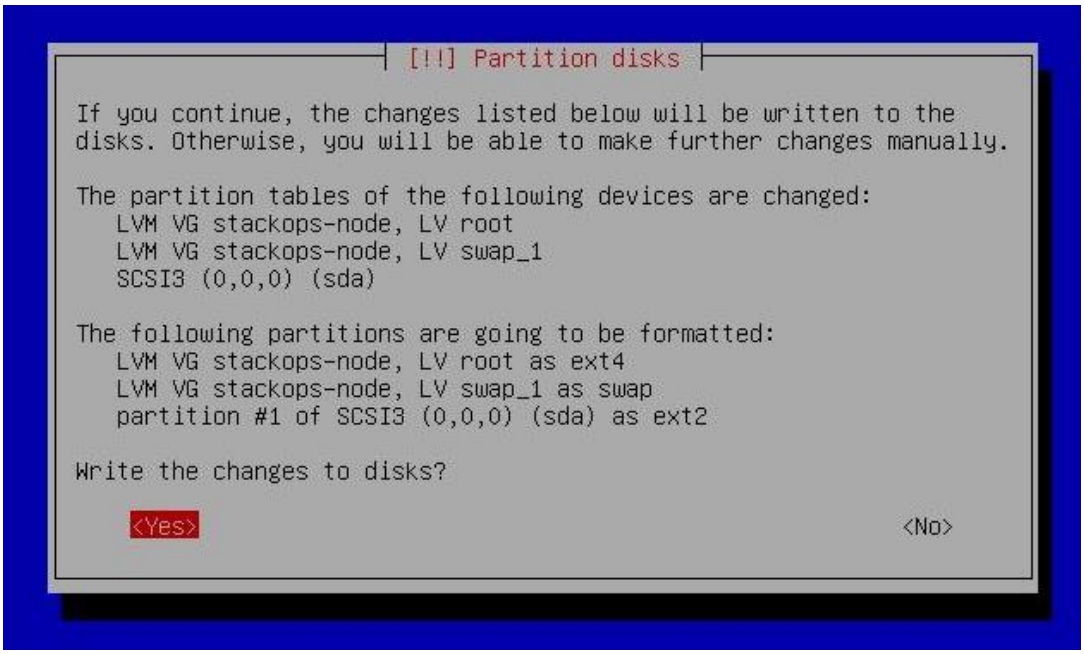


Figura 2.13. Aplicar Cambios a los Discos

Confirmando este paso, comienza a aplicar los cambios en el disco.

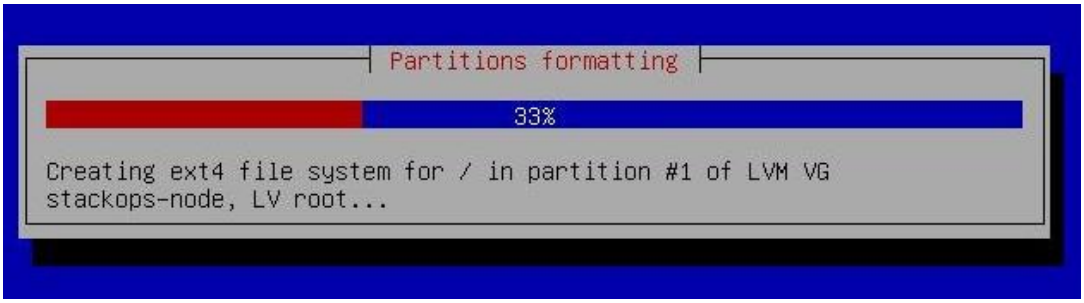


Figura 2.14. Formateando Particiones

A partir de aquí, continúa avanzando el proceso de instalación del sistema operativo base, la instalación de paquetes del sistema, etc.

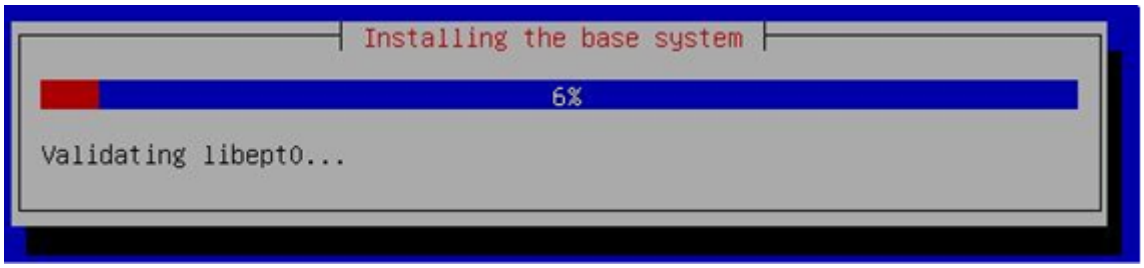


Figura 2.15. Instalando S.O. Base

Comienza instalando la base del sistema.

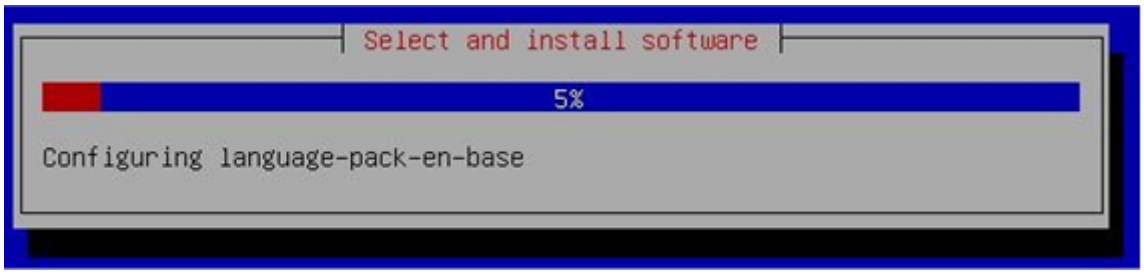


Figura 2.16. Instalando Componentes

Selecciona e instala el software en función de lo previamente configurado.

Figura 2.17. Finalizando Instalación

Una vez terminado el proceso de instalación, es necesario reiniciar el servidor para poder finalizar la instalación del sistema operativo base. En la *Figura 2.17* confirmamos el reinicio.

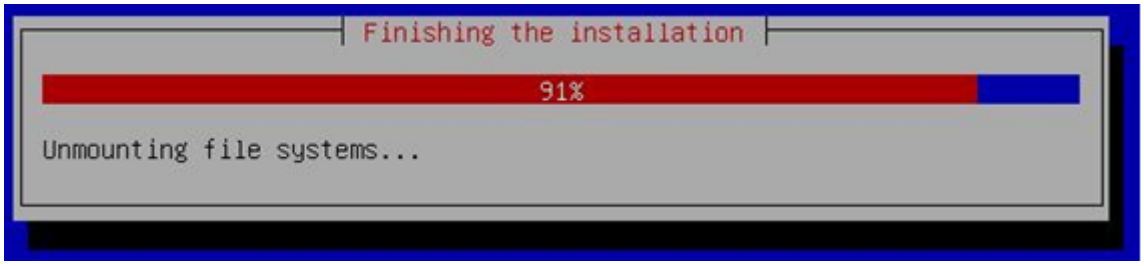


Figura 2.18. Instalación Completa

Realiza las últimas operaciones y reinicia el sistema.

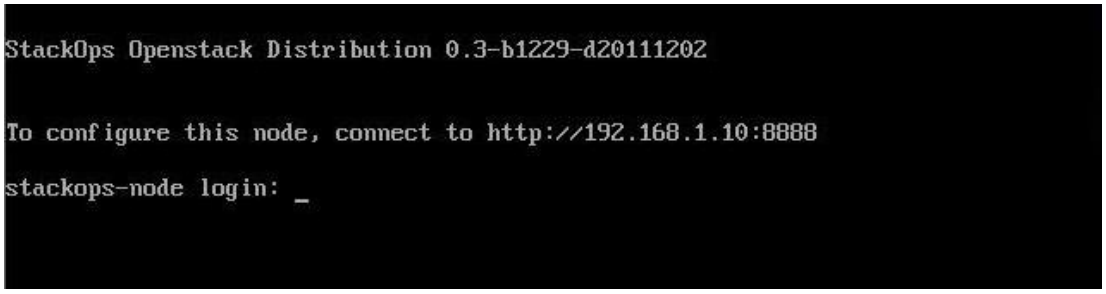


Figura 2.19. Inicio del Sistema *StackOps*

Es necesario realizar esta misma instalación en todos los servidores que forman parte de la nube. La única diferencia entre las distintas instalaciones radica en la dirección IP que asignamos a cada uno de ellos. En este proyecto, simplemente se ha ido incrementando en uno la dirección de cada uno de los nodos.

Por tanto, tras la instalación del sistema operativo base en todos los servidores, disponemos de una nube cuya estructura preliminar es:

- Los nodos que componen la nube se encuentran dentro de una misma red privada con direcciones IP desde la 192.168.7.10 (Nodo Controlador) hasta la 192.168.7.15 (dirección de red del último de los Nodos de Cómputo).
- Cuenta con cuatro procesadores, cuya potencia de procesamiento será sumada tras conformar la nube, por lo que es posible contar con un sistema conjunto de veinticuatro núcleos.
- De la misma forma ocurre con los recursos de memoria RAM y almacenamiento en disco duro.

2.3. Arquitecturas de Nube *StackOps*

Una vez que disponemos de todos los servidores preparados para formar el clúster, es hora de pasar a configurar *StackOps* y hacer que todos ellos trabajen como una Nube de servidores. Lo primero es decidir, de entre las distintas configuraciones de Nube que propone *StackOps*, elegir la que más se ajuste a nuestros requerimientos y las limitaciones que puedan tener nuestros servidores. Las distintas configuraciones que propone *StackOps* son las siguientes:

- *Single Node*
- *Dual Node*
- *Multinode*

Para poder tomar esta decisión de la mejor forma posible es necesario conocer los elementos básicos que conforman una nube *StackOps*. Son cuatro:

- **Controlador**: nodo desde el que se accede, vía Web, al sistema *Cloud* y que ejerce las principales tareas de gestión de la nube.
- **Gestor de la Red**: nodo que controla todas las transferencias que existen entre la red interna de la nube y el exterior.
- **Gestor de Almacenamiento**: nodo encargado de gestionar el almacenamiento de todos los datos que se alojan en el sistema. Controla dónde se almacena y gestiona la seguridad en los datos (redundancias).
- **Cómputo**: nodo o nodos que ejecutan las tareas de cómputo de la nube y en los que se despliegan las instancias virtuales.

Estos elementos han de formar parte de cualquiera de las estructuras anteriormente citadas, siendo indiferente el número de servidores que conformen la nube.

Conociendo en qué consisten los elementos principales del sistema Cloud, pasamos a describir los tres tipos de configuraciones de nube que ofrece *StackOps*.

2.3.1. *Single Node*

Es el modo más sencillo, ya que únicamente consta de **un servidor**. Todas las tareas de gestión de envío de datos a través de la red se simplifican sobremanera. El servidor único se configura como Controlador y se incluyen en él todas las tareas de gestión de almacenamiento y de red. Por supuesto, al solo disponer de un servidor, dentro de éste se llevan a cabo todas las tareas de cómputo.

Por tanto, todos los elementos que conforman una nube *StackOps* anteriormente comentados, están incluidos en el servidor único.

El funcionamiento de una nube *Single Node* es análogo al de una nube multinodo, sólo que el rendimiento obviamente es menor ya que todas las máquinas virtuales y datos se encuentran en el nodo único y todas las tareas son ejecutadas sobre él.

El diagrama correspondiente a la arquitectura es el siguiente (véase la *Figura 2.20*).

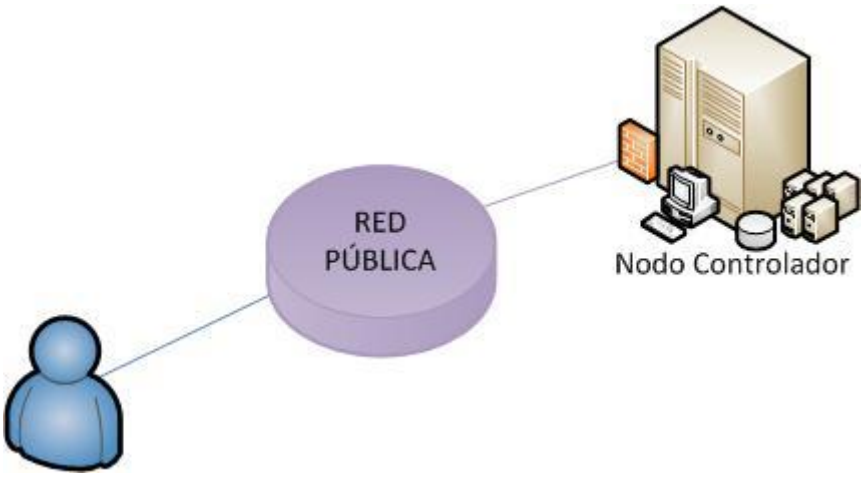


Figura 2.20. Arquitectura *StackOps Single Node*

Vemos como el usuario accede a la nube mediante una conexión a Internet, mientras que en el otro lado de la conexión queda expuesta la estructura interna de la nube de nuestro servidor. Todo esto es transparente al usuario. Puede ver representados junto al servidor todos los elementos de la nube representados con pequeñas figuras, controlador (), cómputo (), red () y almacenamiento ().

Los requisitos mínimos para implementar esta arquitectura *Cloud* con un solo nodo se muestran en la *Tabla 2.2*.

Servidor	Configuración Mínima	Configuración Recomendada
Nodo Controlador, de Red, de Almacenamiento y de cómputo.	64 bits x86 2GB RAM 1 HDD SATA 30GB 1 NIC 1Gb	2 x 64 bits x86 32GB RAM 2 HDD SAS/SATA/SSD 32GB Raid 1 2 HDD SATA 2TB Raid 1 2 NIC 1Gb

Tabla 2.2. Requisitos *Single Node*

2.3.2. *Dual Node*

En este tipo de estructura de nube existen dos tipos de nodos, el **Nodo Controlador** y el (los) **Nodo(s) de Cómputo**. En el Nodo Controlador se configuran las tareas de controlador de la nube además de las de gestión de la red y el almacenamiento. Los nodos de cómputo son destinados únicamente al alojamiento de las máquinas virtuales y los datos. Igualmente, puede configurar el Nodo Controlador como un Nodo de Cómputo más.

Esta configuración de sistema *Cloud* (véase la *Figura 2.21*) ofrece un rendimiento muy superior al anteriormente descrito ya que cuenta con más recursos, pasaríamos de un solo servidor a, como mínimo, dos (un Nodo Controlador y un Nodo de Cómputo). La ventaja de esta configuración respecto a la posterior (*Multinode*), radica en que los requisitos para los recursos de los servidores son más ajustados.

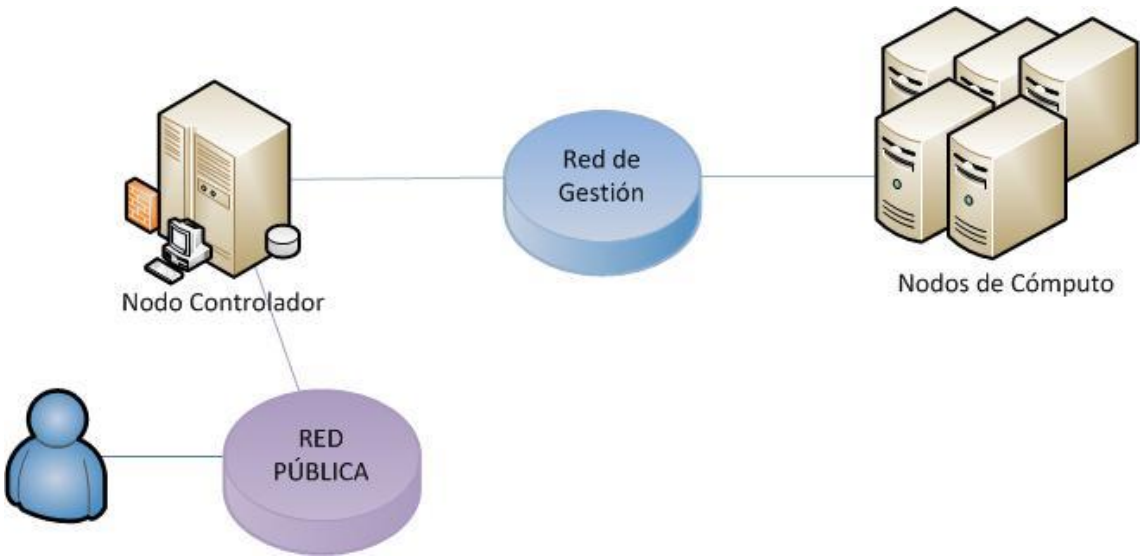


Figura 2.21. Arquitectura *StackOps Dual Node*

Puede ver como el usuario de la nube accede a ella de la misma manera que en el caso de nodo único. Sin embargo, tiene acceso solamente al Nodo Controlador, que es donde vemos que están configuradas las tareas de gestión de la nube. En el Controlador vemos

representadas las tareas de control de la red y del almacenamiento. Al otro lado, formando el propio clúster, aparecen los Nodos de Computo conectados al Controlador por la red interna de gestión de la nube.

Los requisitos mínimos y recomendados para esta configuración se muestran en la *Tabla 2.3. Requisitos Dual Node.*

Servidor	Configuración Mínima	Configuración Recomendada
Nodo Controlador, de Red y de Almacenamiento.	64 bits x86 1GB RAM 1 HDD SATA 30GB 2 NIC 1Gb	1 x 64 bits x86 4GB RAM 2 HDD SSD 32GB Raid 1 2 HDD SATA 2TB Raid 1 3 NIC 1Gb ó 3 NIC 10GboE
Nodo de Cómputo	64 bits x86 2GB RAM 1 HDD SATA 30GB 2 NIC 1Gb	2 x 64 bit x86 - Intel VT ó AMD-V 32GB RAM 2 HDD SAS/SATA 2TB Raid 1 2 HDD SAS/SATA/SSD 32GB Raid 1 2 NIC 1Gb ó 2 NIC 10GboE

Tabla 2.3. Requisitos *Dual Node*

2.3.3. *Multinode*

Por último, vamos a presentar en este apartado la más compleja de las configuraciones estándar que plantea *StackOps* para desplegar nuestra nube. Esta configuración, es la que más recursos precisa y la más restrictiva en cuanto a los requisitos hardware de los servidores.

Son necesarios, como mínimo, **cuatro servidores**. Tres de estos servidores ejercen como controladores de la nube, del almacenamiento y de la red. Por otro lado, es necesario un servidor más para cómputo, pudiendo incrementar el número de Nodos de Cómputo tanto como queramos.

Esta configuración es bastante más compleja de gestionar que las anteriores (véase la *Figura 2.22*), ya que se triplican las redes internas. En lugar de tener una red de gestión de la distribución *Dual Node*, existen tres topologías de red distintas. La **"Red de Gestión"** conecta el Nodo Controlador con todos los elementos de la nube y por ella viajan todas las instrucciones y tareas que envía el controlador para gestionar el funcionamiento de la nube. Por otro lado, la **"Red de Almacenamiento"** que conecta el Nodo Almacenamiento con los Nodos de Cómputo y que dirige dónde se almacenan los datos y cómo se distribuyen las distintas replicas de los mismos, controlando a su vez la seguridad de la información. Para finalizar, la **"Red de Servicios",** que conecta el Nodo de Red con los Nodos de Cómputo y que surte de conexión con el exterior a los servicios de la nube.

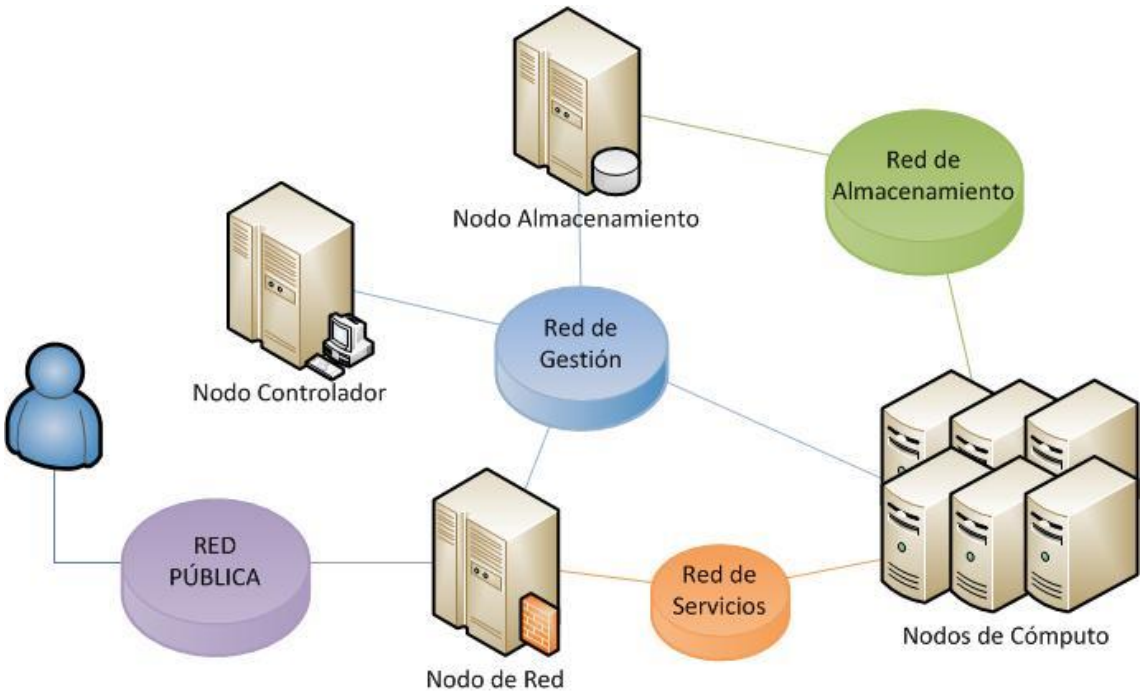


Figura 2.22. Arquitectura *StackOps Multinode*

Puede comprobar como todo lo anteriormente comentado está reflejado en el sistema. Disponemos de un **Nodo de Red** configurado para gestionar todos los servicios que corren en los Nodos de Cómputo, un **Nodo Controlador** conectado con todos los nodos que gestiona el sistema, un **Nodo de Almacenamiento** que se encarga de la gestión de los datos y uno o varios **Nodos de Cómputo**. También puede diferenciar las tres redes internas mencionadas.

Otro aspecto interesante a tener en cuenta es que todos los nodos pueden ejercer además de Nodo de Cómputo, con lo que es posible aumentar el rendimiento, siempre y cuando los servidores de los que disponemos sean suficientemente potentes.

La gestión de un sistema tan complejo provoca que aumenten los requisitos de los recursos hardware. Algo que salta a primera vista es el aumento en la gestión de las redes, que provoca que los servidores necesiten más NICs (*Network Interface Controller*). Vemos a continuación en la *Tabla 2.4* los requisitos mínimos y los recomendados para esta topología:

Servidor	Configuración Mínima	Configuración Recomendada
Nodo Controlador	64 bits x86 1GB RAM 1 HDD SATA 10GB 1 NIC 1Gb	1 x 64 bits x86 4GB RAM 2 HDD SAS/SATA/SSD 200GB Raid 1 o superior 1 NIC 1Gb ó 1 NIC 10GboE
Nodo de Red	64 bits x86 1GB RAM 1 HDD SATA 10GB 2 NIC 1Gb	1 x 64 bits x86 4GB RAM 2 HDD SAS/SATA/SSD 10GB Raid 1 o superior 3 NIC 1Gb ó 3 NIC 10GboE
Nodo de Almacenamiento	64 bits x86 1GB RAM 1 HDD SATA 30GB 1 NIC 1Gb	1 x 64 bit x86 4GB RAM 2 HDD SAS/SATA/SSD 200GB Raid 1 2 HDD SAS/SATA 2TB Raid 2 NIC 1Gb ó 1 NIC 10GboE
Nodo de Cómputo	64 bits x86 2GB RAM 1 HDD SATA 30GB 2 NIC 1Gb	2 x 64 bit x86 - Intel VT o AMD-V 32GB RAM 2 HDD SAS/SATA 2TB Raid 1 2 HDD SAS/SATA/SSD 32GB Raid 1 3 NIC 1Gb ó 2 NIC 10GboE 2 NIC 1Gb ó 2 NIC 10GboE

Tabla 2.4. Requisitos *Multinode*

La distribución *StackOps* propone las tres arquitecturas para el sistema *Cloud* que hemos visto comentadas en los apartados anteriores. De esas tres, hemos decidido decantarnos por la segunda de ellas, ***Dual Node***. La topología *Single Node* queda descartada desde el primer momento, ya que contamos con cinco servidores y sería desperdiciar recursos montar una nube de un solo servidor. Sin embargo, si podía haber cierta duda entre *Dual Node* o *Multinode*. Elegimos la primera de ellas, por la complejidad de gestión de la segunda. El rendimiento que precisamos para nuestro proyecto queda de sobra cubierto por la arquitectura *Dual Node*. Por ello y por su sencillez de configuración y gestión, nos decidimos por esta topología.

2.4. Configuración de la Nube

Como hemos visto en el apartado anterior, se ha decidido tomar como arquitectura para la nube el modelo "*Dual Node*" en el que vamos a configurar un servidor como Nodo Controlador y el resto de ellos como Nodos de Cómputo. El controlador se configura también como Nodo de Cómputo.

Así las cosas, procedemos a realizar los sencillos pasos a seguir para la configuración final de la nube. Es aquí donde se da la principal diferencia entre esta distribución *Cloud Computing* y el resto de las del mercado. Es tan sencillo como conectarse vía Web a la dirección IP del servidor que realiza las funciones de Nodo Controlador. En nuestro caso nos conectamos de la siguiente forma:

```
http://192.168.7.10/8888
```

Al conectarnos, se abre un asistente de instalación (véase la *Figura 2.23*) que comienza con un primer paso en el que nos solicita darnos de alta en la comunidad *StackOps*, si no lo hemos hecho con anterioridad.

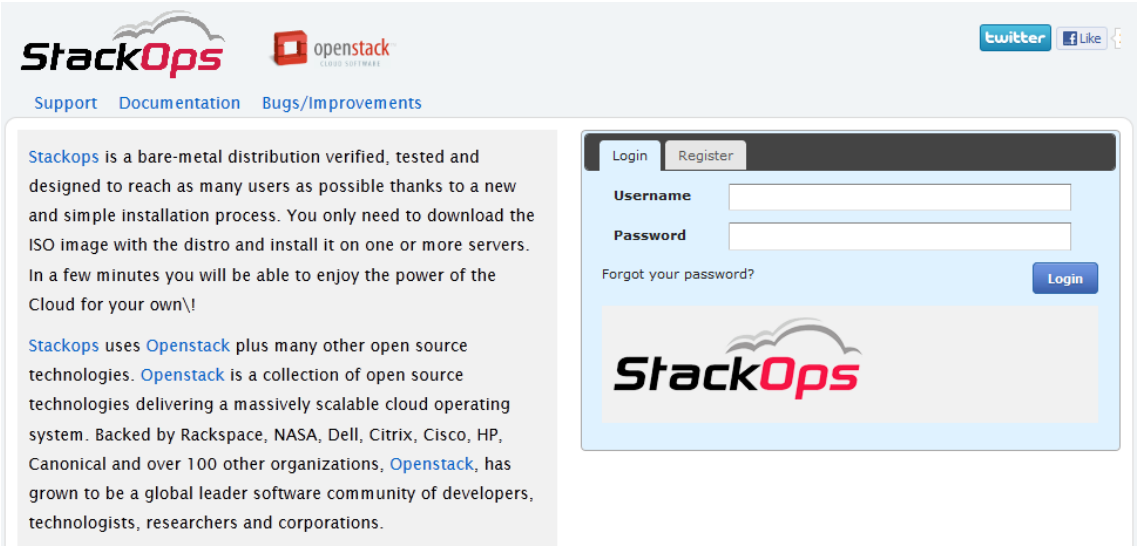


Figura 2.23. Autenticación *StackOps*

Si no se dispone de una cuenta de usuario, basta con pulsar sobre "*Register*", cumplimentar los campos del formulario que aparece en pantalla y crear una nueva. Una vez se disponga de las credenciales de acceso, pulse *"Login"* y acceda como administrador del sistema.

En la *Figura 2.24* podemos ver la Web de inicio. Aquí simplemente pulse sobre *"Create a new Nova Zone from scratch",* ya que aún no hay nada configurado. Puede ver que en el margen derecho aparece una breve descripción del siguiente paso, seleccionar una estructura de nube. Están los tres modos anteriormente comentados: *Single Node, Dual Node* y *Multinode*.

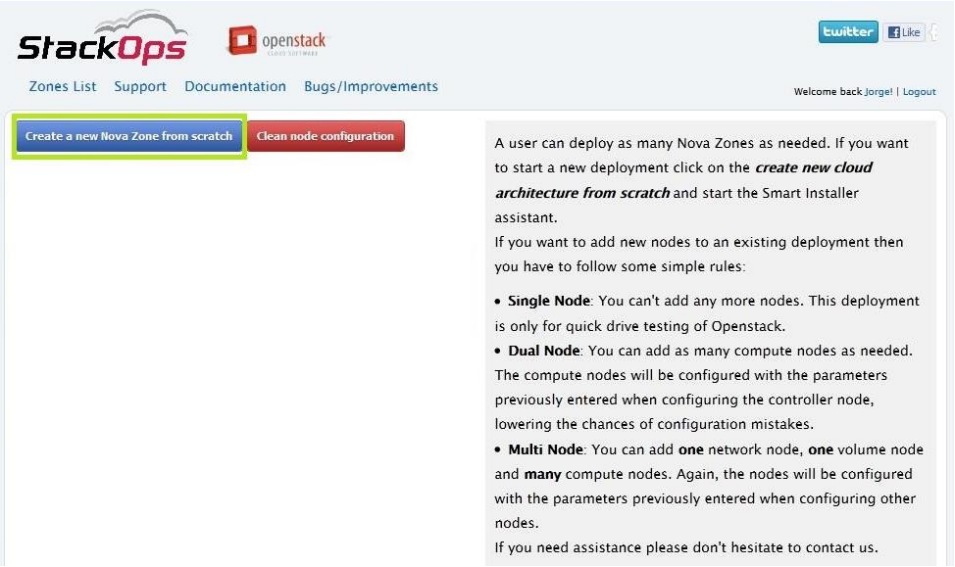


Figura 2.24. Crear una Nueva Nube

Pulse el botón comentado para pasar al siguiente paso ("*Select your deployment architecture*") en el que nos da la opción de escoger la topología de nube que desea configurar (véase la *Figura 2.25*).

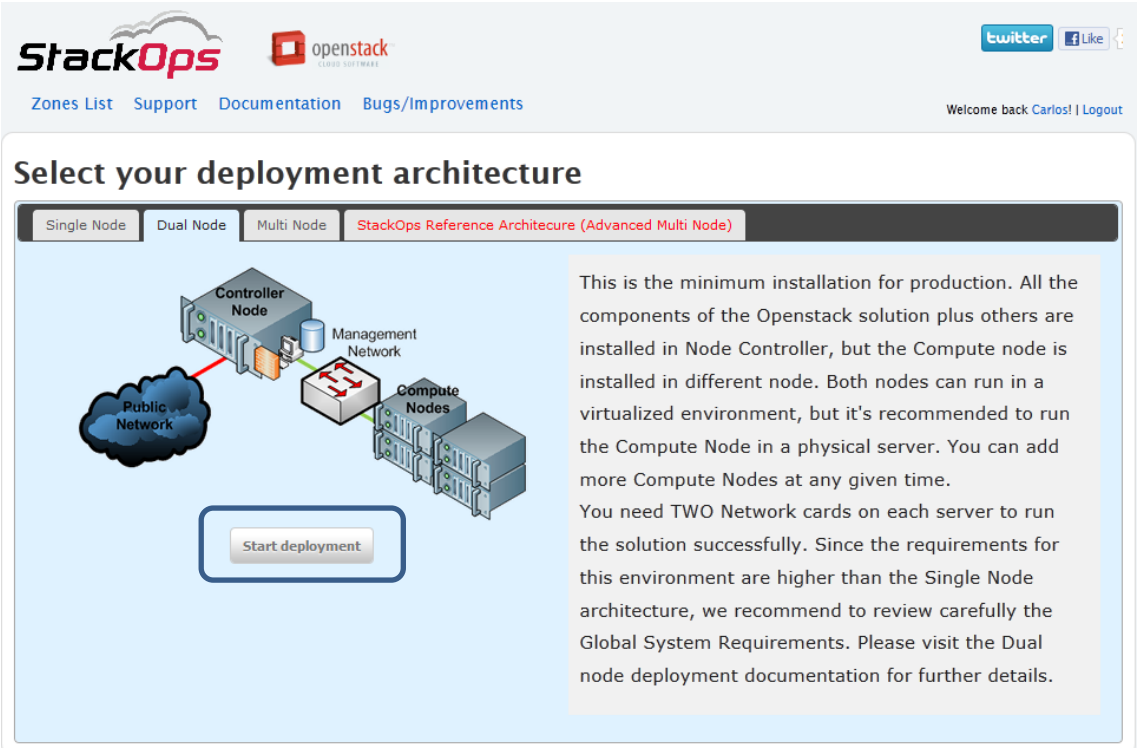


Figura 2.25. Elección *Dual Node*

Seleccionamos para nuestra nube el modo *"Dual Node"* como ya habíamos comentado. Puede leer una breve descripción junto con la imagen que representa la arquitectura. Pulse sobre *"Start Deployment"* y así comienza la configuración, aparece la pantalla que puede ver en la *Figura 2.26*.

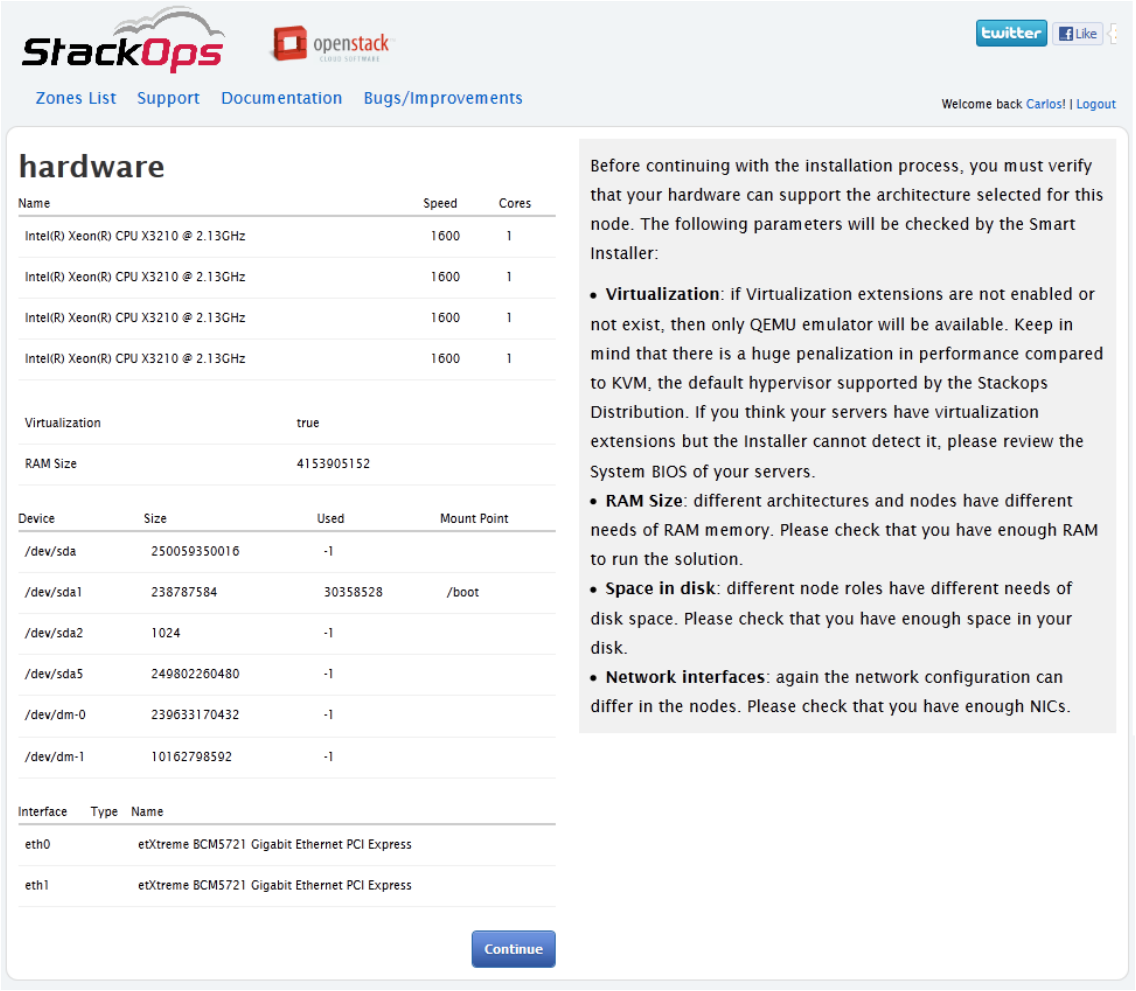


Figura 2.26. Resumen Hardware Controlador

Puede observar en esta *Figura 2.26* un resumen sobre el hardware del servidor que queremos configurar como *Node Controller*. Comprobamos que posee cuatro núcleos, virtualización hardware, 4Gb de RAM, disco duro de 250Gb y dos interfaces de red: recursos suficientes que cumplen con los requisitos de esta configuración.

Pulse *"Continue"* y avanzamos en la configuración a un punto en el que trata aspectos software (véase la *Figura 2.27*).

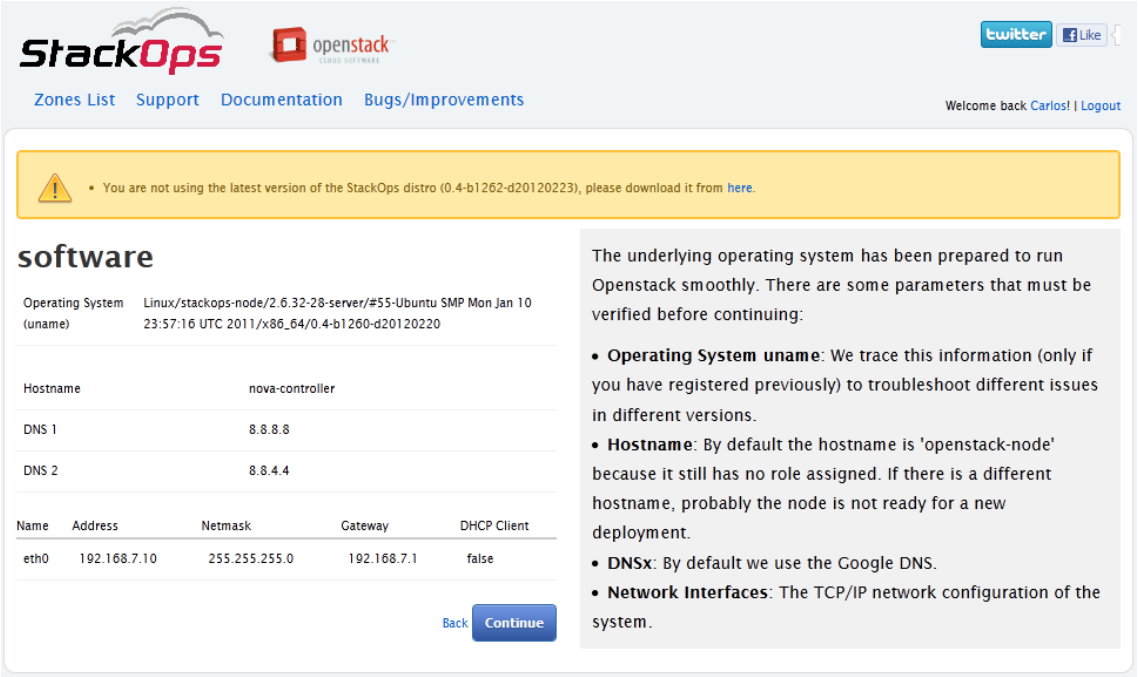


Figura 2.27. Resumen Software Controlador

Comprobamos que detecta correctamente el sistema operativo ya instalado. Además puede ver dos direcciones DNS y la interfaz de red *"eth0"* con la IP del servidor y la del *router* que tiene asociado dentro de la red.

Pulse sobre *"Continue"* y acceda al siguiente punto de la configuración que trata sobre aspectos de la topología de red del sistema *Cloud Computing* que estamos desplegando (véase la *Figura 2.28*).

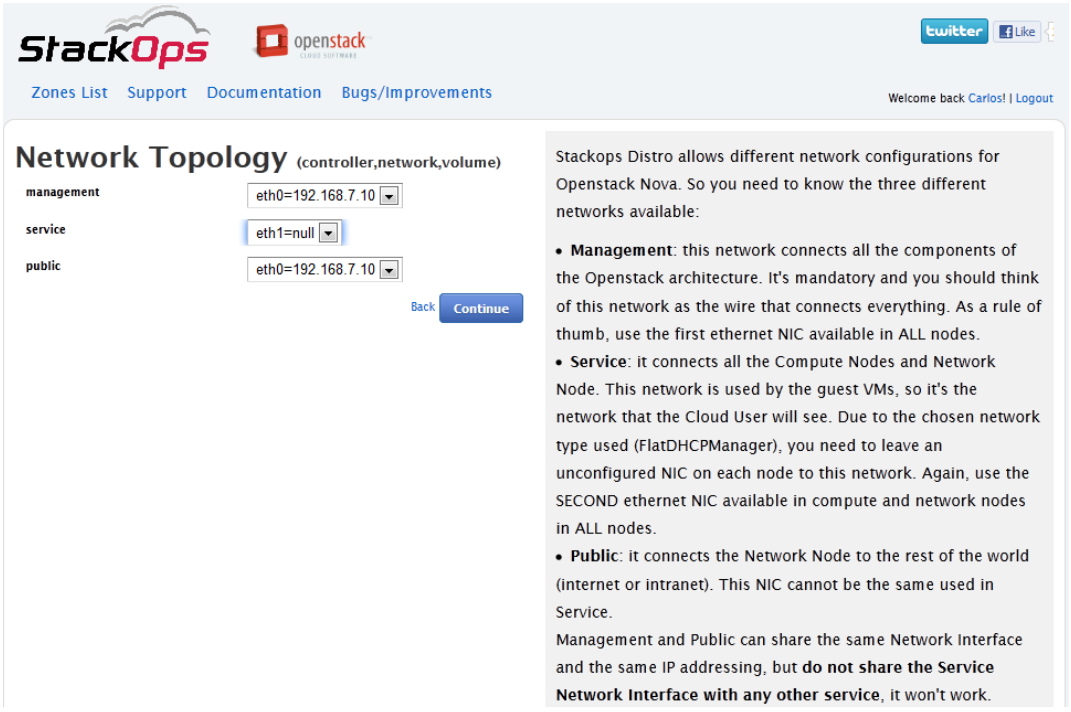


Figura 2.28. Topología de Red Controlador

En el paso "*Network topology*" tenemos que realizar la primera configuración de importancia. Tenemos que asignar las interfaces de red que precisa esta arquitectura de nube:

- La **Red de Gestión** (*Management*) conecta todos los nodos de la nube y utiliza la primera interfaz de red de todos ellos (*eht0*).
- La **Red de Servicios** (*Service*) conecta los Nodos de Cómputo y crea la red de nodos. Es utilizada por las máquinas virtuales y debe usar una interfaz distinta para todos los nodos.
- La **Red Pública** (*Public*) conecta todos los nodos con el resto del mundo. No puede ejecutarse bajo la misma interfaz que la Red *Service*.

Avanzamos en la instalación y llegamos a un apartado en el que se recogen diversos aspectos de configuración globales, todos ellos con valores por defecto (véase la *Figura 2.29 y 2.30*).

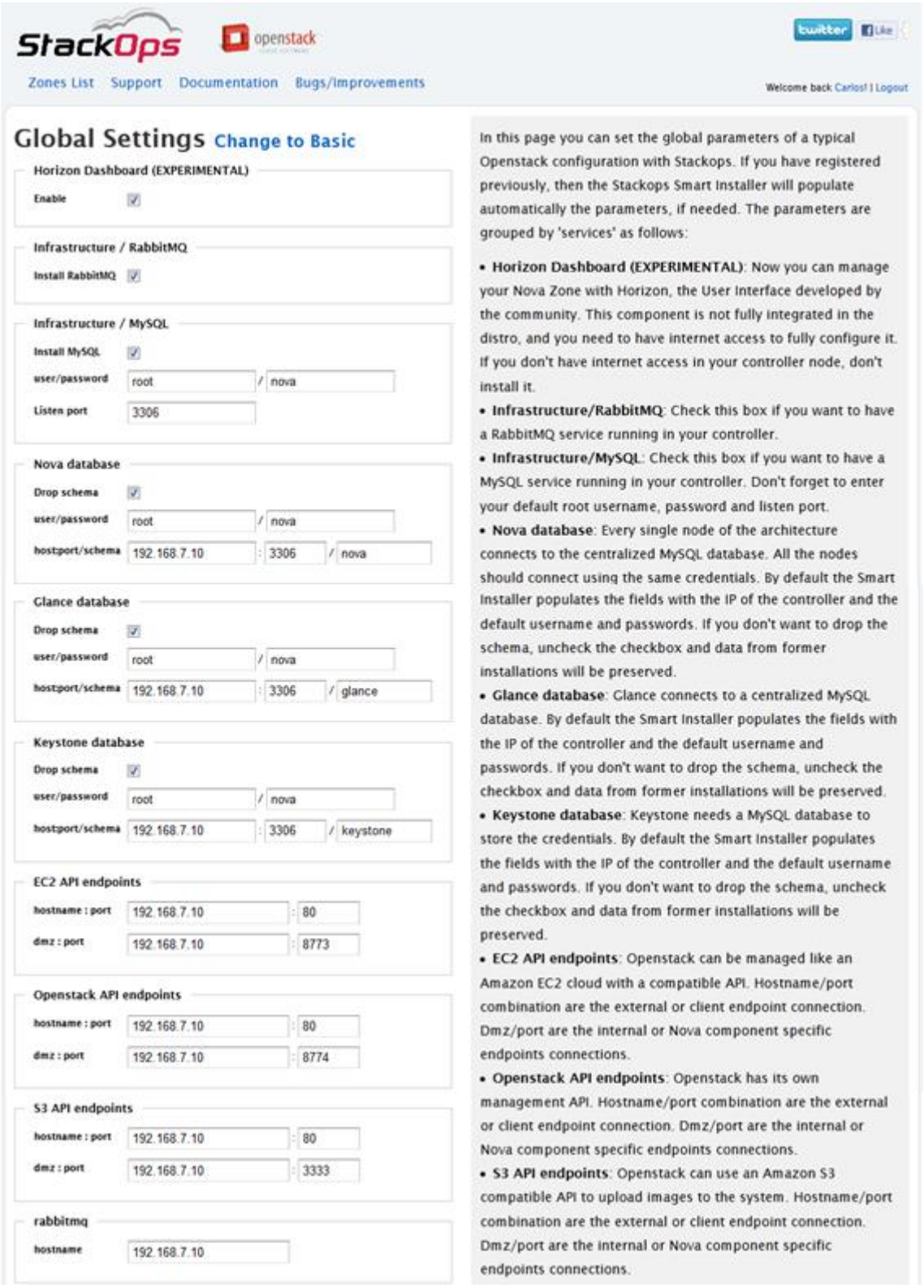
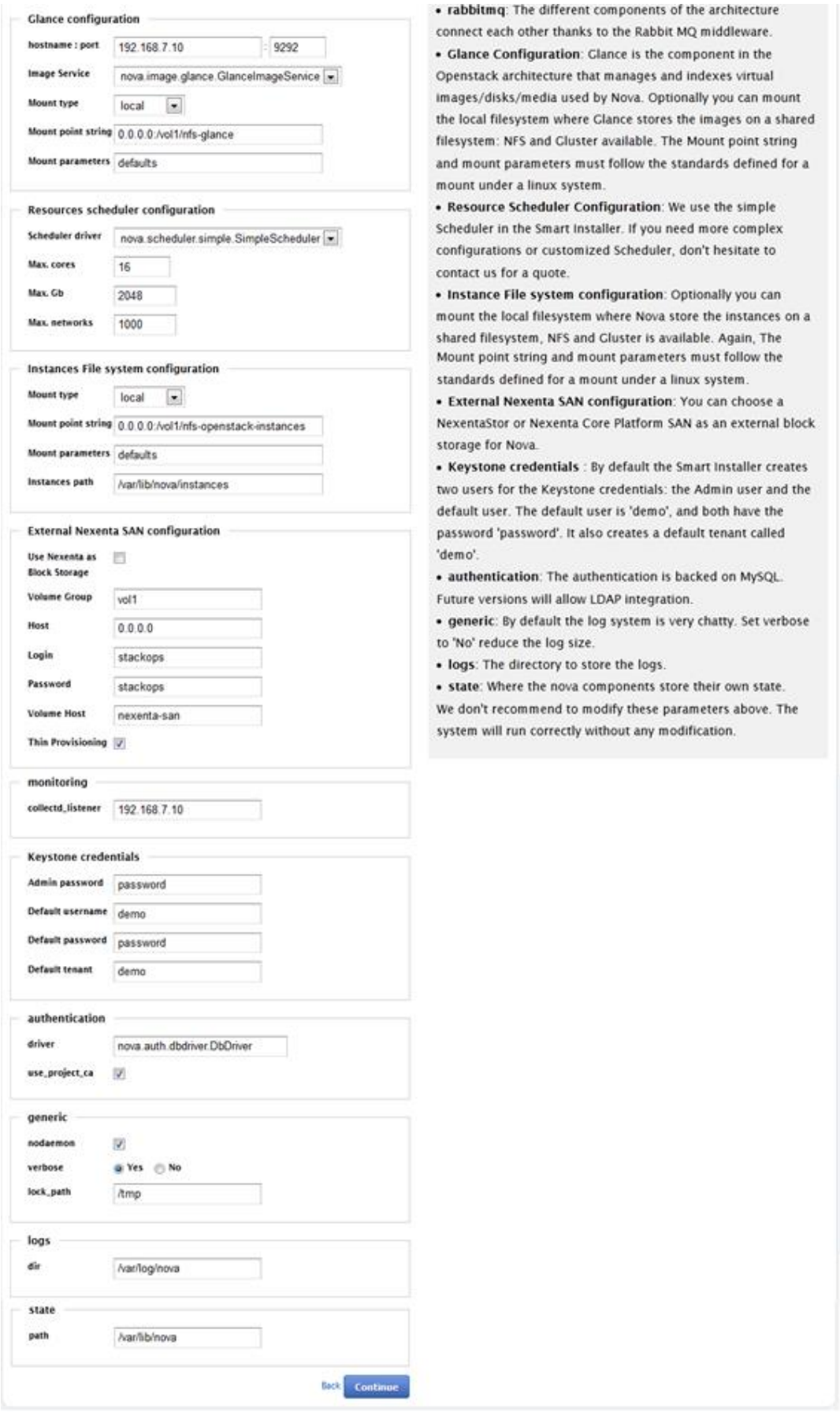


Figura 2.29. Configuración General (1 de 2)



- **rabbitmq**: The different components of the architecture connect each other thanks to the Rabbit MQ middleware.
- **Glance Configuration**: Glance is the component in the Openstack architecture that manages and indexes virtual images/disks/media used by Nova. Optionally you can mount the local filesystem where Glance stores the images on a shared filesystem: NFS and Gluster available. The Mount point string and mount parameters must follow the standards defined for a mount under a linux system.
- **Resource Scheduler Configuration**: We use the simple Scheduler in the Smart Installer. If you need more complex configurations or customized Scheduler, don't hesitate to contact us for a quote.
- **Instance File system configuration**: Optionally you can mount the local filesystem where Nova store the instances on a shared filesystem, NFS and Gluster is available. Again, The Mount point string and mount parameters must follow the standards defined for a mount under a linux system.
- **External Nexenta SAN configuration**: You can choose a NexentaStor or Nexenta Core Platform SAN as an external block storage for Nova.
- **Keystone credentials** : By default the Smart Installer creates two users for the Keystone credentials: the Admin user and the default user. The default user is 'demo', and both have the password 'password'. It also creates a default tenant called 'demo'.
- **authentication**: The authentication is backed on MySQL. Future versions will allow LDAP integration.
- **generic**: By default the log system is very chatty. Set verbose to 'No' reduce the log size.
- **logs**: The directory to store the logs.
- **state**: Where the nova components store their own state.

We don't recommend to modify these parameters above. The system will run correctly without any modification.

Figura 2.30. Configuración General (2 de 2)

En las figuras anteriores es posible apreciar los diferentes campos globales que pueden ser modificados y configurados según se desee. Puede ver y configurar, si lo considera oportuno, la totalidad de parámetros globales, componentes y servicios de nuestra infraestructura *OpenStack*: infraestructuras *RabbitMQ* y *MySQL*, las bases de datos *Nova, Glance* y *Keystone*¸ los puntos finales para las APIs EC2, OpenStack y S3, el *hostname* para *rabbitmq*, así como la configuración de *Glance*, el planificador de recursos, el sistema de ficheros utilizado para las instancias, los directorios para *logs,* etc. Como puede comprobar es posible realizar una configuración avanzada en detalle orientada a la distribución de los servicios.

Pulse una vez más *Continue* y pasamos al siguiente punto de la instalación (véase la *Figura 2.31*) en el que aparece información sobre el Nodo Controlador que está configurando.

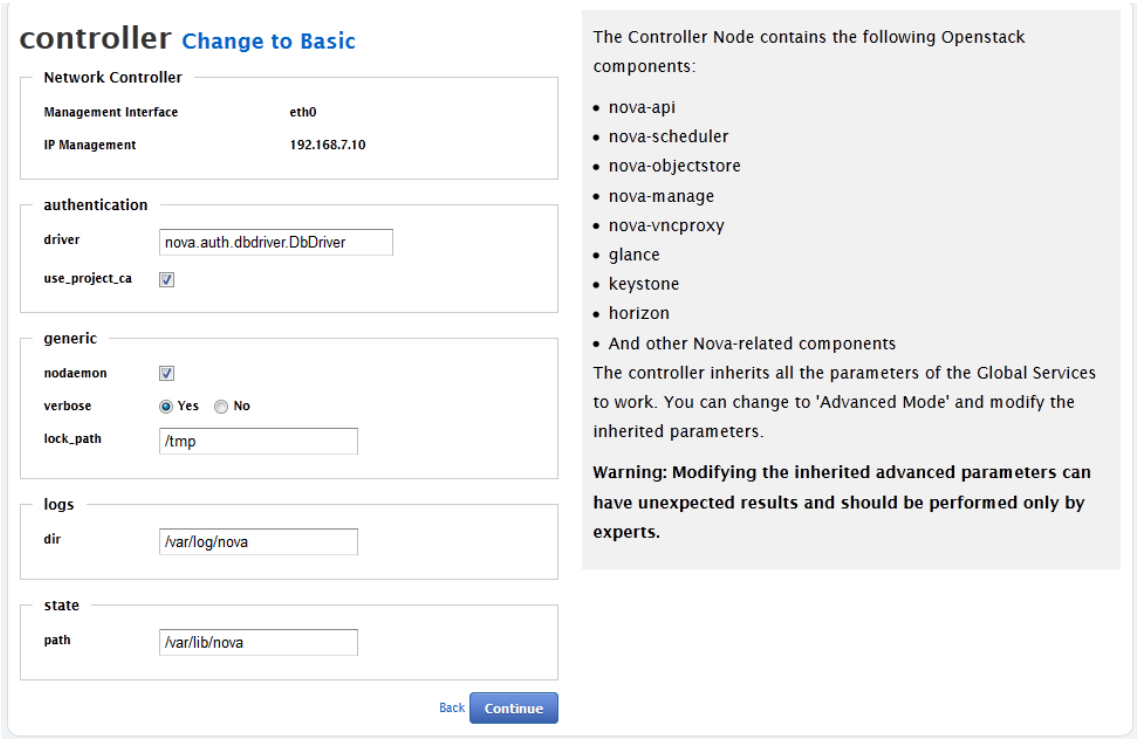


Figura 2.31. Configuración Controlador

Primero nos informa de la IP del mismo y de la interfaz de red de gestión. En la parte derecha vemos los componentes que se van a instalar. Puede ver el driver que gestiona la autenticación en el sistema *Cloud*. Nos solicita algunos directorios para almacenar distintos archivos como son los *"logs"*, o los ficheros de estado, si estamos conformes con los que vienen por defecto, continuamos.

A continuación, es el momento de configurar diversos aspectos sobre las redes de comunicación de la nube (véase la *Figura 2.32*).

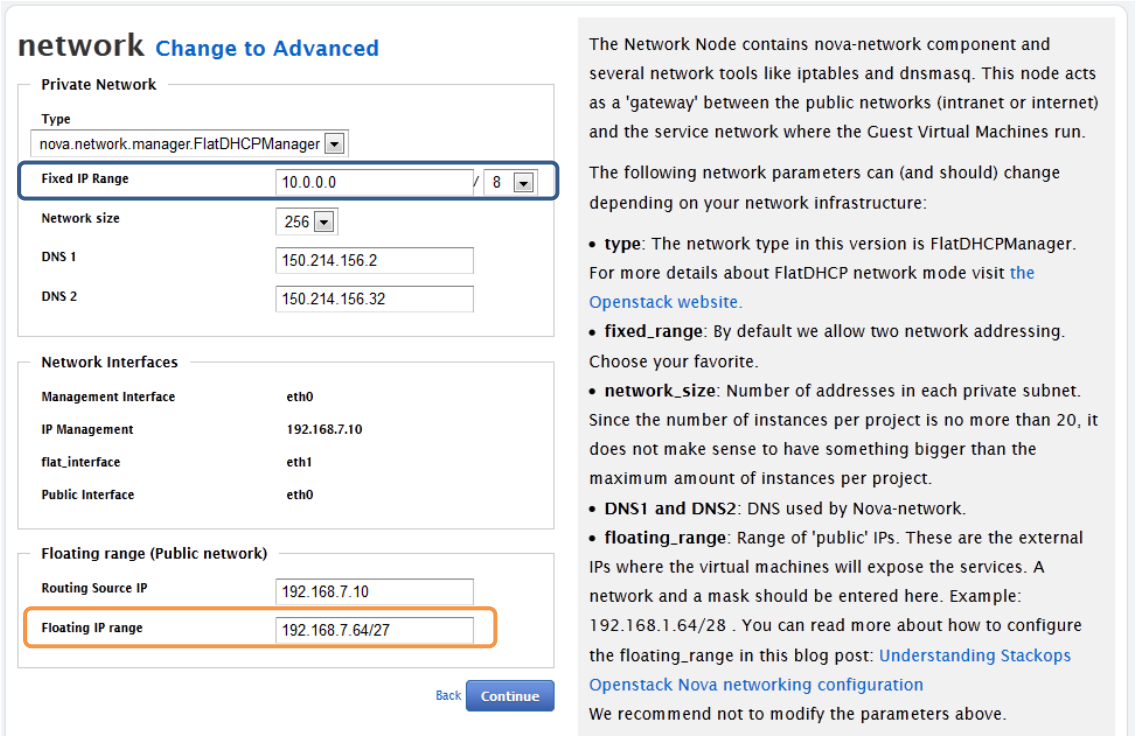


Figura 2.32. Configuración de Red

En la primera sección hay que facilitar unos datos de configuración muy importantes para establecer la red privada. En esta red irán desplegándose las instancias virtuales, por ello hemos de proporcionar un patrón sobre el que se irán concediendo direcciones IP privadas a estas máquinas. Por esto facilitamos en el campo "*Fixed IP Range*" un rango de direcciones de red que puede ver destacado en azul.

Otro campo a destacar es el que está destacado en naranja "*Floating IP range*". En él hay que indicar el rango de direcciones de red que servirá para asociar las instancias virtuales con una dirección pública que les dé salida a Internet. Así las cosas, las máquinas virtuales tendrán dos direcciones IP, por tanto dos interfaces, una de ellas privada (red privada) y otra pública (red de servicios y red pública). Además incluimos las direcciones DNS que utilizamos en el laboratorio, para una correcta salida a Internet.

Continuamos con la instalación, pasando ahora a configurar el almacenamiento en el sistema *Cloud* (véase la *Figura 2.33*).

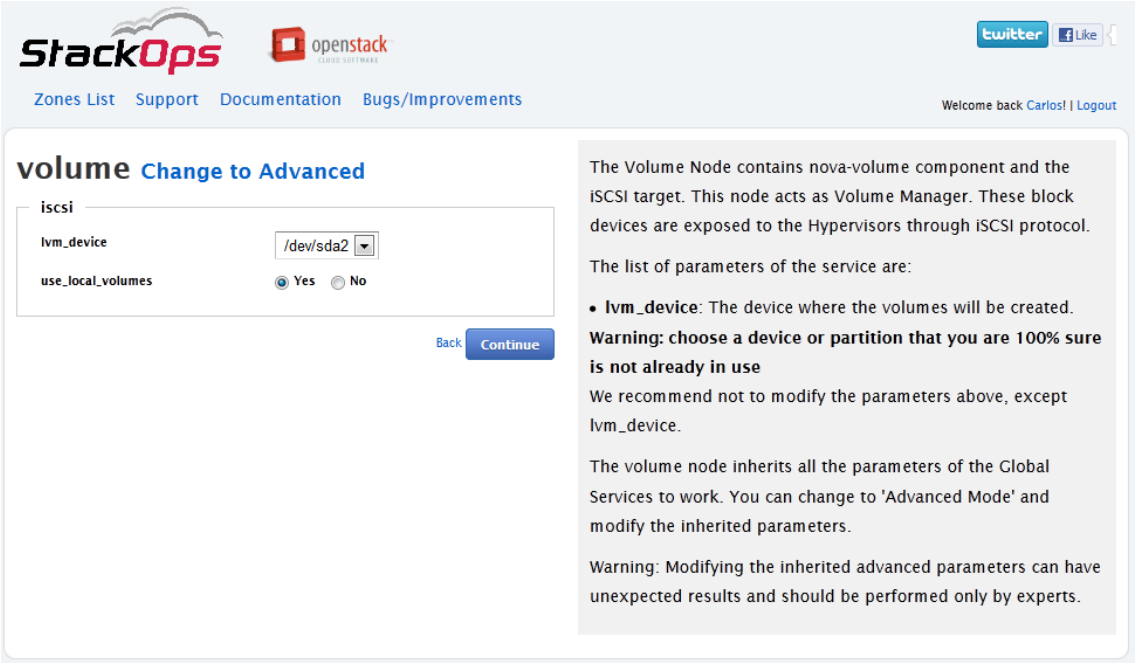


Figura 2.33. Configuración Almacenamiento Controlador

En esta ocasión, tan solo hay que seleccionar el dispositivo de almacenamiento que ejerza como "*nova-volume*" que actúa como Nodo de Almacenamiento y que establece el control sobre los datos del sistema. Es posible realizar una configuración avanzada más exhaustiva pulsando sobre "*Change to Advanced*".

Llegamos ya al final del proceso de configuración en el que completamos información de la nube utilizada para poder identificarla posteriormente (véase la *Figura 2.34*). Así, si posteriormente creamos nuevas nubes, podremos distinguir unas de otras, tanto por su nombre como por la descripción de la misma.

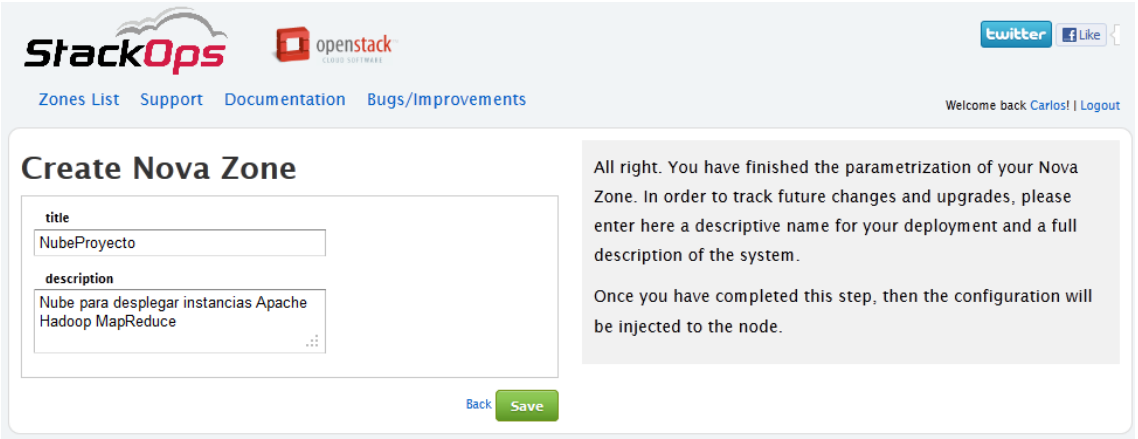


Figura 2.34. Nombre y Descripción de la Nube

Como hemos comentado, introducimos el nombre para la nube rellenando el campo "*title*" y una descripción de la misma completando el campo "*description*". Cuando hayamos rellenado los campos correctamente, guardamos todos los datos de la configuración pulsando el botón "*Save*" que vemos resaltado en verde.

Llegamos al final de la instalación e indicamos que se inicie el despliegue de los datos de configuración (véase la *Figura 2.35*).

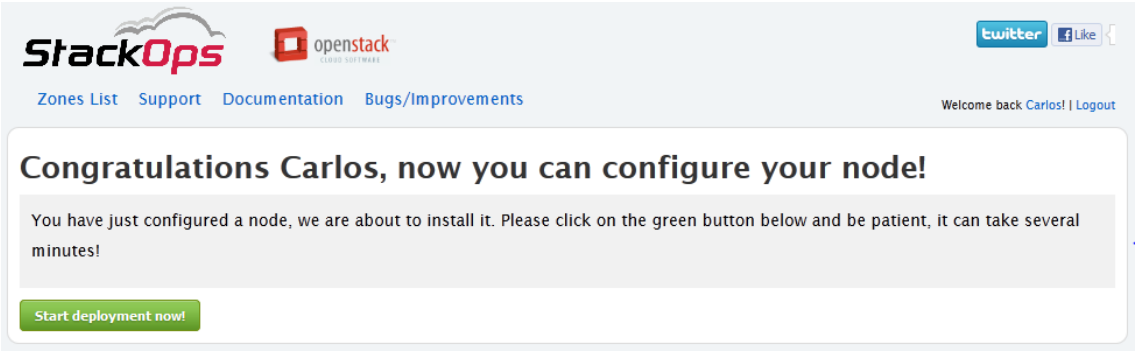


Figura 2.35. Aplicar Configuración

Pulsamos *"Start deployment now!"* y esperamos a que finalice la instalación. Este proceso de instalación suele tener una duración media de quince minutos ya que se realiza la configuración de todos los datos indicados anteriormente.

Una vez concluye la configuración, ya tenemos la parte del controlador de la nube correctamente configurada. Una vez hecho esto, volvemos al primer paso de la instalación, en el que puede ver que aparece una nube sobre la que es posible agregar Nodos de Cómputo.

Añadir Nodos de Cómputo

A continuación se muestra el proceso para añadir Nodos de Cómputo (véase la *Figura 2.36*).

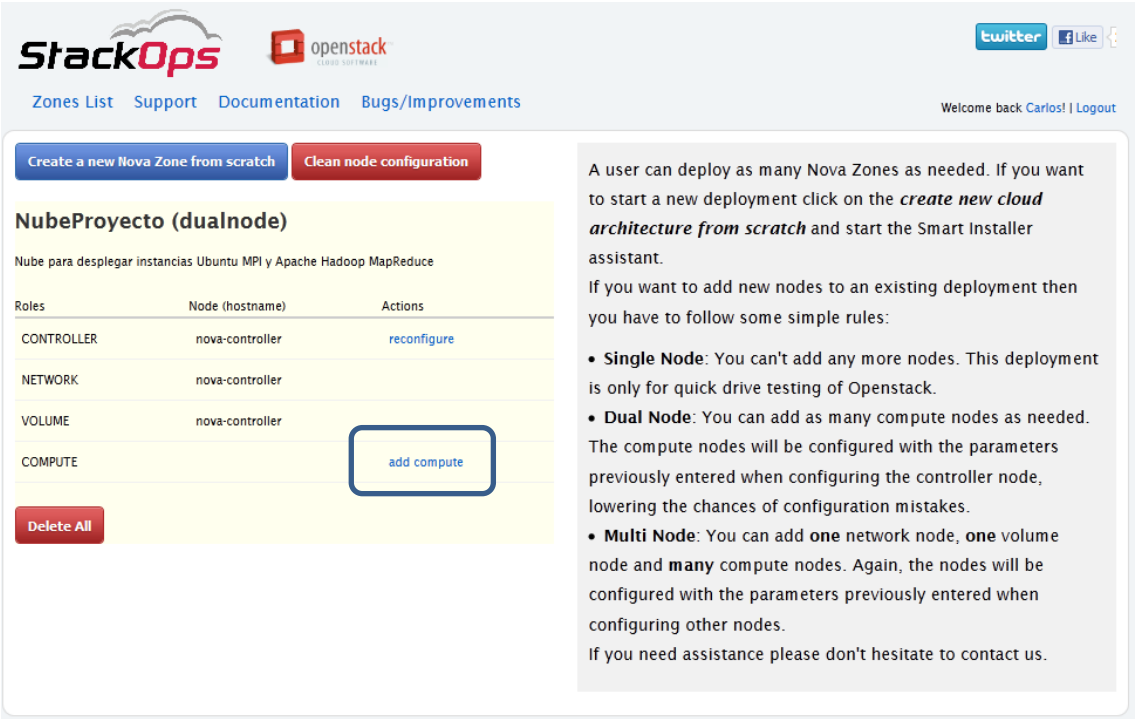


Figura 2.36. Resumen Nube

Si nos fijamos en la *Figura 2.24*, que es una de las primeras imágenes de la instalación, puede ver que ésta es la misma ventana, solo que en esta ocasión aparece la nube ya desplegada con el nombre y la descripción anteriormente introducidos. La interfaz Web nos ofrece distintas opciones llegados a este punto, como son crear otra nueva nube ("*Create a new Nova Zone from scratch*"), limpiar la configuración en todos los nodos que pertenezcan al *Cloud* ("*Clean node configuration*") o eliminar la nube ("*Delete All*"). También puede ver como el Nodo Controlador está correctamente configurado y además alberga los Nodos de Red y Almacenamiento. Si pulsamos sobre "*reconfigure*" vuelve a ejecutarse el instalador que acabamos de realizar en los pasos anteriores, pero esta vez incluyendo los datos ya configurados en todos los campos. Sin embargo, en nuestro caso, vamos a añadir Nodos de Cómputo a nuestra configuración *Cloud Computing* pulsando sobre el botón resaltado con el cuadro azul, *"add compute"*.

Como hemos dicho, comenzamos a añadir Nodos de Cómputo en la nube desplegada. Para ello, nos conectamos vía Web desde los servidores que queremos ir incluyendo. A continuación, vamos a realizar el proceso de configuración para uno de los servidores. Solo vamos a documentar los pasos de instalación para uno de ellos, ya que el proceso es el mismo para todos los servidores. Abrimos el navegador y nos conectamos a la IP del servidor que queremos convertir en Nodo de Cómputo de la nube, en este caso:

```
http://192.168.7.11/8888
```

Automáticamente el Controlador de *StackOps* detecta que ya hay una nube en la red y carga la ventana del navegador que vimos en la *Figura 2.36,* solo que esta vez lo hace desde la IP del nuevo servidor.

Como ya comentamos antes, iniciamos la adición del nuevo servidor a la nube como Nodo de Cómputo pulsando en el enlace *"add compute"*. Carga la ventana que vemos a continuación en la *Figura 2.37*.

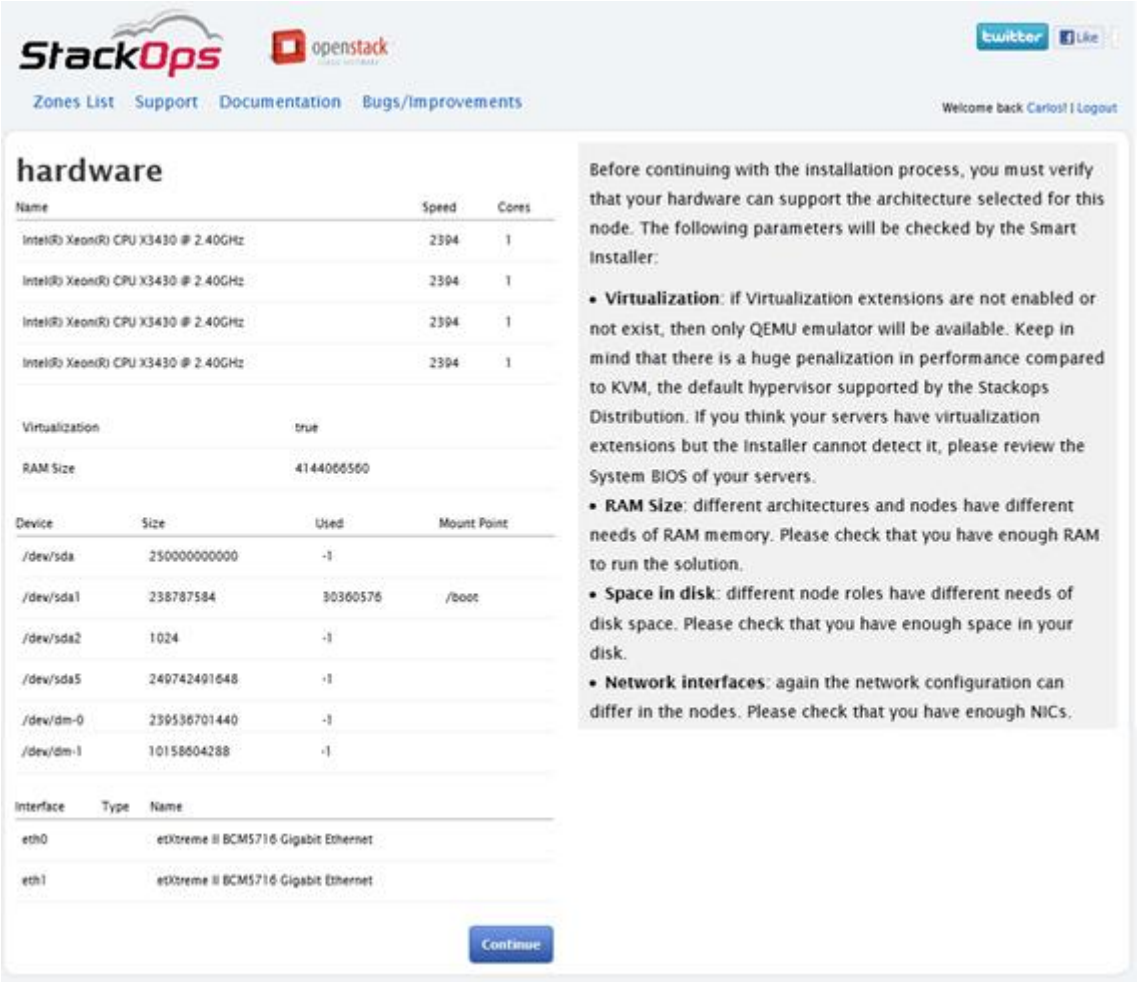


Figura 2.37. Resumen Hardware Nodo Cómputo

La *Figura 2.37* muestra un resumen del hardware del equipo muy similar a la que nos presentaba en el caso de la instalación del Nodo Controlador (véase *Figura 2.26*). Aparecen los cuatro procesadores, la presencia del *flag* de virtualización por hardware, 4Gb de memoria RAM, 250Gb de disco duro y las dos interfaces de red.

Continuamos con la instalación del Nodo de Cómputo. Al igual que en el caso del hardware, en esta ocasión se trata de una ventana que ya conocemos y que vimos en el caso del Nodo Controlador (véase la *Figura 2.27*). En la *Figura 2.38* aparece el sistema operativo base instalado y algunos datos de red como las direcciones DNS y las direcciones de la interfaz de red eth0.

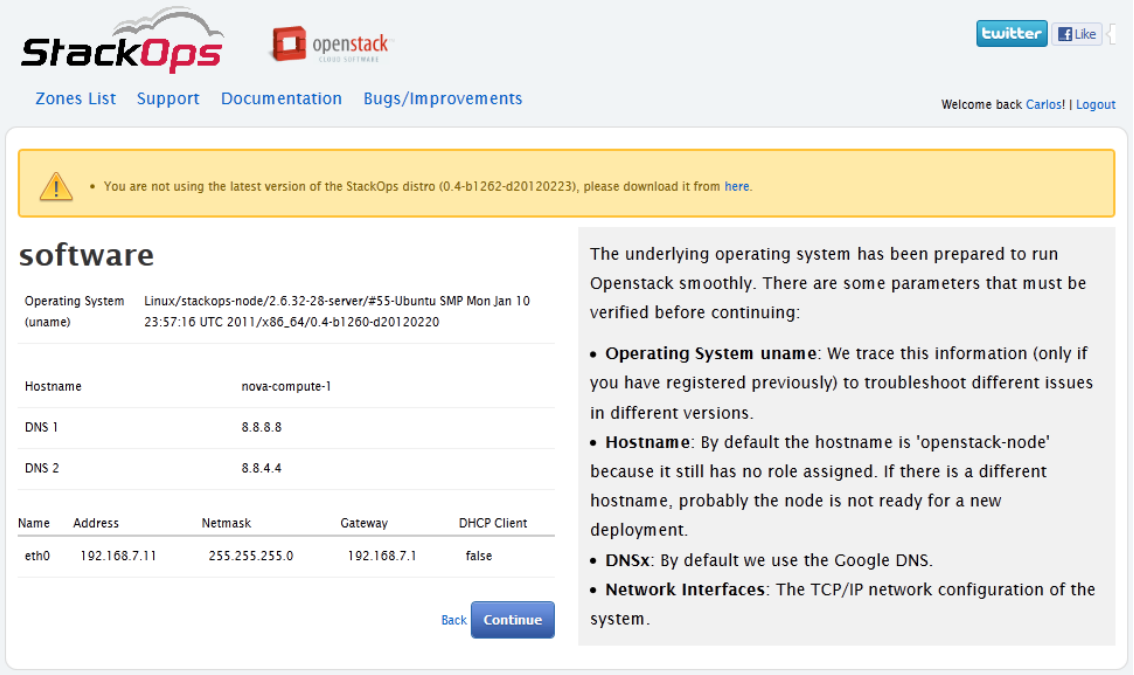


Figura 2.38. Resumen Software Nodo Cómputo

Avanzamos en la configuración pulsando el botón "*Continue*" y pasamos a la configuración de red (véase la *Figura 2.39*).

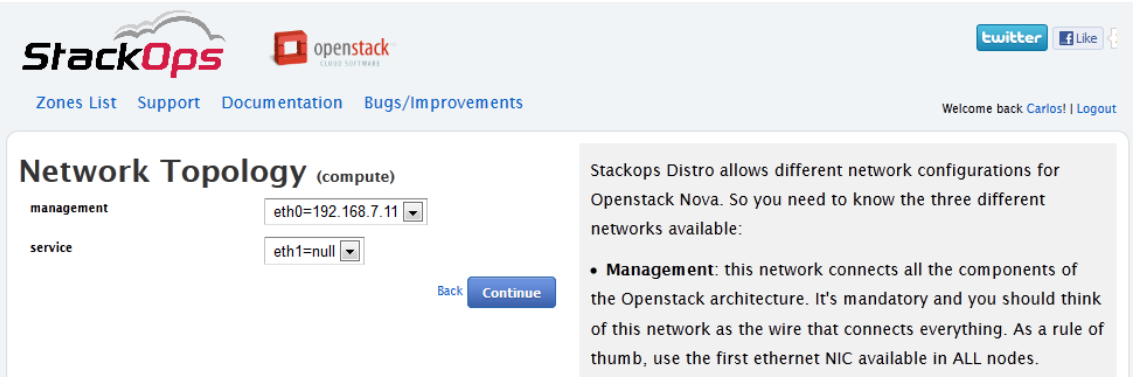


Figura 2.39. Topología de Red Nodo Cómputo

En este caso, la configuración de red difiere de la del Nodo Controlador, ya que no es necesario indicar la interfaz de red a utilizar por la red pública. Esto es debido a que ese servicio se le otorga el Nodo Controlador. Solo es necesario indicar la interfaz para la red de gestión y para la red de servicios interna. Como ya comentamos anteriormente, ambas redes han de situarse con interfaces diferentes.

Tras esto, seguimos con la configuración, pulsando de nuevo el botón "*Continue*". Este paso es importante (véase la *Figura 2.40*), ya que vamos a configurar los datos de la conexión con el servidor que ejerce de Nodo Controlador. Los Nodos de Cómputo ejecutan el hipervisor KVM/QEMU, por ello es necesario indicar que incluimos la librería en la instalación. Puede elegir entre KVM o QEMU, nosotros hemos elegido la primera. KVM es un hipervisor más contrastado y completo que QEMU, por eso nuestra elección. Además hemos de incluir la IP del controlador para que se establezca la conexión entre ambos.

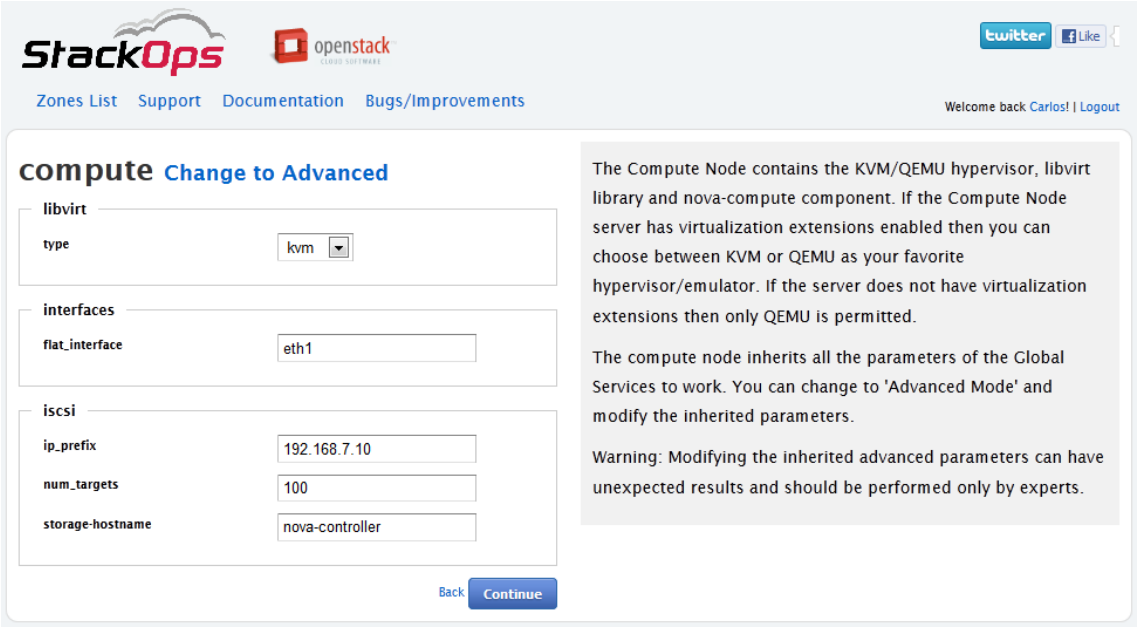


Figura 2.40. Configuración General Nodo Cómputo

Habiendo indicado los datos necesarios, avanzamos y se muestra la *Figura 2.41.*

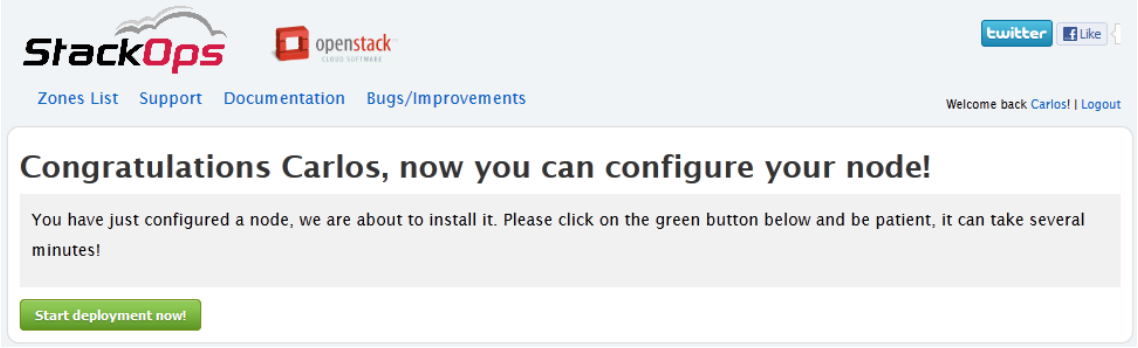


Figura 2.41. Añadir Nodo Cómputo

En este momento finaliza la adición de un Nodo de Cómputo a nuestra nube. Igual que en la primera ocasión, iniciamos el proceso de despliegue de los cambios realizados en la nube, sabiendo que es largo y que tardará algún tiempo.

Una vez pasado el tiempo de configuración, hemos de repetir los pasos anteriores tantas veces como servidores queramos incluir como Nodos de Cómputo. En nuestro caso, tras adherir todos los servidores de los que disponemos, hemos logrado configurar una nube con los

siguientes servidores trabajando de manera conjunta, como muestra *Figura 2.42* que recoge la información de resumen de la Web *StackOps*:

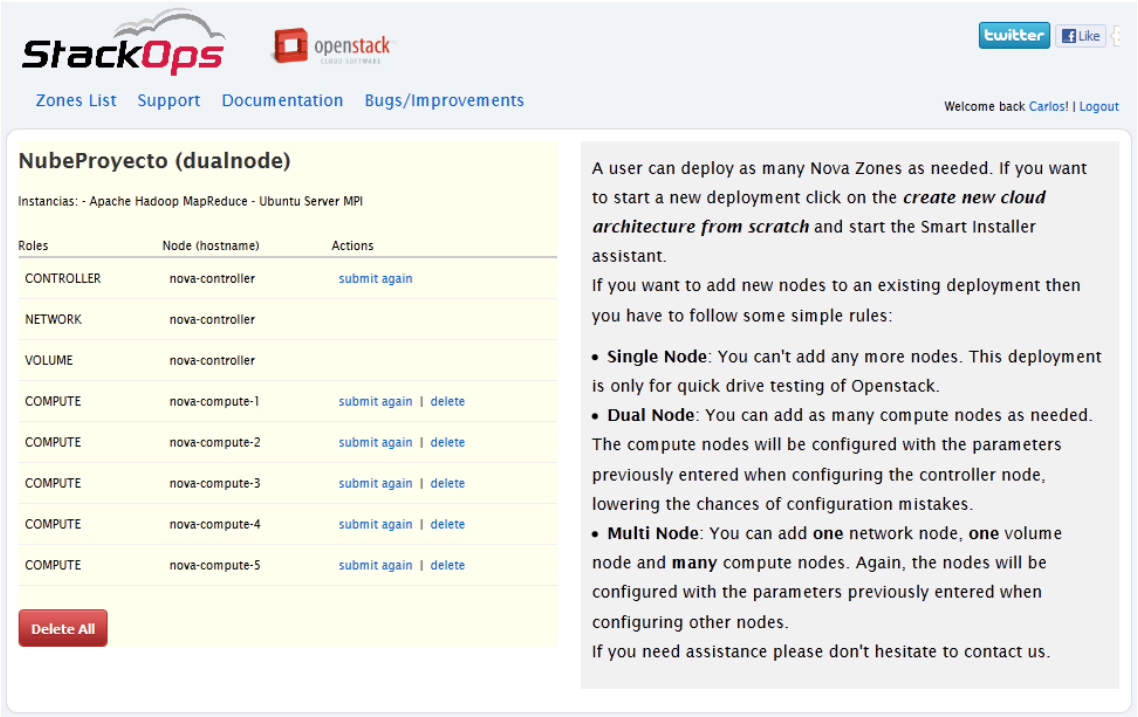


Figura 2.42. Resumen Nube Completa

Como puede comprobar en la *Figura 2.42*, se tiene un Nodo Controlador y cinco servidores que trabajan como Nodos de Cómputo. Ya están dispuestos todos los nodos del sistema para trabajar conjuntamente conformando nuestro entorno *Cloud Computing.*

2.4.1. Configuración del Controlador

Antes de dar por finalizada la configuración de la nube, hemos de realizar unas operaciones sobre el servidor que actúa de Nodo Controlador. Lo primero es comprobar el estado básico de la plataforma, (véase la *Figura 2.43*), ejecutando el siguiente comando en el Nodo Controlador:

```
./nova-manage
```

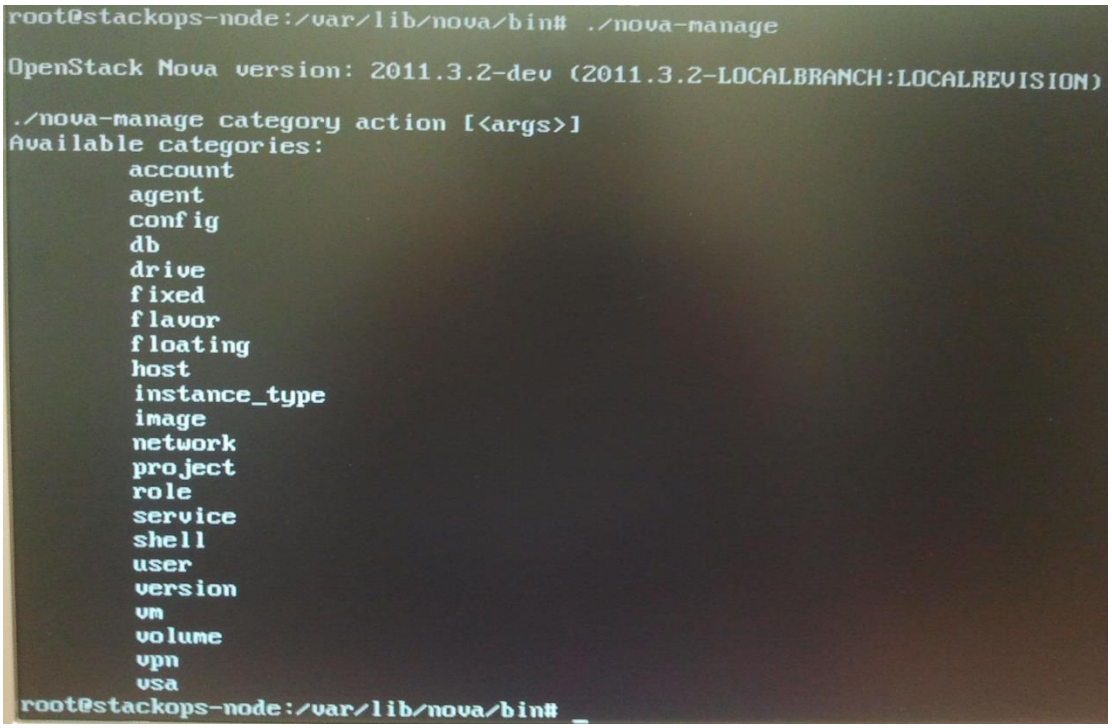


Figura 2.43. *Nova Manage*

El resultado de la ejecución del comando anterior es un listado de opciones que se ejecutan conjuntamente con él y que nos ofrecen distintos resultados. Puede obtener información sobre las imágenes que hay en el sistema, la red, el proyecto, los servicios, usuarios, etc. En nuestro caso, vamos a realizar un listado de los servicios en ejecución, para comprobar el correcto estado del entorno. Para ello, disponemos del siguiente comando:

```
./nova-manage service list
```

En la *Figura 2.44*, puede ver como el Nodo Controlador y los Nodos de Cómputo funcionan correctamente y en todas sus funciones. Para poder diferenciar los nodos que están ejecutándose de manera correcta de los que no lo hacen, hemos de fijarnos en la columna status y ver si aparece una cara sonriente ": -)" o un símbolo de triple equis "XXX".

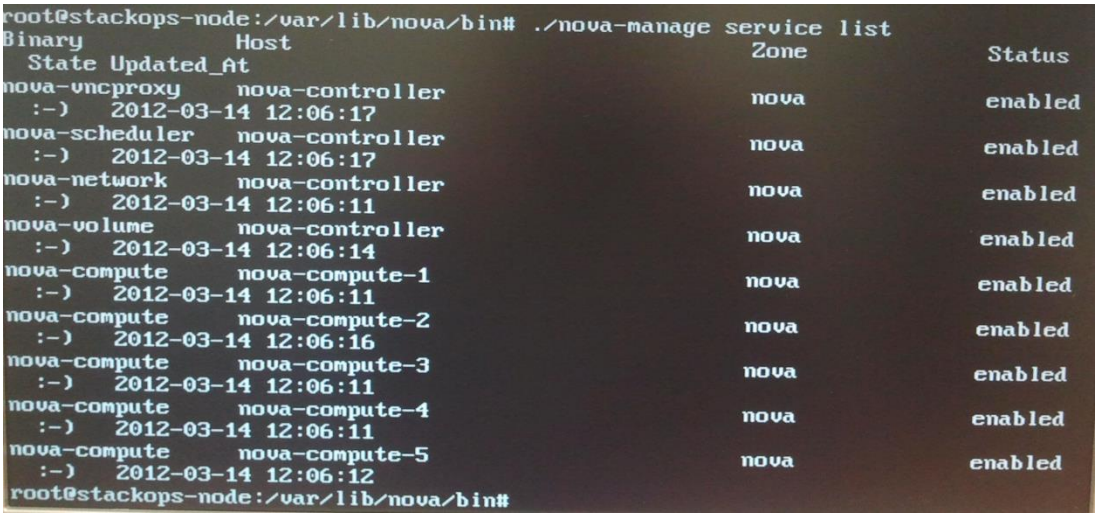


Figura 2.44. *Nova Manage* Lista de Servicios

Por defecto la distribución *StackOps* utiliza *Keystone* para la gestión de los usuarios de la plataforma. Hay dos usuarios creados por defecto, uno de ellos *'admin'* con la clave *'password'* y *'admin:password'* para las credenciales de EC2. Desde *StackOps* no se recomienda utilizar estas credenciales como usuario del *Cloud*. Es recomendable utilizar el otro usuario *'demo'* con clave *'password'* y *'demo:password'* para las credenciales de EC2. Para configurar todas las variables de entorno para el usuario 'demo', ha de ejecutarse el script "`setenv.sh`" (véase la *Figura 2.*45) que encontramos en el directorio "`/var/lib/stackops`" y que genera todo lo necesario. Este *script* puede modificarse si es necesario crear algún otro usuario.

Figura 2.45. *Script setenv.sh*

Existe un BUG reconocido por *StackOps* y para poder eliminarlo, hemos de ejecutar el comando:

```
export OS_AUTH_STRATEGY=keystone
```

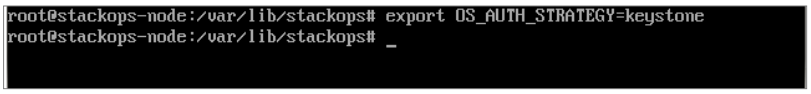


Figura 2.46. BUG *StackOps*

Para completar la instalación, vamos a desplegar una máquina virtual y así cuando nos conectemos a la interfaz Web de la nube, ya tendremos un patron para ir desplegando instancias. Crear un patrón supone desplegar una máquina virtual, aplicarle la configuración deseada y grabarla para ser desplegada posteriormente tantas veces como queramos. Con esto obtenemos una ventaja importante, ya que puede ir creando nuevas máquinas, ahorrándonos los pasos de configuración e instalación de sistema operativo y demás servicios o aplicaciones.

En las distribuciones *StackOps* se incluye un *script*, localizado en el directorio "`/var/lib/stackops`" y llamado "`pububuntu1004.sh`" (véase la *Figura 2.47*), que descarga una máquina virtual con el sistema operativo *Ubuntu 10.04*, que es el que vamos a utilizar para la creación de las las instancias virtuales ya que nos servirá de base para el clúster que posteriormente queremos crear. Además, en dicho directorio, aparecen otros *scripts* que

contienen los enlaces de descarga de otras máquinas virtuales preconfiguradas como son "`pubcentos57.sh`", "`pubcentos62.sh`", "`pubdebian6.sh`" o "`pubttylinux.sh`".

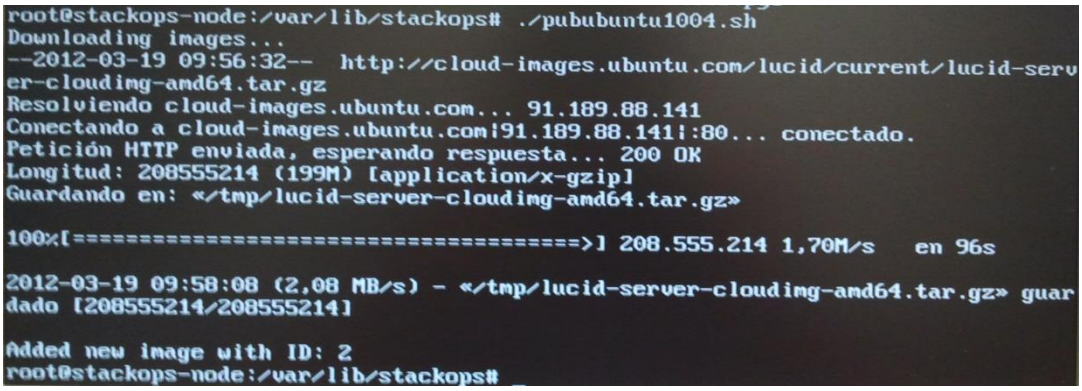


Figura 2.47. Máquina Virtual *Ubuntu*

Está ya desplegada correctamente la nube, se ha configurado el Nodo Controlador, los usuarios y preparado la imagen de una máquina virtual que es utilizada para desplegar instancias en la nube. Damos por finalizado por tanto el proceso de configuración básica del sistema *Cloud*. A continuación acceda a la nube por medio de la interfaz Web y puede comenzar a desplegar instancias virtuales en nuestra nube.

2.5. Interfaz Web *Horizon*.

Una vez hemos desplegada correctamente nuestra nube, llega el momento de comenzar a sacarle el máximo partido. Ya sabemos que una de las ventajas de los entornos *Cloud Computing* es la facilidad para crear y eliminar máquinas virtuales sin tener que preocuparnos del hardware. Para ello, *StackOps* incluye una interfaz Web llamada *Horizon* que permite gestionar el entorno *Cloud*, de forma muy sencilla e intuitiva. Este entorno Web incluye todo lo necesario para la gestión de la nube, tanto a nivel de instancias y máquinas virtuales como a nivel de gestión general del sistema.

Para conectarnos a la interfaz Web, hemos de introducir en el navegador la dirección IP del Nodo Controlador, esta vez sin ningún puerto. Es decir,

```
http://192.168.7.10
```

Con esto se abre una ventana en el navegador como la que puede ver en la *Figura 2.48*, en la que nos solicitan las credenciales de usuario, anteriormente configuradas.

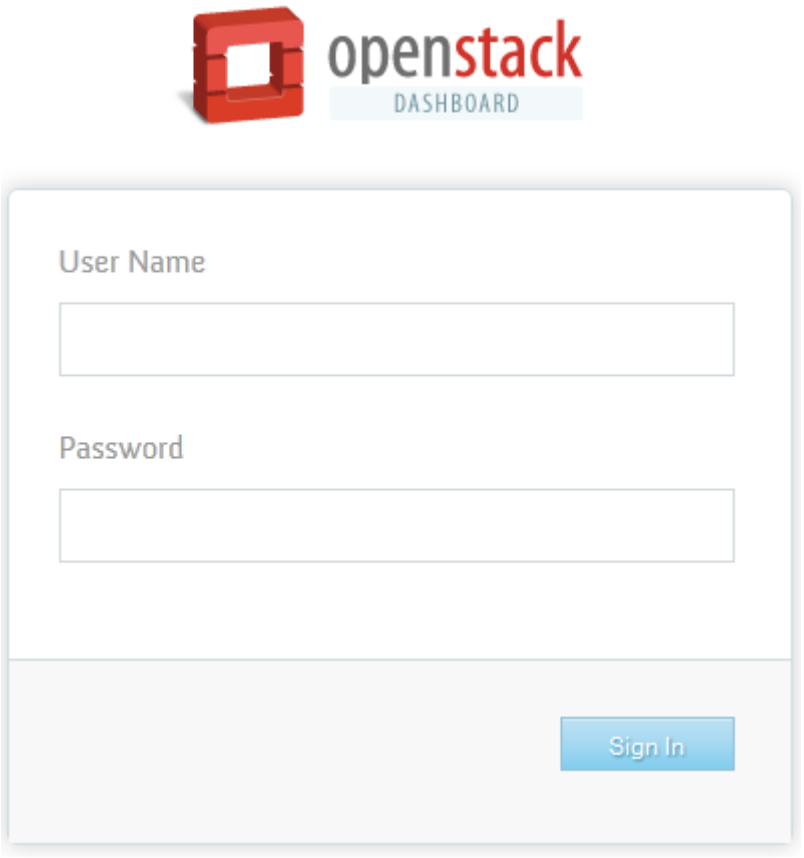


Figura 2.48. Autenticación *Horizon*

Indique sus credenciales y acceda, ahora sí, al controlador Web de la nube que se divide en dos secciones:

- Una de ellas es el **panel de usuario** desde el que puede desplegar todas las funcionalidades de la nube.
- La otra es el **panel de sistema**, en el que puede acceder a toda la información del sistema.

2.5.1. *System Panel*

Hemos comentado que la interfaz Web de nuestra nube se divide en dos paneles. Uno de ellos es "*System Panel*", que contiene toda la información referente a nuestro sistema, su estado y servicios que están en ejecución.

Overview

La primera sección contiene un resumen general de los recursos del sistema y el estado de los mismos (véase la *Figura 2.49*). Muestra información general sobre la utilización de los diferentes recursos disponibles en la infraestructura completa: *estado* de la infraestructura, número de *cores* disponibles y en uso, Gb de memoria RAM disponibles y en uso, Gb de disco de almacenamiento disponibles y en uso. Como puede imaginar, los recursos disponibles y en uso son computados teniendo en cuenta todos los recursos de los *nodos compute* que se encuentran funcionando.

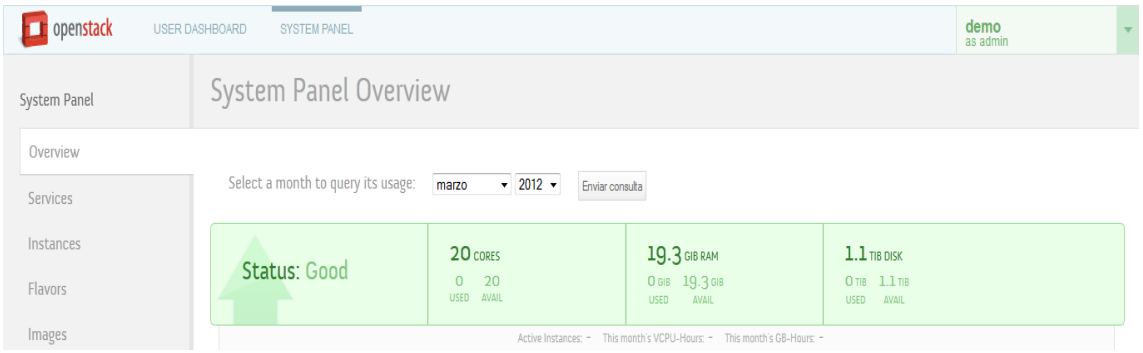


Figura 2.49. Panel Sistema *Horizon*

Puede consultar el uso de los diferentes recursos por mes, así como el número de instancias activas, el total de horas de VCPU consumidas en el mes y el total de Gb consumidos en el mes.

En el caso de nuestra nube, vemos que disponemos en nuestro caso de 20 núcleos, 19.3Gb de RAM (realmente es un total de 20Gb, pero 700Mb los reserva el sistema) y 1.1Tb de almacenamiento.

Services

La siguiente categoría muestra la misma información que vimos en la *Figura 2.44* del *Apartado 2.4.1. Configuración del Controlador* cuando ejecutamos el comando:

```
./nova-manage service list
```

Services permite consultar el estado actual de los diferentes servicios *OpenStack*: *nova-compute, nova-network, nova-schedule, nova-vncproxy, nova-volume, compute, identity, image*. Para cada uno de ellos puede ver en qué *host* se ejecutan, diferente información sobre el estado del mismo, si se encuentran habilitados y en ejecución. Alguno de ellos puede ser deshabilitado desde aquí mismo (véase la *Figura 2.50*).

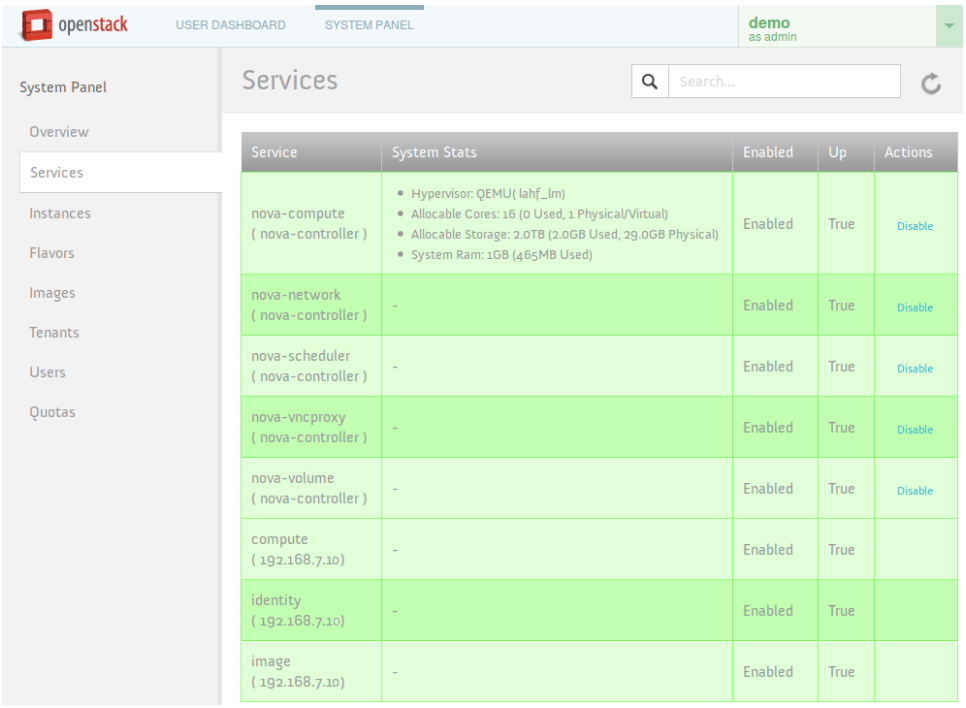


Figura 2.50. Servicios de la Nube

Instances

Desde *Instances* puede ver un listado de las instancias actuales desplegadas en el sistema (véase la *Figura 2.51*). En este momento, al no haber desplegado aún ninguna instancia, aparece vacío. Para cada una de las instancias se muestra su identificador, nombre, grupo al que pertenece, imagen base sobre la que fue creada, tamaño en recursos de procesamiento, memoria y disco, las direcciones de red asignadas y su estado. Además se facilitan enlaces para llevar a cabo ciertas operaciones sobre las instancias como terminarlas, reiniciarlas, ver sus *logs*, conectarnos por VNC a su consola, editar su información o tomar un *snapshot*.

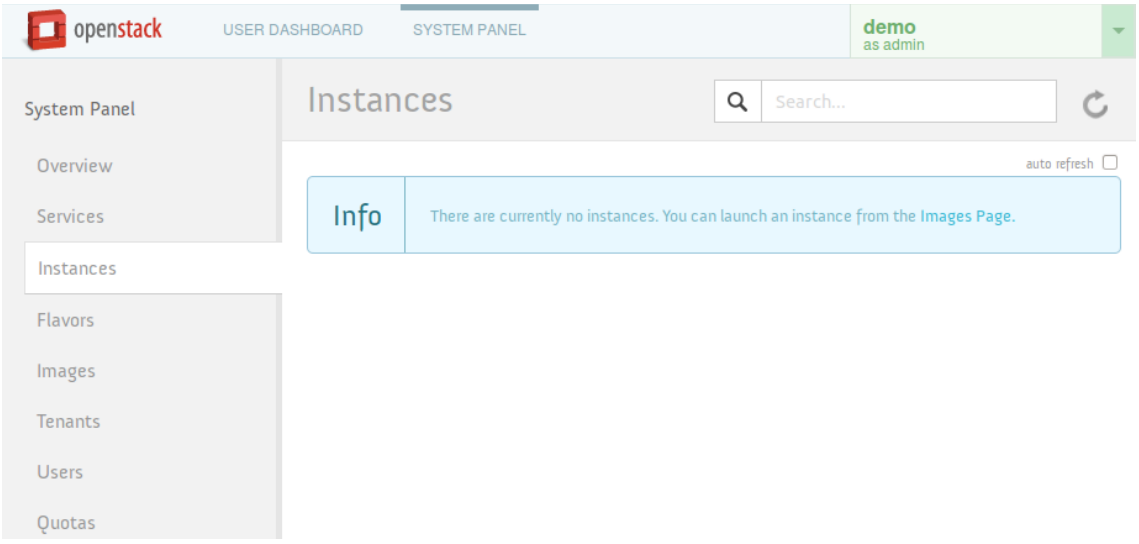


Figura 2.51. Lista de Instancias Nube

Flavors

En la categoría *Flavors* (véase la *Figura 2.52*) se recogen los diferentes tipos de instancias, en función de la cantidad de recursos de procesamiento, memoria y disco, que es posible instanciar. En el momento de instanciar una nueva máquina virtual debe elegir, además de la imagen patrón a utilizar como fichero de disco, el tipo de instancia de entre los mostrados en esta tabla. Puede añadir nuevos *flavors* o eliminar los existentes.

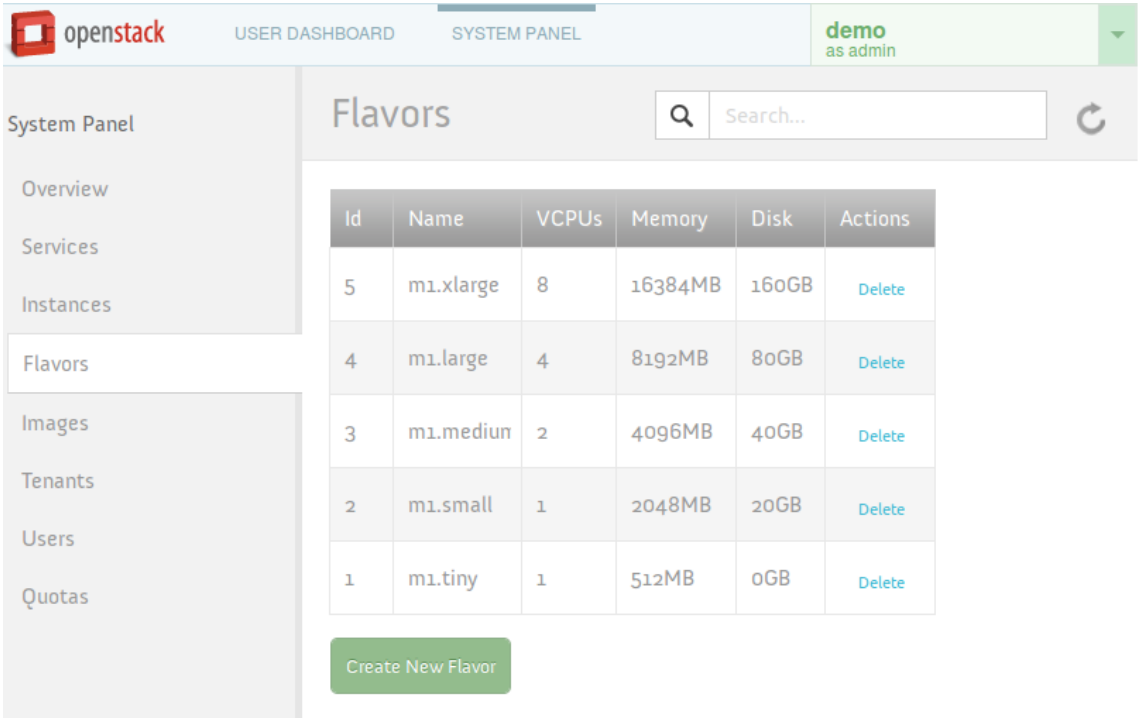


Figura 2.52. Tipos de Instancias

Images

Muestra un listado con las diferentes imágenes de disco disponibles en el sistema para ser instancias. Para cada una de ellas se recoge su identificador, su nombre, tamaño que ocupa en disco, si es de uso público, cuándo fue creada, cuándo ha tenido la última modificación sobre ella y su estado (véase la *Figura 2.53*). Pueden ser editadas o eliminadas. Si nos fijamos, aparece la imagen *Ubuntu 10.04.2* que añadimos en el apartado anterior de configuración de *OpenStack* mediante el *script pububuntu1004.sh*. Además permite ejecutar una de las operaciones más importantes en la nube: **la puesta en marcha de instancias para una imagen de disco dada**.

Figura 2.53. Lista de Imágenes

Tenants

Desde la categoría *Tenants* (véase la *Figura 2.54*) se administra los diferentes proyectos a los que es posible asignar determinados recursos de la infraestructura. Para cada proyecto o *tenant* puede consultar su identificador, descripción, si se encuentra habilitado o no, así como realizar diferentes acciones sobre el mismo: eliminar, editar, ver sus miembros, o modificar las cuotas asignadas para los recursos. Como es lógico, también puede añadir nuevos *tenants* si así lo necesita.

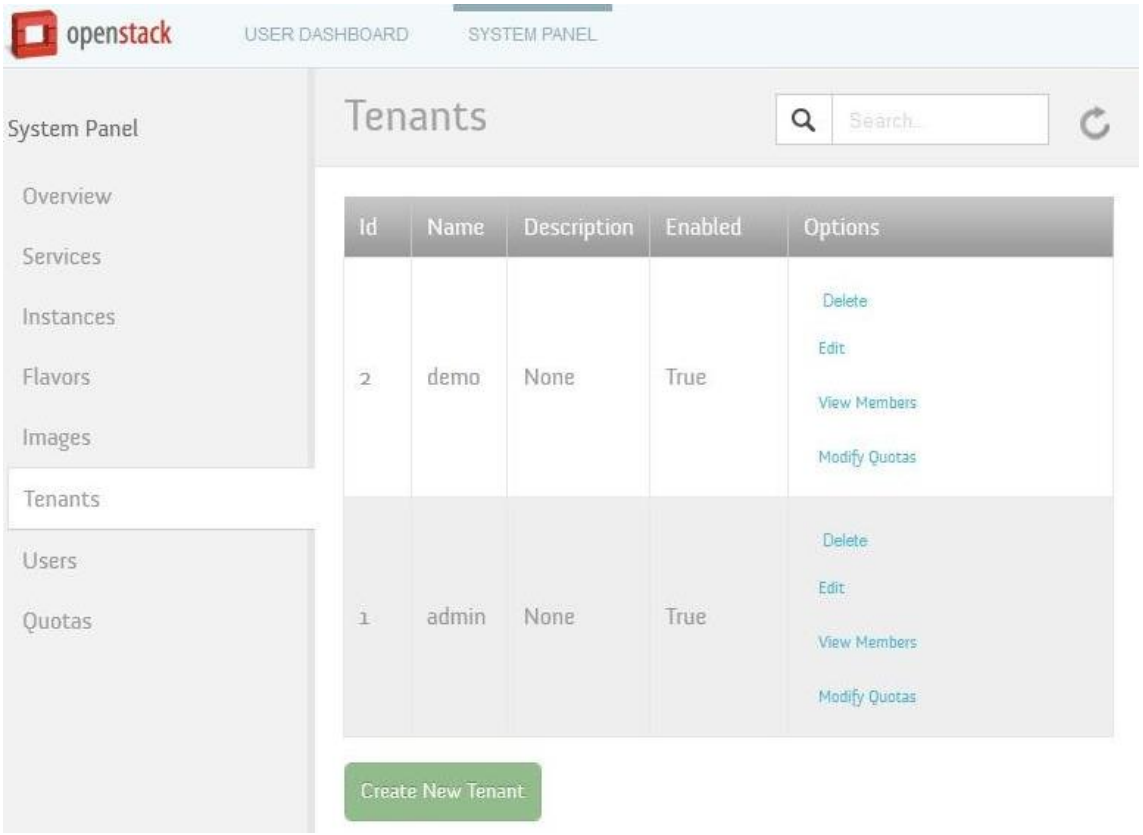


Figura 2.54. Administrar Recursos en Proyectos

Inicialmente, dispone de dos *tenats* creados, uno para el usuario *'demo'* y otro para *'admin'*. Si se desea puede reservar recursos para cada uno de ellos específicamente o dejar libres todos los recursos, en nuestro caso hemos elegido esta última opción. También puede incluir más usuarios a cualquier *tenant* o crear nuevos *tenants*.

Users

Muestra los usuarios registrados en el sistema (véase la *Figura 2.55*). Como se ha comentado anteriormente, de manera inicial encontramos los usuarios *admin* y *demo*. Puede crear nuevos usuarios, editar los existentes, deshabilitarlos, eliminarlos… y consultar su estado (identificador, nombre, correo electrónico, *tenant* predeterminado).

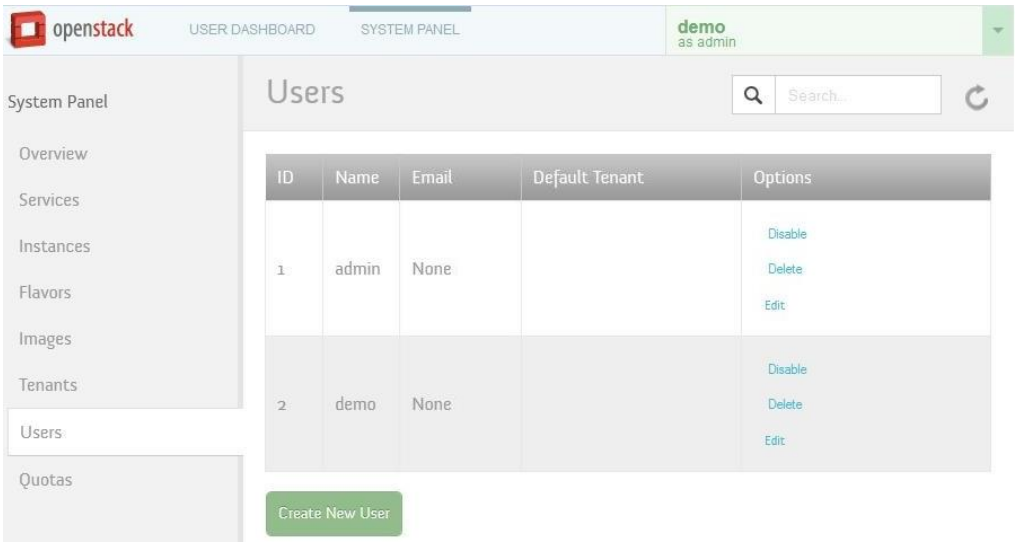


Figura 2.55. Usuarios de la Nube

En la *Figura 2.55* vemos los dos usuarios que ya conocemos y puede modificar su información, deshabilitarlos o eliminarlos. Además puede crear nuevos usuarios si es necesario.

Quotas

Finalmente para el usuario *admin* en el panel de control *System Panel* aparece la categoría *Quotas*. Como puede observar en la *Figura 2.56,* desde aquí puede consultar las cuotas aplicadas de forma predeterminada a cada uno de los diferentes recursos de la infraestructura, como por ejemplo *gigabytes*, memoria RAM, número de direcciones de red, número máximo de instancias, volúmenes, *cores*, etc.

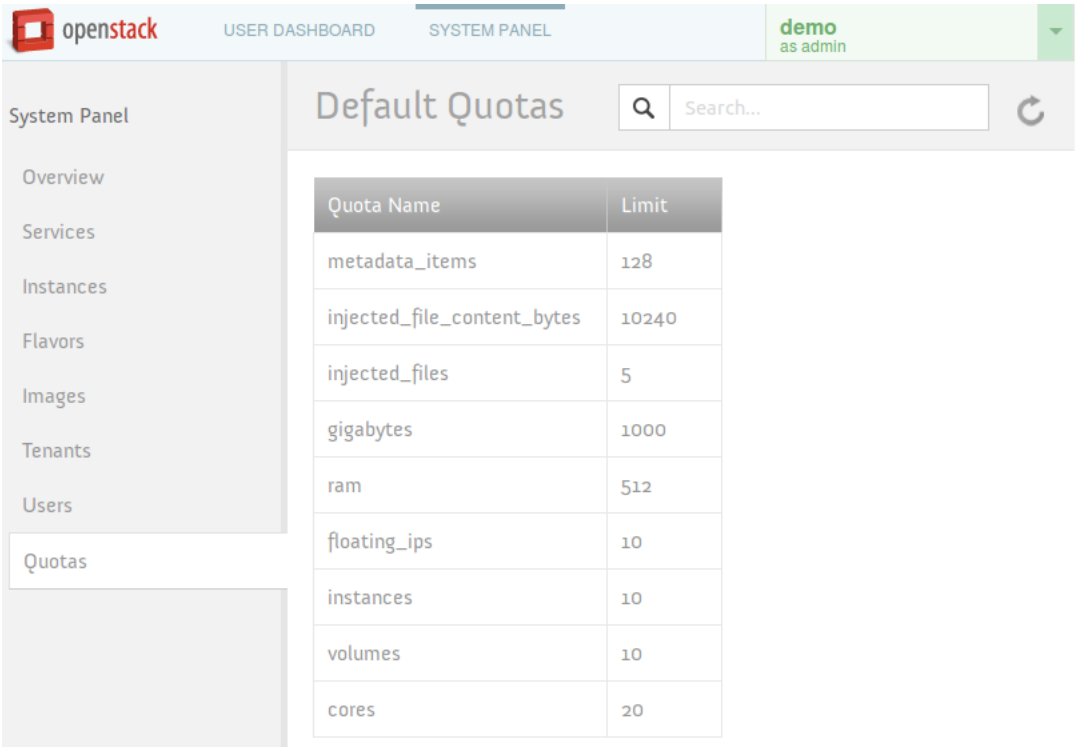


Figura 2.56. Cuotas de los Recursos de la Nube

2.5.2. *User Dashboard*

Este segundo panel de la interfaz es el referente al usuario y su interacción con las imágenes e instancias virtuales de la nube. Vemos una a una todas las categorías del mismo.

Overwiew

La información presentada es relativa a instancias e imágenes (véase la *Figura 2.57*) y dado que no tenemos ninguna en ejecución, en la pantalla de resumen nos muestra un mensaje indicando este hecho y que puede poner en marcha instancias desde el apartado "*Images*".

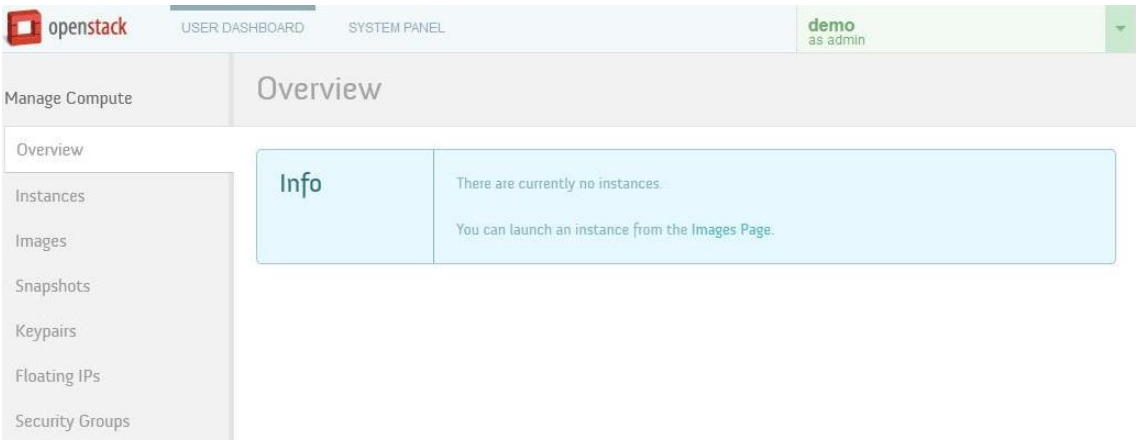


Figura 2.57. Resumen Panel de Usuario

Sobre este segundo panel vamos a realizar una breve presentación de las diferentes categorías que incluye. Más adelante profundizaremos en las mismas durante el proceso de despliegue de una instancia de máquina virtual. La información que puede consultar en cada una de las categorías o pestañas que incluye es la siguiente:

- En ***Instances*** aparecen las máquinas virtuales que hay en el sistema con algunos datos de ellas y el estado en el que se encuentran (ejecución, terminando, en construcción…). Además algunas acciones que se pueden ejecutar directamente desde la instancia, como son *terminarla, reiniciarla, ver el archivo 'log'*, etc.
- En ***Images*** vemos todas las imágenes de máquinas virtuales de las que disponemos para ser ejecutadas. En nuestro caso, aparece la de *Ubuntu*¸ ya creada. Desde aquí puede poner en marcha nuevas instancias virtuales pulsando sobre el botón *'Launch'*.
- En ***Snapshots*** encontramos imágenes tomadas del estado de instancias en ejecución (véase la *Figura 2.58*). En esta categoría son listados los diferentes *snapshots* que hayamos realizado de nuestras máquinas virtuales.

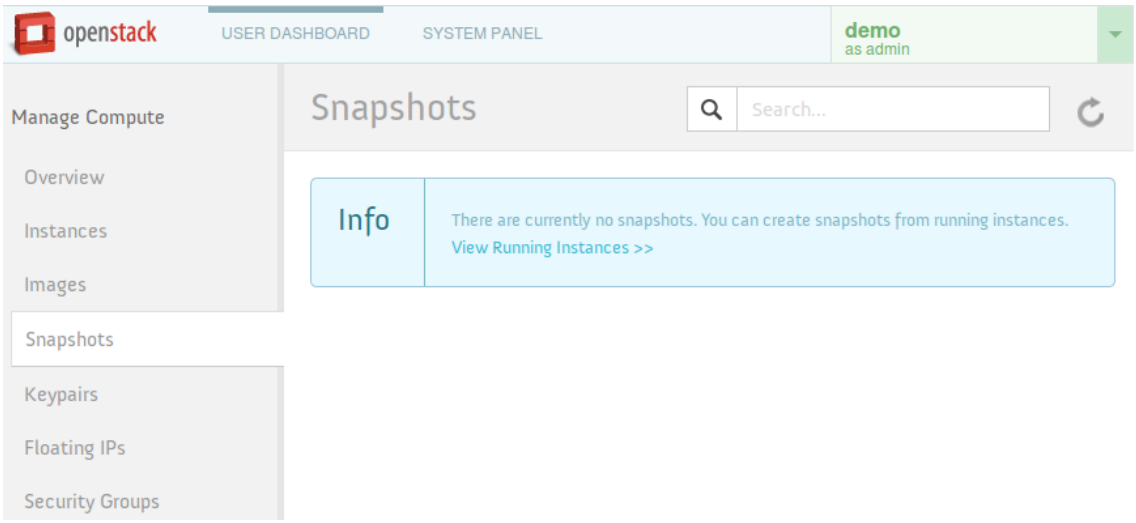


Figura 2.58. Lista de Capturas de Instancias

- En ***Keypairs*** se almacenan los certificados de conexión que nos permiten conectarnos a las instancias. Permite crear, importar y administrar los pares de claves que utiliza el usuario para la conexión con las instancias desplegadas. Los pares de claves contienen las credenciales de acceso a las instancias por lo que su uso y mantenimiento es de gran importancia. Como comprobaremos después, es obligatoria la selección de un par de claves en el proceso de puesta en marcha de una nueva instancia en la nube.
- En ***Floating IPs*** puede generar IPs públicas para asociarlas a las instancias en ejecución las cuales ya disponen de IPs privadas y así poder conectarnos a ellas.
- En ***Security Groups*** se incluyen los diferentes *grupos de seguridad* o grupos de reglas de seguridad que se aplican a las instancias de máquinas virtuales(véase la *Figura 2.59*). Puede añadir nuevos grupos o editar y eliminar los existentes, así como las reglas que los definen.

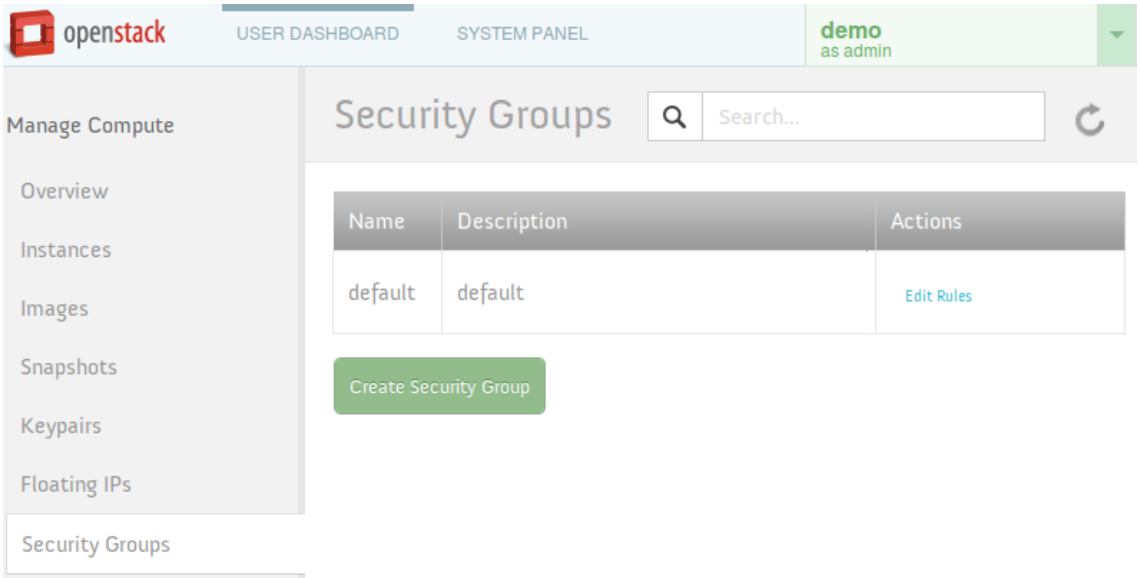


Figura 2.59. Grupos de Seguridad

Como ya hemos comentado antes, no se han explicado en profundidad estas secciones, ya que en el siguiente apartado, van a ser utilizadas y por tanto vamos a conocer a fondo su funcionalidad. Además la mayoría de ellas no contienen información, debido a que aún no se ha creado ninguna instancia, o par de claves o dirección IP, etc.

2.6. Desplegando Instancias

Para desplegar instancias virtuales de imágenes que tenemos almacenadas en la nube, hay que seguir una serie de pasos previos que explicamos a continuación.

Lo primero es generar un certificado que permita la conexión SSH, a través de un cliente SSH, con la instancia virtual desplegada. Para ello, accedemos al panel *"Dashboard"* y vamos al apartado *"Keypairs"* (véase la *Figura 2.60*).

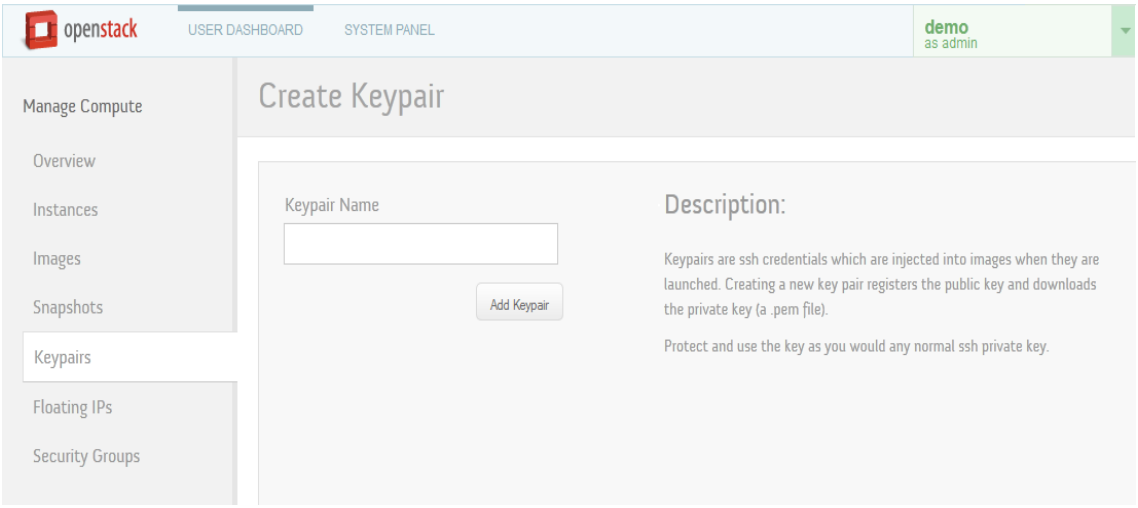


Figura 2.60. Crear *Keypair*

Damos un nombre al certificado y pulse sobre *"Add Keypair"* (véase la *Figura 2.61*). Acto seguido, nos dará la opción de almacenar el certificado en nuestra computadora. Es necesario guardarlo en algún lugar que recordemos, ya que posteriormente lo necesitaremos para poder conectarnos a las instancias.

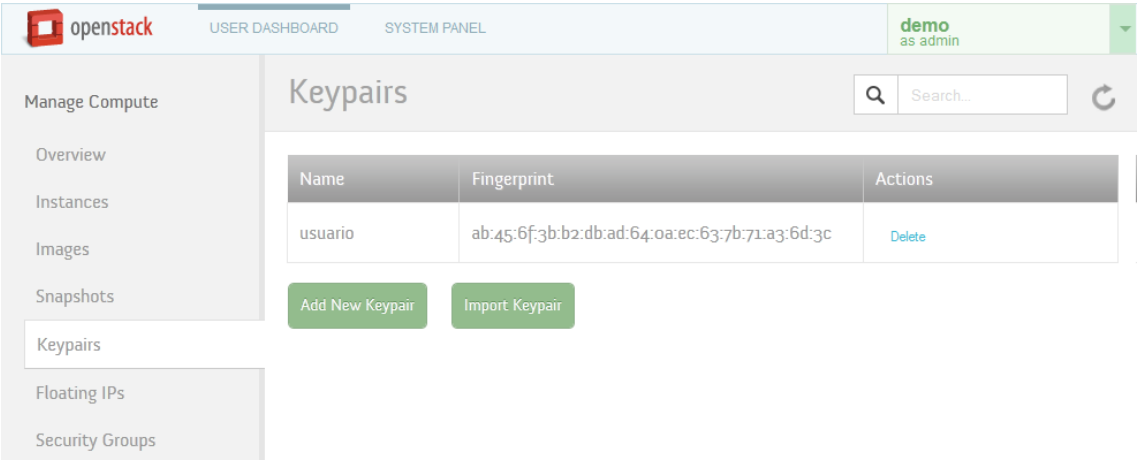


Figura 2.61. Lista *Keypairs*

Una vez guardado, ya tenemos el certificado creado. Para poder continuar con el proceso previo a ejecutar una instancia, debemos acceder a la pestaña *"Images"* (véase la *Figura 2.62*).

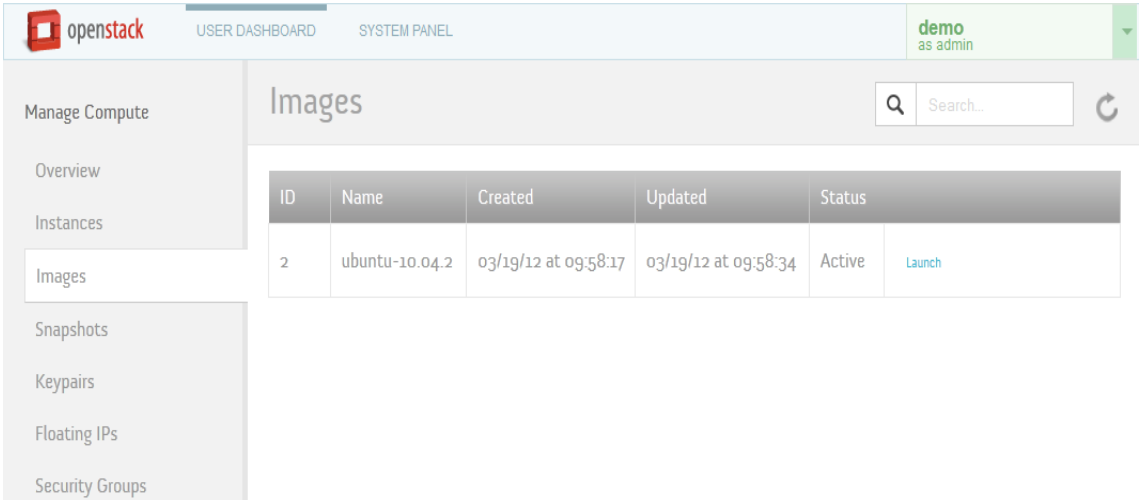


Figura 2.62. Lista de Imágenes Usuario

Vemos la imagen de *Ubuntu* que creamos y pulsamos el botón *"Launch"* para poner en marcha una instancia de dicha imagen.

El siguiente paso puede verse en la *Figura 2.*63. Indique el nombre de la instancia, una descripción (opcional), el tipo de máquina virtual que deseamos crear (éstos son los que se describieron en el apartado *Flavors* del *System Panel*), el certificado y el grupo de seguridad. Hay varios tipos de máquinas virtuales, difieren en la cantidad de recursos que consumen (procesadores, RAM y disco duro). Al especificar el certificado creado con anterioridad, podremos conectarnos correctamente a la instancia, porque al hacerlo se nos pedirá ese certificado indicado. Al coincidir el par de claves, la conexión será efectiva.

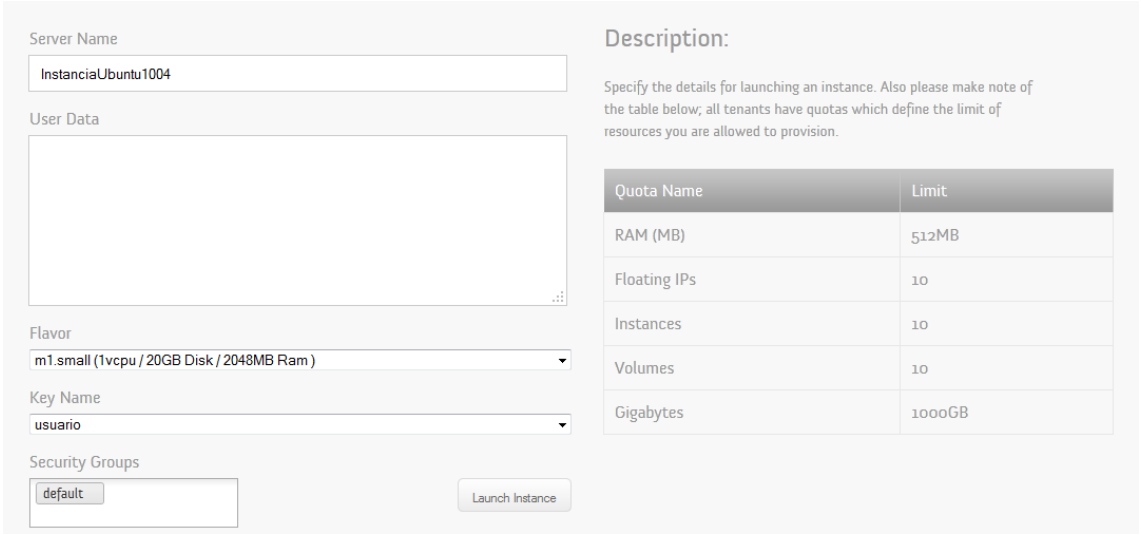


Figura 2.63. Desplegar una Imagen

Tras cumplimentar todos los campos necesarios y pulsar el botón "*Launch instance*", el proceso tiene lugar. Una vez terminado, abrimos la sección "*Instances*" y comprobamos que se ha creado correctamente (veáse *Figura 2.64*).

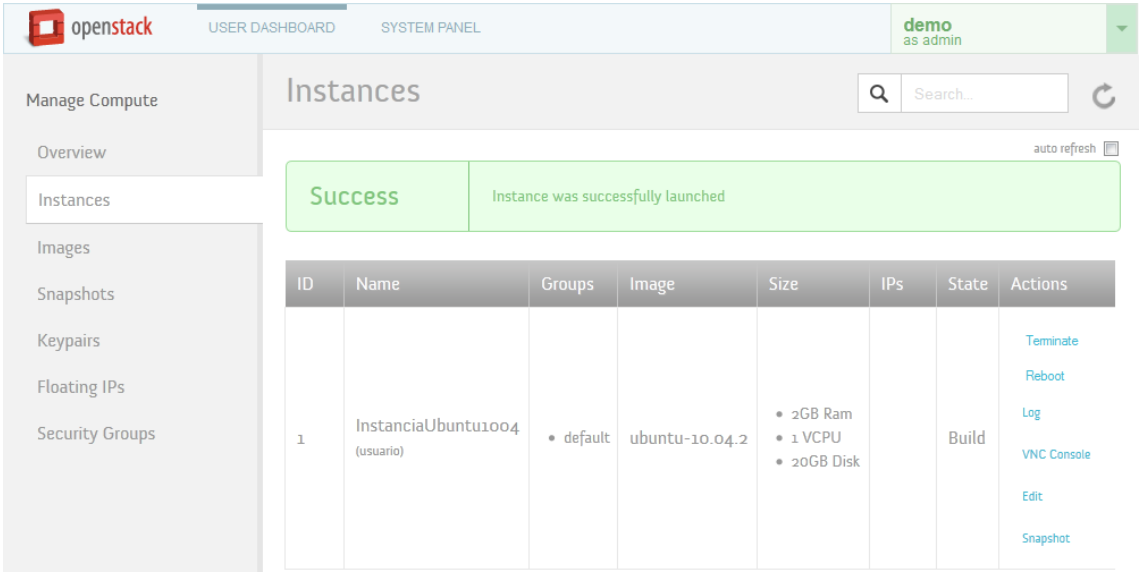


Figura 2.64. Lista de Instancias Usuario

Aparece el mensaje "*Success*, que indica que la instancia ha sido puesta en marcha correctamente. Compruebe que la máquina virtual está en construcción y que aún no se le ha asignado ninguna dirección IP. Observamos que se han asignado los recursos que indicamos en el paso anterior y que la instancia proviene de la imagen de *Ubuntu*.

Mientras que finaliza la construcción de la instancia, puede ir asignándole una IP pública, puesto que la dirección que se le asigna en la construcción es privada dentro del rango que proporcionamos en la configuración de la nube (10.0.0.x). Para ello, vamos a la sección *"Floating IPs"* (veáse *Figura 2.65*).

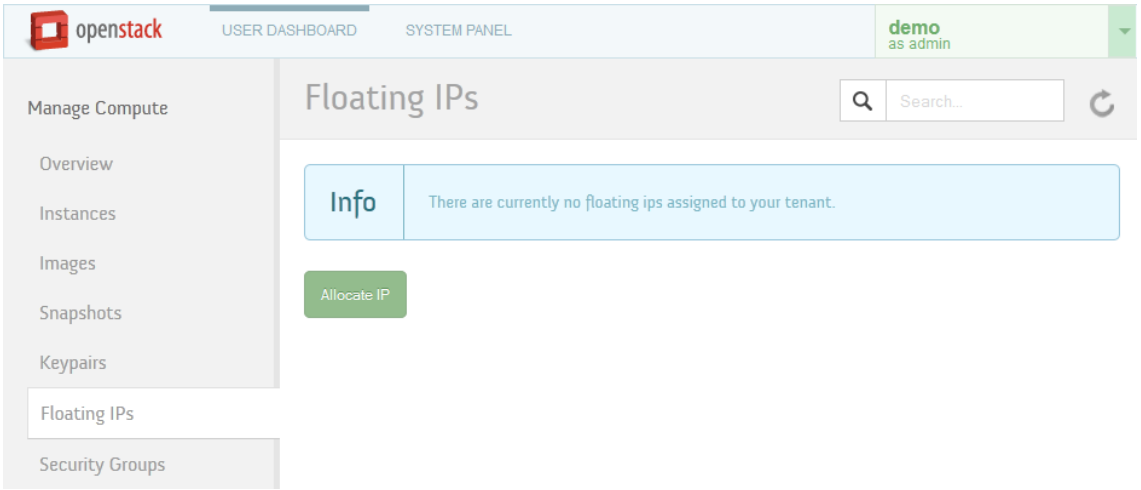


Figura 2.65. Lista de IPs

Observamos que actualmente no hay ninguna IP disponible para asignar a una instancia. Pulse sobre el botón *"Allocate IP"* (veáse *Figura 2.66*) y genere una IP pública dentro del rango que aportamos en la configuración de la nube. En este caso indique una dirección (192.165.7.65) y a partir de ella son desplegadas nuevas direcciones.

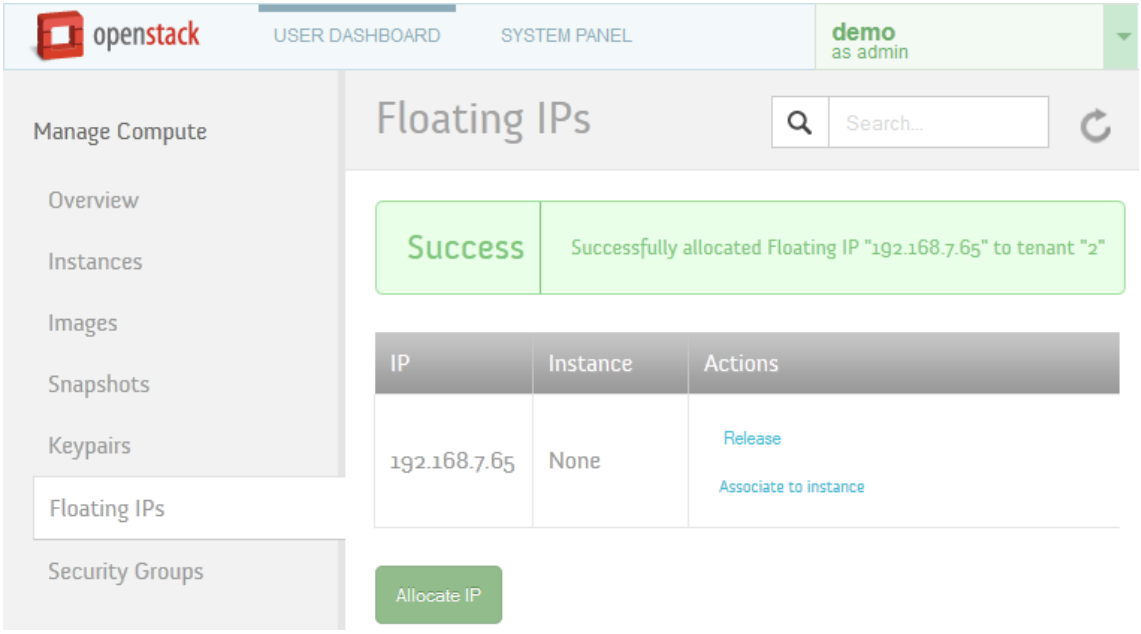


Figura 2.66. Crear una Nueva IP

Creada la dirección IP vemos que genera direcciones comenzando por la fijada al desplegar la nube. Teniendo ya una dirección disponible, el siguiente paso es asociarla a una instancia. Lo hacemos gracias al enlace disponible para ello "*Associate IP*" (veáse *Figura 2.67*).

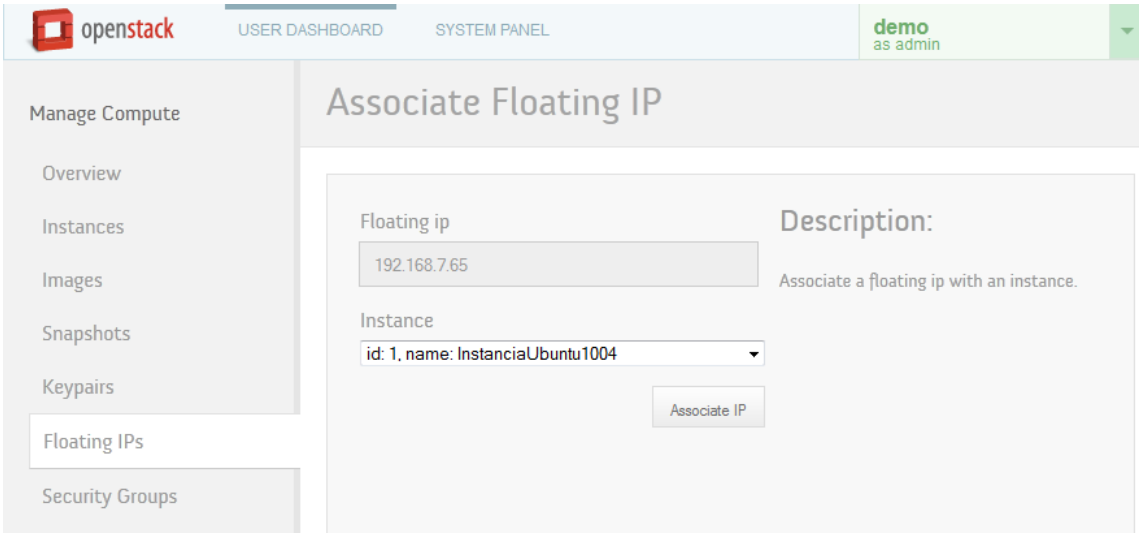


Figura 2.67. Asociar IP a Instancia

Elegimos la instancia a la que queremos asignar la dirección IP, en nuestro caso como solo tenemos una, ya aparece directamente seleccionada en la barra de selección (veáse *Figura 2.68*). Asociamos la dirección a la instancia.

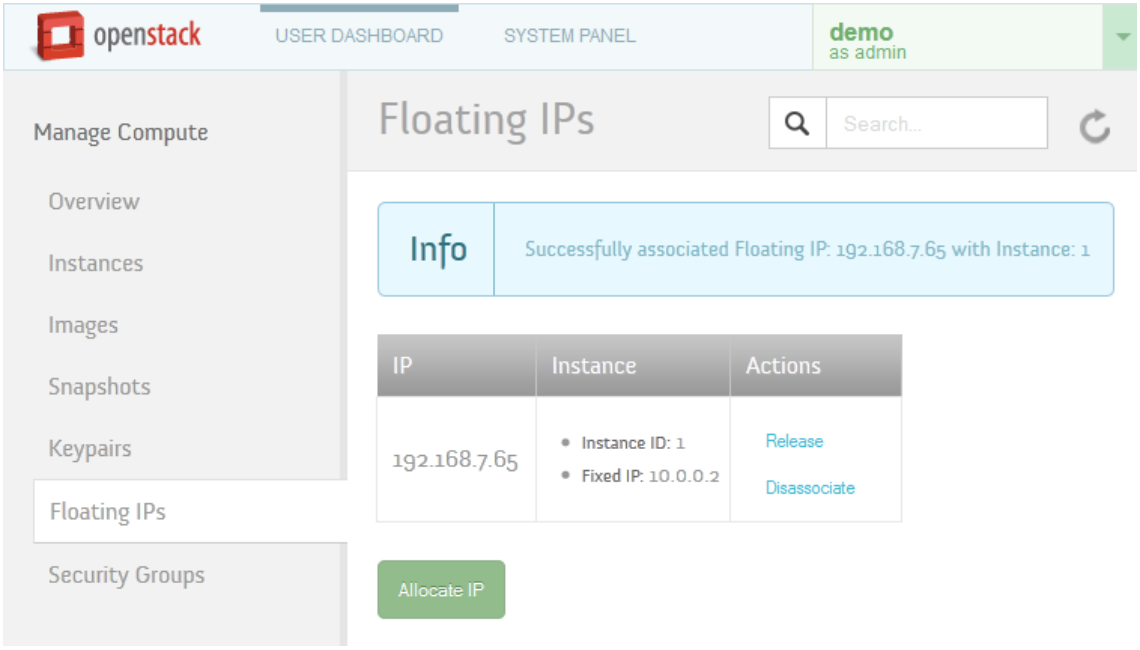


Figura 2.68. Lista de IPs Asociadas

Comprobamos como aparece ahora la misma dirección asociada a una instancia con el identificador *ID:1* y una dirección privada a la que se asocia esta pública. Esa dirección es la que se ha asignado a la instancia como dirección privada de la red interna de la nube. Disponemos además de dos enlaces más, uno para romper la asociación (*"Disassociate"*) y otro para liberar la dirección IP de la instancia (*"Release"*).

A continuación, acuda a la pestaña *Instances* (veáse *Figura 2.69*) y compruebe que la instancia ya se encuentra activa, con sus dos direcciones asociadas.

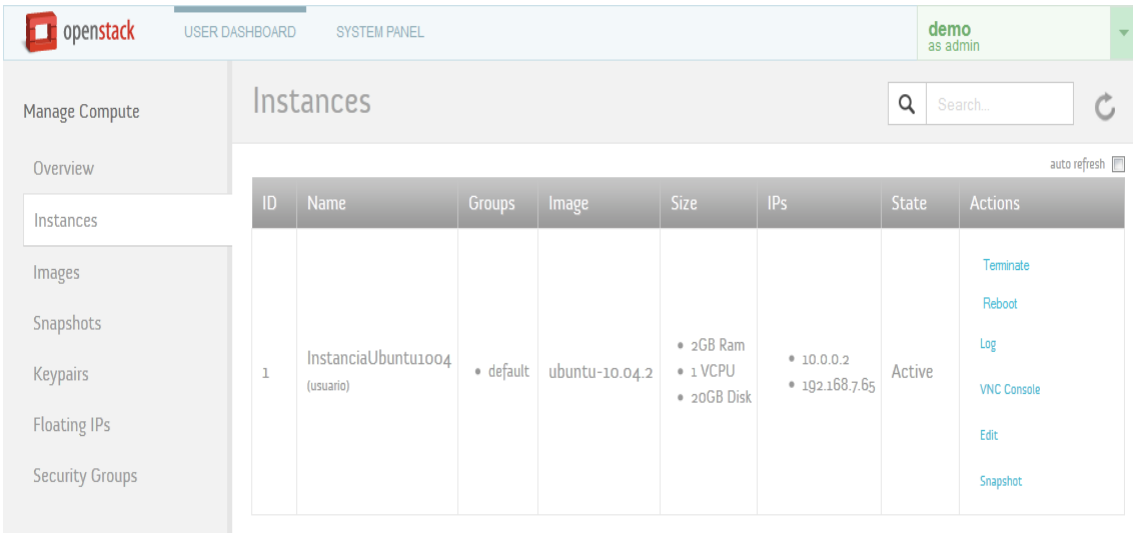


Figura 2.69. Lista de Instancias con IP asociada

Al encontrarse activa, la instancia ya está lista para ejecutarse. Se le ha asignado automáticamente una dirección IP privada para la red interna de la nube y otra pública asignada por nosotros para poder conectarnos a ella desde el exterior del *Cloud*. En la última columna de la instancia, puede apreciar una serie de acciones, son las siguientes:

- *Terminate:* finaliza la instancia.
- *Reboot:* reinicia la instancia.
- *Log:* abre el fichero "Log" donde se recogen los distintos eventos de la máquina (veáse *Figura 2.70*).

```
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Linux version 2.6.32-38-server (buildd@allspice) (gcc version 4.4.3 (Ubuntu 4.4.3-4ubuntu5)
[    0.000000] Command line: root=/dev/vda console=ttyS0
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Centaur CentaurHauls
[    0.000000] BIOS-provided physical RAM map:
```

Figura 2.70. Fichero Log

- *VNC Console:* abre una consola para acceder a la instancia (veáse *Figura 2.71*).

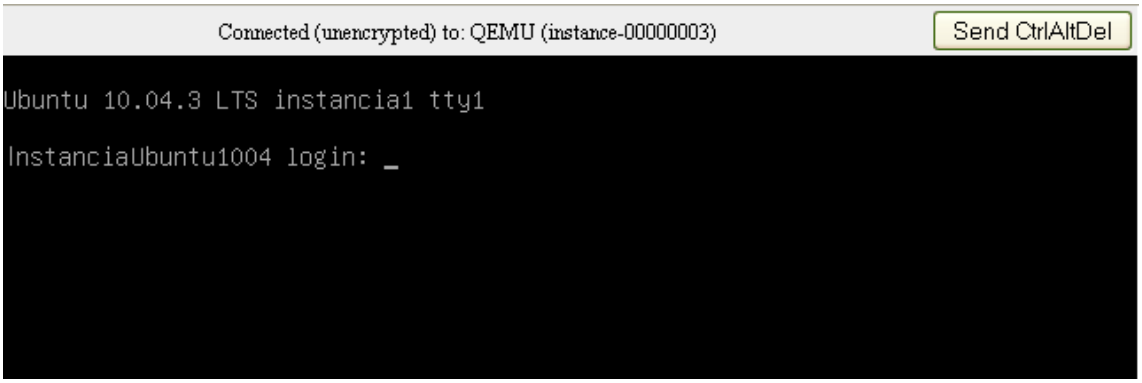


Figura 2.71. Consola VNC

- *Edit:* permite cambiar alguno de los campos que indicamos al crear la instancia.
- *Snapshot:* sirve para crear una captura del estado de la instancia.

El siguiente paso es preparar nuestro ordenador para poder conectarnos a la nube. Lo primero que necesitamos es tener instalado un cliente SSH. En nuestro caso, hemos elegido "*Bitvise Tunnelier"*, aunque es válido cualquier cliente SSH. A continuación vemos su utilización en este proceso. Lo primero es descargarlo de la Web oficial (http://dl.bitvise.com/BvSshClient-Inst.exe) e instalarlo. En la *Figura 2.72* vemos la ventana principal del programa.

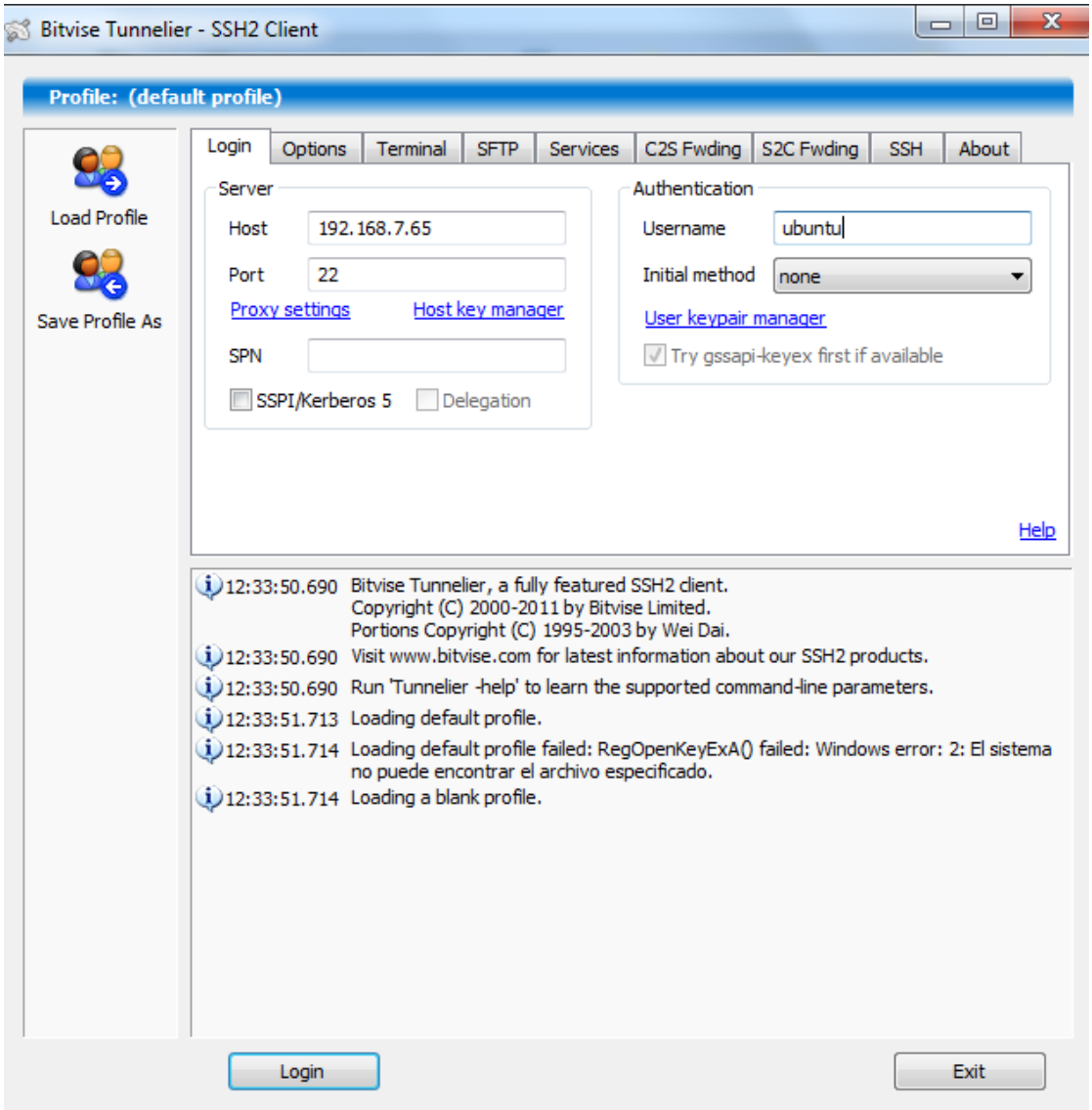


Figura 2.72. ***Bitvise Tunnelier***

Introducimos los valores en los campos que indican la dirección del *host* al que queremos conectarnos, el puerto (22 SSH por defecto) y vamos a la Sección de autenticación, en la que es necesario escribir el nombre de usuario para la conexión. Configuramos la conexión pulsando sobre sobre *"User keypair manager",* que es un gestor de conexión que establece el enlace entre nuestro equipo y el host de la instancia virtual que creamos utilizando el certificado de par de claves de conexión (veáse *Figura 2.73*).

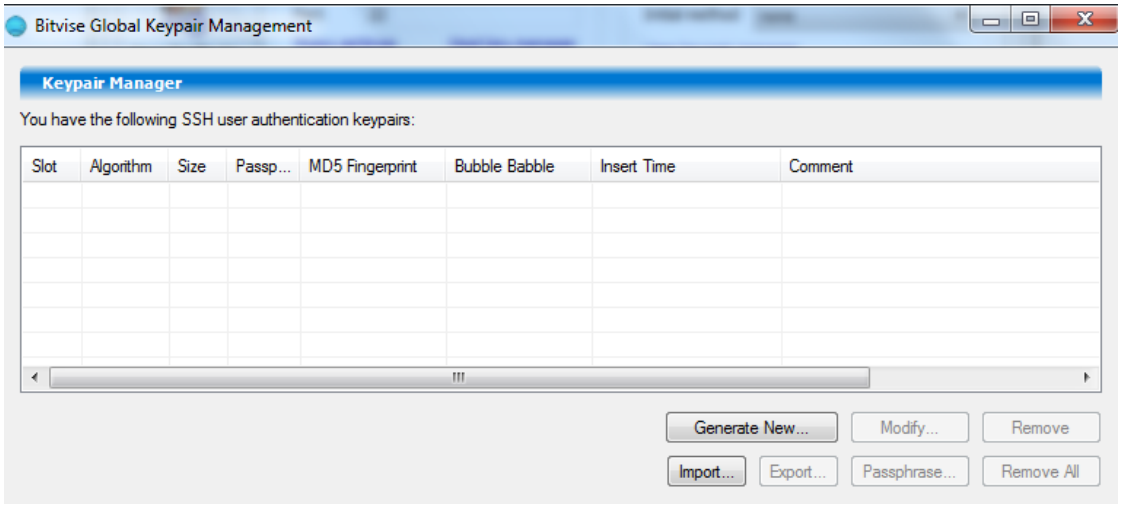


Figura 2.73. Asociar *Keypairs*

Generamos una nueva conexión; para ello, importamos el certificado creado. Se abre el explorador de archivos y así puede proporcionar el certificado que descargamos al crear el par de claves.

Una vez seleccionado, aparece un resumen (veáse *Figura 2.74*) que contiene la información del certificado y pulse el botón "*Import*" para realizar la importación.

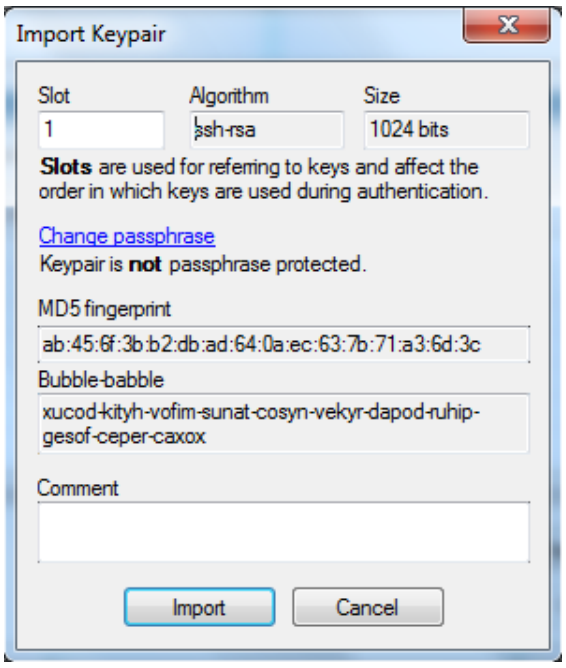


Figura 2.74. Importar *Keypair*

Como es posible apreciar en la *Figura 2.75*, disponemos del certificado en el gestor de conexión, por tanto generamos una conexión nueva.

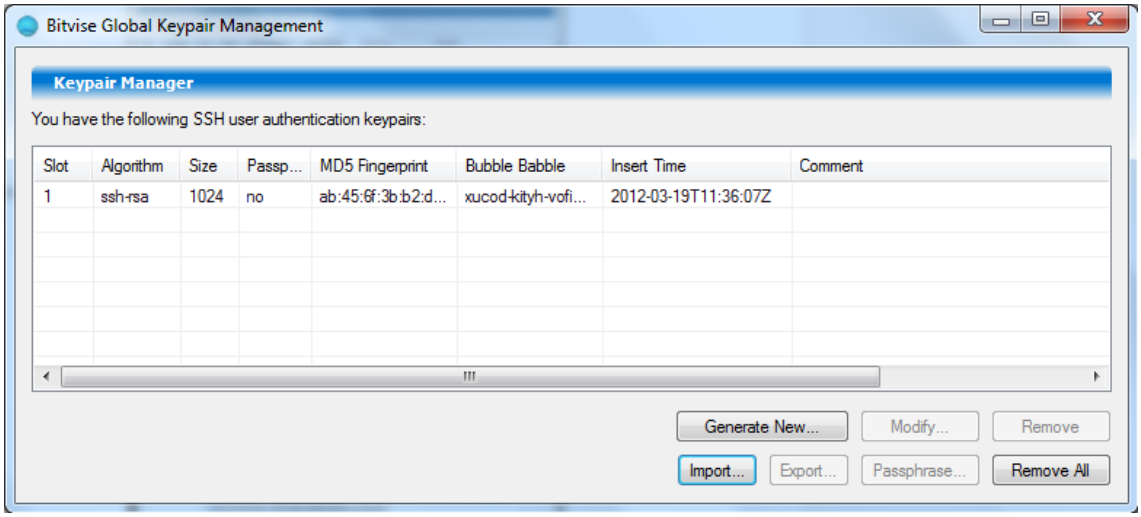


Figura 2.75. *Keypair* Asociado

El gestor de conexión toma los datos de conexión previamente configurados, el par de claves importado y a partir de ahí establece la conexión. Recomendamos pulsar sobre *"Accept and Save"* para que así se guarden los datos de conexión para posteriores ocasiones (veáse *Figura 2.76*).

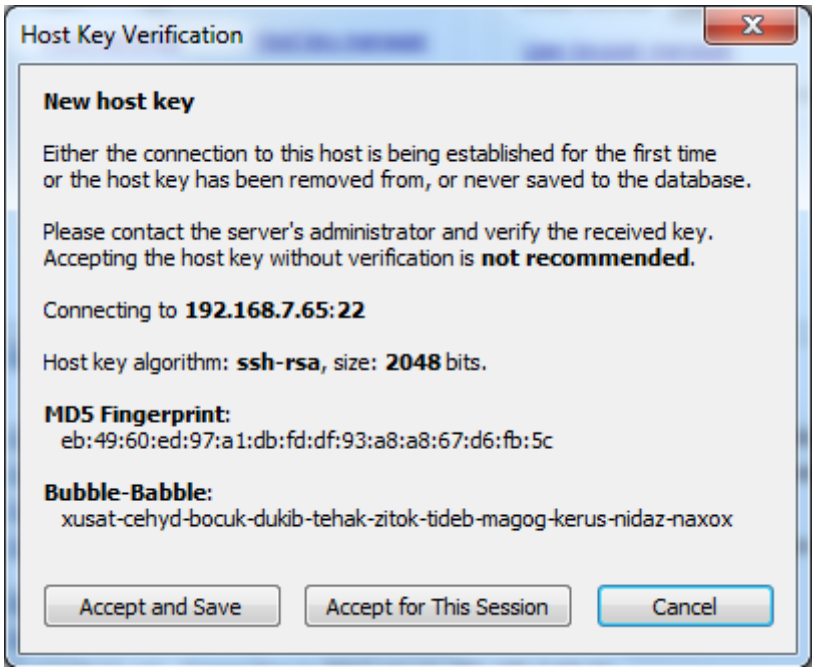


Figura 2.76. Nueva Conexión

Como vemos en la *Figura 2.77* pulsamos el botón OK para aceptar el establecimiento de la conexión.

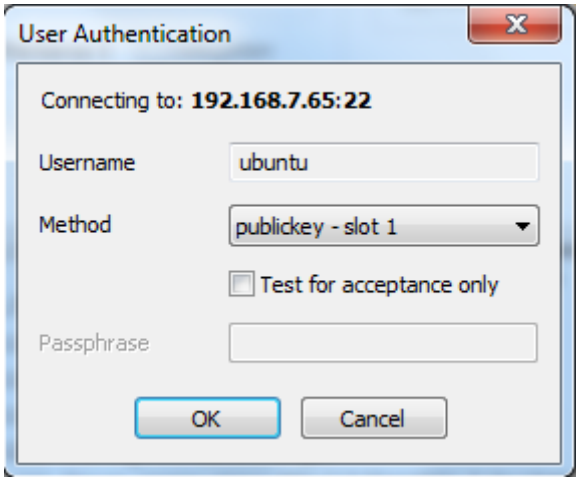


Figura 2.77. Autenticación Nueva Conexión

Vemos en la *Figura 2.78* como se completa la conexión. Si desea acceder a la consola de la máquina, puede abrir un terminal con *"Open New Terminal Console"*.

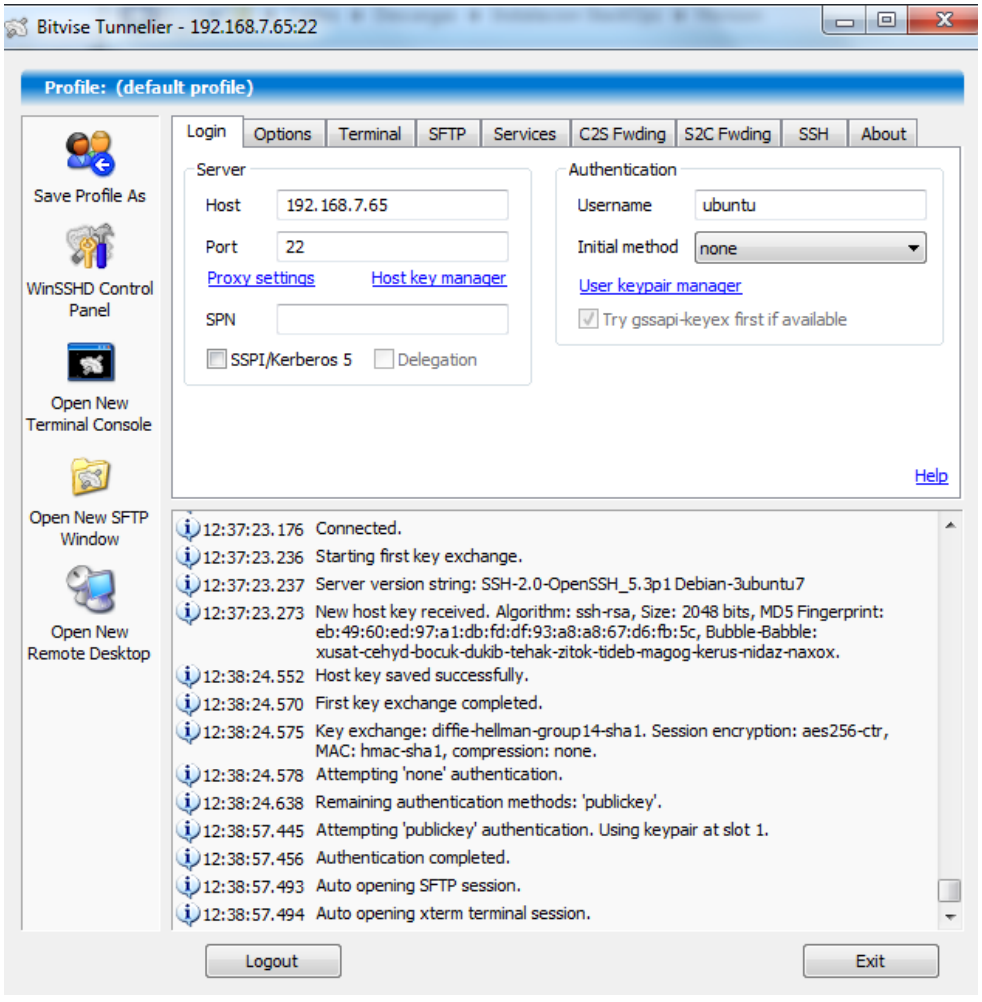


Figura 2.78. Conexión Establecida

Con esto, hemos concluido la puesta en marcha del *Cloud*. A modo de resumen, comenzamos conociendo los recursos con los que contamos para formar la nube. En un segundo paso, instalamos en cada uno de los servidores el sistema operativo de base que como ya comentamos es una distribución llamada *StackOps* basada en *Ubuntu 10.04*. Continuamos presentando las distintas arquitecturas *Cloud Computing* que podíamos conformar, de las cuales elegimos *Dual Node*. El siguiente paso fue configurar esa arquitectura elegida y el Nodo Controlador, así como los Nodos de Cómputo. Para finalizar, presentamos la interfaz Web y ejecutamos una primera instancia.

En el siguiente capítulo realizaremos un estudio del entorno de computación distribuido *Apache Hadoop*. El objetivo es conocer a fondo todas sus características y estudiar si es viable y efectivo desplegar el entorno dentro del paradigma de computación en la nube. Queremos conocer qué tipo de aplicaciones son las que mejor se adaptan a este marco de trabajo distribuido. También es necesario saber sí puede ser configurado sobre instancias virtuales. Si esto fuera posible, podríamos configurar un entorno distribuido de computación dentro del marco *Cloud Computing* que acabamos de finalizar, obteniendo amplios beneficios en la configuración del entorno y sobre todo a la hora de aumentar o disminuir el tamaño del clúster.

3. Estudio del Entorno *Apache Hadoop*

El principal objetivo de este capítulo es conocer a fondo el **entorno de trabajo *Apache Hadoop*** y todos sus aspectos fundamentales. Vamos a comenzar analizando sus características principales y presentando algo de su historia y de cómo surge.

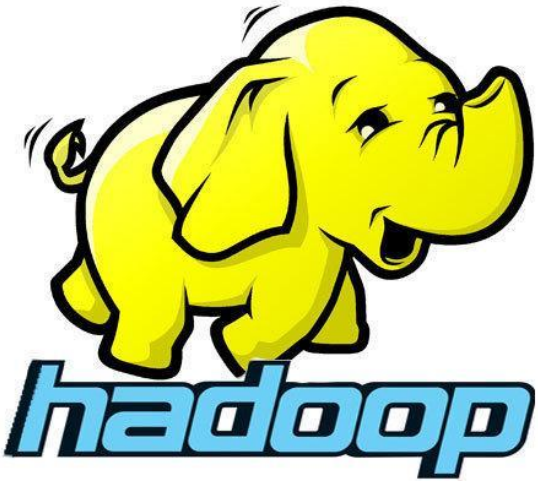


Figura 3.1. Logo *Apache Hadoop*

Es muy importante entender el funcionamiento de su **sistema de ficheros distribuido (HDFS)**. Para ello, es necesario estudiar como base los *Sistemas de Ficheros Distribuidos* para posteriormente profundizar en el que atañe al proyecto. Así lo hacemos, explicamos qué es un SFD, cómo surgen y hacemos una breve enumeración de los principales SFDs.

Otro aspecto destacado a conocer son los diferentes elementos que conforman un sistema de trabajo *Apache Hadoop*, modos de **configuración de clúster** (según la cantidad de nodos) y las principales tareas del mismo.

Para finalizar, se va a valorar su **relación** con los entornos de desarrollo ***Cloud Computing***. La ventaja que produce la computación en la nube sobre este entorno de trabajo es que hace posible desplegar tantas instancias *Hadoop* como sea requerido (siempre que las limitaciones de nuestra nube lo permitan) para formar distintos clústeres y así poder realizar mayor cantidad de pruebas de una forma más sencilla que trabajando directamente sobre los servidores físicos.

El entorno de trabajo *Apache Hadoop* está diseñado para trabajar con aplicaciones basadas en el lenguaje de programación ***MapReduce***. Este lenguaje está basado en *Java* y junto con el

framework Hadoop conforman un sistema que produce un muy alto rendimiento sobre ciertos tipos de aplicaciones. Algunas de estas aplicaciones son comentadas posteriormente.

Figura 3.2. Logo *Apache Hadoop MapReduce*

3.1. *Apache Hadoop*

Es un *framework* inspirado en *Google MapReduce* y *Google File System* para el desarrollo de aplicaciones distribuidas. Es de licencia libre, un proyecto de *Apache* que está siendo desarrollado por multitud de colaboradores. El que más ha aportado al proyecto es *Yahoo!* que utiliza este entorno de trabajo de manera extendida en su negocio. Está basado en el lenguaje de programación *Java* y permite trabajar con miles de nodos de computación y procesar *petabytes* de datos.

Su creador fue *Doug Cutting* que le puso el nombre en honor a su elefante de juguete y fue desarrollado originalmente para un proyecto del motor de búsqueda de *Nutch*. En la F*igura 3.3* lo puede ver posando en una conferencia con su tan famoso peluche.

Figura 3.3. *Doug Cutting*

Un clúster típico *Hadoop* incluye un Nodo Maestro y múltiples Nodos Esclavos. El Nodo Maestro tiene una estructura distinta a los Nodos Esclavos. A continuación vemos de forma esquemática de qué elementos se compone cada uno de ellos:

- **Nodo Maestro (*Master Node*)**
    - *JobTracker o rastreador de trabajo.*
    - *TaskTracker o rastreador de tareas.*
    - *NameNode o nodo servidor de nombres.*
    - *DataNode o nodo de almacenamiento de datos.*
- **Nodo Esclavo (*Compute Node*)**
    - *TaskTracker o rastreador de tareas.*
    - *DataNode o nodo de almacenamiento de datos.*

Los distintos tipos de nodos que componen un clúster y sus elementos son explicados más adelante. Además *Hadoop* requiere tener instalados en todos los nodos del clúster *JRE 1.6 (Java Runtime Environment)* o superior, y SSH.

Para la programación efectiva del trabajo, cada nodo debe conocer y proporcionar su ubicación. Las aplicaciones *Hadoop MapReduce* distribuyen las tareas a los nodos donde se encuentran los datos, minimizando el tráfico en la red. Además, los datos se encuentran replicados conservando copias de los datos en racks distintos reduciendo así al mínimo el impacto de fallos hardware.

Uno de los aspectos más importantes y sorprendentes de *Apache Hadoop MapReduce* es su alta flexibilidad, en muchos aspectos. Por ejemplo, permite crear clústeres desde un solo nodo a miles de ellos. Lo único que se necesita es un nodo máster que gestione tanto las tareas del sistema como el sistema de ficheros, sabiendo que este nodo además puede actuar a su vez de esclavo. A partir de ahí se pueden ir añadiendo nodos hasta un número Ilimitado, con tan solo los recursos disponibles físicamente como límite. Permite la implantación sobre cualquier tipo de servidor, incluso sobre sistemas virtuales. Compatible con casi todos los sistemas operativos, también permite usar el sistema de ficheros distribuido propio (HDFS) o cualquier otro que funcione de manera distribuida como puede ser GFS (*Google File System*). El lenguaje *MapReduce* está principalmente basado en Java pero también existe basado en C++ o *Python* entre otros. Por tanto es muy reseñable su flexibilidad y escalabilidad.

Figura 3.4. Lenguajes MapReduce

Con todo esto, vemos que *Apache Hadoop MapReduce* es un entorno de computación ideal para conformar sistemas que trabajen sobre cualquier tipo de condiciones, bien sobre sistemas físicos o virtuales, y cuyas aplicaciones gestionen grandes cantidades de datos. Permite crear clústeres de tantos servidores o máquinas virtuales como se requiera, incluso pudiendo ir añadiendo nuevos nodos en cualquier momento. Es ideal para trabajar con grandes cantidades de datos y, gracias al SFD, toda esa información se encuentra almacenada entre todos los nodos del clúster de manera organizada manteniendo un alto nivel de seguridad tanto por pérdidas de datos en la redes del clúster como por caída de algún nodo.

3.2. Sistemas de Ficheros Distribuidos

Un Sistema de Ficheros Distribuido (SFD), es un sistema de almacenamiento de archivos que permite acceso mediante red a todos los miembros para poder compartir archivos, impresoras u otros dispositivos de manera concurrente. Los primeros SFDs aparecieron en los años setenta, pero no fue hasta 1985 cuando apareció un sistema ampliamente utilizado. Fue *Sun Microsystems* quien creó el sistema de archivos en red *Network File System* (NFS). Otros sistemas conocidos fueron *Serve Message Block* (SMB) también conocido como *CIFS* o *Andrew File System* (AFS) y *DCE Distributed File System* (DFS) que es un sistema de ficheros distribuido basado casi por completo en AFS.

Los SFDs tienen varios inconvenientes derivados de la seguridad de los datos que viajan por la red, cómo asegurar que el cliente que accede a los datos de un servidor es realmente quien dice ser y quién, por tanto, puede tener acceso a los mismos. Otros problemas que se pueden encontrar son referentes a latencias de envío en la red, cuellos de botella, sobrecarga en la red, fallos de la red o fallos en los servidores que dejen sin datos a los clientes.

Los SFDs se han ido adaptando a las necesidades de los sistemas de los que forman parte y por ello hay gran cantidad de ellos y de muy distintos tipos. Actualmente existen SFDs con autenticación de usuario, tolerantes a fallos, paralelos, encriptados, etc. Además aparecen nuevos sistemas, como los sistemas de Computación Paralela y los sistemas *Cloud Computing,* con nuevas necesidades.

Los sistemas *Cloud Computing* precisan de sistemas de ficheros específicos que soporten sus peculiares características, ya que estos sistemas trabajan con instancias virtuales que se crean, se eliminan, y cuyos datos en algunos casos se guardan y en otros no. Además puede que el cómputo de esos datos se realice en nodos distintos a donde están almacenados dichos datos. Por supuesto necesitan tolerancia a fallos pero, en este caso, pueden producirse tanto sobre los servidores físicos como sobre las instancias virtuales. Esto hace que sea una característica realmente compleja de controlar. En la mayoría de los casos, también precisan cifrado de los datos que viajan por la red entre los nodos que conforman el Cloud, o por lo menos cierta seguridad de los mismos y por supuesto capacidad para soportar ejecuciones paralelas y distribuidas sobre los datos.

En la *Tabla 3.1* puede ver algunos SFDs junto a algunas de sus características.

	Descripción	Características destacables	Versiones	Sistemas Soportados	Web Oficial
NFS	Creado por *Sun Microsystems* con el objetivo de hacer el sistema independiente a la máquina, el SO y el protocolo de transporte	*Kerberos* para autenticación	NFSv2 NFSv3 NFSv4	*UNIX, Solaris, AIX, HP-UX, Linux,MS-DOS, Microsoft Windows, Mac OS, OpenVMS, Novell NetWare e IBM AS/400*	
AFS	Desarrollado en la Universidad *Carnegie Mellon*. Su nombre proviene de *Andrew Carnegie* y *Andrew Mellon*.	Escalable, seguro, *Kerberos* para autenticación, listas de control de acceso, caché que mejora el rendimiento y acceso limitado en caso de caída del servidor o la red.	*Transarc* *OpenAFS* *Arla*	*AIX, Mac OS X, Darwin, Digital UNIX, HP-UX, Irix, Solaris, Linux, Microsoft Windows, FreeBSD y OpenBSD.*	http://grand.central.org http://www.openafs.org http://www.stacken.kth.se/project/arla/
SMB	Creado por IBM pero actualmente se usa una versión ampliamente modificada por *Microsoft* y renombrada como CIFS.	Protocolo de red que pertenece al modelo OSI usado para *Microsoft Windows* y DOS. Inicialmente no usaban TCP/IP para implementar la resolución de nombres	SMB SMB 2 (2.1 y 2.2) *Samba* (*Linux*)	*Windows, MS-DOS, GNU/Linux y UNIX*	http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=9492 http://devel.samba.org

CODA	Desarrollado por Universidad *Carnegie Mellon* desde 1987 bajo la dirección de M. *Satyanarayanan*. Desciende directamente de una antigua versión de AFS (AFS-2) y ofrece muchas características similares.	Tolerante a fallos, puede funcionar sin conexión, software libre, caché para mayor rendimiento, replicado de servidores, autenticación, cifrado y control de acceso y escalable	Actualmente CODA 6	*Linux y Windows XP*	http://coda.cs.cmu.edu/
GFS	Desarrollado por Google Inc, soporta toda su infraestructura informática de procesamiento de información en nube. Diseñado para proveer eficiencia y fiabilidad de acceso a datos usando sistemas masivos de clúster de procesamiento en paralelo.	Tolerante a fallos, preparado para computación paralela, capaz de almacenar grandes cantidades de archivos, concurrencia para múltiples clientes, gran ancho de banda (poca latencia)	La actual versión de Google File System tiene el nombre clave Colossus	Linux	
HDFS	Desarrollado por *Apache Software Foundation* para el entorno de trabajo *Hadoop*. Es de licencia libre y está inspirado en *Google MapReduce* y GFS.	Escrito en *Java*, es escalable y paralelo. Preparado para soportar la infraestructura *MapReduce*, clúster de datos. Replicado de datos a través de múltiples equipos. No proporciona alta disponibilidad.	HDFS 1.0	Multiplataforma	http://hadoop.apache.org/
EFS	Usado sobre NTFS permite el cifrado de archivos a nivel de sistema. Autenticación y control de acceso no	Utiliza claves públicas, privadas, simétricas y asimétricas.		*Windows* (2000, *XP*, *Server* 2003, *Vista*, 7 y	http://technet.microsoft.com/en-us/library/cc721923%28WS.

	son suficiente protección ya que una vez el sistema está en ejecución, cualquier persona con acceso físico a la computadora puede acceder a los datos.	Usuario y Contraseña, DPAPI *Master Key*, RSA, FEK (*File Encryption Key*), SYSKEY.		*Server* 2008) y *Linux*	10%29.aspx

Tabla 3.1. Sistemas de Ficheros Distribuidos

En la *Tabla 3.1* se han descrito brevemente los SFD más utilizados y que podríamos configurar en nuestro clúster *Apache Hadoop*. Hay que destacar que los dos sistemas de ficheros óptimos para desplegar en un entorno como el que queremos crear son:

- *Hadoop Distributed File System (HDFS).*
- *Google File System (GFS).*

Sin embargo, hemos de mencionar que el primero de ellos está creado expresamente para el entorno *Apache Hadoop*, aunque fue desarrollado basándose en las características del segundo. Por ello, son sistemas de ficheros realmente similares.

Para este proyecto, vamos a configurar HDFS como sistema de ficheros ya que así utilizamos un SFD sobradamente testeado y con un amplio soporte construido expresamente para este tipo de entornos de trabajo y que se adapta al mismo perfectamente. En cualquier caso, mencionar que *Apache Hadoop* es tan flexible, que permite casi cualquier SFD. Pasamos entonces a conocer un poco más en profundidad el sistema de ficheros distribuidos HDFS.

3.3. *Hadoop Distributed File System* (HDFS)

Hadoop es un *framework* Java de software libre creado por Doug Cutting y promovido por ***Apache Software Foundation***. Sus principales características son:

- Soporta aplicaciones que trabajen sobre grandes volúmenes de datos (*petabytes*) de manera distribuida.
- Diseñado para manejar aplicaciones que trabajen sobre miles de nodos.
- Alta flexiblidad y escalabilidad que permite crear clústeres con distintas topologías.

Este *framework* se inspiró de la creación de GFS (*Google File System*) y *MapReduce* de *Google*. GFS es un sistema de ficheros distribuido que posee alta tolerancia a fallos. Sin embargo, Apache decidió crear un SFD propio que, con el tiempo, se ha convertido en una de las señas de identidad de este marco de trabajo.

HDFS está diseñado para sistemas que realicen **mínimas escrituras y múltiples lecturas** de datos. Además se crea para instalarse sobre hardware de bajo coste, no precisa grandes servidores ni equipos especialmente potentes. Sin embargo, no se recomienda para sistemas que precisen bajas latencias de acceso a los datos, ni que posean múltiples archivos pequeños, ni que realicen múltiples escrituras.

HDFS sigue una arquitectura cliente/servidor basada en un único nodo servidor denominado ***NameNode*** que maneja el espacio de nombres (namespace) del sistema de ficheros y regula el acceso a los mismos, y varios clientes, llamados ***DataNodes*** que gestionan el almacenamiento de los nodos que contienen.

Internamente un **fichero** es **dividido en bloques** (de 64Mb por defecto) y almacenado en un conjunto de *DataNodes*. Todos los bloques son del mismo tamaño excepto el último. Los

bloques de un fichero son replicados, con el fin de dotar al sistema de tolerancia a fallos (que en casos con tantas máquinas es más una regla que una excepción). El tamaño de bloque y el número de réplicas es parametrizable para cada fichero. La optimización de la colocación de las réplicas en los diferentes *DataNodes* es uno de los factores que distingue a este sistema de ficheros.

En la *Figura 3.5* vemos un esquema representativo de la arquitectura del HDFS.

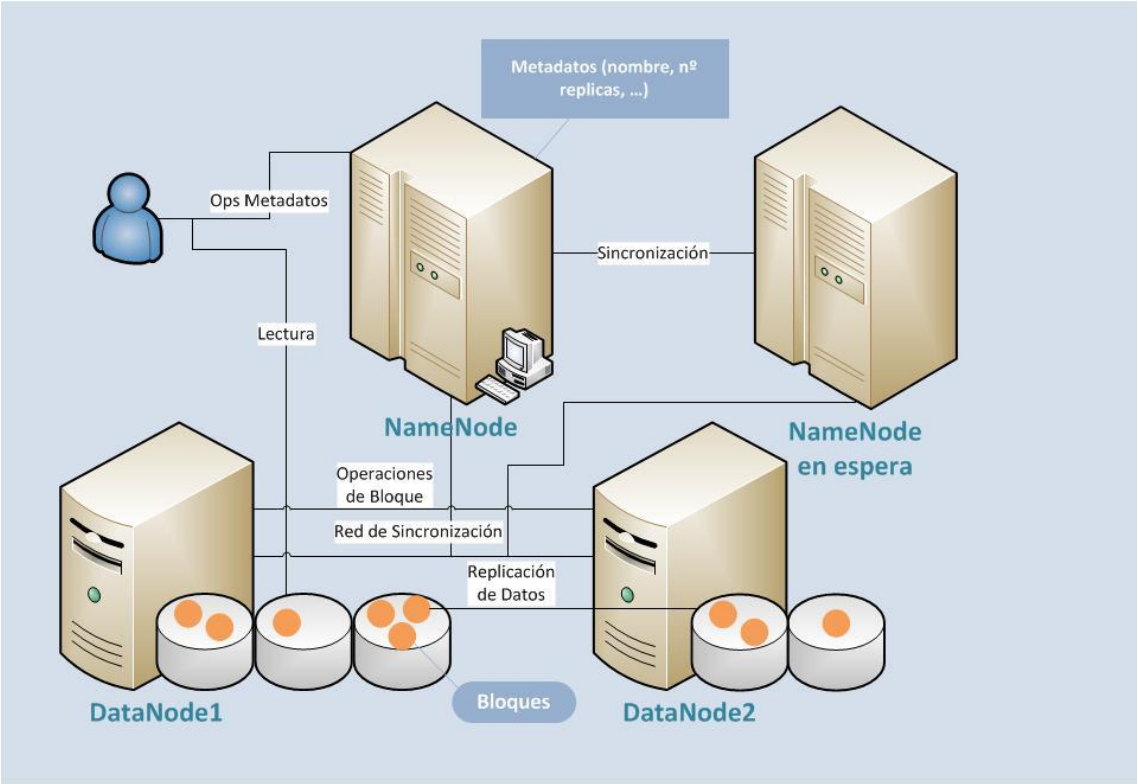


Figura 3.5. Topología HDFS

La arquitectura típica en un clúster *Hadoop* es la que muestra la *Figura 3.5* que suele estar formada por cuatro nodos. Si vemos, dos de ellos son *NameNodes* y los otros dos *DataNodes*. Como ya hemos comentado, una de las principales estrategias de prevención de desastres de este SFD es la replicación de los datos. En un clúster *Hadoop* existe un *NameNode* que es el encargado de almacenar metadatos, es decir, la distinta información acerca de los datos (dónde se almacenan, número de réplicas, etc.). Sin embargo, para mantener la disponibilidad de los datos y en prevención de posibles fallos en el *NameNode* existe otro servidor de nombres replicado que está en constante sincronización con el activo. En este tipo de clúster con dos *NameNodes*, uno activo y otro en segundo plano, es posible configurar el sistema para proporcionar alta disponibilidad de los datos. Es un proceso en desarrollo y se prevé que en poco tiempo esté completamente terminado. El problema actual, es que no se ha conseguido configurar el clúster para que el sistema detecte fallos en un *NameNode* activo y automáticamente tome como activo el *NameNode* secundario.

Por otro lado, están los nodos que almacenan los datos. Lo ideal es que éstos también aparezcan replicados. Los *DataNodes* están conectados con el *NameNode* activo y reciben de él todas las operaciones de bloque que habrán de realizar sobre los datos que almacenan.

Igualmente, todos los nodos del sistema han de estar conectados con ambos servidores de nombres, para que reciban la información de los almacenes de datos y así mantener la integridad del sistema.

3.3.1. Comandos para Gestionar el HDFS

Gracias a lo expuesto a lo largo de la memoria, hemos dado a conocer la importancia del sistema de ficheros distribuido y del peso que tiene en la computación en nuestro entorno. Sabemos que el *framework* de computación distribuida Apache *Hadoop MapReduce* no tendría la importancia y la popularidad que tiene hoy en día si no fuese por el sistema de archivos distribuido *Hadoop Distributed File System* que maneja toda la información con la que trabajan las tareas que se ejecutan en el marco de trabajo.

Es por esto que, antes de exponer cualquier tipo de aplicación *MapReduce*, hemos de comentar todos los comandos que se utilizan para realizar cualquier operación sobre nuestro HDFS, ya que todos estos programas basan su trabajo en grandes volúmenes de datos que han de ser gestionados de manera correcta para obtener el mayor rendimiento de estas tareas.

Los comandos HDFS son invocados mediante la siguiente estructura:

```
>> bin/hadoop dfs <args>
```

Todos ellos toman como argumento un directorio y muchos de ellos son similares a los comandos Unix. En la *Tabla 3.2* listamos todos los comandos, una breve descripción y un ejemplo de aplicación.

COMANDO	DESCRIPCIÓN	EJEMPLO
cat	Copia el fichero del directorio fuente al de salida.	`hadoop dfs -cat URI [URI …]`
chgrp	Cambia el grupo asociado a los archivos. Con –R la ejecución en recursiva.	`hadoop dfs -chgrp [-R] GROUP URI [URI …]`
chmod	Cambia los permisos de los ficheros. Con –R la ejecución es recursiva.	`hadoop dfs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI …]`
chown	Cambia los permisos de los archivos. Con –R la ejecución es recursiva.	`hadoop dfs -chown [-R] [OWNER][:[GROUP]] URI [URI ]`
copyFromLocal	Copia un fichero del directorio local y lo introduce en el HDFS.	`hadoop dfs -copyFromLocal <localsrc> URI`
copyToLocal	Copia un fichero del HDFS a un directorio local.	`hadoop dfs -copyToLocal [-ignorecrc] [-crc] URI <localdst>`
cp	Copia archivos de un directorio a otro de destino dentro del HDFS.	`hadoop dfs -cp URI [URI …] <dest>`
du	Escribe la longitud de un archivo o de los contenidos en un directorio.	`hadoop dfs -du URI [URI …]`
dus	Devuelve la longitud del fichero que recibe como argumento.	`hadoop dfs -dus <args>`
expunge	Vacía la papelera.	`hadoop dfs -expunge`
get	Obtiene ficheros del HDFS para el	`hadoop dfs -get [-ignorecrc] [-crc] <src> <localdst>`

	sistema local.	
getMerge	Toma un directorio fuente como argumento y concatena todos sus archivos en un fichero de destino en el sistema de archivos local.	`hadoop dfs -getmerge <src> <localdst> [addnl]`
ls	Para un fichero devuelve el estado del mismo y para un directorio, su contenido.	`hadoop dfs -ls <args>`
lsr	Similar a *ls* pero de forma recursiva	`hadoop dfs -lsr <args>`
mkdir	Crea los directorios que toma como argumentos.	`hadoop dfs -mkdir <paths>`
moveFromLocal	Mueve un archivo desde el sistema local al HDFS.	`hadoop dfs -moveFromLocal <src> <dst>`
mv	Mueve ficheros de un directorio a otro dentro del HDFS.	`hadoop dfs -mv URI [URI …] <dest>`
put	Copia ficheros del directorio actual y los lleva al directorio que recibe como argumento.	`hadoop dfs -put <localsrc> ... <dst>`
rm	Elimina los ficheros o directorios que recibe como argumento.	`hadoop dfs -rm URI [URI …]`
rmr	Igual que el anterior, pero de manera recursiva.	`hadoop dfs -rmr URI [URI …]`
setrep	Cambia el factor de réplica del fichero especificado. Puede hacerse de manera recursiva con –R.	`hadoop dfs -setrep [-R] <path>`
stat	Devuelve la información del estado de un directorio.	`hadoop dfs -stat URI [URI …]`
tail	Devuelve el último kilobyte del fichero stdout	`hadoop dfs -tail [-f] URI`
test	Evalúa el estado de un directorio en función de: -e: 1 si el fichero existe -z: 0 si la longitud del fichero es cero -d: 1 si la ruta es un directorio.	`hadoop dfs -test -[ezd] URI`
text	Toma un fichero como argumento y lo escribe en formato texto.	`hadoop dfs -text <src>`
touchz	Crea un fichero de longitud cero.	`hadoop dfs -touchz URI [URI …]`

Tabla 3.2. Comandos HDFS

Con estos comandos, es posible gestionar desde la consola todas las opciones que nos permite realizar el sistema de ficheros HDFS. En el *Capítulo 6. Ejemplos de Aplicación del Entorno Apache Hadoop MapReduce* daremos uso a algunos de ellos, con lo que podrá verse un ejemplo práctico de los mismos.

3.3.2. Estrategias de Recuperación de Desastres

Uno de los más importantes objetivos de HDFS es reducir el impacto de los fallos de disco en la disponibilidad general del sistema. En el SFD los datos se encuentran replicados para así minimizar los fallos si alguno de los discos deja de estar activo. También comentamos que en este sistema de ficheros existen dos tipos de nodos, el *NameNode* y los *DataNodes*, por ello hay que plantear estrategias que gestionen posibles errores en ambos. Como en todos los sistemas, puede actuar de un modo preventivo o, a partir de que se produzcan errores, plantear acciones que recuperen esos fallos.

Edit Log Failover

Uno de los aspectos destacados es cómo minimizar el impacto de los fallos de disco en el *NameNode*. La función del *NameNode* es almacenar metadatos que son "datos sobre datos", tales como los propietarios de los archivos, los bits de permiso y demás. El HDFS almacena sus metadatos en el *NameNode* en dos lugares principales: la *FSImage*, y el registro de edición (Edit Log). Configurar el *NameNode* para que almacene múltiples copias de sus metadatos es una buena práctica. Al almacenar dos copias del registro de edición (Edit Log) y *FSImage*, en dos discos duros separados, evitamos que se caiga el *NameNode* si uno de los discos falla.

NameNode Metadata Recovery

Cuando se configura correctamente, HDFS es mucho más robusto contra los metadatos corruptos que un sistema de ficheros local, ya que almacena múltiples copias de todo. Sin embargo, se ha añadido la posibilidad de que un administrador recupere un registro de edición parcial o dañado. Esta nueva funcionalidad se llama recuperación manual de *NameNode* (*NameNode Recovery*). Es un proceso offline en el que un administrador puede ejecutar *NameNode Recovery* para recuperar un registro de edición dañado. Esto puede ser muy útil para detectar eficientemente sistemas de ficheros corruptos.

Se ejecuta de la siguiente manera:

```
./bin/hadoop namenode -recover
```

En este punto, el *NameNode* le pregunta si desea continuar.

```
You have selected Metadata Recovery mode. This mode is intended
to recover lost metadata on a corrupt filesystem. Metadata
recovery mode often permanently deletes data from your HDFS
filesystem. Please back up your edit log and fsimage before
trying this! Are you ready to proceed? (Y/N) (Y or N)
```

Una vez se responde a esto, el proceso de recuperación lee toda la información posible del *Edit Log*. Cuando se produce un error o existe ambigüedad, pregunta cómo continuar.

En el siguiente ejemplo, encontramos un error al intentar leer la transacción con ID=3:

```
11:10:41,443 ERROR FSImage:147 - Error replaying edit log at
offset 71. Expected transaction ID was 3 Recent opcode offsets:
17 71 org.apache.hadoop.fs.ChecksumException: Transaction is
corrupt.     ...
```

```
...       ERROR MetaRecoveryContext:96 - We failed to read txId
3  11:10:41,444  INFO  MetaRecoveryContext:64  -  Enter  'c'  to
continue, skipping the bad section in the log Enter 's' to stop
reading the edit log here, abandoning any later edits Enter 'q'
to quit without saving Enter 'a' to always select the first
choice in the future without prompting. (c/s/q/a)
```

Al tratar de pasar por alto la sección errónea en el registro se dan cuatro opciones: ***continue, stop, quit*** y ***always*** (continuar, parar, salir, y siempre continuar).

- *Continue.* Si el problema es sólo un byte extraviado o dos, o unos pocos sectores defectuosos, esta opción le permitirá saltar este error.

- *Stop.* Detiene la lectura del registro de edición y guarda los contenidos actuales de la FSImage. En este caso, todas las ediciones que aún no se han leído se perderán permanentemente.

- *Quit.* Sale del proceso de *NameNode* sin guardar una FSImage nueva.

- *Always.* Selecciona continuar, y suprime el sistema en el futuro. Una vez que seleccione *always*, el modo de recuperación dejará de preguntar qué opción elegir y siempre seleccionará *continue* en el futuro.

En este caso, hemos elegido continuar:

```
12:22:42,205   INFO   MetaRecoveryContext:105   -   Continuing.
12:22:42,207 INFO FSEditLogLoader:199 - replaying edit log: 4/5
transactions completed. (80%) 12:22:42,208 INFO FSImage:95 -
Edits            file            opt/hadoop/run4/name1/current/
edits_0000000000000000001-0000000000000000005 of size 1048580
edits # 4 loaded in 4 seconds. 12:22:42,212 INFO FSImage:504 -
Saving      image      file      /opt/hadoop/run4/name2/current/
fsimage.ckpt_0000000000000000005    using    no    compression
12:22:42,213   INFO   FSImage:504   -   Saving   image   file
/opt/hadoop/run4/name1/current/fsimage.ckpt_0000000000000000005
using no compression
```

Después, el *NameNode Recovery* se termina. Ahora, puede reiniciar el *NameNode* y continuar de forma normal con las operaciones. El error ha sido reparado, aunque habremos perdido una pequeña cantidad de *metadata*.

Este es un buen método para recuperar metadatos, en caso de una pérdida o error parcial. Sin embargo, es preferible configurar el sistema para que exista una copia válida del *Edit Log* en otro lugar. Es preferible usar esa copia que tratar de recuperar la versión dañada. Así mantenemos un sistema de alta disponibilidad. Si tenemos *NameNode* en modo de espera listo para asumir el control, no hay necesidad alguna de recuperar el *Edit Log*. La recuperación manual es una buena opción cuando no hay otra copia disponible del *Edit Log*.

El mejor proceso de recuperación es el que nunca es necesario hacer. La alta disponibilidad, junto con *Edit Log Failover*, debe significar que la recuperación manual casi nunca es necesaria.

Sin embargo, es bueno saber que el HDFS dispone de herramientas para hacer frente a cualquier cosa que ocurra.

HDFS FSCK

Si bien, hasta ahora hemos visto métodos de recuperación y prevención de errores sobre metadatos. HDFS también dispone de métodos de recuperación de datos.

Cuando el sistema local de archivos ext3 o ext4 se ha dañado, el comando *fsck* se encarga de repararlo. *Fsck* es un proceso offline que examina las estructuras del disco y por lo general ofrece soluciones para arreglarlo si está dañado.

HDFS tiene su propio comando *fsck*, al que se puede acceder ejecutando el comando

```
hdfs fsck
```

Su uso es similar a *ext3 fsck*, determina qué archivos contienen bloques corruptos y ofrece opciones acerca de cómo solucionarlo.

3.4. Configuraciones *Apache Hadoop*

Un clúster típico *Apache Hadoop* suele disponer de dos o más nodos. Sin embargo, la flexibilidad de este sistema permite crear clústeres que van desde un único nodo a miles de ellos. En este apartado vamos a exponer dos tipos de configuraciones, la de un único nodo y la multinodo, ya que se configura de la misma manera un clúster de dos nodos que uno de mil. Además, *Apache Hadoop* permite pasar de un clúster *Single Node* a uno *Multinode* e incluso, en un clúster multinodo ir añadiendo nuevos nodos.

Tanto en los sistemas de nodo único como en los sistemas que aparecen múltiples nodos, hemos de diferenciar dos capas de trabajo. Estas capas son:

- **MapReduce**. Esta capa del entorno de trabajo está destinada a todo lo que tiene que ver con las tareas de las aplicaciones *MapReduce* que se ejecutan. Se compone de dos tipos de elementos, el *JobTracker* y los *TaskTrackers*. En un clúster de un solo nodo, existe uno de cada, pero en un multinodo, hay un *JobTracker* y n *TaskTrackers* (tantos como nodos). El usuario ejecuta aplicaciones sobre los datos del sistema y en esta capa se generan las tareas que se distribuyen entre los nodos. Esta capa está claramente separada de la siguiente, excepto en el momento en el que una tarea llega al nodo en el que se alojan los datos con los que tiene que trabajar. En este momento, ambas capas interaccionan para producir los datos resultantes de la aplicación de dicha tarea y la información almacenada necesaria para la compilación. De esta interacción surgen nuevos datos que pasan a gestionarse en la capa HDFS y una nueva tarea con el resultado de la compilación.

- **HDFS.** Sección del entorno dedicado plenamente a la gestión de los datos del sistema *Apache Hadoop*. En esta capa del sistema, se gestiona el almacenamiento de los datos, la distribución de los mismos y se generan los metadatos que dan la información necesaria sobre los datos del sistema. El elemento principal de esta división del

sistema es el *NameNode* que divide en bloques los datos, decide en qué nodo se almacena cada bloque, el número de réplicas de los datos para mantener un cierto nivel de seguridad ante pérdidas de bloques y genera y almacena todos los metadatos referentes a cada bloque para poder controlar la información que corre por el sistema. Por otro lado, el lugar donde se almacenan los bloques son los *DataNodes*. Generalmente, en un clúster típico, existe un único *NameNode* y tantos *DataNodes* como nodos tenga el sistema.

En las capas de Apache Hadoop aparecen cuatro elementos que son los que realmente gestionan el clúster. Estos elementos están divididos en las dos capas y dos de ellos (uno por capa) ejercen de controladores y los otros dos de elementos de cómputo, bien sea de tareas de aplicaciones o bien de gestión de los datos del sistema. Estos elementos que ejercen el control del clúster son los siguientes:

- ***JobTracker*:** En un clúster hay un único *JobTracker*. Su labor principal es la gestión de los *TaskTrackers*, entre los que distribuye los trabajos *MapReduce* que recibe. Es, por lo tanto, el interfaz principal que tienen los usuarios para acceder al clúster.

- ***TaskTracker*:** Un clúster tiene n *TaskTrackers* que se encargan de la ejecución de las tareas *Map* y *Reduce* en los nodos. Cada uno de los *TaskTrackers* gestiona la ejecución de estas tareas en una máquina del clúster. El *JobTracker* se encarga, a su vez, de controlarlos.

- ***NameNode*:** Las funciones principales del *NameNode* son el almacenamiento y la gestión de los metadatos del sistema de ficheros distribuido y ofrecer el interfaz principal que tiene el usuario para acceder al contenido HDFS. En un clúster hay un único proceso *NameNode* activo. Sin los metadatos que mantiene el *NameNode* no se sabría en qué nodo está cada bloque, además de perderse la información sobre la estructura de directorios. Es, por lo tanto, el componente más importante del clúster, y debe estar redundado. *Hadoop* no ofrece de manera nativa ningún mecanismo de alta disponibilidad, pero sí dispone de herramientas que permiten replicar la información. Entre ellas está el *NameNode* secundario, que permite guardar una copia de los metadatos en una máquina diferente al *NameNode*. Eso sí, no es una copia en tiempo real, y no ofrece *failover* automático en caso de fallo del primario. Para esto necesitamos usar otro tipo de herramientas, entre las que destacan DRBD y *Heartbeat*.

- ***DataNode*:** Estos procesos ofrecen el servicio de almacenamiento de bloques para el sistema de ficheros distribuido. Son coordinados por el *NameNode*.

Estos son los procesos principales del sistema, pero hay otros, como el *Balancer*, que se encarga de equilibrar la distribución de los bloques entre los *DataNodes*, por ejemplo cuando se añade un nuevo servidor.

Conociendo ya las capas en las que se divide un clúster *Apache Hadoop* y los elementos que conforman la base del sistema y que ejercen el control y el cómputo de los datos y las tareas, vamos, a continuación, a conocer las distintas topologías de clúster que puede desplegar según el número de nodos.

3.4.1. Configuración *Single Node*

Es el modo más sencillo ya que se compone de un solo nodo y en él se configuran las dos capas de las que se compone el clúster. Además de diferenciar ambas secciones, se instalan en cada una de ellas los cuatro elementos descritos en el apartado anterior. Es decir, este clúster con nodo único, se compone de una máquina que ejerce todas las tareas tanto para gestionar los datos como las aplicaciones. En ella se instala el controlador de tareas (*JobTracker*), el ejecutor de las mismas (*TaskTracker*), el *NameNode* que gestiona los datos y los metadatos y el *DataNode* que almacena la información.

Al estar todos los elementos del clúster alojados en una única máquina, el rendimiento del sistema se ve mermado. Sin embargo, la funcionalidad del clúster es análoga a la de otro clúster con miles de nodos, la única diferencia es el rendimiento.

En la *Figura 3.6* se puede apreciar que aparece un único nodo en el que se han configurado todos los elementos. Comprobamos como existe la separación entre la parte del clúster dedicada a las tareas *MapReduce* y el apartado dedicado a los datos (capa HDFS).

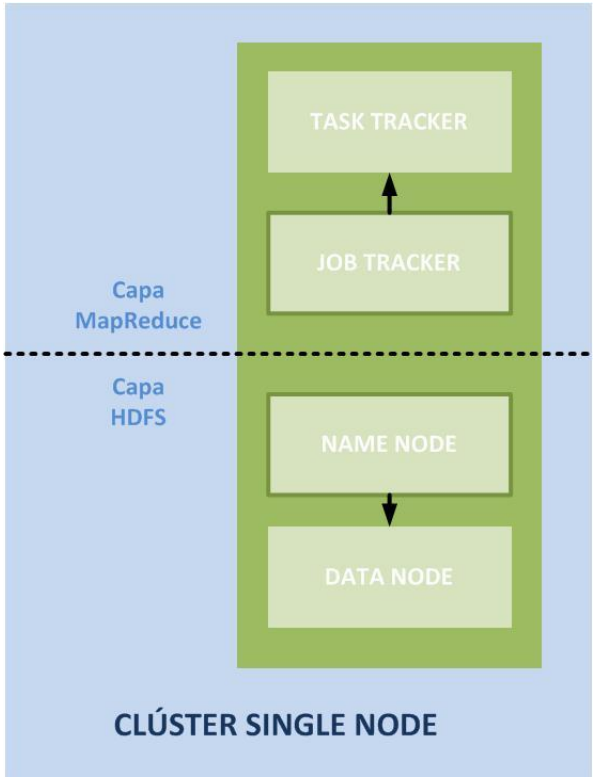


Figura 3.6. Clúster *Hadoop Single Node*

El sistema funciona igual que un clúster multinodo, solo que los datos que se gestionan en el *NameNode*, son distribuidos dentro del mismo disco, eso sí se generan los mismos metadatos que para un clúster superior y los mismos bloques para el almacenamiento de la información. Por otro lado, al *JobTracker* le llegan aplicaciones ejecutadas por el usuario y éste genera las tareas *Map* y *Reduce* que, en lugar de distribuir entre los nodos del clúster, manda al *TaskTracker* que está alojado dentro de la misma máquina.

Como puede apreciar, el funcionamiento es igual que el de un clúster, solo que se verá afectado el rendimiento del sistema.

3.4.2. Configuración *Multinode*

Un clúster *Apache Hadoop Multinode* se compone de un Nodo Máster y tantos Nodos Esclavos como queramos desplegar según los límites que posea nuestro entorno. Esos límites, tienen que ver con los recursos hardware de los que dispone el entorno. Según los recursos de los que dispongamos, procesamiento, memoria, etc., es posible desplegar clústeres con mayor número de nodos.

En ocasiones, formar un clúster y configurar todos los nodos del mismo para que trabajen de manera conjunta se convierte en algo realmente complejo. Sin embargo, en el caso que nos ocupa, formar un entorno de trabajo bajo esta tecnología es algo más sencillo de lo que pueda parecer y a la vez realmente flexible y adaptable, ya que permite incluir cualquier tipo de equipo (máquinas virtuales incluidas) y tantas cuantas deseemos, incluso una vez concluida la configuración inicial.

Formar un clúster *Apache Hadoop* con múltiples nodos comienza realizando la configuración anteriormente expuesta sobre el nodo que máster. Es decir, la instalación comienza formando un *Single Node Clúster* sobre una de las máquinas. Esta máquina pasará a ser el Nodo Máster que ejercerá de nodo controlador del clúster.

A partir de aquí, los siguientes pasos corresponden con ir añadiendo tantos Nodos Esclavos como queramos añadir al clúster.

En la *Figura 3.7* puede ver que el Nodo Máster queda configurado de la misma manera que el nodo único del entorno *Single Node*. Este nodo contiene tanto el *JobTracker* como el *NameNode*, que ejercen de controladores de las tareas *MapReduce* y del sistema de ficheros respectivamente. También puede comprobar como el Nodo Máster contiene instalado tanto el *TaskTracker* como el *DataNode*, por lo que este nodo ejerce tanto de controlador del sistema como de nodo de cómputo.

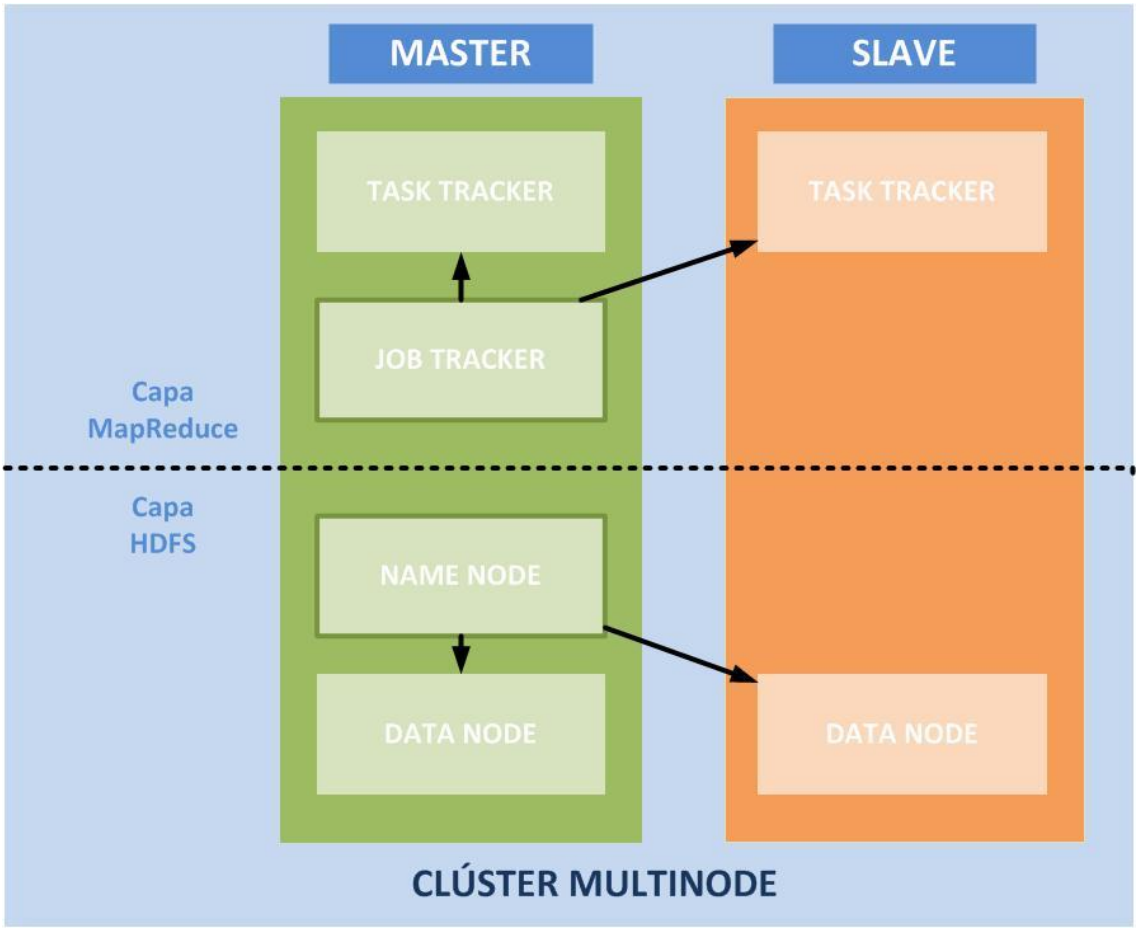


Figura 3.7. Clúster *Hadoop Multinode*

Por otro lado, en el Nodo Esclavo solo se configuran el *Data Node* en la capa HDFS y el *TaskTacker* en la capa *MapReduce*. Estos nodos, sólo actúan de nodos de cómputo para las tareas *MapReduce* y de almacenamiento para el sistema de ficheros distribuido.

Si se fijan en la *Figura 3.7*, el *JobTracker* se comunica tanto con el *TaskTracker* alojado en su mismo nodo como con el resto de *TaskTrackers* que estén configurados como propios del clúster. El *JobTracker* envía a cada *TaskTrackers* las tareas que atañen a los datos alojados en sus *DataNodes*. Es decir, cada *TaskTracker* ejecuta tareas que utilicen los datos que almacenan dentro del *DataNode* configurado en su misma máquina. Por ello, el *NameNode* alojado en el Nodo Máster, ha de tener el control sobre los datos, dónde se alojan, cómo están divididos los bloques y dónde están almacenadas las réplicas de los mismos.

Dada una tarea, el *JobTracker* genera las órdenes MapReduce necesarias para su ejecución. Estas órdenes se generan manteniendo una comunicación previa con el *NameNode*. Éste proporciona al *JobTracker* la información necesaria sobre los datos que la tarea va a operar para que así puedan generarse las órdenes y ser enviadas a la máquina o máquinas que corresponda.

El *NameNode*, dada una nueva entrada de datos, realiza una división en bloques de los mismos, generalmente de 64Mb. Una vez divididos los datos, envía una copia de los mismos a los *DataNodes* en los que se van a ir almacenando. Además, dependiendo de la configuración

del clúster, genera tantas réplicas como dicte dicha configuración y envía las copias a otros nodos distintos. Así puede mantenerse cierta seguridad sobre los datos.

Un clúster multinodo en *Apache Hadoop* es más seguro y eficiente si asignamos otro nodo como Nodo Máster secundario. Es decir, este nodo ejerce de Nodo Máster en caso de caída o pérdida total del Nodo Máster primario. Dispone de una copia de todos los metadatos y se encuentra en constante comunicación con el maestro activo y conoce todas las nuevas órdenes o tareas que se vayan ejecutando. Por tanto en caso de pérdida parcial o total del nodo primario, el nodo secundario toma el control, manteniéndose por completo la funcionalidad del sistema.

La principal característica de los clústeres *Apache Hadoop* es su flexibilidad y adaptación a todo tipo de condiciones. Hemos visto cómo es posible desplegar clústeres que van desde un nodo a tantos como queramos. Además, dado un clúster multinodo, puede variar la configuración según le interese; puede configurarse un Nodo Máster que a su vez sea Nodo Esclavo o, si así lo preferimos, que únicamente ejerza las tareas de maestro. Es decir, un nodo que solamente tenga configurados los elementos *JobTracker* y *NameNode* o que a su vez (como vimos en la *Figura 3.7*), tenga configurados también el *TaskTracker* y el *DataNode*. Esto depende de las características de la máquina que empleemos como Nodo Máster. Por otro lado, hemos comentado que es bueno replicar tanto los datos de los *DataNodes* como el *NameNode* en sí, incluidos sus metadatos. Esto es algo opcional ya que, si no se desean replicar los datos o el *NameNode*, el clúster funciona correctamente; sin embargo es recomendable hacerlo. Por defecto, el sistema de ficheros distribuido de *Apache Hadoop* (HDFS) replica los datos tres veces en los *DataNodes*, pero este valor es configurable.

3.4.3. Elección de la Topología del Clúster

Para la realización del proyecto, vamos a seguir directamente las recomendaciones de *Apache Hadoop*. En primera instancia, creamos un clúster *Single Node* sobre una máquina virtual. Sobre este clúster de nodo único, realizamos una serie de pruebas para familiarizarnos con el entorno de trabajo.Posteriormente, añadimos nuevos nodos y creamos un clúster multinodo.

Comenzaremos agregando una nueva instancia virtual configurada con *Apache Hadoop* al clúster e incluiremos nuevas copias virtuales, tantas como nuestro entorno de computación *Cloud Computing* nos permita. Así, alcanzaremos el rendimiento máximo de nuestro sistema.

El desarrollo de este proceso de despliegue y configuración del clúster, se lleva a cabo en el *Capítulo 5. Puesta en Marcha del Entorno Apache Hadoop/MapReduce.*

3.5. Relación *Cloud Computing - Apache Hadoop*

Como ya hemos ido comentando a lo largo del capítulo, *Apache Hadoop* proporciona un entorno de computación distribuido sorprendentemente flexible.

Apache Hadoop permite instalarse y configurarse en cualquier tipo de equipo, sea cual sea su sistema operativo o sus características de rendimiento a nivel hardware. Además, permite utilizar máquinas virtuales. El único requerimiento de este entorno de computación distribuida es que los equipos que conformen el clúster tengan instalado *Java*, ya que, al estar basado en él, lo necesitan para funcionar, y que tengan también configurado el servicio SSH.

Por otro lado, hemos destacado en el *Apartado 3.4.2. Configuración Multinode*, que pueden formarse clústeres *Hadoop* que van desde un nodo único a múltiples nodos, simplemente configurando uno de ellos como Nodo Máster y añadiendo posteriormente nuevos Nodos Esclavos.

El hecho de que *Apache Hadoop* permita instalarse en instancias virtuales y que sea tan flexible a la hora de aumentar o disminuir el tamaño de sus clústeres sin que el funcionamiento del mismo se vea afectado, lo convierten en un entorno **idóneo para trabajar bajo el paradigma de computación *Cloud Computing*** (véase la *Figura 3.8*).

Figura 3.8. *Cloud Computing* y *Apache Hadoop*

Por tanto, llegamos a la conclusión de que la mejor opción para obtener el máximo rendimiento de este entorno de computación distribuido es implantarlo sobre una nube y así realizar pruebas con tantos nodos como queramos en el clúster.

3.6. Aplicaciones de *Apache Hadoop*

Apache Hadoop MapReduce es un *framework* **ampliamente asentado** en el mercado y que obtiene grandes resultados. Todo ello le ha llevado a ser probablemente el **entorno de trabajo de computación distribuida con mayor popularidad,** hasta tal punto de que en el sitio web oficial del proyecto hay un apartado expresamente dedicado para mencionar todas esas empresas que han realizado proyectos con su entorno de trabajo. En este apartado de la web puede encontrar un **glosario de empresas** (véase la *Figura 3.*9) que aglutina a cientos de ellas. Por ello añadimos el enlace a dicho glosario web y además vamos a mencionar algunas de ellas, las más conocidas, a modo de reseña.

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

Figura 3.9. Empresas *Apache Hadoop*

- **eBay.** Tiene un clúster con 532 nodos con 8 núcleos por nodo y una capacidad de almacenamiento de 5,3PB. Usado para optimización de búsqueda e investigación

- **Fox (*audience network*).** Tres clústeres de cuarenta, setenta y treinta núcleos con ocho núcleos por servidor con capacidades de almacenamiento de dos, tres y cuatro terabytes respectivamente. Usados para registro y análisis de datos de audiencia, minería de datos y aprendizaje máquina.

- **Facebook.** Lo utiliza para almacenar copias de registro interno, fuentes de datos de grandes dimensiones, generación de informes y análisis de datos y aprendizaje máquina. Los dos clústeres de mayor tamaño que poseen hoy en día son; uno de 1.100 nodos con 8.800 núcleos y 12PB de almacenamiento y otro de 300 nodos con 2.400 núcleos y 3PB de almacenamiento.

- **Yahoo!.** Utiliza más de 100.000 CPUs en más de 40.000 computadoras con *Apache Hadoop* ejecutándose. Su mayor clúster es de 4.500 nodos con 8 núcleos y 4TB por nodo. Usados para investigación, sistema de anuncios, búsqueda web y pruebas a gran escala para apoyar el desarrollo de *Apache Hadoop* en grandes clústeres.

- **IBM.** Utiliza *Apache Hadoop* en dos iniciativas, *Blue Cloud Computing Cluster* y *University Initiative to Address Internet-Scale Computing Challenges.*

- **AOL.** Usan *Apache Hadoop*, como muchas empresas, para la ejecución de algoritmos avanzados para el análisis del comportamiento y la orientación, además de para la generación de estadísticas y el estilo de procesamiento ETL. El clúster usado para el análisis del comportamiento y la orientación cuenta con 150 equipos, *Intel Xeon*, doble núcleo con 16GB de RAM y 800GB de disco duro.

- **Amazon.** Utilizado en la construcción de índices de búsqueda de productos de Amazon. Procesan millones de sesiones diarias de análisis de datos. Los clústeres varían de 1 a 100 nodos.

- **Google.** El entorno en el que desarrollan su estudio sobre *Apache Hadoop* es la *University Initiative to Address Internet-Scale Computing Challenges.* Aplicaciones propias de *Google* que utilizan *MapReduce*, son el Traductor Google (*Google Translator*), el procesador de imágenes de satélite (*Google Maps*), el aprendizaje automático y análisis de páginas (*Google Analytics*), *Google Trends* y por supuesto, la más usada, el motor de búsqueda.

- **MercadoLibre.com.** Clúster de 20 nodos con 12 núcleos por nodo 32GB de RAM y 53,3TB de almacenamiento. Usado para la conexión de clientes a aplicaciones online, operaciones de procesamiento de registros. Hacen uso de *Java, Pig, Hive y Oozie.*

- **Twitter.** Usan *Apache Hadoop* para almacenar y procesar *Tweets*, archivos de registro y muchos otros tipos de datos generados por *Twitter*. Usan la distribución *Hadoop Cloudera's* CDH2 y almacenan los datos comprimidos en formato LZO.

- **Adobe.** Utilizan *Apache Hadoop* para almacenamiento y procesamiento de datos de muchas áreas de servicios para usuarios o para datos estructurados de uso interno. Poseen clústeres de 30 nodos que pretenden ampliar a 80.

- **Hosting Habitat**. Dispone de una versión modificada de *Apache Hadoop* y *Nutch* con la que ofrecen los servicios clásicos de *hosting* y servicios propios que mejoran la calidad de sus servicios.

- **Linkedin.** Utiliza varios clústeres que van desde los 120 hasta los 1.200 nodos, con gran capacidad de almacenamiento. Todos sus sistemas basan su funcionamiento en sistemas *Apache Hadoop*.

- **Last.fm.** Utiliza un clúster de 40 nodos *quad-core* con 16GB de RAM y 4TB de almacenamiento por nodo usado para cálculo de gráficos, análisis de datos y distintas pruebas. Otro clúster de 20 nodos *quad-core* con 32GB de RAM y 5TB de disco duro por nodo usado para análisis de perfiles, análisis estadístico, presentación de informes a nivel de cookies, etc.

Puede comprobar como el uso del entorno de computación distribuida *Apache Hadoop MapReduce* está realmente extendido en el mundo empresarial lo que, unido a sus brillantes características comentadas a lo largo de todo el capítulo, hacen de él un marco de trabajo muy interesante de ser estudiado.

En el siguiente capítulo, realizaremos un estudio del entorno de programación que acompaña a este marco de trabajo, *MapReduce.* Sabemos que el entorno de programación y el marco de trabajo distribuido *Apache Hadoop* están diseñados para trabajar conjuntamente y así alcanzar un gran rendimiento en computación distribuida con aplicaciones que trabajen con grandes cantidades de datos. Aun así, necesitamos conocer a fondo el entorno de programación *MapReduce*, profundizar en sus características, ventajas, desventajas y posteriormente, estudiar cómo crear programas en este lenguaje.

4. Estudio de la Programación *MapReduce*

En el capítulo anterior, "Estudio del Entorno *Apache Hadoop*", dimos a conocer un entorno de computación distribuida capaz de trabajar con clústeres de hasta miles de nodos y un gran volumen de datos. Se estudió su sistema de ficheros distribuido *Hadoop Distributed Files System*, parte fundamental de este entorno.

Figura 4.1. Logo *MapReduce*

Sin embargo, en muchos párrafos del capítulo anterior, se menciona un *framework* de programación que trabaja conjuntamente al entorno *Hadoop*, creando clústeres de computación distribuida de alto rendimiento capaces de gestionar entornos con miles de nodos y hasta petabytes de datos.

Este capítulo pretende hacer un análisis en profundidad de este marco de programación llamado ***MapReduce***, dar a conocer sus **inicios, características, funcionalidad, implementaciones, etc**. Se van a plantear dos tipos de aplicaciones, una para poner a prueba el entorno *Apache Hadoop MapReduce* que se va a desplegar en el capítulo posterior y otra para utilizar una librería creada para trabajar con imágenes en este entorno llamada HIPI (*Hadoop Image Processing Interface*).

4.1. Introducción

En la actualidad el motor de búsqueda más usado es *Google*. Todos valoramos su velocidad, ya que es capaz de realizar búsquedas en milisegundos sobre una gran cantidad de páginas web. Si buscamos "*MapReduce*" en *Google*, la consulta nos devuelve 3.400.000 resultados en 0.58 segundos. ¿Cómo puede ser esto posible?

Cuando *Yahoo* y *Microsoft* eran los líderes en Internet (véase la *Figura 4.2*), *Google* intentó venderles su idea y éstas hicieron caso omiso al negocio. Dichas empresas, con sus grandes presupuestos, apostaban por las supercomputadoras, los grandes servidores y por infraestructuras de alto coste físico. Sin embargo, *Google* no podía competir con eso, ya que

por aquel entonces no era la empresa líder que es hoy en día. Por ello decidió poner en práctica su idea de utilizar servidores de mucho menor coste pero cuyo uso de los recursos fuese realmente eficiente, implementando clústeres gestionados con *MapReduce*. El tiempo les ha dado la razón.

Figura 4.2. *Yahoo* y *Microsoft*

La idea es la siguiente, si dispone de un computador con una capacidad de almacenamiento de 1TB, contando con que la tasa de transferencia sea de 100Mb/s se necesitaría más de dos horas y media para completar la lectura del disco. Para la escritura el tiempo sería mayor. Por ello, es preferible tener cien discos que sumen la misma cantidad de espacio de almacenamiento y leerlos en paralelo, con lo que reduciríamos el tiempo a poco menos de dos minutos. Ahora bien, aumentar el número de dispositivos, incrementa la posibilidad de que alguno falle, por ello hay que implementar técnicas de replicación. También es muy frecuente que durante la ejecución de una tarea se necesitase acceso a datos de varios discos, todo esto, hace que sea necesario un sistema que lo gestione. *MapReduce* acompañado de un buen Sistema de Gestión de Ficheros Distribuido es la solución de *Google* al problema.

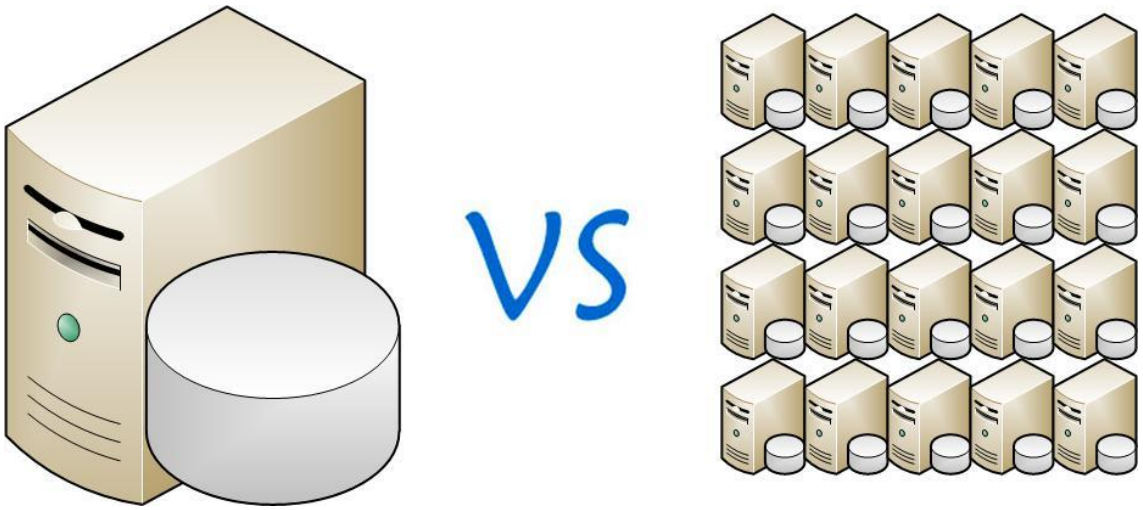


Figura 4.3. Gran Servidor vs Clúster

MapReduce es un modelo de programación creado por Google para procesar grandes cantidades de datos de forma paralela en clústeres de computadoras. Existen varias implementaciones de *MapReduce*, escritas en Java (las más utilizadas), C++, *Python* y otros lenguajes. Inicialmente, este modelo de programación fue creado con la habilidad de hacer una consulta sobre un gran conjunto de datos, dividiéndola y ejecutándola en paralelo sobre muchos servidores a la vez. Si bien hemos comentado anteriormente que es básico que *MapReduce* vaya acompañado de un sistema de ficheros distribuido que le de soporte. Por tanto, ante una consulta, la programación *MapReduce* divide dicha tarea entre cada uno de los nodos que componen el clúster para que realicen la tarea en modo local y luego sea combinada para obtener el resultado. Un sistema de programación *MapReduce* hace

balanceado de carga en las tareas y establece un control con los nodos que fallen, dotando al conjunto de tolerancia a fallos.

4.2. Características

El entorno de programación está orientado a aplicaciones que procesen gran cantidad de datos. Trabaja junto a un sistema de ficheros con mecanismos de replicación de datos, por lo que dotan al sistema de fiabilidad en la información, además de que es tolerante a fallos en la red o por desaparición de algún nodo. El sistema sobre el que se ejecutan las aplicaciones *MapReduce* es distribuido y por tanto la programación es paralela. Es también tolerante a fallos sobre los nodos que funcionan de manera incorrecta y la carga de trabajo es repartida entre los nodos. El sistema es portable entre nodos con hardware y software heterogéneo y por tanto, es muy escalable, ya que es posible añadir nodos de trabajo que sean esclavos de un mismo máster. **La programación *MapReduce* es funcional** ya que aplica una misma función a elementos de una lista de entrada y genera una lista de elementos de salida.

Para terminar de asimilar las características de la programación vamos a sintetizar todas ellas en la siguiente lista:

- **Programación funcional**. Funciones *map* y *reduce* que producen, ante una lista de datos de entrada, un conjunto de elementos de salida.
- **Orientado a grandes cantidades de datos** que se procesan por lotes. Estos datos **se almacenan en bloques** entre los distintos nodos del sistema y son **controlados por el** nodo de datos (***DataNode***).
- **Sistema de Ficheros fiable y tolerante a fallos. Replicación de metadatos** del nodo de control.
- **Portabilidad del sistema en nodos con hardware y software heterogéneos**. Por tanto, **sistema escalable** que permite añadir nuevos nodos.
- Sistema distribuido que lleva a cabo la **programación paralela.** Realiza **balanceado de la carga de trabajo**.
- **Control de los nodos que fallan, tolerancia a fallos del sistema**.
- Puede implantarse un sistema a coste cero ya que todo el software puede encontrarse en **licencia libre**. Además el que sea libre aporta más ventajas como personalización del software, adaptabilidad, continua actualización, soporte en comunidades, etc.
- **Alta compatibilidad** ya que puede instalarse en sistemas operativos G*NU/Linux* y *Windows*, de forma **virtualizada o sobre servidores físicos**, con gran variedad de sistemas de ficheros distribuidos.
- **Basado en Java**.

- Existen varias implementaciones del lenguaje de programación que permiten explotar las distintas funcionalidades del mismo.

4.3. Funcionamiento

Mapreduce está **basado** en dos funciones: ***Map*** **y** ***Reduce***. La computación toma pares de **entrada clave-valor** y produce como **salida otro par clave-valor**. Los pares de entrada son generados por el sistema a partir de los datos con los que trabaja el usuario. Es decir, la aplicación trabaja sobre cierta información del sistema, dichos datos, son repartidos por los nodos ejecutores del sistema organizados por una clave, de ahí los pares clave-valor. El usuario de la librería expresa la computación en las dos funciones mencionadas anteriormente.

Map, escrito por el usuario, toma los pares de entrada clave-valor y produce pares intermedios con la misma estructura. La librería *MapReduce* agrupa todos los valores intermedios por clave y los pasa a la función Reduce.

La función *Reduce*, igualmente escrita por el usuario, acepta los valores intermedios agrupados por clave y trata de reducirlos para formar el menor número posible de valores. El conjunto resultante se trata de una lista en la que puede ir comprobando todos los valores obtenidos a través de un iterador.

En resumen, el modelo de programación esta basado en la computación de los siguientes conceptos:

1. **Iteración** sobre los datos de entrada.
2. **Computación de los pares** clave-valor para cada dato de entrada.
3. **Agrupación** de los valores intermedios por clave.
4. **Iteración** sobre los grupos resultantes.
5. **Reducción** de cada grupo.

El clásico ejemplo con el que se explica la funcionalidad de la programación *MapReduce* es el de contar el número de veces que aparece una palabra en una biblioteca. Para ello hay que coger libro por libro, recorrer todas sus páginas y contar palabra por palabra cuantas veces se repite el término en cuestión.

La ejecución *MapReduce*, sigue el siguiente proceso. Por un lado la función *Map* reparte los libros entre todas los equipos disponibles para que cada una busque dentro de los libros que se le asignen y cuente el número de repeticiones de cada término. La función combinadora, cuenta y agrupa cada palabra con el número de repeticiones en cada libro. La función *Reduce* toma cada palabra y suma el número de repeticiones en todos los libros. Finalmente obtenemos la cantidad de valores que aparece cada término dentro de la librería.

El código que lleva a cabo este ejemplo es el siguiente:

```
map(String name, String document):
```

```
  // clave: nombre del documento
  // valor: contenido del documento
  for each word w in document:
    EmitIntermediate(w, 1);

reduce(String word, Iterator partialCounts):
  // word: una palabra
  // partialCounts: una lista parcial de la cuenta agregada
  int result = 0;
  for each v in partialCounts:
    result += ParseInt(v);
  Emit(result);
```

También puede servir para tratar de entender la computación de *MapReduce* el esquema que se muestra en la *Figura 4.4* y que explica el ejemplo comentado anteriormente.

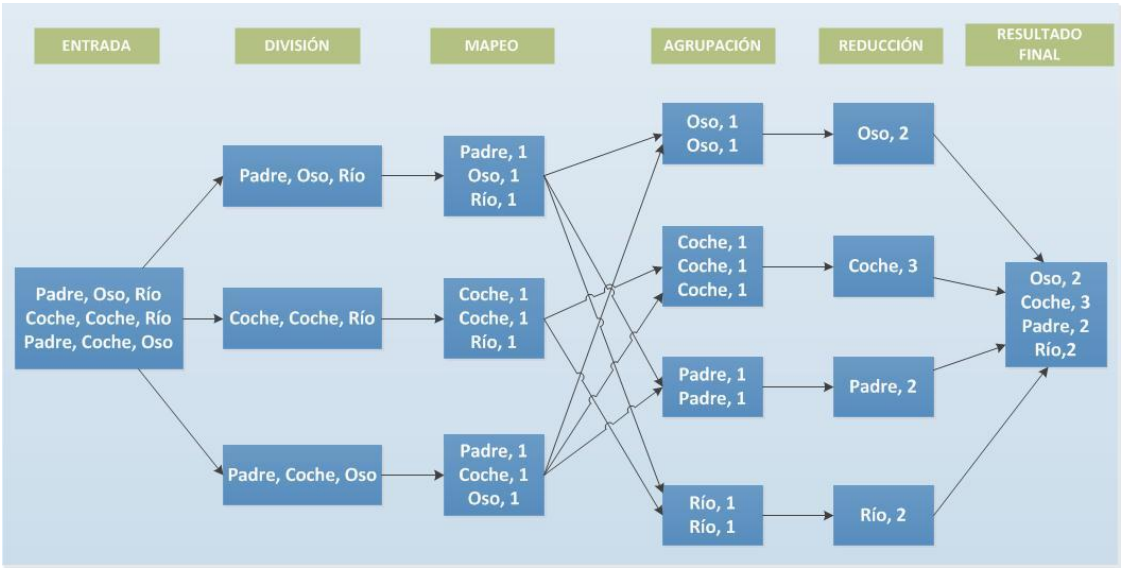


Figura 4.4. Esquema *MapReduce*

La ejecución sigue el siguiente guión:

1. En los datos de entrada vemos tres líneas que corresponden a cada uno de los libros. Éstos son repartidos entre tres equipos que van a procesarlos.

2. La función *Map* separa y cuenta las palabras dentro de cada libro agrupándolas en pares clave-valor, que en este caso son palabra-repeticiones.

3. En el siguiente paso, la función combinadora, agrupa cada una de las palabras con su número de apariciones.

4. Finalmente la función *Reduce* los aglutina y produce la lista final con cada palabra y sus apariciones en la biblioteca de datos recibida como entrada.

En todo programa *MapReduce*, dado que está basando en *Java*, debe haber un método "`main()`" o principal desde el que se pone en marcha la ejecución y en el que deben de aparecer una serie de funciones básicas para configurar el trabajo *MapReduce*. Para crear un

nuevo trabajo *MapReduce* ha de llamarse a la clase "JobConf" y crear un nuevo trabajo. Sobre este objeto se configuran todos los aspectos fundamentales del programa *MapReduce.*

4.3.1. Clase JobConf

Es una clase ***Java*** incluida en la *API* de *Hadoop* que contiene métodos para configurar todos los parámetros de un trabajo *MapReduce*, por ello supone la interfaz principal de un usuario para crear y ejecutar un trabajo *MapReduce* en el entorno *Hadoop*. Generalmente utilizamos esta clase para especificar ciertas variables del entorno de programación como son:

- ***Map (Mapper).*** Reparte los datos de entrada y genera los pares intermedios. Hemos de indicar la clase que realiza esta funcionalidad, ejecutando el método `setMapperClass(Clase Map)`.

- ***Combiner.*** Combina los pares intermedios y los agrupa. Al igual que en el caso anterior, especificamos la clase que realiza las combinaciones de los valores intermedios llamando al método `setCombinerClass(Clase Combinadora)`.

- ***Reduce (Reducer).*** Reúne y cuenta los valores intermedios generando la salida. Se especifica con el método `setReducerClass(Clase Reduce)`.

- **Archivos de entrada.** Es necesario indicar el directorio de los ficheros de entrada y el formato. En *MapReduce* existen formatos especiales de archivos de entrada que documentaremos más adelante. Hemos de indicar la variable *InputFormat* que contiene el tipo de los archivos de entrada y llamar a los métodos que definen la ruta de los archivos de entrada (`setInputPaths (JobConf, ruta ...)` / `addInputPath (JobConf, Path)` y `setInputPaths (JobConf, String)` / `addInputPaths (JobConf, String)`)

- **Archivos de salida.** Al igual que en el caso anterior, se indica el directorio de salida y el formato. Hemos de configurar la variable *OutputFormat* y llamar a los métodos para definir la ruta de salida (por ejemplo `setOutputPath (Path)`).

El entorno trata siempre de ejecutar fielmente el trabajo según lo descrito por *JobConf*, aunque pueden darse situaciones que no lo permitan, como son:

- Los parámetros de configuración estén marcados como tipo "*final*" por los administradores y por tanto, no pueden ser alterados.

- Algunos parámetros de configuración del trabajo pueden establecerse sin problemas (por ejemplo `setNumReduceTasks (int)`), pero otros interactúan con el resto de la estructura del entorno y/o de la configuración del trabajo y pueden presentar problemas si quieren modificarse desde *JobConf*, como puede ser `setNumMapTasks (int)`.

Opcionalmente, *JobConf* puede utilizarse para especificar otras facetas avanzadas del trabajo tales como:

- El `Comparator` que va a ser utilizado (en las operaciones *Reduce*).
- Los archivos que se disponen en la `DistributedCache`, ya sean pares intermedios y/o pares de salida de trabajos,
- Depuración proporcionada mediante *scripts* por el usuario (`setMapDebugScript(String)/setReduceDebugScript(String)`)
- El número máximo de intentos de ejecución por tarea, tanto para la función *Map* como Reduce (`setMaxMapAttempts(int)/setMaxReduceAttempts(int)`)
- Porcentaje máximo de fallo de tareas *Map* y *Reduce* que pueden ser tolerados por el trabajo (`setMaxMapTaskFailuresPercent(int)/setMaxReduceTaskFailuresPercen(int)`), etc.

Estas últimas variables (agrupadas por guiones) pueden ser o no modificadas en un trabajo *Hadoop*, pero las comentadas anteriormente mediante puntos sí han de ser especificadas. En los siguientes apartados vamos a tratar esos aspectos fundamentales de todo trabajo *Apache Hadoop* en mayor profundidad, aportando más características de los mismos.

El entorno de computación distribuida *Apache Hadoop* proporciona una *API* de programación Java *MapReduce* en la que puede encontrar, entre otras, la clase *JobConf* y todos sus métodos. Incluimos la dirección Web y recomendamos acceder a ella y consultarla para conocer más a fondo todos los procesos de cómputo que se encuentran.

http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapred/JobConf.html

4.3.2. Función *Map*

La función *Map* o *Mapper* toma los datos de entrada y los transforma en los pares de entrada (clave-valor). Posteriormente toma esos valores, los procesa y genera los pares intermedios. Estos pares intermedios contienen la clave con la que se agrupan los resultados de la computación *Map* y el valor de esa computación. Es decir, si nos trasladamos al ejemplo de los libros, la clave es la palabra y el valor el número de repeticiones. Los pares de entrada y los intermedios no tienen que ser del mismo tipo.

Para cada dato de entrada, que se configura con *InputFormat* y ha de tener alguno de los formatos permitidos en *MapReduce,* se genera un *InputSplit* que realiza una llamada al método *Mapper*. El número de *Maps* viene determinado por el tamaño de los datos de entrada. El número de bloques de los datos de entrada es lo que determina los *Maps* por nodo. Al configurar un nuevo trabajo, *JobConf*, se realizan las llamadas pertinentes al método *Mapper* correspondientes, atendiendo al número de *InputSplit* generadas.

Dada una tarea *Mapper*, se pueden generar de cero a muchos pares de salida, los cuales no tiene por qué ser del mismo tipo que los de entrada. Todos los pares de salida, son recogidos por el `OutputCollector.collect()`.

Estos valores intermedios, son asociados con una clave dada y posteriormente agrupados por el entorno de trabajo y pasados al método *Reducer* que determina la salida. El usuario puede controlar el agrupamiento especificando el comparador mediante el `JobConf.setOutputKeyComparatorClass(Class)`.

El usuario puede especificar un combinador desde el `JobConf.setCombinerClass(Class)` para llevar a cabo la agrupación de datos intermedios, lo que reduce la cantidad de datos transferidos desde el *Mapper* al *Reducer*.

Los resultados intermedios, siempre ordenados, se almacenan en un formato simple

```
(key-len, clave, valor-len, valor)
```

Las aplicaciones pueden controlar si, y cómo, los resultados intermedios se comprimen y el *CompressionCodec* que va ser utilizado a través del *JobConf*.

Por tanto, podemos concluir que la funcionalidad principal de un programa *MapReduce* se encuentra alojada en el método *Mapper*. Este procedimiento es el que recibe los datos de entrada en formato de entrada correspondiente, los procesa (según lo programado) y genera los datos intermedios con su clave. Estos datos son agrupados gracias a la clave y al comparador, también configurable por el programador y pueden ser comprimidos para ser enviados a la tarea *Reduce*, si así lo estima el programador.

La *API* de *Hadoop MapReduce* también incluye información sobre la interfaz de programación *Mapper*. Más información en:

http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapred/Mapper.html

4.3.3. Función Reduce

Cuando creamos un trabajo *MapReduce* y realizamos la configuración con un objeto *JobConf*, hay que asignar un método que se encargue de la ordenación y reducción de los pares intermedios generados por el *Mapper*. El método *Reduce* toma un conjunto de valores intermedios que comparten una clave y los reduce a un conjunto más pequeño de valores. El número de tareas *Reduce* para el trabajo puede establecerse por el usuario a través de `JobConf.setNumReduceTasks (int)`.

En general, las implementaciones *Reduce* son aprobadas por el *JobConf* para el trabajo a través de `JobConfigurable.configure (JobConf)` y han de tener la estructura `Reduce (WritableComparable, Iterator, OutputCollector, Reporter)`, método preparado para cada lista que agrupa los pares de entradas (`list <key, values)>`).

El proceso *Reduce* tiene tres fases principales:

1. ***Shuffle*** (Barajar). Es la primera etapa del proceso Reduce, en la que se recibe como entrada la salida ordenada de los *Mappers*. En esta fase, el marco recupera la partición correspondiente de la salida de todos los mapeadores, a través de HTTP. Además los pares de entrada son repartidos entre todas las tareas *Reduce*.

2. ***Sort*** (Ordenar). En esta fase, el *framework Apache Hadoop MapReduce* agrupa las entradas *Reduce* por clave y las ordena (los pares de diferentes *Mappers* han de tener la misma clave de salida).

    - ***Secondary Sort*** (Ordenación secundaria). Si son requeridas reglas de equivalencia para agrupar los pares de claves intermedios antes de reducirlos, ha de especificarse un comparador (*Comparator*) a través del método `JobConf.setOutputValueGroupingComparator (Class)`. El método `JobConf.setOutputKeyComparatorClass (Class)` se puede utilizar para controlar cómo las claves intermedias se agrupan. Ambos procesos pueden ser utilizados en combinación para simular clasificación secundaria en los valores.

    Shuffle y las fases de clasificación ***Sort*** y ***Secondary Sort*** se producen simultáneamente, mientras que las salidas de los *Mappers* van siendo recibidas. Así, se van recibiendo pares intermedios de todos los *Mappers* y se van procesando sin llegar a acabar la etapa del *Map.*

3. ***Reduce*** (Reducir). En esta fase, el método `reduce (WritableComparable, Iterator, OutputCollector, Reporter)` es llamado para cada par de datos de entrada agrupados en `list <key, values)>`. La salida de la tarea *Reduce* normalmente es escrita por el sistema de archivos (*FileSystem*) a través de `OutputCollector.collect (WritableComparable, Writable)`. Las aplicaciones pueden utilizar el *Reporter* para conocer el progreso, establecer el nivel de mensajes de estado de la aplicación y actualizar los contadores (*Counters*), o simplemente para conocer que los nodos están activos. La salida del reductor no está ordenada.

Numero de tareas Reduce

Según la documentación aportada por *Apache Hadoop MapReduce*, el número correcto de tareas puede calcularse con la siguiente formula:

Reduce es 0.95 o 1.75 multiplicado por (<Número de nodos> * `mapred.tasktracker.reduce.tasks.maximum`).

Si se toma como valor de referencia 0.95 para realizar la multiplicación, las tareas *Reduce* se ejecutarán en cuanto vayan llegando las salidas de los *Mappers.* Sin embargo, si toma el valor 1.75, los nodos más rápidos terminarán su primera ronda de *Reduce* y podrán ejecutar una segunda oleada de tareas *Reduce* haciendo un trabajo mucho mejor en cuanto a equilibrio de carga.

Aumentar el número de tareas *Reduce* incrementa la carga del entorno, pero a su vez se produce un mayor equilibrio de la carga y disminuye el coste de los fallos.

Los factores de escala anterior son valores un poco menores de un número entero para reservar unos pocas tareas *Reduce* en el marco de trabajo para tareas especulativas o fallidas.

No configurar un reductor

Es posible establecer el número de tareas *Reduce* en cero. Es decir, no configurar un reductor. En este caso, las salidas de los *Mappers* van directamente al sistema de archivos, en la ruta de salida establecida en el *JobConf* por `setOutputPath (Path)`. Las salidas de las tareas *Map* no son ordenadas antes de ser escritas en el sistema de archivos.

Recomendamos, como en los casos anteriores, acceder en la *API* de *Hadoop MapReduce* a su sección dedicada al *Reducer*.

http://hadoop.apache.org/docs/r0.20.0/api/org/apache/hadoop/mapred/Reducer.html

4.3.4. Formatos de Archivos de Entrada

El entorno de programación *Apache Hadoop MapReduce* incluye formatos de entrada específicos y propios del marco de trabajo. Todo trabajo que sea declarado en este entorno, ha de contener como datos de entrada ficheros en alguno de estos formatos. *InputFormat* describe los formatos de entrada de los archivos para las aplicaciones *MapReduce.*

El entorno *Hadoop MapReduce* incluye varios formatos predefinidos de tipos de archivos, siendo *TextInputFormat* el *InputFormat* por defecto. Esta preparado para leer archivos de texto plano sin formato divididos por líneas. El marco de trabajo detecta archivos de entrada con la extensión *'.gz'* y automáticamente los descomprime usando el *CompressionCodec* apropiado. Sin embargo debe tenerse en cuenta que los archivos comprimidos no se pueden dividir y cada archivo comprimido es procesado en su totalidad por un único *mapper*. Otros formatos de archivos de entrada son:

- *CombineFileInputFormat.*
- *CompositeInputFormat.*
- *DBInputFormat.*
- *FileInputFormat.*
- *MultiFileInputFormat.*
- *NLineInputFormat.*
- *SequenceFileInputFormat.*

Es recomendable acudir a la *API* de *Apache Hadoop MapReduce* para consultar las especificaciones de cada uno de estos formatos de archivo.

http://hadoop.apache.org/docs/r0.20.2/api/org/apache/hadoop/mapred/InputFormat.html

El entorno de trabajo *MapReduce* sigue las siguientes operaciones para cada trabajo:

- Validar las especificaciones de entrada del trabajo, configuradas en el *JobConf*.
- Dividir y repartir los archivos de entrada en los correspondientes *InputSplit* lógicos, cada uno de los cuales se asigna a un *Mapper*.
- Proporcionar la implementación *RecordReader* utilizada para leer y almacenar los registros de entrada *InputSplit* lógicos para el procesamiento por parte del Mapper asignado.

El comportamiento predeterminado de las implementaciones *InputFormat* basadas en archivos, por lo general sub-clases de *FileInputFormat*, es dividir la entrada en instancias *InputSplit* lógicas basadas en el tamaño total, en bytes, de los archivos de entrada. Sin embargo, el tamaño de bloque del sistema distribuido de archivos se trata como un límite superior para las divisiones de los ficheros de entrada. Un límite inferior del tamaño de división se puede ajustar a través `mapred.min.split.size`.

Hay que tener en cuenta que, los *InputSplit* lógicos en los que se dividen los archivos de entrada basándose en el tamaño de bloque no siempre son suficientes para algunas aplicaciones, ya que los límites de registro deben ser respetados. En tales casos, las aplicaciones deben implementar un *RecordReader*, quien es responsable de respetar los límites de registros y presenta una grabación orientada a registro de los *InputSplit* lógicos de cada tarea.

InputSplit

Representa los datos para ser procesados por un *Mapper* individual. Típicamente *InputSplit* presenta una vista orientada a los bytes de la entrada, y es responsabilidad de *RecordReader* procesar y presentar una vista orientada a registro.

FileSplit es la *InputSplit* por defecto. Se establece en la variable `map.input.file` la ruta del archivo de entrada para la división lógica.

RecordReader

Lee los pares <clave, valor> de un *InputSplit*. El *RecordReader* convierte la vista orientada a byte de la entrada, suministrada por el *InputSplit*, y genera un registro orientado a las implementaciones *Mapper* para su procesamiento. *RecordReader* asume así la responsabilidad de procesar los límites de registro y presenta las tareas con claves y valores.

4.3.5. Formatos de Archivos de Salida

Al igual que para los archivos de entrada el entorno de programación *Apache Hadoop MapReduce* incluye formatos de salida específicos y propios del marco de trabajo. *OutputFormat* describe las especificaciones de los ficheros de salida para un trabajo *MapReduce*.

TextOutputFormat es el formato de archivo de salida *OutputFormat* por defecto y, al igual que en los archivos de entrada, está especialmente indicado para el trabajo con archivos de texto plano sin formato. Otros formatos de archivos de salida con los que nos podemos encontrar son:

- *DBOutputFormat.*
- *FileOutputFormat.*
- *MultipleOutputFormat.*
- *SequenceFileOutputFormat.*

Si se desea conocer en profundidad estos formatos de archivos de entrada o el resto de métodos que acompañan a la interfaz de los ficheros de salida, se recomienda acudir a la *API* oficial del entorno.

http://hadoop.apache.org/docs/r0.20.2/api/org/apache/hadoop/mapred/OutputFormat.html

La secuencia de acciones que lleva a cabo el marco de trabajo *MapReduce* para un programa en cuanto a los ficheros de salida *OutputFormat* consiste en:

- Validar las especificaciones de salida del trabajo como, por ejemplo, comprobar que el directorio de salida existe.
- Proporcionar la implementación *RecordWriter* utilizada para escribir los archivos de salida del trabajo. Los archivos de salida se guardan en el sistema de archivos distribuido.

OutputCommitter

Es el encargado de aportar la confirmación para una tarea de salida de un trabajo *MapReduce*. El entorno de trabajo utiliza el *OutputCommitter* para:

1. **Configurar el trabajo durante la inicialización**. Por ejemplo, crear el directorio de salida temporal para el programa durante la inicialización del trabajo. La configuración del trabajo se hace en una tarea por separado cuando el trabajo está en el estado de preparación "*PREP*" y después de la inicialización de tareas. Una vez que se completa la tarea de inicialización, el trabajo se traslada a un estado de ejecución "*RUNNING*".
2. **Liberador de espacio después de la finalización del trabajo**. Por ejemplo, eliminar el directorio de salida temporal después de la finalización del trabajo. La limpieza del entorno de trabajo se realiza por una tarea separada al finalizar la ejecución. El trabajo se declara *"SUCCEDED/FAILED/KILLED"* después de que la tarea de limpieza se complete.

3. **Configuración de la salida de la tarea temporal**. Esta tarea se realiza conjuntamente con la inicialización.

4. **Comprobar si una tarea requiere una confirmación**. Esto es para evitar que el procedimiento de confirmación si una tarea no la precisa.

5. **Confirmación de la tarea de salida**. Una vez que la tarea se lleva a cabo, ha de comprobarse su salida siempre que sea necesario según las especificaciones.

6. **Desechar las tareas no confirmadas**. Si la tarea ha sido "*FAILED/KILLED*", la salida será limpiada. Si la tarea no puede ser limpiada (en el bloque de excepciones), otra nueva tarea separada se pondrá en marcha con el mismo id para poder hacer la limpieza.

FileOutputCommitter es la *OutputCommitter* por defecto. Las tareas de configuración y limpieza del trabajo ocupan "*slots*" del entorno *MapReduce*, que se llevan a cabo en el *TaskTracker*. Las tareas de limpieza del trabajo "*JobCleanup*", de limpieza de tareas "*TaskCleanup*" y de configuración del trabajo "*JobSetup*" tienen la más alta prioridad y en ese orden.

Tareas Side-Effect Files

En algunas aplicaciones, ciertos componentes de las tareas deben crear y/o escribir en archivos secundarios, que difieren de los de salida reales del trabajo.

En tales casos, pueden producirse problemas con dos instancias del mismo *Mapper* o *Reducer* ejecutándose al mismo tiempo (por ejemplo, las tareas especulativas) tratando de abrir y/o escribir en el mismo archivo (*path*) en el sistema de ficheros. Es por esto que los programadores deben utilizar nombres únicos para los archivos, incluyendo en el nombre el intento de la tarea y no solo el identificador de la misma. Deben utilizar el '*attemptid*', por ejemplo `attempt_200709221812_0001_m_000000_0`.

Para evitar estos problemas en el entorno *MapReduce*, cuando el *OutputCommitter* tiene el control *FileOutputCommitter*, mantiene un especial "`${mapred.output.dir}/_temporary_/_${taskid}`" (subdirectorio temporal de salida para cada tarea) accesible a través de "`${mapred.work.output.dir}`" (directorio de salida del trabajo) para todos los intentos de las tareas en el sistema de ficheros donde se almacena el resultado de cada uno de ellos. Tras la finalización satisfactoria de la tentativa de una tarea, solo en ese caso, los archivos en el directorio "`${mapred.output.dir}/_temporary/_${taskid}`" se mueven a "`${mapred.work.output.dir}`". Por supuesto, el marco de trabajo descarta todos los subdirectorios de trabajo infructuosos creados por las distintas tentativas de las tareas. Este proceso es completamente transparente para la aplicación y por supuesto para el usuario.

La parte escritora de la aplicación puede actuar de forma preventiva sobre este problema creando los archivos secundarios requeridos en "`${mapred.work.output.dir}`" durante la ejecución de una tarea a través de `FileOutputFormat.getWorkOutputPath()`. Así, el marco de trabajo, podrá tomarlos como válidos de manera similar que en el caso anterior para tentativas exitosas de las tareas, lo que elimina la necesidad de crear directorios únicos por tareas-intento. Tendríamos en un

mismo directorio todos los archivos secundarios (directorio de salida "`${mapred.work.output.dir}`"), ahorrándonos la creación de un directorio para cada fichero con su identificador.

Nota: *El valor de "`${mapred.work.output.dir}`" durante la ejecución de una determinada tentativa de una tarea es en realidad "`${mapred.output.dir}/_temporary/_${taskid}`", y este valor es establecido por el entorno MapReduce. Por lo tanto, basta con crear los archivos secundarios en el directorio que devuelve `FileOutputFormat.getWorkOutputPath()` de la tarea MapReduce para tomar ventaja sobre esta problemática con los archivos secundarios.*

Toda la discusión es válida para los entornos de trabajo que son configurados con reductor = NINGUNO (es decir, reduce=0) ya que la salida de cada mapper, en ese caso, va directamente a HDFS y por ello se producen errores con ficheros secundarios.

RecordWriter

Encargado de escribir los pares <clave, valor> definitivos resultantes de la ejecución en un archivo de salida en el directorio configurado en el *JobConf* del trabajo *MapReduce*. Las salidas del *RecordWriter* son escritas directamente en el *FileSystem*.

4.4. Librería HIPI (*Hadoop Image Processing Interface*)

Como hemos visto en los apartados anteriores, un trabajo *Hadoop MapReduce* se compone, entre otras cosas, de un método principal en el que se crea un objeto de configuración *JobConf*. Utilizamos este objeto para configurar todos los aspectos necesarios en un trabajo. Una parte de los puntos de configuración básicos, es referente a los tipos de los archivos de entrada y salida. Si acude a las secciones anteriores en los que se trataba esta temática, apartados "4.3.4. Formatos de Archivos de Entrada" y "4.3.5. Formatos de Archivos de Salida", o incluso revisando la *API* oficial de *Apache Hadoop MapReduce*, puede comprobar que no existe ningún formato específico para trabajar con imágenes.

Dado que el entorno de computación distribuida *Apache Hadoop MapReduce* está especialmente indicado para aplicaciones que trabajan sobre grandes volúmenes de datos, no parece descabellado querer utilizar este entorno para desplegar marcos de trabajo que operen con cantidades elevadas de imágenes. Las imágenes en alta resolución pueden tener un peso de almacenamiento realmente alto y un ambiente de desarrollo en el que se almacenen estas imágenes y se realicen operaciones sobre las mismas, parece realmente adecuado.

Si pensamos en el mundo multimedia en el que nos encontramos, surgen numerosos ejemplos de posibles aplicaciones. Hoy en día, casi todas las personas tenemos un *Smartphone* con el que realizamos fotografías en cualquier situación y luego queremos "subirlas" a las distintas redes sociales o almacenarlas en un entorno de almacenamiento en la nube. Todas estas situaciones serían aptas para ser desplegadas en un entorno *Apache Hadoop MapReduce*¸ ya que una imagen (o conjunto de ellas) es "subida" a la red social una sola vez y sobre ella se realizan operaciones como pueden ser comentarios, etiquetas, obtención de información, etc. Esta es la situación óptima de trabajo del entorno de computación distribuida que nos ocupa,

como ya se ha comentado anteriormente, las aplicaciones que más se adaptan son las que realizan pocas escrituras y múltiples accesos a los datos.

Al no existir ningún formato de archivos de entrada o salida para imágenes, se hace imposible crear aplicaciones de este entorno que computen sobre imágenes. Sin embargo, gracias a la ***University of Virginia*** (http://www.virginia.edu/) disponemos hoy en día de una interfaz de procesamiento de imágenes en *Apache Hadoop MapReduce,* llamada ***Hadoop Image Processing Interface*** (HIPI) (http://hipi.cs.virginia.edu/).

Figura 4.5. Logo HIPI

Esta librería para el entorno *Hadoop MapReduce* proporciona una *API* para la realización de tareas de procesamiento de imágenes en un entorno de computación distribuida. La ejecución utilizando esta librería, es similar a la de un trabajo común en *MapReduce*, solo se diferencia en que hemos de preparar un contenedor de imágenes que es la entrada de la aplicación. En la *Figura 4.*6 puede ver la distribución de un trabajo:

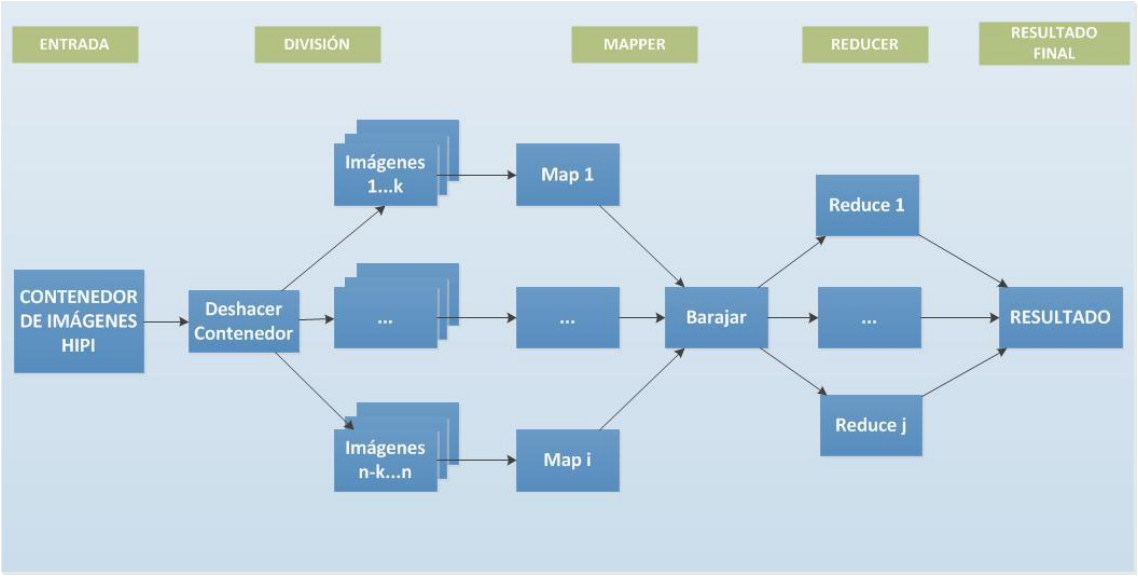


Figura 4.6. Esquema HIPI

Hemos de preparar como entrada del programa un contenedor propio de esta librería, un ***HipiImageBundle*** (HIB). Un HIB es un conjunto de imágenes combinadas en un archivo grande junto con ciertos metadatos que describen algunos aspectos referentes a de las imágenes. Un

HIB puede ser creado a partir de un conjunto ya existente de imágenes, o a través de otra fuente, como puede ser un conjunto de descargas.

Una vez que se recibe el contenedor de imágenes HIB como entrada, los pasos a seguir por la ejecución son los mismos que los de una ejecución normal. Es decir, tras deshacer el contenedor, se reparten las imágenes entre los nodos de ejecución que haya en el clúster y se generan las tareas *Mapper* pertinentes. Los pares de datos intermedios <clave, valor> son barajados y repartidos de manera organizada entre los *Reducer* que son los encargados de devolver el resultado final de la ejecución. Los tipos de pares intermedios en este caso, varían según el tipo de aplicación que se programe, por ejemplo, se pueden agrupar las imágenes según el tipo de cámaras con la que se han tomado; la clave sería el tipo de cámara (*Canon*, *Olympus*, etc.) y el valor, el identificador de la imagen.

Con el fin de mejorar la eficacia de algunos trabajos, HIPI permite a un usuario especificar una **función de sacrificio** que descarta imágenes que no cumplan con un conjunto determinado de criterios (por ejemplo, la resolución de la imagen debe ser inferior a 10 megapíxeles). Si queremos añadir esta funcionalidad, hemos de especificar la clase *CullMapper* que es invocada para cada imagen y que realiza la prueba de sacrificio, eliminando de la ejecución las que no cumplan con los requisitos. Las imágenes se presentan a esta clase como una *FloatImage* y un *ImageHeader* asociado.

Aunque HIPI no modifica directamente el comportamiento de *Hadoop MapReduce,* una vez que el *Mapper* toma el control, el usuario puede modificar los parámetros de ejecución específicos para las tareas de procesamiento de imágenes a través del objeto *HipiJob* durante la configuración del trabajo.

Para poder comenzar a realizar aplicaciones *Hadoop MapReduce HIPI* hemos de conocer y dominar el comportamiento de cuatro clases principales. Estas clases aportan el conocimiento básico para programar con este lenguaje y la librería de tratamiento de imágenes que nos ocupa. Las clases más utilizadas por un desarrollador son la *HipiImageBundle*, *FloatImage*, CullMapper y la clase de configuración de un trabajo HIPI, *HipiJob*. Proporcionan la mayor parte de la funcionalidad que un usuario medio necesita y se describen brevemente a continuación. La API completa para HIPI se puede encontrar en la siguiente página Web de documentación (http://hipi.cs.virginia.edu/doc/api/index.html).

4.4.1. *HIPI Image Bundle (HIB)*

Un HIB es una colección de imágenes que se almacenan en un solo archivo, de forma análoga a los archivos ".tar" en *GNU/Linux*. Los HIBs se implementan a través de la clase *HipiImageBundle* y están diseñados para poder utilizarse directamente en el marco de trabajo *Hadoop MapReduce*.

Las operaciones básicas que han de conocerse para poder trabajar con los contenedores de archivos HIB las comentamos a continuación.

Creación de un HIB

Para crear un nuevo contenedor, hemos de crear un nuevo objeto *HipiImageBundle* y pasarle por parámetro el directorio dónde queremos que se almacene el contenedor generado y el objeto de configuración que vamos a utilizar.

```
Configuration conf = new Configuration();
HipiImageBundle hib = new HipiImageBundle(new Path("/path/to/file.hib"), conf);
hib.open(AbstractImageBundle.FILE_MODE_WRITE, true);
```

Figura 4.7. Creación de un HIB

Hemos mencionado que se ha de incluir en la declaración un objeto de configuración de la clase *Configuration*. Este objeto proporciona acceso a los datos de configuración que se especifican a la hora de desplegar el clúster Hadoop. Muchos de esos valores de configuración son tipo *"Final"*, con lo que solo pueden ser consultados. Sin embargo algunos de ellos si pueden ser modificados para la ejecución del trabajo por el contenedor de imágenes HIB. Si queremos consultar en profundidad los métodos que permiten acceder a consultar o modificar datos de configuración, recomendamos acudir a la *API* de *Hadoop*:

http://hadoop.apache.org/docs/current/api/org/apache/hadoop/conf/Configuration.html

En la última línea de código de la *Figura 4.7* se abre el nuevo contenedor en modo escritura.

Añadir imágenes a un HIB

Una vez que se crea un contenedor HIB, el siguiente paso es incluir imágenes dentro. En la *Figura 4.8* puede ver cómo hacerlo:

```
File file = new File("/path/to/file.jpg");
FileInputStream fis = new FileInputStream(file);
hib.addImage(fis, ImageType.JPEG_IMAGE);
```

Figura 4.8. Añadir Imágenes a un HIB

Se declara un fichero tipo *File* y por parámetro facilitamos el directorio dónde se encuentra el archivo. Convertimos ese *File* declarado en un *FileInputStream,* clase que se utiliza para leer los archivos por bytes. Con esto, ya tenemos el fichero preparado para incluirlo en el contenedor. Para ello, utilizamos el método `addImage(InputStream,ImageHeader.ImageType)` incluyendo el *FileInputStream* y el tipo de imagen (*JPEG_IMAGE, PNG_IMAGE, PPM_IMAGE, UNSUPPORTED_IMAGE*).

Fusionar dos HIBs

La librería también contempla la posibilidad de, teniendo un contenedor almacenado, añadirlo a otro nuevo contenedor. En el caso que estamos describiendo, sabemos que hemos creado un contenedor al que hemos llamado "hib". Pues bien, ahora queremos añadir a éste un nuevo contenedor que tenemos almacenado en cierto directorio siguiendo el procedimiento que vemos en la *Figura 4.9*.

```
Path temp_path = new Path("/path/to/file.hib");
HipiImageBundle input_bundle = new HipiImageBundle(temp_path, conf);
hib.append(input_bundle);
```

Figura 4.9. Fusionar HIBs

Creamos un directorio con la localización del contenedor a añadir. Creamos un contenedor nuevo con ese directorio y llamamos al método `append(HipiImageBundle)` proporcionando el contenedor que queremos fusionar.

Leer las imágenes contenidas en un HIB

Sabiendo ya las directrices básicas para crear un HIB, parece claro que el siguiente paso debe ser saber leer las imágenes que estos contienen. Puede hacerse una a una desde el principio o de manera aleatoria. El modo aleatorio puede conocerlo a través de *HipiImageBundle.FileReader* en la siguiente dirección Web:

http://hipi.cs.virginia.edu/doc/api/hipi/imagebundle/HipiImageBundle.FileReader.html

Para el acceso convencional leyendo imágenes desde la primera a la última puede ejecutar (véase *Figura 4.10*):

```
while (hib.hasNext()) {
      ImageHeader imageHeader = hib.readHeader();
      FloatImage image = hib.readImage();
}
hib.close();
```

Figura 4.10. Leer HIB

Lo primero que ha de hacerse es tener declarado y abierto el contenedor, sino este código daría lugar a errores. Creamos un bucle "*while*" que realice iteraciones hasta recuperar todas las imágenes contenidas en el HIB. De cada imagen se puede leer la información (metadatos) gracias el método `readHeader()`, o bien la imagen completa con el método `readImage()`. Tras leer el contenedor completo, ha de cerrarlo.

Para conocer en profundidad la clase *HipiImageBundle* y sus métodos, acuda a la *API* de HIPI y consultar la información que allí se presenta.

http://hipi.cs.virginia.edu/doc/api/hipi/imagebundle/HipiImageBundle.html

En cualquier caso, es recomendable conocer ciertos **datos importantes** a la hora de trabajar con contenedores de imágenes.

- Los contenedores deben abrirse en modo *FILE_MODE_READ* o *FILE_MODE_WRITE*, dependiendo de si queremos el HIB en modo lectura o escritura. Una vez abierto en un modo, no puede cambiarlo, hay que cerrar el HIB y abrirlo en el otro modo.

- A la hora de crear un nuevo HIB, disponemos de varios constructores. En ellos puede declarar tanto el número de réplicas de la información como la longitud de bloque de las divisiones. Este parámetro es muy importante ya que controla la forma en que las imágenes de un HIB se distribuyen entre las tareas de los *Mappers*. El *ImageBundleInputFormat* divide un HIB en secciones basadas en el tamaño de bloque del propio contenedor para distribuir eficazmente el trabajo. Sin embargo, si el tamaño de bloque es demasiado grande, entonces un pequeño número de servidores se encargan del procesamiento de todas las imágenes del HIB. Por lo tanto, se aconseja que el tamaño de bloque se ajuste de tal manera que el número de bloques en los que se divida el contenedor sea aproximadamente el número de nodos en el clúster.

- Pueden crearse nuevos contenedores bien como explicamos anteriormente o bien de otros modos, como pueden ser una lista de descargas. Se recomienda acceder a la documentación que proporciona la librería para conocer los distintos modos de creación de contenedores.

4.4.2. *Float Image*

La entrada principal para asignar tareas en HIPI es la clase *FloatImage*. Se trata de una simple representación de un archivo de imagen como un conjunto de píxeles especificados con un solo punto flotante de precisión. Varias operaciones comunes se pueden realizar en un *FloatImage*:

- **Crear una *FloatImage.*** (véase *Figura 4.11*) Necesario para poder hacer cualquier operación con imágenes.

```
FileInputStream file = new FileInputStream("/path/to/image.jpg");
FloatImage image = JPEGImageUtil.getInstance().decodeImage(file);
```

Figura 4.11. Crear *FloatImage*

- **Obtener o modificar los valores de los píxeles.** (véase *Figura 4.12*) Útil para poder obtener o modificar colores de las imágenes.

```
float red = image.getPixel(0, 0, 0);
float green = image.getPixel(0, 0, 1);
float blue = image.getPixel(0, 0, 2);
image.setPixel(0, 0, 0, red / 2);
image.setPixel(0, 0, 1, green / 2);
image.setPixel(0, 0, 2, blue / 2);
```

Figura 4.12. Obtener Píxeles

- **Convertir a escala de grises.** (véase *Figura 4.13*) Sirve para eliminar los colores y transformar la imagen a su análoga en escala de grises. Para realizar ciertas operaciones (como detección de formas) es muy útil esta operación.

```
FloatImage gray_image = image.convert(FloatImage.RGB2GRAY);
```

Figura 4.13. Escala de Grises

- **Operaciones aritméticas.** (véase *Figura 4.14*) Puede emplear esta funcionalidad para escalar la imagen u obtener secciones de la misma (altura o anchura).

```
FloatImage image2 = new FloatImage(image.getWidth(),
                image.getHeight(), image.getBands());
image2.add(2.0f);
image.scale(image2);
image.scale(2.0f);
image.add(image2);
image.add(2.0f);
```

Figura 4.14. Operaciones Aritméticas

- ***Image Cropping.*** (véase *Figura 4.15 y 4.16*) Técnica para eliminar el exterior de una imagen y así poder obtener el tamaño y dimensiones deseados**.**

```
FloatImage crop = image.crop(0, 0, 20, 20);
```

Figura 4.15. *Image Cropping*

Un ejemplo de esta técnica es el que vemos a continuación:

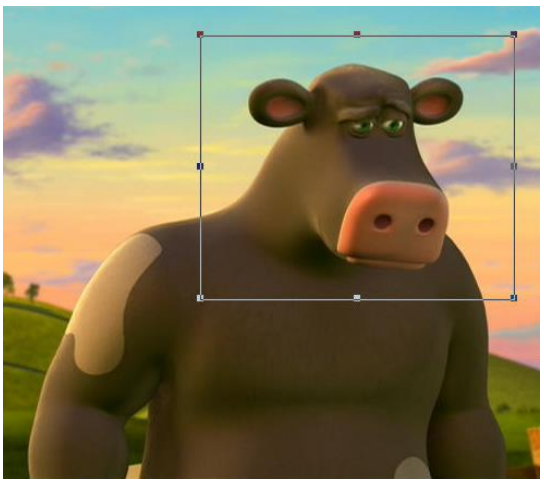


Figura 4.16. Ejemplo *Cropping*

FloatImage es en realidad un conjunto de tres dimensiones de valores en punto flotante. Por lo tanto, también se puede tratar como una matriz y utilizarse en consecuencia. La función `FloatImage::GetData` convierte la *FloatImage* en su representación como matriz y se puede utilizar directamente para realizar operaciones estándares de matriz. La creación de una

FloatImage como una matriz de píxeles también se puede realizar a través de uno de los constructores de *FloatImage*. Por ello recomendamos acudir a la documentación de la API.

http://hipi.cs.virginia.edu/doc/api/hipi/image/FloatImage.html

4.4.3. *Cull Mapper*

HIPI proporciona una forma para que los usuarios puedan descartar o desechar imágenes que no cumplan una serie de criterios. La clase *CullMapper* define un tipo especial de *Mapper Hadoop* que contiene una función extra `CullMapper::cull` que puede utilizarse para comprobar a través de la cabecera de las imágenes (*ImageHeader*) ciertos criterios que deban cumplir las mismas y así desechar las *FloatImage* asociadas.

En el ejemplo siguiente, *CullMapper* (hereda directamente de *Mapper*) se utiliza para procesar las imágenes digitales tomadas con una cámara *Canon PowerShot S500* y exigir que las dimensiones sea 2.592x1944. La función `CullMapper::cull` (véase la *Figura 4.*17) devuelve un booleano que indica si la imagen descrita por *ImageHeader* debe desecharse (true) o deben ser transformadas o desestimadas(false).

```
public static class MyMapper extends CullMapper<ImageHeader,
                        FloatImage, NullWritable, FloatImage>
{
    public boolean cull(ImageHeader key) throws IOException,
                                        InterruptedException {
        if(key.getEXIFInformation("Model").equals("Canon PowerShot S500")
                                && key.width == 2592 && key.height == 1944)
            return false;
        else
            return true;
    }
    public void map(ImageHeader key, FloatImage value, Context context)
                                throws IOException, InterruptedException {
```

Figura 4.17. *CullMapper*

Dependiendo de los criterios que queramos que nuestras imágenes cumplan, cambia el contenido del método *CullMapper* que se debe programar. Sin embargo, la estructura del mismo no cambia, así que puede tomar este método como ejemplo para posteriores implementaciones.

En cualquier caso, a continuación puede encontrar la referencia Web a la API con la documentación de esta clase.

http://hipi.cs.virginia.edu/doc/api/hipi/imagebundle/mapreduce/CullMapper.html

4.4.4. *HIPI Job*

HIPI proporciona una extensión de la clase de trabajo estándar *Hadoop* (*JobConf*) que permite al programador establecer los parámetros que son comunes en escenarios donde HIPI es utilizado. Las dos operaciones principales que se pueden realizar utilizando un HipiJob son:

- Activar/desactivar la ejecución especulativa.
- Activar/desactivar la compresión de los registros de salida del *Mapper*.

El primer método (`HipiJob::set{Map,Reduce}` Ejecución especulativa) controla si *Hadoop* debe ejecutar varias instancias de la misma tarea *Map* para aumentar potencialmente el rendimiento. Generalmente esta opción suele ser buena y por ello debe estar habilitado. Esto garantiza que si un nodo en particular en el clúster *Hadoop* está experimentando una degradación del rendimiento, todo el trabajo no se verá afectado. *Hadoop* automáticamente va a "matar" la tarea más lenta, y utilizar el resultado de una instancia diferente. Por supuesto, esto conlleva cierta sobrecarga de tareas más de lo que realmente se necesita, aunque suele compensar en el rendimiento global.

La segunda operación (`HipiJob::setCompressMapOutput`) activa o desactiva la compresión de los registros de salida de las tareas *Map* antes de que sean enviados a las tareas *Reduce*. Esta opción debe estar habilitada si hay una cantidad significativa de datos que están siendo transferidos entre los dos conjuntos de tareas. Tenga en cuenta que la eficacia de este enfoque se controla implícitamente por lo bien que los registros se puedan comprimir y la relación entre el tamaño de los registros, el ancho de banda del clúster y el tiempo que se tarda en comprimir y descomprimir los archivos.

A pesar de que estas son las dos opciones más características que aporta este nuevo objeto para controlar los trabajos *HipiJob*, es recomendable acudir a la API para conocer el resto de los métodos que en el objeto aparecen y que nos pueden ser útiles, así como revisar la herencia entre esta clase y la *Job* principal.

http://hipi.cs.virginia.edu/doc/api/hipi/imagebundle/mapreduce/HipiJob.html

4.4.5. Puesta en Marcha

Una vez que conocemos los aspectos básicos de esta librería, llega el momento de incluirla en nuestro clúster *Apache Hadoop MapReduce*. El requisito, obviamente es tener configurado y en ejecución alguno de los modos de clúster *Hadoop* que ya se expusieron en el *Capítulo 3. Estudio del Entorno Apache Hadoop*.

En el siguiente capítulo pondremos en marcha el clúster *Hadoop* mencionado y configuraremos HIPI tal y como vamos a indicar a continuación; aun así vamos a explicar el procedimiento para incluir la librería en un clúster de este tipo. Hemos de descargar el código fuente de HIPI desde la sección de descargas del portal Web de la librería.

http://hipi.cs.virginia.edu/downloads.html

Disponemos de tres versiones diferentes para descargar:

- Iniciación (*Release Candidate*).
- Código Fuente (*Source Code*).
- Desarrollador (*Developer*).

La versión *Developer* contiene el código fuente completo, incluyendo programas de ejemplo y programas experimentales que pueden utilizarse para realizar pruebas de rendimiento de HIPI. La versión *Source Code* dispone del árbol completo de código fuente de HIPI, pero no contiene los experimentos. Por último, la versión de iniciación solo contiene los archivos necesarios para utilizar HIPI junto con el código fuente de los programas de ejemplo.

Para utilizar la API HIPI, debe agregar el archivo '.jar' que ha descargado anteriormente al directorio fuente del entorno *Hadoop*.

```
hduser@nodomaster:/usr/local/hadoop$
```

Figura 4.18. Directorio *Hadoop*

La forma de configurar la instalación de HIPI difiere según el tipo de sistema. En el caso que nos ocupa, un clúster *Apache Hadoop Multinode* basta con descomprimir el archivo en el directorio mencionado y modificar el fichero "*build.xml*". Este fichero de configuración contiene la ruta de las librerías que queremos utilizar en el entorno de programación. Ha de añadir la ruta en la que se descomprima la librería. A continuación, en la *Figura 4.19,* puede ver cuál es el fichero que estamos mencionando.

```
hduser@nodomaster:/usr/local/hadoop$ ls
bin             docs                        ivy             NOTICE.txt
build.xml       hadoop-0.20.2-ant.jar       ivy.xml         README.txt
c++             hadoop-0.20.2-core.jar      lib             src
CHANGES.txt     hadoop-0.20.2-examples.jar  librecordio     webapps
conf            hadoop-0.20.2-test.jar      LICENSE.txt
contrib         hadoop-0.20.2-tools.jar     logs
```

Figura 4.19. Contenido Directorio *Hadoop*

El contenido del mismo aparece en la *Figura 4.20*.

```
<project basedir="." default="all">

<target name="setup">
<property name="hadoop.home" value="/hadoop/hadoop-0.20.1" />
<property name="hadoop.version" value="0.20.1" />
<property name="hadoop.classpath" value="${hadoop.home}/hadoop-${hadoop.version}-core.jar" />
<property name="metadata.jar" value="3rdparty/metadata-extractor-2.3.1.jar" />
</target>
...
```

Figura 4.20. *build.xml*

Se ha de incluir en las propiedades "*hadoop.home"* y *"hadoop.version"* el directorio y la versión de nuestra instalación de *Hadoop*. Así ya tendríamos dispuesta la librería HIPI para la ejecución de aplicaciones *MapReduce* orientadas a la computación sobre imágenes.

En el capítulo siguiente llevaremos a cabo la ejecución de los ejemplos que incluye la librería para así probar su funcionamiento.

4.5. Implementaciones *MapReduce*

A lo largo del capítulo hemos realizado un estudio de la programación *MapReduce* enfocándola exclusivamente al entorno de computación distribuida *Apache Hadoop*. Sin embargo, puede encontrar en el mercado otros marcos de trabajo que utilizan la programación *MapReduce*. A continuación sintetizamos en una tabla algunas de los ejemplos más notables que puede encontrar, detallando brevemente algunas de sus características principales.

Distribución	Características	SFD	Licencia	Web
Amazon Elastic MapReduce 	Servicio web que permite usar instancias de máquinas virtuales con *framework Hadoop*. Disponibles distintas versiones de máquinas, eso sí, de pago.	HDFS	Pago	http://aws.amazon.com/es/elasticmapreduce/
Google MapReduce 	Servicio que permite usar, de manera limitada, la programación *MapReduce*. Es un servicio que puede usarse para iniciarse en *MapReduce*.	GFS	*GNU GPL*	http://code.google.com/p/appengine-mapreduce/ http://code.google.com/intl/es-ES/edu/parallel/mapreduce-tutorial.html
Apache Hadoop MapReduce 	El más popular servicio de licencia libre de *MapReduce* que permite crear tu propio clúster con un solo nodo o en modo multinodo. Es muy flexible ya que permite todo tipo de sistemas y soporta todo tipo de SFD, aunque recomienda el uso del suyo propio.	HDFS	*GNU GPL*	http://hadoop.apache.org/mapreduce/

Aster Data MapReduce 	Versión de *MapReduce* enfocada al trabajo con grandes bases de datos SQL. Integrada con *Hadoop*.	HDFS	Pago. Posibilidad de obtener una versión demo.	http://www.asterdata.com/resources/mapreduce.php
Greenplum MapReduce 	Versión de *MapReduce* enfocada al trabajo en Computación Paralela sobre sistemas de almacenamiento en gran escala. Combinación *MapReduce-SQL*. Integrada con *Hadoop*.	HDFS	Pago.	http://www.greenplum.com/technology/mapreduce
Disco MapReduce 	Entorno de computación distribuida desarrollado por *Nokia Research Center*. Funciona sobre *Debian* o sobre distribuciones basadas en *Debian* como *Ubuntu* en arquitecturas de 64bits.	DDFS	*GNU GPL*	http://discoproject.org/
Holumbus MapReduce 	Creado por Stefan Schmidt para una tesis en la Universidad de Ciencias Aplicadas de Wedel, Alemania. Usa un sistema de ficheros propio, basado en *GFS*. Todo el sistema está basado en *Hadoop* y *Google MapReduce*.	HFS	*GNU GPL*	http://holumbus.fh-wedel.de/trac/wiki/MapReduce

Tabla 4.1. Implementaciones *MapReduce*

Si se fija en la información aportada por la tabla, puede apreciar como varias de las implementaciones que se ofrecen están basadas en *Hadoop* e incluso utilizan su sistema de ficheros distribuido HDFS. También aparecen distribuciones enfocadas al trabajo exclusivo con bases de datos. Por su versatilidad, flexibilidad y por ser la más aplicable y extensible de todas las distribuciones, hemos decidido trabajar con ***Apache Hadoop MapReduce***, siendo además *GNU GPL*. Ya destacamos en capítulos anteriores la flexibilidad y adaptabilidad de los entornos *Apache Hadoop,* configurables sobre casi cualquier tipo de equipo. También se mencionó su

escalabilidad, permitiéndonos desplegar clústeres de un solo nodo y ampliarlos hasta límites de miles de ellos.

Por ello, concluimos que la mejor forma de iniciarnos en el desarrollo de aplicaciones *MapReduce* es hacerlo utilizando el marco de trabajo con mayor soporte, más extendido y que además presenta las mejores características, presentando éste además su sistema de ficheros distribuido propio.

En el capítulo siguiente, vamos a desplegar el clúster *Apache Hadoop MapReduce* que se ha expuesto e incluiremos en él la librería HIPI tal y como se ha mostrado en el apartado anterior.

5. Puesta en Marcha del Entorno *Apache Hadoop/MapReduce*

En los capítulos *1. Introducción a Cloud Computing* y *3. Estudio del Entorno Apache Hadoop,* se expuso la base teórica con la que adquirir el conocimiento suficiente para poder llevar a cabo uno de los objetivos del proyecto, poner en marcha un clúster *Apache Hadoop*. Por otro lado, en el *Capítulo 2. Puesta en Marcha del Cloud* se desplegó la nube que forma la base para el clúster *Apache Hadoop* que vamos a configurar.

El entorno de computación distribuida *Apache Hadoop*, dadas sus características, se adapta perfectamente al paradigma de computación *Cloud Computing*, ya que es altamente flexible y permite instalarse y configurarse en cualquier equipo (incluso virtuales) o sistema operativo. Permite añadir o eliminar nodos al clúster sin que el funcionamiento se vea afectado, solo afecta el rendimiento. Este tipo de operaciones dentro de un sistema *Cloud Computing* son realmente ágiles y sencillas, por lo que puede apreciar que el trabajo conjunto de estos dos entornos de trabajo es realmente apropiado.

En este capítulo va a poder ver como formar un clúster con múltiples nodos, tantos como las limitaciones de nuestra nube nos permita. Crear un clúster *Hadoop* parte de la instalación y configuración del entorno en un único servidor conformando un clúster con nodo único. Vamos a configurar una máquina virtual con el modo *Apache Hadoop Single Node*, el clúster de *Apache Hadoop* más sencillo. El segundo paso, es crear una imagen de esa máquina virtual para crear distintas instancias virtuales de la misma. Teniendo varias instancias virtuales en funcionamiento, se forma un clúster multinodo, que es el modo de computación más potente de *Apache Hadoop*.

5.1. *Hadoop Single Node*

Para la instalación de *Apache Hadoop* en modo *Single Node*, disponemos de una máquina virtual con sistema operativo *Ubuntu Server* 10.04.2 LTS (Lucid Lynx). Esta máquina virtual proviene de la imagen instalada en el *Capítulo 2. Puesta en Marcha del Cloud* en el que ejecutamos un *script* que descargó dicha imagen. Lo que ha de hacer es poner en marcha una instancia de de la imagen y conectarse a ella vía SSH, así puede comenzar a configurarla. Todo el proceso de ejecutar una instancia y conectarse a la misma quedó explicado en el *Apartado 2.6. Desplegando Instancias*, por ello, en esta ocasión pasamos directamente a configurar el entorno *Apache Hadoop*.

Esta distribución puede implantarse sobre cualquier distribución *GNU/Linux* o *Win32,* por tanto disponemos de un entorno perfectamente válido. Previo a la configuración del framework *Hadoop*, ha de cumplir una serie de requisitos previos, que puede ver a continuación.

Durante todo el proceso de instalación se van a incluir figuras para hacer más sencillo el procedimiento.

5.1.1. Prerrequisitos

Pasamos a describir todos los requisitos que la instalación de *Hadoop* precisa y a realizar los pasos que sea necesario aplicar a la instancia virtual para cumplirlos.

Sun Java 6

Hadoop requiere la máquina de *Java* a partir de su versión 1.5.x instalada. Sin embargo, es recomendable instalar la versión 1.6.x.

Puede instalar *Java* ejecutando las siguientes instrucciones

```
>> sudo add-apt-repository "deb http://archive.canonical.com/ lucid
partner"

>> sudo apt-get update

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

>> sudo update-java-alternatives -s java-6-sun
```

El JDK completo se emplaza en el directorio */usr/lib/jvm/java-6-sun*. Esta ruta ha de recordarla, ya que es necesario indicarla en algunos archivos de configuración de *Hadoop.*

Para comprobar que la instalación es correcta, ejecute el comando que aparece en la *Figura 5.1*:

```
hduser@nodomaster:~$ java -version
java version "1.6.0_26"
Java(TM) SE Runtime Environment (build 1.6.0_26-b03)
Java HotSpot(TM) 64-Bit Server VM (build 20.1-b02, mixed mode)
hduser@nodomaster:~$
```

Figura 5.1. Versión *Java*

Crear un usuario exclusivo para Hadoop

Vamos a crear un usuario dedicado para *Hadoop*, cuya cuenta disponga de acceso a todos los ficheros de la instalación. Esto no es algo necesario, pero si recomendable, ya que mantiene aislada la configuración del resto de usuarios o de aplicaciones software que se encuentren en nuestro equipo. El usuario se llama `hduser` y el grupo al que pertenece `hadoop`. Para crear el usuario ejecute:

```
>> sudo addgroup hadoop
```

Posteriormente, cree el grupo y acceda con el nuevo usuario (véase *Figura 5.2*):

```
root@hadoopsinglenode:~# addgroup hadoop
Adding group `hadoop' (GID 1001) ...
Done.
root@hadoopsinglenode:~# adduser --ingroup hadoop hduser
```

Figura 5.2. Añadir Grupo

Además, otórguele permisos `root` al usuario `hduser` modificando el archivo de configuración */etc/sudoers* (véase *Figura 5.3*) .

```
root@hadoopsinglenode:~# vi /etc/sudoers
root@hadoopsinglenode:~#
```

Figura 5.3. Editar *sudoers*

```
# /etc/sudoers
# This file MUST be edited with the 'visudo' command as root.
# See the man page for details on how to write a sudoers file.

Defaults        env_reset

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL) ALL

# Privilegios usuario hduser
hduser  ALL=(ALL) NOPASSWD:ALL

# Uncomment to allow members of group sudo to not need a password
# (Note that later entries override this, so you might need to move
# it further down)
# %sudo ALL=NOPASSWD: ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# ubuntu user is default user in ec2-images.
# It needs passwordless sudo functionality.
ubuntu  ALL=(ALL) NOPASSWD:ALL
```

Figura 5.4. Fichero *sudoers*

Configuración SSH

El siguiente paso es tener bien configurado *SSH*, ya que *Hadoop* necesita acceso para controlar los nodos. En el caso *Single Node* ha de configurar el acceso *SSH* a `localhost` para el usuario `hduser` creado anteriormente.

Teniendo el servicio *SSH* iniciado, cree una clave *RSA* para la autenticación del `hduser`. Como se puede ver en la *Figura 5.5*, creamos la clave con contraseña vacía. Por lo general esto no es recomendable, pero en nuestro caso debemos hacerlo así, ya que hay que permitir el acceso automático a *Hadoop*, sin interacción de ningún usuario. Las equipos que componen el clúster se comunican entre sí usando éste servicio.

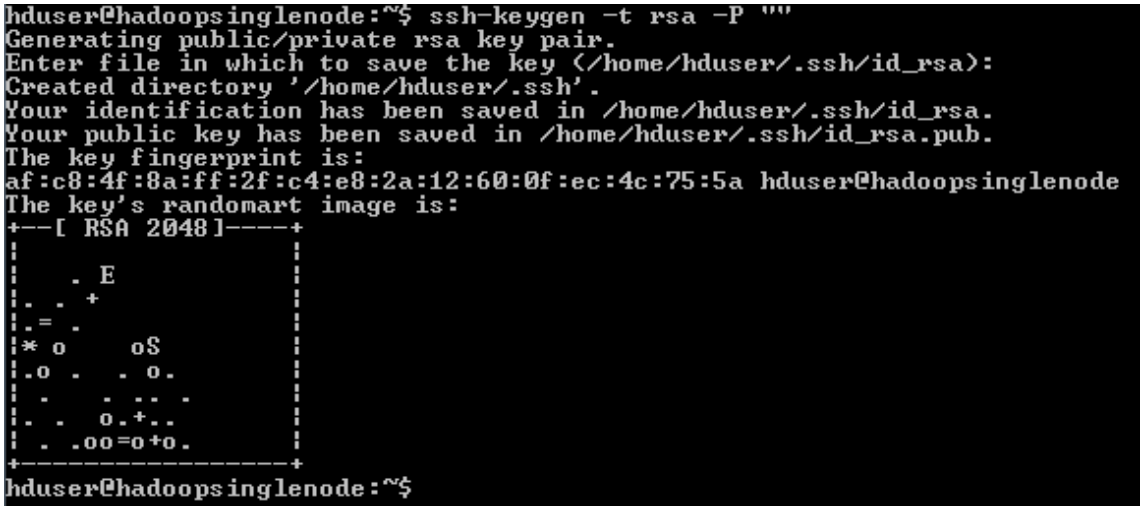


Figura 5.5. Crear Clave SSH

El siguiente paso es permitir el acceso *SSH* al host local con la clave creada (véase la *Figura 5.6*).

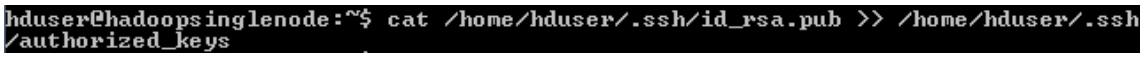


Figura 5.6. Acceso clave RSA

Finalmente, compruebe la conexión `localhost` para el usuario `hduser` se puede establecer correctamente (véase la *Figura 5.7*). En este paso es necesario almacenar la huella digital de la clave del host en el archivo `known_hosts` que se encuentra en el directorio *.shh/* del usuario `hduser`.

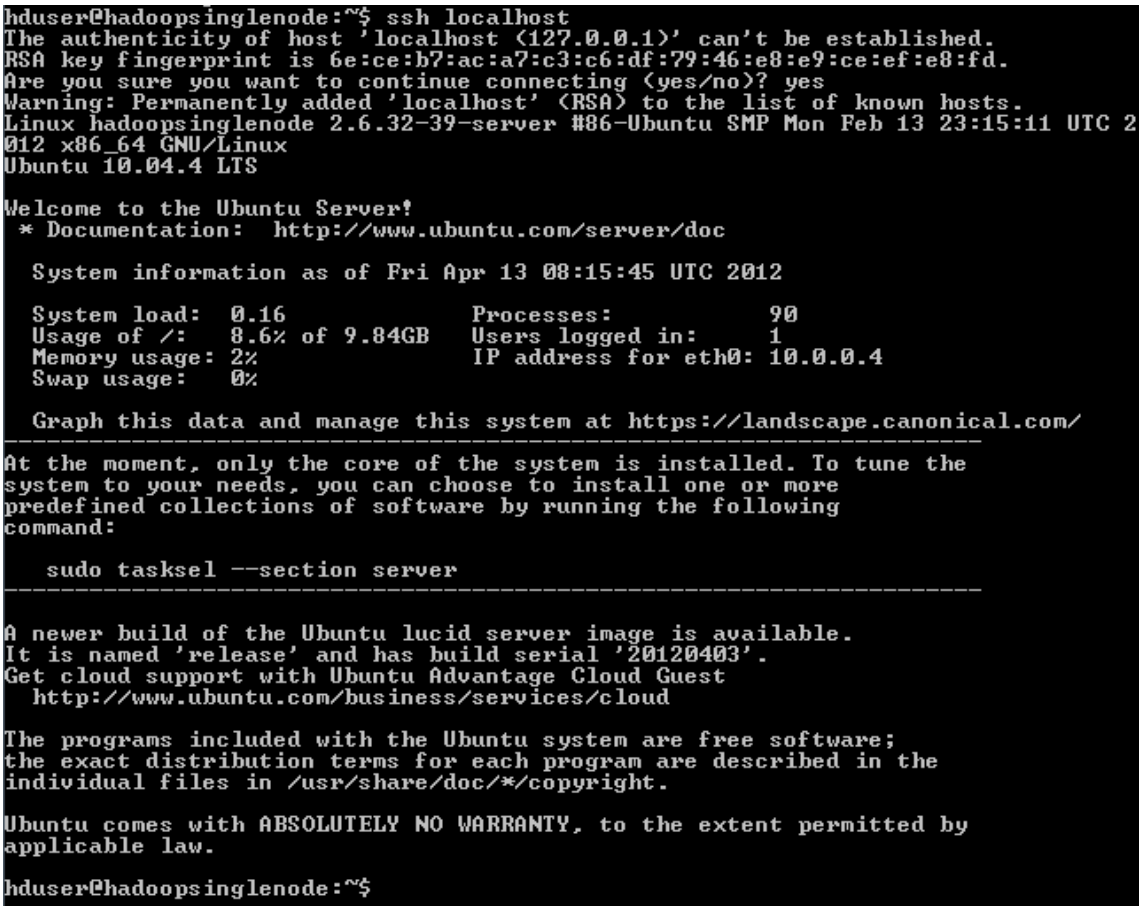


Figura 5.7. Comprobar Acceso *Localhost*

Al establecer la conexión vemos como automáticamente se añade la clave a la lista de `known_hosts`. Con esto ha terminado la configuración *SSH*. Escriba el comando `exit` para volver al nodo local.

Deshabilitar IPv6

Un problema relacionado con IPv6 en *Ubuntu* es que al usar la dirección IPv4 0.0.0.0 en algunos de los pasos de configuración de red de *Hadoop Single Node* puede dar lugar a que *Ubuntu* utilice direcciones IPv6 automáticamente y deje de utilizar los valores configurados, dando lugar a errores. Para poder deshabilitar el uso de direcciones IPv6, ha de modificar el archivo */etc/sysctl.conf*, añadiendo las siguientes líneas.

```
hduser@hadoopsinglenode:~$ sudo vi /etc/sysctl.conf
hduser@hadoopsinglenode:~$
```

Figura 5.8. Editar *sysctl.conf*

Añada al archivo las líneas que aparecen al final de la *Figura 5.9*.

```
#################################################################################
# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through
# redirection. Some network environments, however, require that these
# settings are disabled so review and enable them as needed.
#
# Ignore ICMP broadcasts
#net.ipv4.icmp_echo_ignore_broadcasts = 1
#
# Ignore bogus ICMP errors
#net.ipv4.icmp_ignore_bogus_error_responses = 1
#
# Do not accept ICMP redirects (prevent MITM attacks)
#net.ipv4.conf.all.accept_redirects = 0
#net.ipv6.conf.all.accept_redirects = 0
# _or_
# Accept ICMP redirects only for gateways listed in our default
# gateway list (enabled by default)
# net.ipv4.conf.all.secure_redirects = 1
#
# Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1

# Disable IPv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Figura 5.9. Fichero *sysctl.conf*

Es necesario reiniciar el sistema para que los cambios surtan efecto. Tras hacerlo, puede comprobar que se ha deshabilitado con éxito con el comando que puede ver en la *Figura 5.10*.

```
hduser@hadoopsinglenode:~$ cat /proc/sys/net/ipv6/conf/all/disable_ipv6
1
hduser@hadoopsinglenode:~$
```

Figura 5.10. Comprobar IPv6 Deshabilitada

En caso de devolver 0, no habría surtido efecto el cambio.

5.1.2. Instalación y Configuración de *Hadoop Single Node*

Tras cumplir todos los requisitos previos que se exigen desde la documentación oficial de *Apache Hadoop*, puede comenzar a realizar la instalación del entorno de un solo nodo de computación.

Al igual que con los prerrequisitos, iremos describiendo paso a paso los cambios en la configuración que vamos a ir aplicando sobre nuestra máquina virtual.

Descarga del software

El primer paso para la instalación consiste en descargar *Apache Hadoop* de *Apache Download Mirrors* (http://www.apache.org/dyn/closer.cgi/hadoop/core), y descomprimir el paquete que se obtiene de la descarga en el directorio */usr/local/hadoop*. También es recomendable asegurarse de que todos los archivos son propiedad del usuario `hduser` y del grupo `hadoop`.

En la *Figura 5.11* se muestra cómo descargar la versión de *Hadoop* en el directorio */usr/local*. Tras la descarga, se descomprime y se mueve al directorio */usr/local/hadoop*. Para asegurarse de que todo el contenido del directorio es del usuario `hduser` ejecute el comando `'chown'` y evitará problemas de permisos de acceso, escritura o lectura.

```
hduser@hadoopsinglenode:~$ cd /usr/local/
hduser@hadoopsinglenode:/usr/local$ sudo wget http://apache.rediris.es/hadoop/co
re/hadoop-0.20.2/hadoop-0.20.2.tar.gz
--2012-04-13 08:27:40--  http://apache.rediris.es/hadoop/core/hadoop-0.20.2/hado
op-0.20.2.tar.gz
Resolving apache.rediris.es... 130.206.1.5
Connecting to apache.rediris.es|130.206.1.5|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 44575568 (43M) [application/x-gzip]
Saving to: `hadoop-0.20.2.tar.gz'

100%[======================================>] 44,575,568  1.32M/s   in 24s

2012-04-13 08:28:04 (1.77 MB/s) - `hadoop-0.20.2.tar.gz' saved [44575568/4457556
8]

hduser@hadoopsinglenode:/usr/local$ sudo tar xzf hadoop-0.20.2.tar.gz
hduser@hadoopsinglenode:/usr/local$ sudo mv hadoop-0.20.2 hadoop
hduser@hadoopsinglenode:/usr/local$ sudo chown -R hduser:hadoop hadoop
hduser@hadoopsinglenode:/usr/local$
```

Figura 5.11. Descarga *Apache Hadoop*

Modificar el archivo $HOME/.bashrc

A continuación, ha de añadir una serie de variables al archivo de configuración `.bashrc` que sirven para indicar al sistema cuál es el directorio del entorno, dónde se encuentra instalado *Java,* el tipo de compresión del sistema de ficheros y el directorio donde se exportarán los ejecutables de *Hadoop*. También debe cambiar dos alias referentes a dos comandos de *Hadoop* que sirven para ejecutar acciones sobre el sistema de ficheros; por ejemplo estos alias, acompañados de un directorio, nos mostrarían su contenido, similar a "*ls*" en *Linux* (véase la *Figura 5.13*).

```
hduser@hadoopsinglenode:/usr/local$ sudo vi /home/hduser/.bashrc
hduser@hadoopsinglenode:/usr/local$
```

Figura 5.12. Editar fichero *bashrc*

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop

# Set JAVA_HOME
export JAVA_HOME=/usr/lib/jvm/java-6-sun

# Some convenient aliases and functions for running Hadoop-related commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"

# LZO compression
lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}

# Add Hadoop /bin directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

Figura 5.13. Fichero *bashrc*

Configuración del Sistema

Es necesario modificar los archivos de configuración para adaptarlos a nuestro sistema. Todos los archivos de configuración que se van a modificar se encuentran en el directorio */usr/local/hadoop/conf*.

- **hadoop-env.sh.** En este archivo aparecen varias variables de entorno. En nuestro caso solo necesita modificar la variable que contiene el directorio del *JDK* de *Java*.

```
hduser@hadoopsinglenode:/usr/local/hadoop/conf$ sudo vi hadoop-env.sh
hduser@hadoopsinglenode:/usr/local/hadoop/conf$
```

Figura 5.14. Editar *hadoop-env.sh*

Una vez abierto el fichero, sustituya

```
# The java implementation to use.  Required.
# export JAVA_HOME=/usr/lib/j2sdk1.5-sun
```

Figura 5.15. Antes de modificar *hadoop-env.sh*

por

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

Figura 5.16. Después de modificar hadoop-env.sh

- **core-site.xml.** Permite definir el directorio de archivos temporales de *Hadoop* y su localización. Además se define el nombre con el que el sistema de ficheros (HDFS) se identifica en el clúster.
  Lo primero es crear el directorio temporal y asignarlo a `hduser`. La localización del directorio temporal es */usr/local/hadoop/tmp*.

```
hduser@nodomaster:/usr/local/hadoop$ mkdir -p tmp
hduser@nodomaster:/usr/local/hadoop$ chown hduser:hadoop tmp
hduser@nodomaster:/usr/local/hadoop$ chmod 750 tmp
hduser@nodomaster:/usr/local/hadoop$
```

Figura 5.17. Crear Directorio Temporal

Una vez creado el directorio temporal, edite el fichero de configuración y a añadir las dos variables mencionadas que sirven para indicar cuál es el directorio de archivos temporales (*hadoop.tmp.dir*) y la localización del *NameNode* del sistema de ficheros (*fs.default.name*).

```
hduser@hadoopsinglenode:/usr/local/hadoop/conf$ sudo vi core-site.xml
hduser@hadoopsinglenode:/usr/local/hadoop/conf$
```

Figura 5.18. Editar core-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
   <name>hadoop.tmp.dir</name>
   <value>/usr/local/hadoop/tmp</value>
   <description>A base for other temporary directories.</description>
</property>

<property>
   <name>fs.default.name</name>
   <value>hdfs://localhost:54310</value>
   <description>The name of the default File System.</description>
</property>
</configuration>
~
```

Figura 5.19. Fichero core-site.xml

- **mapred-site.xml.** Permite indicar dónde se encuentra el *JobTracker*.

```
hduser@hadoopsinglenode:/usr/local/hadoop/conf$ sudo vi mapred-site.xml
hduser@hadoopsinglenode:/usr/local/hadoop/conf$
```

Figura 5.20. Editar *mapred-site.xml*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
   <name>mapred.job.tacker</name>
   <value>localhost:54311</value>
   <description>The host and port that the MapReduce job tracker runs at </descr
iption>
</property>
</configuration>
~
~
```

Figura 5.21. Fichero *mapred-site.xml*

- **hdfs-site.xml.** Permite definir el número de veces que desee que se repliquen los datos en el sistema. En este caso, al estar definiendo un clúster con un solo nodo, debe indicar el valor de replicación uno.

```
hduser@hadoopsinglenode:/usr/local/hadoop/conf$ sudo vi hdfs-site.xml
hduser@hadoopsinglenode:/usr/local/hadoop/conf$
```

Figura 5.22. Editar *hdfs-site.xml*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
   <name>dfs.replication</name>
   <value>1</value>
   <description>Default block replication</description>
</property>
</configuration>
~
```

Figura 5.23. Fichero *hdfs-site.xml*

5.1.3. Formatear el Sistema de Ficheros HDFS desde el *NameNode*

Una vez aplicados todos los cambios a los ficheros de configuración, el paso a seguir, previo a arrancar los servicios de *Hadoop*, es formatear el sistema de ficheros.

El formateo se realiza desde el *NameNode* y al ser un clúster *Single Node*, todo se hace desde la misma máquina. Se ha de ejecutar el siguiente comando:

```
hadoop/bin>> hadoop namenode -format
```

Véa el resultado en la *Figura 5.24*.

```
hduser@hadoopsinglenode:/usr/local/hadoop/bin$ hadoop namenode -format
12/04/13 09:01:32 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = hadoopsinglenode.novalocal/10.0.0.4
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 0.20.2
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/b
ranch-0.20 -r 911707; compiled by 'chrisdo' on Fri Feb 19 08:07:34 UTC 2010
************************************************************/
12/04/13 09:01:32 INFO namenode.FSNamesystem: fsOwner=hduser,hadoop
12/04/13 09:01:32 INFO namenode.FSNamesystem: supergroup=supergroup
12/04/13 09:01:32 INFO namenode.FSNamesystem: isPermissionEnabled=true
12/04/13 09:01:32 INFO common.Storage: Image file of size 96 saved in 0 seconds.
12/04/13 09:01:32 INFO common.Storage: Storage directory /usr/local/hadoop/tmp/d
fs/name has been successfully formatted.
12/04/13 09:01:32 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoopsinglenode.novalocal/10.0.0.4
************************************************************/
hduser@hadoopsinglenode:/usr/local/hadoop/bin$
```

Figura 5.24. Formatear *NameNode*

Tras formatear el nodo, puede arrancar el clúster. Hay varias formas de arrancar el clúster, pero una de ellas arranca todos los nodos del sistema. El comando que arranca todas las funciones del sistema es (véase la *Figura 5.25*):

```
hadoop/bin>> start-all.sh
```

```
hduser@hadoopsinglenode:/usr/local/hadoop/bin$ start-all.sh
starting namenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-nameno
de-hadoopsinglenode.out
localhost: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hd
user-datanode-hadoopsinglenode.out
localhost: starting secondarynamenode, logging to /usr/local/hadoop/bin/../logs/
hadoop-hduser-secondarynamenode-hadoopsinglenode.out
starting jobtracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-jobt
racker-hadoopsinglenode.out
localhost: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop
-hduser-tasktracker-hadoopsinglenode.out
hduser@hadoopsinglenode:/usr/local/hadoop/bin$
```

Figura 5.25. Iniciar el Clúster *Single Node*

En la *Figura 5.25* puede ver cómo se arrancan *Namenode*, *DataNode*, *JobTracker* y *TaskTracker*. En cualquier caso, puede comprobar que se han iniciado correctamente todos los nodos con la siguiente orden:

```
hadoop/bin >> jps
```

```
hduser@hadoopsinglenode:/usr/local/hadoop/bin$ jps
1457 Jps
1145 SecondaryNameNode
989 DataNode
1353 TaskTracker
1211 JobTracker
841 NameNode
hduser@hadoopsinglenode:/usr/local/hadoop/bin$
```

Figura 5.26. JPS Clúster *Single Node*

El comando *jps*, devuelve todas las máquinas virtuales de *Java* que haya ejecutándose en el sistema. Además indica su identificador.

5.2. Hadoop Multinode

Como hemos comentado en la introducción, el objetivo final es presentar la creación de un clúster *Apache Hadoop Multinode* configurando varias instancias virtuales dentro de un *Cloud OpenStack* desplegado gracias a *StackOps*. El primer paso previo a la creación de un clúster *Hadoop* es configurar un nodo con *Hadoop Single Node*. Ese nodo funciona como controlador del clúster aunque puede ser también configurado como nodo esclavo.

Teniendo instalado *Hadoop* en modo *Single Node* en un servidor, puede configurar varias máquinas virtuales para que trabajen juntas formando un clúster. Para ello ha de crear una imagen de esa instancia. A partir de la imagen se despliegan nuevas máquinas virtuales y realizando los pasos necesarios de configuración, se conforma el clúster.

Para alcanzar el máximo rendimiento, vamos a desplegar tantas instancias virtuales como los recursos de nuestra nube permitan. Así conseguimos crear un clúster con el máximo número de procesadores y la mayor cantidad de almacenamiento.

Una vez estén todas las máquinas virtuales que se pretende que conformen el clúster ejecutándose en la nube, ha de aplicar los pasos de configuración necesarios en cada una de ellas y así crear el entorno de trabajo distribuido *Apache Hadoop Multinode*.

5.2.1. Crear Imagen de la Instancia *Hadoop Single Node*

El primer paso es utilizar la máquina virtual en la que hemos configurado *Hadoop Single Node* como patrón para el resto de las instancias que decidimos que conformen el clúster. Sabemos que la instancia está ejecutándose en nuestro clúster, pues bien, *OpenStack* dispone de un método para crear imágenes de instancias en ejecución utilizando *KVM* y *XEN*. Ha de entrar como administrador al *frontend* de nuestra nube *StackOps*. Para este caso, en vez de acceder vía web, ya que podemos hacerlo, vamos a trabajar directamente sobre el servidor que está configurado como "nodo controlador". Primero ha de comprobar que tiene la herramienta "*qemu-img 0.14*" o posterior. Esta herramienta puede usarse para dar formato a imágenes de huéspedes virtualizadas, dispositivos de almacenaje adicional y almacenamiento de redes. Compruebe que esta herramienta está instalada correctamente y en la versión indicada.

La principal ventaja que se obtiene de crear la imagen de la instancia ya configurada, es que, para formar el clúster multinodo, puede aprovechar la configuración *Hadoop* realizada, utilizando máquinas que ya tienen instalado el clúster *Single Node* que, por tanto, ya cumplen con los requisitos previos a la instalación y además solo requieren los pasos de configuración propios del modo *Multinode.*

Ejecute el siguiente comando,

```
>> dpkg -l | grep qemu
```

Ha de obtener un resultado similar al de la *Figura 5.27*.

```
ii  qemu            0.14.0~rc1+noroms-0ubuntu4~ppalucid1     dummy transitional pacakge from qemu to qemu
ii  qemu-common     0.14.0~rc1+noroms-0ubuntu4~ppalucid1     qemu common functionality (bios, documentati
ii  qemu-kvm        0.14.0~rc1+noroms-0ubuntu4~ppalucid1     Full virtualization on i386 and amd64 hardwa
```

Figura 5.27. *grep qemu*

Para poder crear la imagen de la máquina virtual en ejecución, ha de listar todas las que están ejecutándose en el sistema y guardar el identificador de la que quiera almacenar como patrón de desarrollo del sistema multinodo. Para listar las máquinas virtuales, puede, bien acceder vía web a la nube o desde el mismo servidor configurado como "nodo controlador" ejecutando:

```
>> nova list
```

Obtendrá algo similar a lo que aparece en la *Figura 5.28*.

```
+-----+------------------------+--------+--------------------+
|  ID |          Name          | Status |      Networks      |
+-----+------------------------+--------+--------------------+
|  1  | InstanciaUbuntu1004 | ACTIVE | private=10.0.0.2   |
+-----+------------------------+--------+--------------------+
```

Figura 5.28. Lista Instancias

Como hemos comentado antes, accediendo a la web de administración de la nube, puede ver las instancias que están ejecutándose y, en definitiva, acceder a los mismos datos que con "*nova list".* Acceda al apartado *"instances"* (véase la *Figura 5.29*) y así puede ver las imágenes activas.

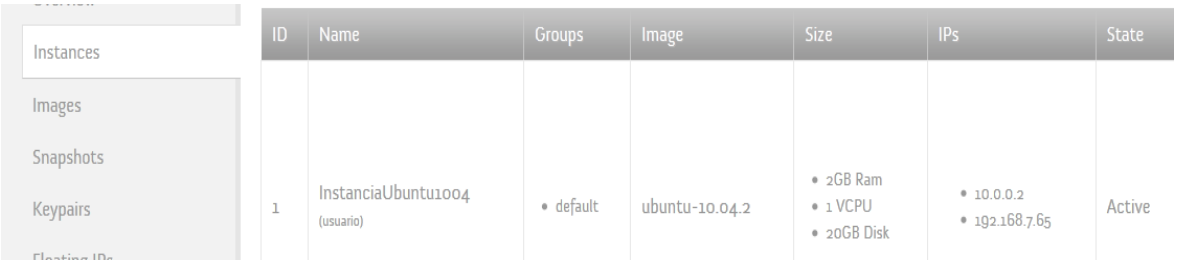


Figura 5.29. Instancias en *Horizon*

En cualquier caso, de una forma u otra, guarde el número identificador de la instancia y ejecute la siguiente línea de código directamente sobre el *frontend* del *Cloud*:

```
>> nova image-create id_Instancia Nombre_Imagen
```

Siguiendo con el primer ejemplo, ejecute la siguiente línea:

```
>> nova image-create 1 hadoopSinglenode
```

Una vez creada la imagen de nuestra instancia virtual activa, ejecute:

```
>> nova image-list
```

y así verá todas las imágenes almacenadas en el sistema. Inicialmente aparecerá la imagen recién creada con estado *"SAVING"*. Posteriormente pasará a estado *"ACTIVE"*, esto indica que ya está lista para ser iniciada tantas veces como desee, pudiendo crear a partir de ella nuevas máquinas virtuales.

El resultado es el que se aprecia en la *Figura 5.30*.

```
+----+-----------------------------------------+--------+
| ID |                   Name                  | Status |
+----+-----------------------------------------+--------+
| 20 | hadoopSinglenode                        | ACTIVE |
| 6  | ttylinux-uec-amd64-12.1_2.6.35-22_1-vmlinuz | ACTIVE |
| 7  | ttylinux-uec-amd64-12.1_2.6.35-22_1-initrd  | ACTIVE |
| 8  | ttylinux-uec-amd64-12.1_2.6.35-22_1.img     | ACTIVE |
+----+-----------------------------------------+--------+
```

Figura 5.30. Instancia Creada

Ya dispone de la imagen de la instancia en ejecución creada y configurada con *Hadoop Single Node*. Puede crear tantas instancias como desee y éstas serán copias virtuales de la imagen que acaba de crear, todas con la misma configuración.

5.2.2. Instanciación de Máquinas Virtuales para Formar el Clúster

Para aprovechar al máximo los recursos de la nube y así poder alcanzar el máximo rendimiento del clúster, vamos a crear tantas máquinas virtuales como los recursos de nuestra nube permitan.

Lo primero es conocer con qué recursos se cuenta. Como vimos en el *Capítulo 3. Estudio del Entorno Apache Hadoop*, la interfaz Web *Horizon* de *OpenStack* nos ofrece un resumen en el Panel de Sistema con todos los recursos hardware de la nube. En ese resumen, puede ver la siguiente información (véase la *Figura 5.31*).

Figura 5.31. Estado de los Recursos del Clúster

Contamos por tanto con veinte procesadores, casi veinte gigas de memoria RAM y algo más de un terabyte de disco duro. Inicialmente, puede ver que todos los recursos están libres.

A continuación, pasamos a desplegar instancias de la imagen creada en el apartado anterior. Para ello, vaya a la sección *"Images"* en el *"User Dashboard"* (véase la *Figura 5.32*) que es desde donde se ponen en marcha dichas instancias. Busque la que fue creada en el apartado anterior, llamada "*hadoopSingleNode*" y pulse sobre el botón *"Launch"* para desplegar nuevas instancias. Todo este proceso esta explicado al detalle en el *Apartado 2.6. Desplegando Instancias*.

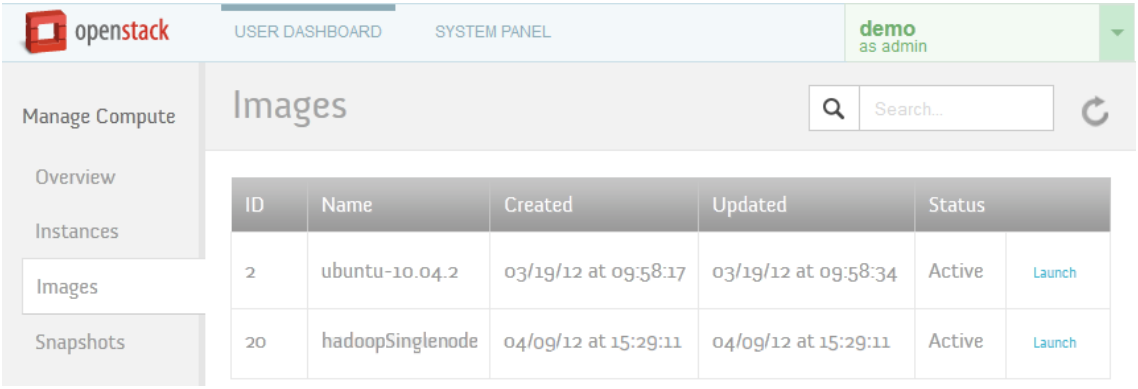


Figura 5.32. Lista Imágenes Horizon

Al ejecutar una nueva instancia, se abre una ventana que contiene el siguiente formulario (véase la *Figura 5.33*). Tiene que rellenarlo tal y como se hizo en el *Capítulo 2. Puesta en Marcha del Cloud*, pero en este caso, aparece la propiedad *"Flavor"* en la que se selecciona el tipo de instancia que se quiere generar.

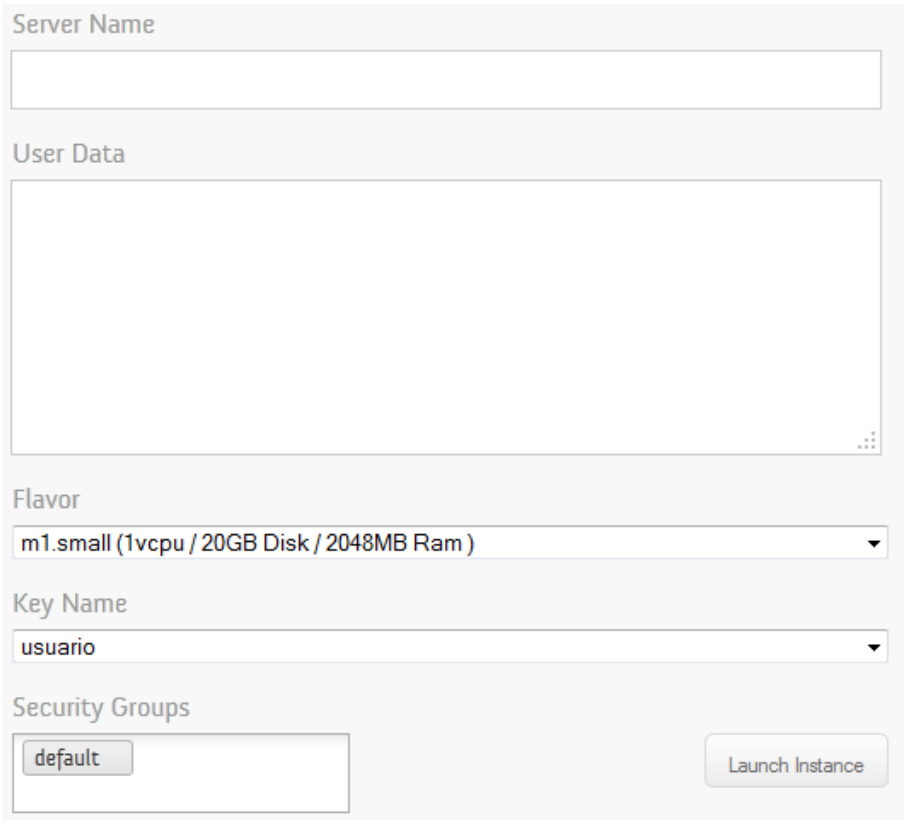


Figura 5.33. Nueva Instancia

Es muy importante decidir el tipo de máquinas virtuales que va a desplegar, ya que buscamos obtener el máximo rendimiento del *Cloud*, instanciando tantas veces como podamos la imagen, sumando entre las instancias los máximos recursos.

El siguiente paso, es conocer el tipo de nodo que puede crear, es decir, los recursos que consumen cada una de ellas. Así puede elegir la que más se adapte a sus necesidades y con la que más pueda aprovechar los recursos hardware. Es decir, en este tipo de clústeres es

preferible crear máquinas algo menos potentes si así permite crear un número más elevado de ellas. Es mejor crear diez máquinas de un tipo medio que cuatro o cinco más potentes. *OpenStack* ofrece varios tipos de máquinas que podrían adaptarse a nuestros recursos:

- ***Tiny*:** 1 procesador con 512Mb de RAM y 0Gb de almacenamiento reservado.
- ***Small*:** 1 procesador con 2Gb de RAM y 20Gb de almacenamiento reservado.
- ***Medium*:** 1 procesadores con 4Gb de RAM y 40Gb de almacenamiento reservado.
- ***Large*:** 1 procesadores con 8Gb de RAM y 80Gb de almacenamiento reservado.
- ***XLarge*:** 1 procesadores con 16Gb de RAM y 160Gb de almacenamiento reservado.

De los tipos de máquinas que ofrece *OpenStack*, hay varias de ellas que descartamos en primera instancia. El modelo "*Tiny*" no es útil para el proyecto ya que no posee almacenamiento reservado, por tanto no se puede configurar sobre él el sistema de ficheros distribuido *HDFS*. En la *Tabla 5.*1, podemos ver un resumen de las posibles configuraciones. Contamos con veinte procesadores y algo menos de veinte gigabytes de RAM, con lo que si utiliza máquinas "*XLarge*" solo se puede desplegar una de ellas (limitación de RAM). Las instancias tipo "*Large*" tampoco parecen ser las más adecuadas, ya que debido a la cantidad de RAM disponible, solo es posible desplegar dos de ellas. Algo similar ocurre con las de tipo "*Medium*", al contar con cuatro gigabytes de RAM, nuestro *Cloud* permite crear cuatro instancias de este tipo y una más tipo "*Small*", en total cinco máquinas. Otra opción es crear todas las máquinas tipo "*Small*".

Clúster	Tipos de Instancias	Hardware Utilizado
1	1 x XLarge 1 x Small	9 Procesadores 18Gb RAM 180Gb HDD
2	2 x Large 1 x Small	9 Procesadores 18Gb RAM 180Gb HDD
3	4 x Medium 1 x Small	9 Procesadores 18Gb RAM 180Gb HDD
4	9 x Small	9 Procesadores 18Gb RAM 180Gb HDD

Tabla 5.1. Posibles Combinaciones de Clúster

Si se fija en la *Tabla 5.1*, puede observar que todos los tipos de clústeres usan la misma cantidad de recursos. Sin embargo, hay dos de ellos que descartamos directamente. Los tipos 1 y 2 no son seleccionados debido a que solo contienen dos y tres máquinas respectivamente, hecho que no permite configurar el HDFS para obtener el máximo rendimiento ni tampoco realizar las réplicas de los datos de una manera efectiva. Por tanto, los clústeres tipo 3 y 4 parecen ser los que más nos convienen. Nos decantamos por el tipo cuatro, ya que cuantas

más máquinas tengamos en el clúster, más se asemeja a un gran sistema distribuido y mejor podremos configurar el HDFS para replicar los datos.

Dado que en el proyecto se pretende realizar distintas pruebas de aplicaciones sobre sistemas distribuidos en un entorno *Cloud Computing*, nos decantamos por formar el clúster que contiene más máquinas para así asemejar el entorno lo más posible a un gran sistema distribuido.

Por tanto, siguiendo los pasos expuestos en el *Capítulo 2. Puesta en Marcha del Cloud*, puede comenzar a desplegar nuevas instancias virtuales hasta completar el número de máquinas que se ha propuesto en el clúster número cuatro de la *Tabla 5.1*.

Finalmente, alcanzamos el objetivo de este apartado, desplegar tantas instancias como permiten los recursos de nuestra nube tratando de obtener el máximo rendimiento e intentando conformar un entorno distribuido lo más real posible. Si vamos al "*Overview*" del "*User Dashboard*" (véase la *Figura 5.34*) podemos ver el resumen de las máquinas virtuales que hemos instanciado.

Server Usage Summary (Show Terminated) Download CSV »

ID	Name	User	VCPUs	Ram Size	Disk Size	Flavor	Uptime	Status
20	NodoMaster	admin	1	2GB	20GB	m1.small	11 horas, 20 minutos	Active
21	NodoEsclavo1	admin	1	2GB	20GB	m1.small	11 horas, 10 minutos	Active
22	NodoEsclavo2	admin	1	2GB	20GB	m1.small	2 horas, 16 minutos	Active
23	NodoEsclavo3	admin	1	2GB	20GB	m1.small	2 horas, 16 minutos	Active
24	NodoEsclavo4	admin	1	2GB	20GB	m1.small	2 horas, 6 minutos	Active
25	NodoEsclavo5	admin	1	2GB	20GB	m1.small	2 horas, 5 minutos	Active
26	NodoEsclavo6	admin	1	2GB	20GB	m1.small	2 horas, 3 minutos	Active
27	NodoEsclavo7	admin	1	2GB	20GB	m1.small	2 horas, 2 minutos	Active
28	NodoEsclavo8	admin	1	2GB	20GB	m1.small	2 horas, 1 minuto	Active

Figura 5.34. Instancias que Van a Formar Clúster

Estas son las instancias virtuales desplegadas a partir de la imagen creada y con las que se va a formar el clúster. Todas ellas son de tipo "*Small*" y están denominadas, una de ellas como Nodo Máster y el resto como Nodos Esclavos. En los siguientes apartados del capítulo, se va a configurar todos los nodos para formar el clúster *Hadoop Multinode*.

5.2.3. Prerrequisitos

Antes de comenzar la configuración del clúster *Hadoop Multinode* conéctese a todas las instancias creadas y abra una consola para cada una de ellas (véase la *Figura 5.35*). Gran parte de las acciones de configuración han de efectuarse sobre cada una de las instancias, así que conviene tener todas preparadas para poder aplicar los cambios.

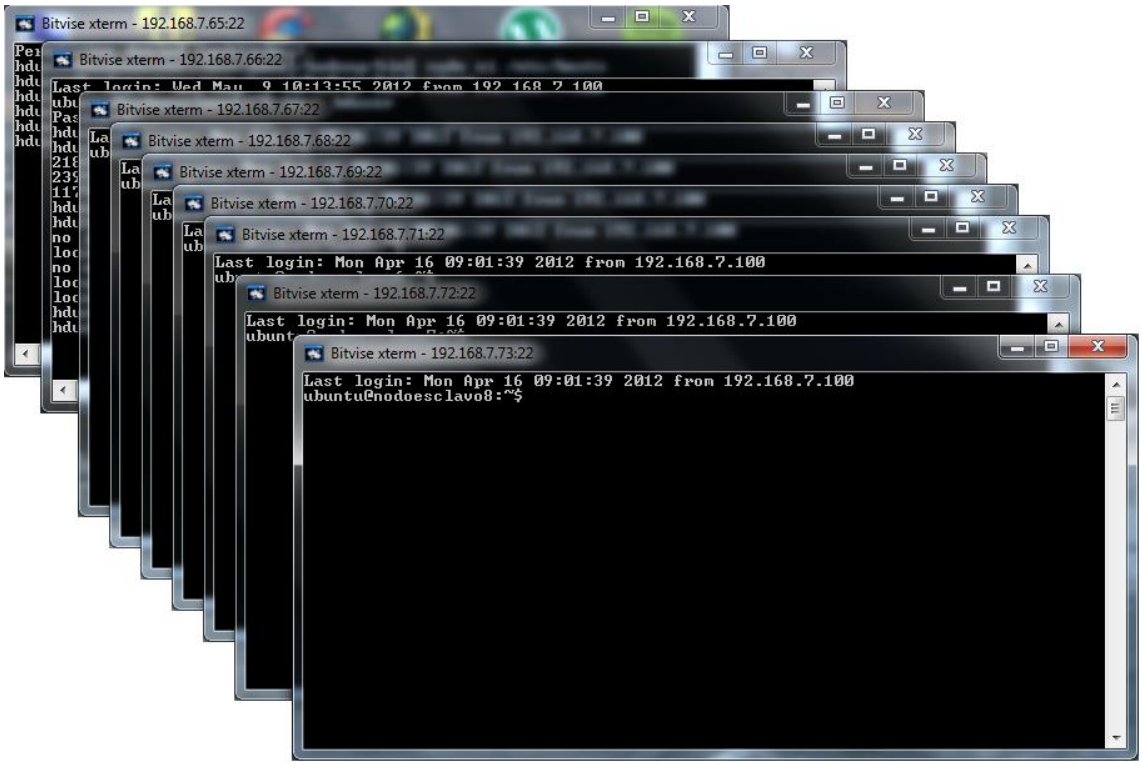


Figura 5.35. Máquinas Virtuales del Clúster

Ya dispone de todas las máquinas preparadas para aplicarle los cambios de configuración que precisen, comenzamos por tanto con los prerrequisitos de la instalación.

Configurar el sistema en modo Single Node

La instalación de *Hadoop Multinode* obliga a que todas las máquinas que conforman el clúster tengan configurado *Hadoop* en modo *Single Node* en ellas. Por ello, podemos decir que este modo precisa menos pasos previos a la instalación, ya que al tener el modo *Single Node* configurado, varios de esos requisitos previos están cumplidos. Esto es así porque las instancias creadas provienen todas de la imagen que grabamos en el *Apartado 5.2.1. Crear Imagen de la Instancia Hadoop Single Node* y que tiene *Hadoop Single Node* instalado. Además, es necesario que en cada una de las máquinas que se vayan a configurar para formar parte del clúster, se detengan todos los servicios de *Hadoop*. Para ello ejecute en cada una de ellas:

```
>> hadoop/bin/stop-all.sh
```

Configuración de la Red

Es necesario modificar todos los ficheros de configuración de red */etc/hosts* para que todos los nodos que van a formar el clúster se reconozcan unos a otros en la misma red. Debe incluir en cada uno de los ficheros de cada una de las instancias las direcciones IP de todos los nodos del clúster. Realice este paso en cada una de las máquinas abriendo el fichero de configuración de red */etc/hosts* (véase la *Figura 5.36*) en modo edición y aplicando los cambios que aparecen a continuación.

```
hduser@nodomaster:/usr/local/hadoop/bin$ sudo vi /etc/hosts
hduser@nodomaster:/usr/local/hadoop/bin$
```

Figura 5.36. Editar */etc/hosts*

Una vez que disponga del fichero abierto, incluya todas las direcciones IP de cada uno de los nodos y añada un alias para identificarlos (véase la *Figura 5.37*).

```
10.0.0.2 master
10.0.0.3 slave1
10.0.0.4 slave2
10.0.0.5 slave3
10.0.0.7 slave4
10.0.0.8 slave5
10.0.0.9 slave6
10.0.0.10 slave7
10.0.0.11 slave8
```

Figura 5.37. Fichero */etc/hosts*

Ha completado el cambio en el Nodo Máster, repita este proceso en cada uno de los Nodos Esclavos con la misma información.

Configuración SSH

Como ya vimos en el modo *Single Node*, es necesario que todas las máquinas que conforman el clúster puedan conectarse vía SSH sin necesidad de interacción del usuario y sin que se solicite ninguna contraseña. Para ello, debe intercambiar las claves entre el nodo máster y el resto de esclavos, proceso que ya realizamos de manera local en la configuración de nodo único.

Intercambie la clave del Nodo Máster con el resto de nodos con el siguiente comando (véase la *Figura 5.38*):

```
>> ssh-copy-id -i /home/hduser/.ssh/id_rsa.pub hduser@slave1
```

```
hduser@nodomaster:~$ ssh-copy-id -i /home/hduser/.ssh/id_rsa.pub hduser@slave1
The authenticity of host 'slave1 (10.0.0.3)' can't be established.
RSA key fingerprint is e8:e2:20:0b:11:a2:ae:f2:25:f8:82:fa:4d:1d:6e:b2.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'slave1,10.0.0.3' (RSA) to the list of known hosts.
Now try logging into the machine, with "ssh 'hduser@slave1'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

Figura 5.38. Intercambiar Claves

Como puede comprobar en la *Figura 5.38*, se envía la clave creada en la configuración anterior para el usuario *'hduser'* al mismo usuario en el primer Nodo Esclavo. Repita esta acción para

cada uno de los Nodos Esclavos, nombrando cada uno de ellos con el alias asignado a cada IP en el apartado anterior *Configuración de la Red*.

Para asegurarse de que le intercambio de claves es efectivo, conéctese mediante SSH tanto al Nodo Máster como a cada uno de los Nodos Esclavos. Empiece conectándose al maestro (véase la *Figura 5.39*).

Figura 5.39. Conexión SSH Máster

En la *Figura 5.39* puede comprobar que la conexión es efectiva, por tanto, escriba en la consola *'exit'* y conéctese a continuación a los esclavos.

Igualmente, puede comprobar en la *Figura 5.40* que ha sido exitosa la conexión, así que vuelva a escribir en la consola el comando *'exit'* y repita el proceso de conexión para el resto de nodos.

Figura 5.40. Conexión SSH Esclavos

Cada vez que se establece una conexión SSH desde un nodo a otro, se agrega al archivo de conexión `known_hosts` la IP, el alias y la clave de intercambio para que en posteriores conexiones no sea necesario aceptar el intercambio de claves, como ha ocurrido en la *Figura 5.40* tratando de conectarnos SSH máster-máster.

5.2.4. Instalación y Configuración *Hadoop Multinode*

Llegados a este punto, comenzamos con la configuración específica de la topología de clúster con múltiples nodos maestro-esclavo (véase la *Figura 5.41*). Antes de comenzar con la instalación, vamos a recordar la estructura de este clúster, como ya hicimos en el *Capítulo 3. Estudio del Entorno Apache Hadoop*.

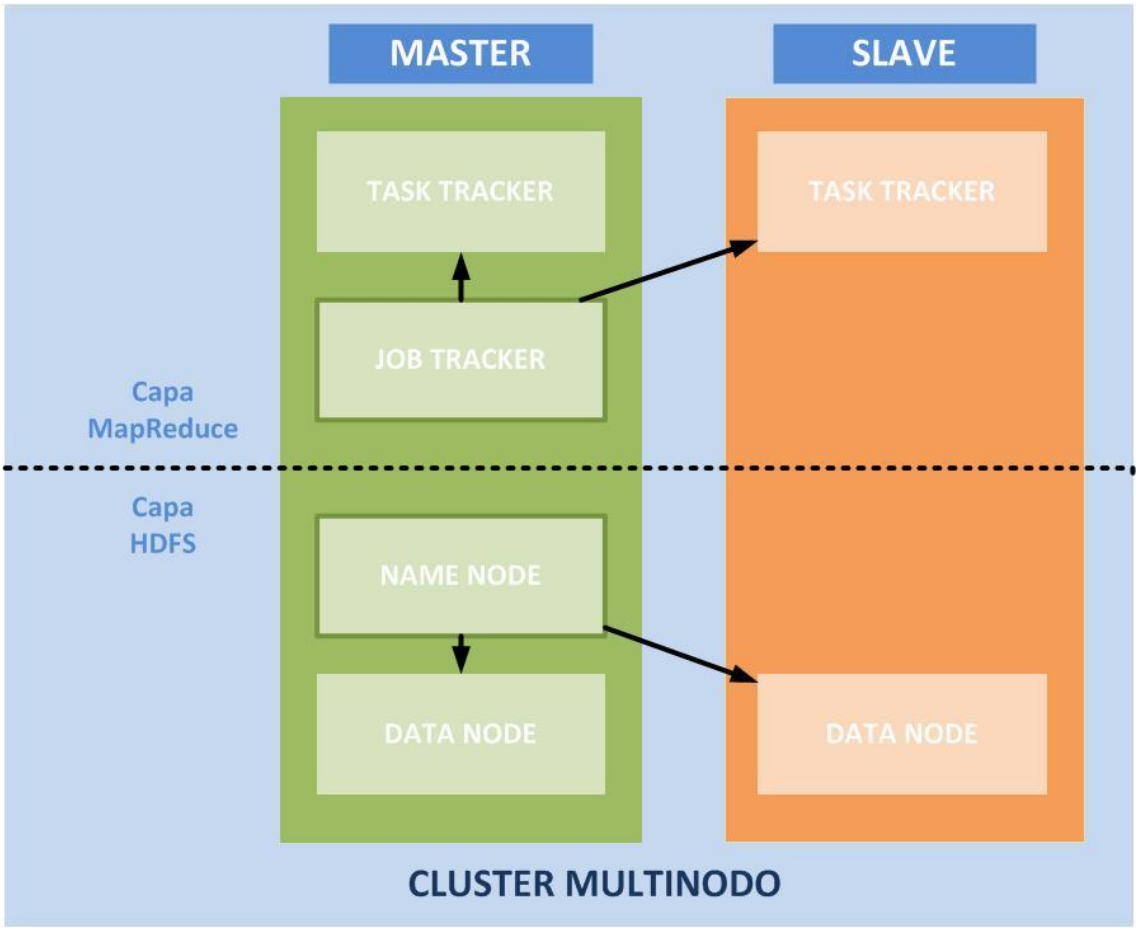


Figura 5.41. Arquitectura Clúster *Apache Hadoop Multinode*

Recuerde que en todo clúster *Hadoop*, bien sea de nodo único o múltiples nodos, se diferencian dos capas. La primera de ellas, la del sistema de ficheros distribuido HDFS que se encarga de gestionar todos los datos y metadatos gracias al *NameNode*. Éste es instalado únicamente en el máster y se encarga de decidir dónde se almacenan los datos y sus replicas. Además emite los metadatos con los que referencia cada uno de los bloques de datos que se almacenan en el sistema. En cada uno de los esclavos y en nuestro caso en el máster también se configuran los *DataNode* o almacenes de datos del sistema. Por otro lado, la segunda capa denominada *MapReduce,* en la que se gestionan todas las tareas del usuario. El *JobTacker* es el que divide las tareas de las aplicaciones y las envía a cada uno de los *TaskTracker* o actuadores de dichas tareas. Ambas capas se relacionan en este punto, ya que el *JobTracker* ha de conocer, gracias a los metadatos del *DataNode*, dónde se encuentran los datos sobre los que se ejecutan las aplicaciones, para enviar las tareas a dichos nodos.

Puede apreciar que en nuestro clúster, el nodo máster también va a ser configurado como esclavo, ya que contiene tanto el *DataNode* como el *TaskTracker* configurado.

Una vez revisada la topología de clúster que va a ser desplegada, pasamos a continuación a aplicar los cambios de configuración precisos.

Modificar Ficheros de Configuración (Solo Máster)

Tiene que modificar una serie de archivos de configuración que se encuentran en el directorio */hadoop/conf/* y que en este caso solo vamos a hacerlo sobre el Nodo Máster.

Directorio *hadoop/conf/masters*

El primero de ellos es el fichero ***conf/masters*** en el que se define en qué máquinas *Hadoop* arrancará los nodos (*NameNodes*) secundarios en el clúster del sistema. En nuestro caso, vamos a definir como nodo maestro primario y secundario la misma máquina.

Como ya explicamos en el *Capítulo 3. Estudio del Entorno Apache Hadoop*, puede configurar un *NameNode* secundario que este inactivo y que tome el control del sistema en caso de pérdida del *NameNode* primario. Este servidor de nombres secundario tendrá una copia de los mismos datos del primario, además de que ha de ser informado, al igual que el activo, de todas las transferencias de bloques en el clúster. Por tanto, tener un *NameNode* secundario es muy útil para mantener la seguridad, pero es costoso ya que se han de replicar todos los datos y metadatos del primario y aumentan sobremanera las transferencias en la red. Dado que se está trabajando en un entorno *Cloud Computing*, que resulta realmente sencillo crear o eliminar máquinas para generar nuevos clústeres y que buscamos el máximo rendimiento de las aplicaciones, no se va a configurar un *NameNode* secundario en otro nodo distinto del clúster.

Por tanto, edite el fichero que mencionamos anteriormente.

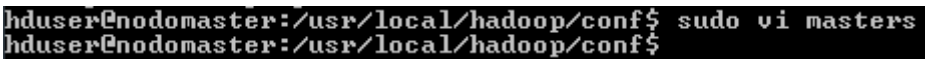


Figura 5.42. Editar *masters*

Ahora incluya el alias del nodo maestro.

Figura 5.43. Fichero *masters*

Guarde los cambios y así queda configurado el Nodo Máster como único *NameNode* primario y secundario.

Directorio *hadoop/conf/slaves*

Tras definir el nodo que actúa como maestro en el sistema, debe definir los nodos que son esclavos y ejecutan las tareas del sistema, teniendo configurados en ellos los *DataNodes* y los *TaskTrackers*. Ya dijimos anteriormente que el Nodo Máster va a ser también Nodo Esclavo, por tanto ha de editar el fichero añadiendo línea a línea el alias de cada uno de los nodos.

Primero edite el fichero:

```
hduser@nodomaster:/usr/local/hadoop/conf$ sudo vi slaves
hduser@nodomaster:/usr/local/hadoop/conf$
```

Figura 5.44. Editar *slaves*

Una vez abierto en modo edición, añada en cada línea los alias:

```
master
slave1
slave2
slave3
slave4
slave5
slave6
slave7
slave8
~
~
```

Figura 5.45. Fichero *slaves*

Una de las ventajas que se ha mencionado en varias ocasiones a lo largo de esta memoria, es la flexibilidad de este clúster. Pues bien, en este paso se ve reflejada esa flexibilidad, ya que si desea añadir más esclavos o eliminar algunos de ellos, basta con modificar este fichero y aplicar posteriormente los cambios (más adelante veremos cómo hacerlo).

En estos dos pasos en los que hemos establecido los Nodos Esclavos y el Nodo Maestro se ha configurado el sistema para que a la hora de aplicar los cambios de configuración, se instalen en el nodo definido como Máster el *NameNode* y el *JobTracker* y en todos los nodos (incluido el maestro) los *DataNodes* y los *TaskTrackers*.

Modificar Ficheros de Configuración (Todas las máquinas)

A continuación, se van a realizar una serie de cambios en los ficheros que se encuentran en el directorio */hadoop/conf/*, pero en esta ocasión tiene que hacerlo para cada una de los nodos que conformarán nuestro clúster.

Antes de realizar los cambios mencionados, debe hacer una operación que no es indispensable, pero que si previene de posibles errores de permisos en la carpeta temporal del directorio *Hadoop* que se creó en el modo *Single Node*. Al ejecutar la operación final de la configuración, `bin/hadoop namenode -format`, se generan una serie de archivos de configuración dentro de ese fichero temporal. Cuando finalice los pasos de configuración, tiene que repetir la operación de formateado. Como estos archivos son creados automáticamente, no tienen específicados permisos de lectura y escritura para el usuario *'hduser'* y al volver a formatear el *NameNode* se tienen que reescribir esos ficheros. Se produce un error de permisos y por eso es preferible, bien conceder permisos para el usuario *'hduser'* sobre directorio o bien eliminarlo y crear uno nuevo. Esto último es lo que se hace en esta ocasión y que debe hacerse en todas las máquinas del clúster.

```
hduser@nodomaster:/usr/local/hadoop$ rm -R tmp
hduser@nodomaster:/usr/local/hadoop$ ls
bin             docs                        ivy             NOTICE.txt
build.xml       hadoop-0.20.2-ant.jar       ivy.xml         README.txt
c++             hadoop-0.20.2-core.jar      lib             src
CHANGES.txt     hadoop-0.20.2-examples.jar  librecordio     webapps
conf            hadoop-0.20.2-test.jar      LICENSE.txt
contrib         hadoop-0.20.2-tools.jar     logs
hduser@nodomaster:/usr/local/hadoop$ mkdir -p tmp
hduser@nodomaster:/usr/local/hadoop$ chown hduser:hadoop tmp
hduser@nodomaster:/usr/local/hadoop$ chmod 750 tmp
hduser@nodomaster:/usr/local/hadoop$
```

Figura 5.46. Eliminar Fichero Temporal

Con esto se asegura que no se produzcan errores indeseados con permisos de escritura y ya puede pasar a realizar los cambios mencionados en el directorio */hadoop/conf*.

Tiene que modificar los mismos tres ficheros de configuración que en el modo *Single Node*, pero en esta ocasión con cambios orientados a la topología multinodo que está instalando. Los archivos en los que vamos a aplicar los cambios son los siguientes:

- ***core-site.xml.*** En este fichero, en el modo *Single Node* indica la ubicación del directorio de archivos temporales y la ubicación del *NameNode* del sistema de ficheros distribuido.

  Como en este caso pasamos a un clúster multinodo, tiene que indicar a todos los nodos cuál es la localización del *NameNode* del clúster. En */etc/hosts* asigna un alias a cada una de las máquinas del sistema junto a su dirección IP, así que ha de usar este alias para indicar que el *NameNode* se encuentra en el nodo máster.

  Abra el fichero para poder editarlo:

```
hduser@nodomaster:/usr/local/hadoop/conf$ sudo vi core-site.xml
hduser@nodomaster:/usr/local/hadoop/conf$
```

Figura 5.47. Modificar *core-site.xml*

Ha de modificar la propiedad que vé en la *Figura 5.48* y que configuró en la instalación *Single Node* y quitar *'localhost'* cambiándolo por la denominación que le otorgó al nodo maestro en la configuración de red.

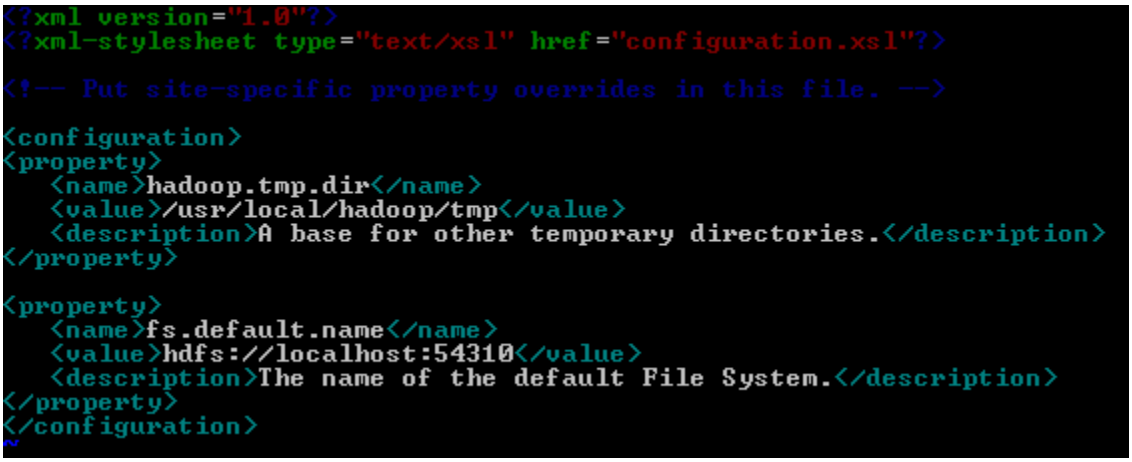


Figura 5.48. Fichero *core-site.xml* antes

El fichero queda así:

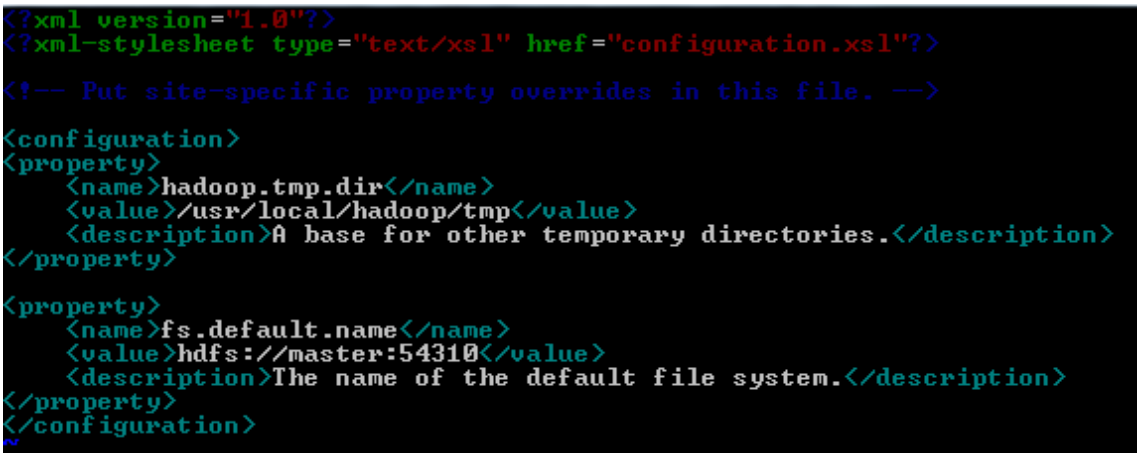


Figura 5.49. Fichero *core-site.xml* después

- ***mapred-site.xml.*** De igual manera que en el caso anterior, en este fichero tiene que modificar una dirección para que apunte a la máquina correcta.

  En "*mapred.site.xml*" indique la localización del *JobTracker*, que en el caso *Single Node*, al ejecutarse todo en la misma máquina, se hizo apuntando a *'localhost'*. Por tanto, edite el fichero.

Figura 5.50. Editar *mapred-site.xml*

Al abrirlo lo encontraremos tal y como fue configurado para *Single Node* y como se aprecia en la *Figura 5.51*.

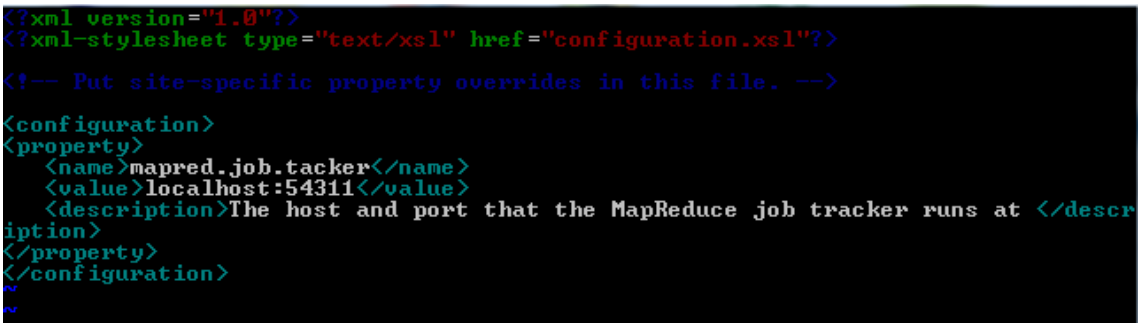


Figura 5.51. Fichero *mapred-site.xml* antes

Ha de cambiar *'localhost'* por el alias de la IP del máster:

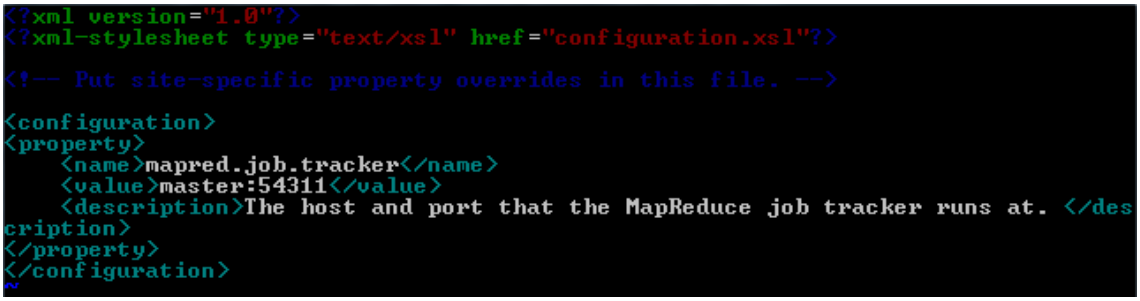


Figura 5.52. Fichero *mapred-site.xml* después

- ***hdfs-site.xml.*** Para finalizar, configure el número de veces que desea que el *NameNode* replique los bloques de datos en el sistema de ficheros distribuido.

  En el caso de la configuración *Single Node* se fijó el número de replicas en uno. Si acude a las recomendaciones de *Apache Hadoop* para clústeres de múltiples nodos, se indica que para una seguridad efectiva el número de replicas ha de ser tres. Vamos a seguir esta recomendación.

  Como en los casos anteriores, abra el fichero para poder editarlo:

Figura 5.53. Modificar *hdfs-site.xml*

En la *Figura 5.54* puede comprobar que en *Single Node* el valor de replica fue uno.

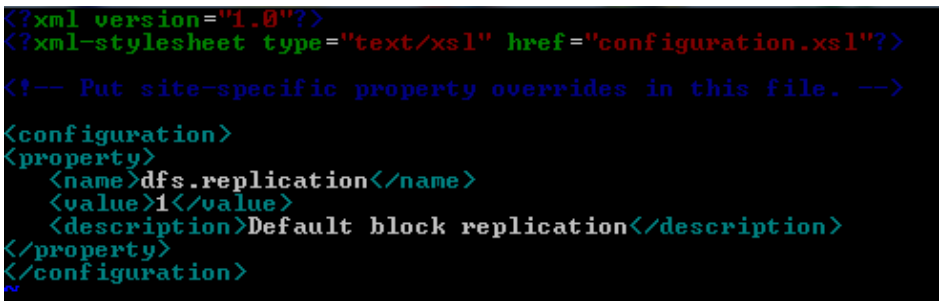


Figura 5.54. Fichero *hdfs-site.xml* antes

Por tanto, debe cambiar ese valor por el que nos recomiendan desde *Apache Hadoop*.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
    <name>dfs.replication</name>
    <value>3</value>
    <description>Default block replication.</description>
</property>
</configuration>
~
```

Figura 5.55. Fichero *hdfs-site.xml* después

Con estas modificaciones, concluye la instalación del clúster multinodo *Apache Hadoop*. En este caso, se ha creado una configuración que cubre nuestras necesidades y que si quisiéramos podría incluir ciertas modificaciones más.

5.2.5. Formatear el Sistema de Ficheros HDFS desde el *NameNode*

Al igual que en la topología *Single Node*, para finalizar la instalación del clúster multinodo y que se apliquen todos los cambios de configuración efectuados en los pasos anteriores, tiene que formatear el sistema de ficheros distribuido desde el *NameNode*.

Ejecute la orden de formateado igual que en el caso anterior, con el comando:

```
>> bin/hadoop namenode -format
```

En la *Figura 5.56* se aprecia el resultado.

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop namenode -format
12/05/18 10:41:05 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = nodomaster/10.0.0.2
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 0.20.2
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/b
ranch-0.20 -r 911707; compiled by 'chrisdo' on Fri Feb 19 08:07:34 UTC 2010
************************************************************/
12/05/18 10:41:05 INFO namenode.FSNamesystem: fsOwner=hduser,hadoop
12/05/18 10:41:05 INFO namenode.FSNamesystem: supergroup=supergroup
12/05/18 10:41:05 INFO namenode.FSNamesystem: isPermissionEnabled=true
12/05/18 10:41:05 INFO common.Storage: Image file of size 96 saved in 0 seconds.
12/05/18 10:41:05 INFO common.Storage: Storage directory /usr/local/hadoop/tmp/d
fs/name has been successfully formatted.
12/05/18 10:41:05 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at nodomaster/10.0.0.2
************************************************************/
hduser@nodomaster:/usr/local/hadoop$
```

Figura 5.56. Formatear Clúster *Multinode*

Ha finalizado la configuración. Sin embargo para asegurarse de que los cambios se han efectuado correctamente, inicie el clúster y compruebe que tanto en el *NameNode* como en los Nodos de Cómputo se ejecutan todos los servicios.

En *Single Node* iniciamos el clúster de una sola orden. En este caso, va a hacerse de manera independiente separando, por un lado el HDFS y por otro la capa *MapReduce*.

Iniciando HDFS

En el directorio *hadoop/bin* se encuentran todos los *scripts* que inician los servicios. En este caso ejecute el que inicia el HDFS, de la siguiente forma:

```
>> bin/start-dfs.sh
```

Tras la ejecución del comando que pone en marcha el *script*, comienzan a arrancarse los servicios (véase la *Figura 5.57*).

```
hduser@nodomaster:/usr/local/hadoop$ bin/start-dfs.sh
starting namenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-nameno
de-nodomaster.out
slave5: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodoesclavo5.out
slave2: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodoesclavo2.out
slave6: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodoesclavo6.out
slave7: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodoesclavo7.out
slave1: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodoesclavo1.out
master: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodomaster.out
slave3: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodoesclavo3.out
slave8: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodoesclavo8.out
slave4: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduse
r-datanode-nodoesclavo4.out
master: starting secondarynamenode, logging to /usr/local/hadoop/bin/../logs/had
oop-hduser-secondarynamenode-nodomaster.out
hduser@nodomaster:/usr/local/hadoop$
```

Figura 5.57. Incio del HDFS

Si observa la *Figura 5.57*, puede comprobar que la operación de inicio se efectúa correctamente tanto en el Nodo Máster como en cada uno de los Nodos Esclavo. A pesar de ello, compruebe que los servicios se están ejecutando en cada una de las máquinas, gracias al comando *Java* que devuelve todas las máquinas virtuales que hay ejecutándose en el sistema.

Primero compruébelo en el máster:

```
hduser@nodomaster:/usr/local/hadoop$ jps
2195 NameNode
2515 SecondaryNameNode
2364 DataNode
2569 Jps
hduser@nodomaster:/usr/local/hadoop$
```

Figura 5.58. JPS HDFS Máster

En la *Figura 5.58* se aprecia que en el nodo máster está el *NameNode* en ejecución, el *NameNode* secundario y el *DataNode*. Es correcto, puesto que así fue configurado. Decidimos no replicar el *NameNode* en otro nodo y por ello se configuró en la misma máquina el *NameNode* secundario. Además queremos que el Nodo Máster trabaje también como nodo de cómputo y por ello ha de tener el *DataNode*.

Segundo, ha de comprobar uno a uno los nodos esclavos para asegurarse de que el *DataNode* está activo en ellos:

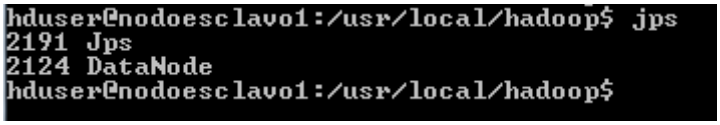


Figura 5.59. JPS HDFS Esclavos

En la *Figura 5.59*, puede ver el resultado que devuelve el comando *jps* en uno de los nodos esclavos. Ejecute el mismo comando en todos los nodos y compruebe que el *DataNode* está activo en todos ellos.

Iniciando MapReduce

Al igual que *Hadoop* dispone de un *script* para arrancar el HDFS, existe otro que inicia la capa de ejecución de tareas *MapReduce*. Ejecute el *script* en el nodo maestro y posteriormente compruebe que los servicios *JobTracker* y *TaskTracker* están activos en los nodos que corresponda.

Lance el *script* que inicia los servicios *MapReduce*, con el comando:

```
>> bin/start-mapred.sh
```

Tras ejecutar el comando, se arrancan los servicios *MapReduce,* como se ve en la *Figura 5.60*.

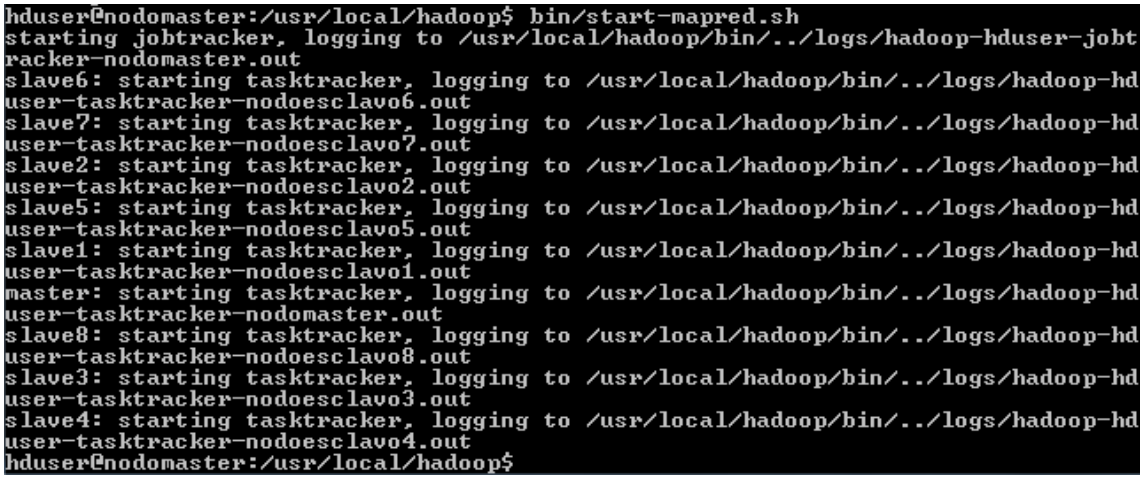


Figura 5.60. Inicio *MapReduce*

De la misma forma que para el HDFS, puede repasar la información de la *Figura 5.60*, comprobar que para el Nodo Máster se ejecuta el *JobTracker* y que el *TaskTracker* es iniciado en todos los nodos incluido el maestro.

En cualquier caso, cerciórese de que todos los servicios están activos en los equipos que corresponden. Para ello utilice el comando *jps* igual que en el caso anterior. Primero para el Nodo Máster.

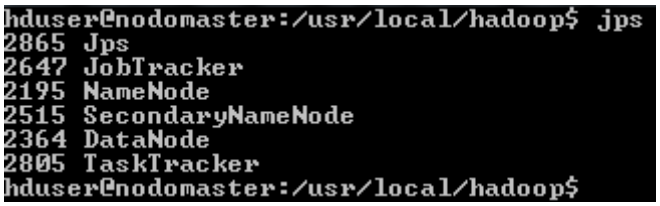


Figura 5.61. JPS *MapReduce* Máster

Puede ver que siguen activos los servicios anteriormente iniciados del HDFS (véase la *Figura 5.58*) y se incluyen ahora el *JobTracker* y el *TaskTracker*.

En segundo lugar, compruebe uno por uno los esclavos.

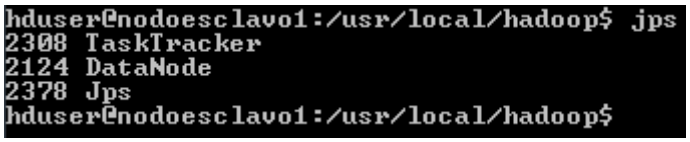


Figura 5.62. JPS *MapReduce* Esclavos

En cada uno de ellos han de mantenerse los servicios anteriores y ha de aparecer el *TaskTracker*.

Ya disponemos del clúster que pretendíamos formar correctamente instalado y configurado. En el siguiente capítulo, pondremos en marcha aplicaciones que pongan a prueba nuestro entorno de computación distribuido multinodo y que nos lleven a obtener de él su máximo rendimiento.

6. Ejemplos de Aplicación del Entorno *Apache Hadoop MapReduce*

En el *Capítulo 5. Puesta en marcha del entorno Apache Hadoop/MapReduce*, configuramos un clúster *Hadoop* multinodo utilizando instancias virtuales de nuestro entorno *Cloud Computing*. Previo a desplegar el clúster de múltiples nodos, se configuró un clúster *Single Node* en el que una sola instancia virtual realiza todas las funciones del entorno de trabajo *Apache Hadoop MapReduce.*

En este capítulo, vamos a realizar diversas pruebas al *framework* configurado en el capítulo anterior. Como hemos comentado, el primer paso fue instalar el modo *Single Node*, por ello la primera fase es probar ese primer clúster y asegurarnos de que todo ha sido configurado correctamente. Posteriormente, desplegamos el modo *Multinode* en el que suponemos que el compilador de *MapReduce* funciona correctamente, ya que las máquinas virtuales del clúster son copias de las del modo *Single Node.* En cualquier caso, vamos a ejecutar la misma prueba y así podremos cerciorarnos de su correcto trabajo. Tras las pruebas de funcionamiento, se realizarán pruebas de rendimiento, por tanto, el siguiente paso consiste en realizar test del entorno *Apache Hadoop MapReduce* para obtener mediciones del tiempo de ejecución ampliando progresivamente la carga de trabajo. Para concluir, haremos uso de la librería HIPI (*Hadoop Image Processing Interface*) con aplicaciones que utilicen imágenes en su ejecución.

6.1. Prueba de Funcionamiento del Entorno

Comenzamos comprobando que la configuración realizada en el apartado anterior de la memoria ha sido exitosa. Para ello, vamos a ejecutar un programa de prueba *MapReduce* para cada uno de los entornos desplegados. Como hemos comentado, el primer paso fue desplegar el clúster modo *Single Node* y posteriormente, sirviéndonos de éste, configuramos el *Multinode*. En este orden, realizaremos las pruebas de ejecución del entorno.

El programa utilizado para las pruebas de ejecución, es un código escrito en *Java MapReduce* llamado *"WordCount"* y cuya funcionalidad es, a partir de unos archivos de entrada en modo texto, diferenciar todas las palabras que se encuentran en esos archivos y escribirlas en forma de lista en un archivo de salida junto con el número de apariciones de cada una de ellas.

En el siguiente diagrama (véase la *Figura 6.1*) se puede observar un seguimiento de la ejecución de esta aplicación sobre una serie de palabras de entrada.

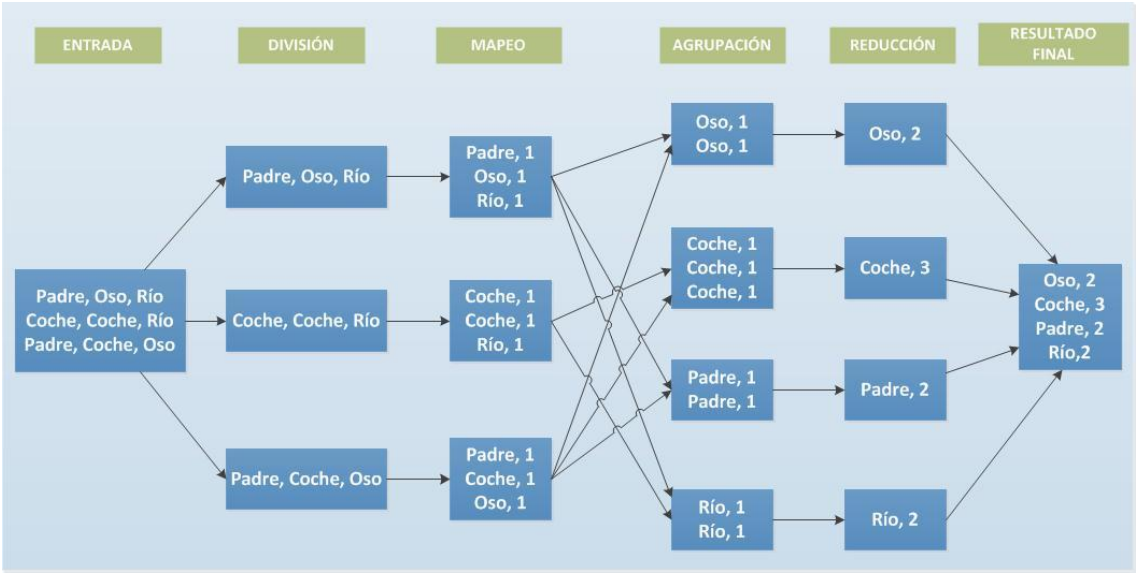


Figura 6.1. Diagrama *MapReduce*

En el primer elemento del diagrama, aparecen una serie de palabras de entrada y, a primera vista, se puede comprobar que todos aparecen repetidos. Esos datos son divididos por el entorno entre los nodos del sistema, que en este caso son tres. Cada uno de los nodos, ejecuta la tarea *Map* con los datos que se encuentran en su *DataNode* y devuelven como salida cada una de las palabras con su contador (siempre será igual a uno para la tarea *Map*). Posteriormente el *framework Hadoop* agrupa todos los resultados por clave y ejecuta el *Reduce*, devolviendo como resultado final un lista con todos los términos y el número de apariciones de cada uno de ellos.

El código del programa es el siguiente:

```
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.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

      public static class Map extends MapReduceBase implements
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,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
                  String line = value.toString();
```

```
                    StringTokenizer tokenizer = new
StringTokenizer(line);
                    while (tokenizer.hasMoreTokens()) {
                          word.set(tokenizer.nextToken());
                          output.collect(word, one);
                    }
              }
      }

      public static class Reduce extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {
          public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
                    int sum = 0;
               while (values.hasNext()) {
                          sum += values.next().get();
               }
               output.collect(key, new IntWritable(sum));
          }
      }

      public static void main(String[] args) throws Exception {
          JobConf conf = new JobConf(WordCount.class);
          conf.setJobName("wordcount");

          conf.setOutputKeyClass(Text.class);
          conf.setOutputValueClass(IntWritable.class);

          conf.setMapperClass(Map.class);
          conf.setCombinerClass(Reduce.class);
          conf.setReducerClass(Reduce.class);

          conf.setInputFormat(TextInputFormat.class);
          conf.setOutputFormat(TextOutputFormat.class);

          FileInputFormat.setInputPaths(conf, new Path(args[0]));
          FileOutputFormat.setOutputPath(conf, new Path(args[1]));

          JobClient.runJob(conf);
      }
}
```

A continuación, desgranamos el código explicando los aspectos destacados del mismo.

El método en el que se programan los aspectos fundamentales del código es, como en todo programa escrito en *Java¸* el método `main ()`. En este procedimiento, se crea un "objeto trabajo" de la clase *JobConf* con el que indicamos al entorno los aspectos de configuración que se indican en un programa *MapReduce*.

Los valores que se configuran en el método `main ()` son los siguientes:

- **Nombre del trabajo:** `conf.setJobName("wordcount");`
- **Clave de los pares de salida:** (texto) `conf.setOutputKeyClass(Text.class);` Este valor se refiere a las palabras que se van a escribir en el fichero de salida.
- **Valor de los pares de salida:** (numérico) `conf.setOutputValueClass (IntWritable.class);` Representa el número de apariciones de la palabra con la que forman el par de salida. Si lo juntamos con el punto anterior, un par de salida podría ser < (Hola, 2) >; en el que la palabra (clave de salida) "Hola" aparece dos veces (valor de salida).
- **Clase *Map:*** `conf.setMapperClass(Map.class);`
- **Clase combinadora:** `conf.setCombinerClass(Reduce.class);`
- **Clase *Reduce:*** `conf.setReducerClass(Reduce.class);`
- **Formato de los archivos de entrada:** (Archivos de texto .txt) `conf.setInputFormat(TextInputFormat.class);`
- **Formato de los archivos de salida:** (Archivos de texto .txt) `conf.setOutputFormat(TextOutputFormat.class);`
- **Directorio de entrada:** `FileInputFormat.setInputPaths(conf, new Path(args[0]));`
- **Directorio de salida:** `FileOutputFormat.setOutputPath(conf, new Path(args[1]));`

Hemos creado un trabajo llamado *WordCount* que recibe como entrada (*InputFormat*) archivos de texto almacenados en el directorio de entrada (*InputPath*) y crea, gracias al método *Map*, pares intermedios con el formato clave-valor < (*Text*, *IntWritable*) >. Posteriormente, la clase *Reduce* se encarga de combinar y reducir (*Combiner* y *Reducer*) los pares intermedios para generar el archivo de salida (*OutputFormat*) de texto con la lista de palabras y sus apariciones, que guarda en el directorio de salida (*OutputPath*).

El entorno, tras ejecutar la siguiente línea de código del *main*, `JobClient.runJob(conf);` comienza la ejecución *MapReduce*, llamando a la clase que en los pasos anteriores fue configurada como clase *Mapper*, que si examinamos el código anterior es la clase con la siguiente cabecera: `public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable>.`

Esta clase contiene un método que es ejecutado como tarea *Map* a todos los nodos del *framework* y cada uno de ellos lo ejecuta sobre los datos asignados en su *DataNode.* Se definen dos variables, una de ellas estática "*one*" que guarda el valor numérico 1 y que se utiliza para generar el par intermedio que tendrá el siguiente formato < (palabra, 1) >. La segunda variable, es "*word*" y sirve para ir guardando las palabras que se van detectando. Si vemos el procedimiento *Map* que aparece en la clase, su funcionalidad es, ante un archivo de

texto de entrada, va separando las palabras que en él aparecen utilizando la clase *Java StringTokenizer* que permite dividir los archivos de texto en *Tokens* o *substrings* separados por espacios en blanco. Por cada palabra que encuentra, emite un par intermedio con esa cadena y el valor uno que representa una aparición de la palabra en cuestión. Realiza esta ejecución hasta que complete el archivo de entrada y si tiene más archivos en su *DataNode* continúa automáticamente el trabajo con ellos.

Tras la ejecución de las tareas *Map* sobre todos los conjuntos de entrada, se generan los pares intermedios compuestos todos por un vocablo y el valor uno, referente a su aparición. El *framework* agrupa todos los pares por clave, que en este caso es el término, y pasa las agrupaciones a los distintos nodos para realizar la segunda ejecución, la tarea *Reduce*¸ cuyas líneas están bajo la cabecera `public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable>` en el código del programa ya incluido anteriormente.

El proceso *Reduce* recibe como entrada los pares compuestos por una clave en modo *"Text"* y el valor "*IntWritable*" uno. Este método, al igual que el anterior, trabaja con un bucle "*while*" en el que se contabilizan, dejando fijo el valor de la clave, cuántas apariciones tiene cada uno de los términos. Para finalizar, se producen los pares de salida en los que se escribe cada clave con la suma de sus apariciones.

El último paso lo lleva a cabo el entorno *Hadoop MapReduce*, creando el archivo de salida con la agrupación de todos los pares creados en las tareas *Reduce* en los que aparece cada vocablo con sus apariciones en la biblioteca de entrada.

Tras ver el código que se va a ejecutar para probar el entorno, comenzamos a realizar los pasos necesarios para llevarla a cabo.

Descarga de libros

Como vimos en el apartado anterior, de lo primero que es necesario disponer para la ejecución del programa *WordCount,* además del código del mismo, es de una biblioteca de entrada que conste de uno a varios libros sobre los que realizar el conteo.

Para poder conseguir una batería de libros con los que realizar la ejecución, vamos a servirnos de la Web del proyecto ***Gutenberg,*** que pone a disposición de todos los usuarios del portal una librería gratuita. Puede descargarse de sus servidores los libros que se precise para las pruebas, ya que cuentan con unos cuarenta mil, pudiendo obtenerlos en varios formatos. La dirección Web es la siguiente:

http://www.gutenberg.org/

Por tanto, el primer paso es obtener y almacenar en nuestro entorno, algunos de los libros que ofrece este proyecto. Lo puede ver en la *Figura 6.2*.

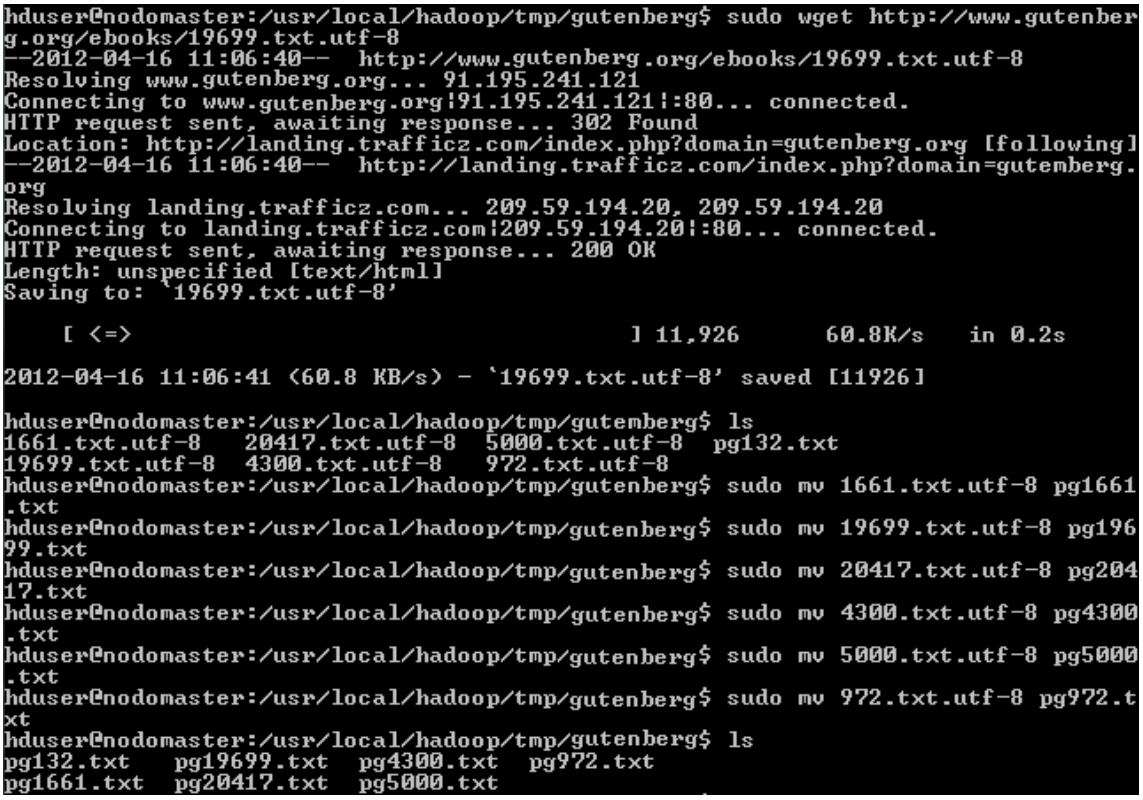


Figura 6.2. Descarga Biblioteca

En la *Figura 6.2* puede ver la descarga del último de los libros y posteriormente se aprecia como son renombrados con un formato de nombre más sencillo, todos ellos con la extensión ".txt". Finalmente, ejecutando el comando "ls" puede comprobarse que se dispone de siete libros para la prueba.

Incluir archivos en el sistema de ficheros HDFS

Con la descarga, ya disponemos de los libros en el disco duro del servidor en el que configuramos el *NameNode*, concretamente en un directorio de archivos temporales (`usr/local/hadoop/tmp/`) creado en la configuración del sistema dentro del emplazamiento utilizado para almacenar los archivos de *Hadoop*. Sin embargo donde es realmente necesario almacenarlos es en el sistema de ficheros de *Hadoop HDFS*. Por ello, hemos de realizar una copia del directorio en el que tenemos almacenados los libros en el sistema de ficheros distribuido y que se reparta entre todos los nodos del sistema, en el caso de que el sistema sea *Multinode*.

Incluimos el directorio con los libros en el HDFS ejecutando el comando "***copyFromLocal***" que es una de las órdenes ejecutables del sistema de ficheros. Esta orden, toma el directorio que se especifica en el comando e introduce el contenido del mismo en el sistema de archivos distribuido. Todas las instrucciones que pueden utilizarse para gestionar el HDFS fueron descritas en el *Apartado 3.3.1. Comandos para Gestionar el HDFS*; para conocer a fondo el comando que vamos a usar a continuación y el resto de los que pueden utilizarse, acudir a ese punto de la memoria.

Finalmente, para incluir los libros en el sistema de ficheros ejecutamos:

```
>> hadoop dfs -copyFromLocal /usr/local/hadoop/tmp/gutenberg /home/hduser/gutenberg
```

Así conseguiremos incluir todos los ficheros que estén dentro del directorio */usr/local/hadoop/tmp/gutenberg* en el sistema de ficheros. Para comprobar que la copia ha sido correcta, ejecutamos dos comandos:

```
>> hadoop dfs -ls /home/hduser

>> hadoop dfs -ls /home/hduser/gutenberg
```

Con el primero comprobamos que se ha creado correctamente el directorio y con el segundo accederemos al contenido para asegurarnos de que los libros se encuentran en el sistema de ficheros distribuido (véase la *Figura 6.3*).

```
hduser@nodomaster:/usr/local/hadoop/bin$ hadoop dfs -copyFromLocal /usr/local/ha
doop/tmp/gutenberg /home/hduser/gutenberg
hduser@nodomaster:/usr/local/hadoop/bin$ hadoop dfs -ls /home/hduser/
Found 1 items
drwxr-xr-x   - hduser supergroup          0 2012-04-16 11:19 /home/hduser/gutenb
erg
hduser@nodomaster:/usr/local/hadoop/bin$ hadoop dfs -ls /home/hduser/gutenberg
Found 7 items
-rw-r--r--   2 hduser supergroup      17443 2012-04-16 11:19 /home/hduser/gutenb
erg/pg132.txt
-rw-r--r--   2 hduser supergroup      11926 2012-04-16 11:19 /home/hduser/gutenb
erg/pg1661.txt
-rw-r--r--   2 hduser supergroup      11926 2012-04-16 11:19 /home/hduser/gutenb
erg/pg19699.txt
-rw-r--r--   2 hduser supergroup      11931 2012-04-16 11:19 /home/hduser/gutenb
erg/pg20417.txt
-rw-r--r--   2 hduser supergroup      11926 2012-04-16 11:19 /home/hduser/gutenb
erg/pg4300.txt
-rw-r--r--   2 hduser supergroup      17453 2012-04-16 11:19 /home/hduser/gutenb
erg/pg5000.txt
-rw-r--r--   2 hduser supergroup      17445 2012-04-16 11:19 /home/hduser/gutenb
erg/pg972.txt
```

Figura 6.3. Insertar Directorio *gutenberg* en HDFS

Como hemos comprobado, ya disponemos de los archivos de entrada para ejecutar el programa de prueba del entorno, en el que se van a listar todas las palabras que hay en la biblioteca de entrada e irán acompañadas con el número de apariciones de cada una de ellas. Por tanto, pasamos a ejecutar el programa.

Ejecutar el trabajo MapReduce

Al descargarnos el paquete que contiene los instaladores de *Apache Hadoop*, obtenemos también el código compilado del ejemplo del conteo de palabras de una biblioteca de entrada. Es decir, dentro del directorio que contiene *Hadoop* podemos encontrar el ejecutable "*.jar*" que, recibiendo como argumentos el directorio con los libros de entrada y un directorio de salida, ejecuta el *WordCount*.

Podemos ejecutar el ejemplo de una forma genérica (válida para todas las versiones de *Hadoop*), ya que en el nombre del ejemplo se incluye la versión de *Hadoop*. Esto se haría con el siguiente comando:

```
>> hadoop jar hadoop*examples*.jar wordcount /home/hduser/gutenberg /home/hduser/gutenberg-output
```

Sin embargo, puede ser que nos produzca algún error si en nuestro caso la estructura del nombre no es tal y como escribimos en la línea de comandos. En cualquier caso probamos (véase la *Figura 6.4*):

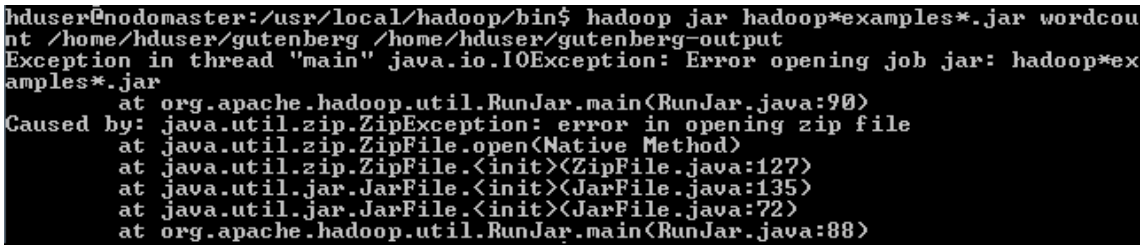


Figura 6.4. Error de Ejecución Ejemplo

Como podemos observar, nos produce un error de compilación, debido a que no encuentra el fichero "*jar*" y no puede abrirlo. Lo que hacemos es buscar en nuestro entorno *Hadoop* dónde se encuentra el contenedor con los ejemplos de compilación y cómo se llama. Ejecutando un "*ls*" en la raíz del entorno, encontramos el archivo, lo vemos a continuación en la *Figura 6.5*.

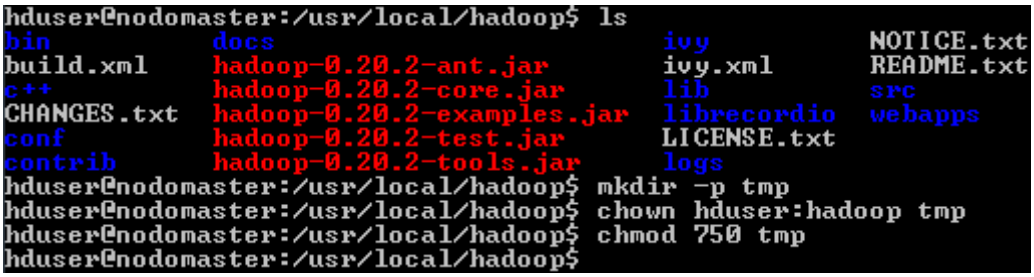


Figura 6.5. Contenido Directorio *Hadoop*

Comprobamos que el formato del nombre no se corresponde con el que tratamos de ejecutar en el primer intento. Para evitar más errores, ejecutamos el ejemplo con el nombre al completo, sin abreviaciones, y así nos aseguramos su correcta ejecución. Escribimos en la línea de comandos:

```
>> hadoop jar hadoop-0.20.2-examples.jar wordcount
/home/hduser/gutenberg /home/hduser/gutenberg-output
```

Con esta orden, se ejecuta el programa *WordCount* incluido en el contenedor de ejecutables "*hadoop-0.20.2-examples.jar*", enviando como argumentos el directorio con los libros de entrada y un directorio de salida para escribir el resultado de la ejecución. Conocemos el resultado:

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop jar hadoop-0.20.2-examples.jar w
ordcount /home/hduser/gutenberg /home/hduser/gutenberg-output/
12/04/18 10:18:25 INFO input.FileInputFormat: Total input paths to process : 7
12/04/18 10:18:26 INFO mapred.JobClient: Running job: job_201204161012_0002
12/04/18 10:18:27 INFO mapred.JobClient:  map 0% reduce 0%
12/04/18 10:18:38 INFO mapred.JobClient:  map 28% reduce 0%
12/04/18 10:18:39 INFO mapred.JobClient:  map 57% reduce 0%
12/04/18 10:18:42 INFO mapred.JobClient:  map 71% reduce 0%
12/04/18 10:18:44 INFO mapred.JobClient:  map 100% reduce 0%
12/04/18 10:18:53 INFO mapred.JobClient:  map 100% reduce 100%
12/04/18 10:18:55 INFO mapred.JobClient: Job complete: job_201204161012_0002
12/04/18 10:18:55 INFO mapred.JobClient: Counters: 17
12/04/18 10:18:55 INFO mapred.JobClient:   Job Counters
12/04/18 10:18:55 INFO mapred.JobClient:     Launched reduce tasks=1
12/04/18 10:18:55 INFO mapred.JobClient:     Launched map tasks=7
12/04/18 10:18:55 INFO mapred.JobClient:     Data-local map tasks=7
12/04/18 10:18:55 INFO mapred.JobClient:   FileSystemCounters
12/04/18 10:18:55 INFO mapred.JobClient:     FILE_BYTES_READ=58676
12/04/18 10:18:55 INFO mapred.JobClient:     HDFS_BYTES_READ=100050
12/04/18 10:18:55 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=117612
12/04/18 10:18:55 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=29376
12/04/18 10:18:55 INFO mapred.JobClient:   Map-Reduce Framework
12/04/18 10:18:55 INFO mapred.JobClient:     Reduce input groups=418
12/04/18 10:18:55 INFO mapred.JobClient:     Combine output records=1594
12/04/18 10:18:55 INFO mapred.JobClient:     Map input records=1348
12/04/18 10:18:55 INFO mapred.JobClient:     Reduce shuffle bytes=58712
12/04/18 10:18:55 INFO mapred.JobClient:     Reduce output records=418
12/04/18 10:18:55 INFO mapred.JobClient:     Spilled Records=3188
12/04/18 10:18:55 INFO mapred.JobClient:     Map output bytes=106469
12/04/18 10:18:55 INFO mapred.JobClient:     Combine input records=3481
12/04/18 10:18:55 INFO mapred.JobClient:     Map output records=3481
12/04/18 10:18:55 INFO mapred.JobClient:     Reduce input records=1594
```

Figura 6.6. Ejecutar Ejemplo *Hadoop*

Si observamos en la *Figura 6.6*, el ejemplo se ejecuta correctamente, detecta los siete libros (“*Total input paths to process: 7”*) y a continuación, asigna un identificador al trabajo y empieza a ejecutar tareas *Map* que se ejecutan para posteriormente completarse la ejecución con las tareas *Reduce*.

Comprobar los resultados

Tras la ejecución del código, parece que efectivamente se ha realizado correctamente el conteo de las palabras de los siete libros de entrada. Sin embargo, eso no lo hemos comprobado realmente; solo hemos visto que la ejecución no ha producido errores. Por tanto, vamos a ejecutar de nuevo los comandos que listan el contenido del directorio que reciben como argumento y así comprobamos que se han creado tanto el directorio como los archivos de salida.

```
>> hadoop dfs -ls /home/hduser

>> hadoop dfs -ls /home/hduser/gutenberg-output
```

Con el primer comando nos aseguramos que se ha creado el directorio de salida y con el segundo que se han generado los archivos de salida.

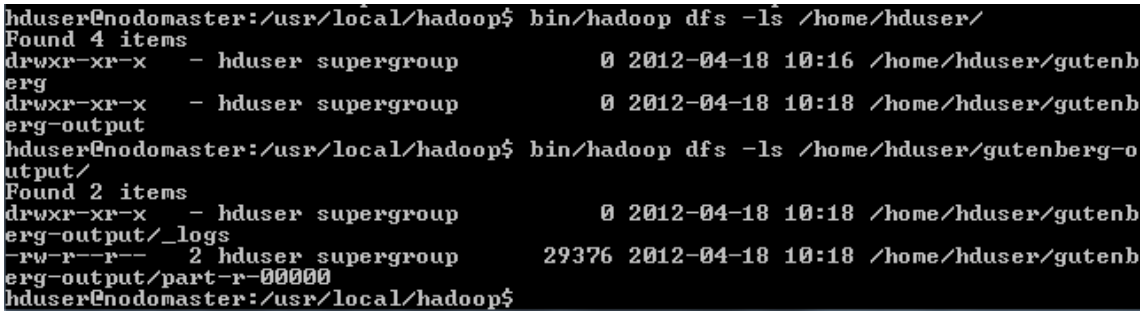


Figura 6.7. Contenido Directorio Raíz HDFS

Efectivamente, ejecutando la primera instrucción apreciamos que se ha creado el directorio de salida y con la segunda vemos que aparece el *log* de la ejecución y el fichero de salida.

El siguiente paso es copiar el fichero de salida al sistema de archivos local para examinar su contenido. Para ello, ejecutamos:

```
>> bin/hadoop dfs -getMerge /home/hduser/gutenberg-output
tmp/gutenberg-output
```

Ya tenemos por tanto, el resultado de la ejecución del ejemplo en nuestro sistema de ficheros local. Vamos a ver su contenido y así, si el resultado es correcto, damos por finalizada la prueba del entorno con resultado satisfactorio.

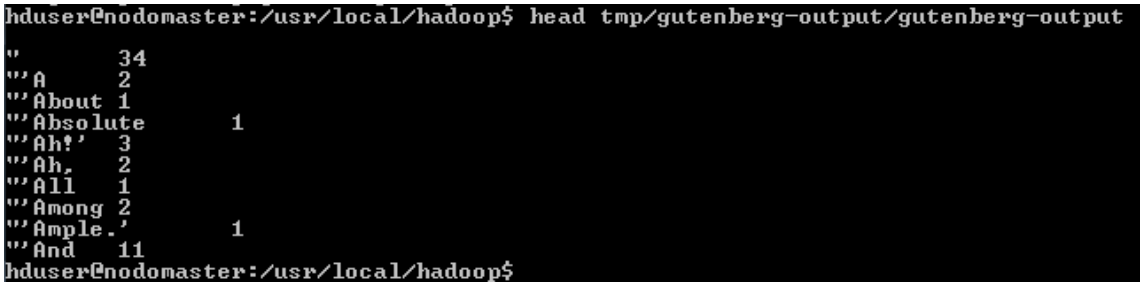


Figura 6.8. Salida Prueba Gutenberg

Podemos ver que aparecen ordenadas las palabras en orden alfabético acompañadas de su número de apariciones, por tanto, concluimos que la instalación del entorno *Apache Hadoop MapReduce* y la compilación del *WordCount* ha sido correcta.

Ejecución en Multinode

Tras configurar *Hadoop Single Node* en nuestra instancia virtual, creamos una imagen de la misma para poder arrancar a la vez las máquinas virtuales con la misma configuración y con ellas crear el clúster *Multinode*. Dado que las instancias utilizadas para crear el clúster de múltiples nodos provienen de la máquina virtual utilizada en el modo *Single Node*, podemos suponer que el compilador *Java MapReduce* funcionará correctamente.

Hemos de recordar que el clúster lo conforman el Nodo Máster que controla el sistema y ocho nodos esclavos ejecutores de las tareas que el trabajo *MapReduce* genere. En la *Figura 6.9*,

recordamos las instancias virtuales que forman parte de nuestro clúster, incluyendo los recursos que las mismas aportan al sistema global.

Server Usage Summary (Show Terminated) Download CSV »

ID	Name	User	VCPUs	Ram Size	Disk Size	Flavor	Uptime	Status
20	NodoMaster	admin	1	2GB	20GB	m1.small	11 horas, 20 minutos	Active
21	NodoEsclavo1	admin	1	2GB	20GB	m1.small	11 horas, 10 minutos	Active
22	NodoEsclavo2	admin	1	2GB	20GB	m1.small	2 horas, 16 minutos	Active
23	NodoEsclavo3	admin	1	2GB	20GB	m1.small	2 horas, 16 minutos	Active
24	NodoEsclavo4	admin	1	2GB	20GB	m1.small	2 horas, 6 minutos	Active
25	NodoEsclavo5	admin	1	2GB	20GB	m1.small	2 horas, 5 minutos	Active
26	NodoEsclavo6	admin	1	2GB	20GB	m1.small	2 horas, 3 minutos	Active
27	NodoEsclavo7	admin	1	2GB	20GB	m1.small	2 horas, 2 minutos	Active
28	NodoEsclavo8	admin	1	2GB	20GB	m1.small	2 horas, 1 minuto	Active

Figura 6.9. Resumen Máquinas Virtuales del Clúster

En cualquier caso, ejecutamos el mismo ejemplo que en el caso anterior y así nos aseguramos que efectivamente el funcionamiento es correcto. Los pasos a seguir para realizar la prueba son los mismos que en la ejecución *Single Node*, por ello obviamos toda la explicación y las imágenes.

Una vez completamos la ejecución, podemos conectarnos a cada uno de los nodos esclavos y leer la información del fichero *log* que guarda los datos de la conexión del *DataNode* con el *NameNode* en el que estamos conectados. Este fichero se llama "*hadoop-hduser-datanode-nodoesclavo1.log*" y almacena toda la información de conexión y envío de transacciones del nodo maestro con el esclavo. Por tanto, editamos el fichero para ver su contenido:

```
hduser@nodoesclavo1:/usr/local/hadoop/logs$ vi hadoop-hduser-datanode-nodoesclav
o1.log
hduser@nodoesclavo1:/usr/local/hadoop/logs$
```

Figura 6.10. Editar Log de los Nodos Esclavos

A continuación, en la *Figura 6.11* mostramos una parte del fichero para comprobar si aparecen datos de conexión con el nodo maestro.

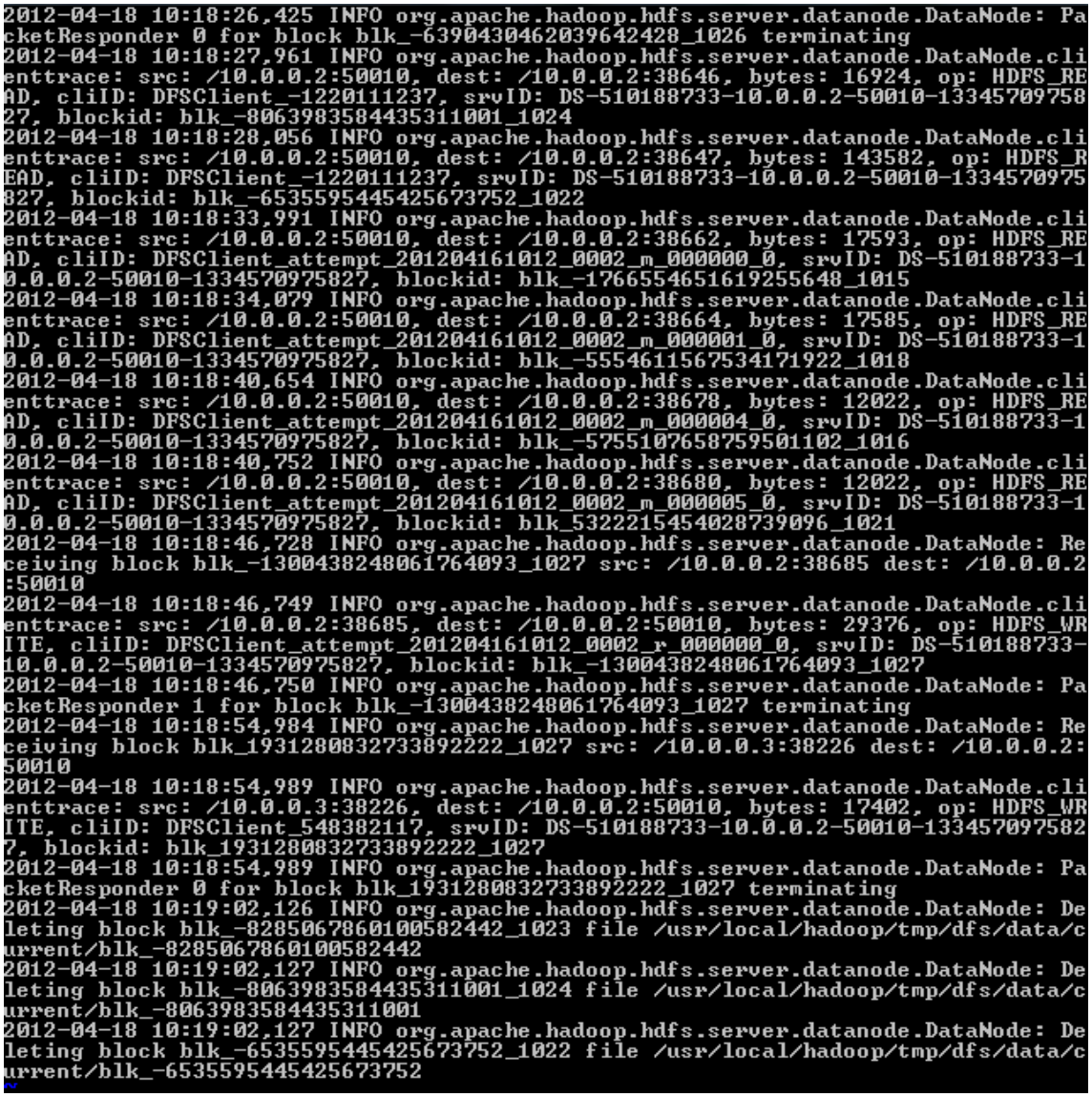


Figura 6.11. Contenido Log Esclavo

Efectivamente se aprecian envíos de bloques con el nodo maestro. Como el resultado de la ejecución ha sido correcto y además aparecen envíos de bloques de información entre el nodo maestro y los esclavos, concluimos que la configuración de *Apache Hadoop Multinode* es correcta y que por ello, podemos realizar ejecuciones que pongan a prueba el rendimiento del sistema.

6.2. Prueba de Rendimiento *MapReduce*

Una vez que conocemos que nuestro clúster multinodo *Apache Hadoop Multinode* funciona correctamente es el momento de comenzar a evaluar hasta dónde puede llegar, es decir, obtener el máximo rendimiento del mismo. Dado que lo más destacado de la programación *MapReduce* en entornos de computación distribuida *Apache Hadoop* es la capacidad de obtener tiempos de ejecución más que aceptables para aplicaciones que trabajan sobre grandes volúmenes de datos, parece claro pensar que lo que hemos de encontrar es una fuente de datos de la que podamos obtener grandes conjuntos de datos de entrada.

Como hemos comentado antes el proyecto Gutenberg pone a nuestra disposición de manera gratuita cerca de cuarenta mil libros, número más que suficiente para poder realizar pruebas de ejecución sobre conjuntos de libros. Por ello, dado que podemos disponer de una gama de conjuntos de entrada muy amplia gracias a la facilidad de obtención de libros con este proyecto, vamos a volver a ejecutar el programa *WordCount*. Sin embargo, en esta ocasión seremos más exigentes con el volumen de los datos de entrada que iremos aumentando progresivamente.

Por tanto, para las pruebas de rendimiento de nuestro clúster *Apache Hadoop* vamos a realizar ejecuciones del programa *WordCount* midiendo los tiempos de ejecución para conjuntos de datos de entrada compuestos de diez libros en la primera prueba y llegando hasta mil en la última de ellas.

Al igual que en el caso de la prueba del entorno, el primer paso de esta ejecución es incluir el conjunto de libros de entrada en el sistema de ficheros distribuido. Dado que esta operación se convertirá en una acción costosa en cuanto a tiempo de ejecución según vayan aumentando el número de libros, vamos a medir el tiempo de duración de la misma para poder incluir estos tiempos en las comparativas posteriores.

Entonces, los resultados de las pruebas de rendimiento están valorados con dos variables, el tiempo que consume el entorno en incluir los datos de entrada en el sistema de ficheros distribuido HDFS y el tiempo de cómputo del código del programa *WordCount*.

Antes de realizar cualquier prueba, llevamos a cabo las descargas de los libros y los disponemos en conjuntos preparados para cada una de las ejecuciones planteadas.

Además para estas pruebas de rendimiento, usaremos el código *Java* que incluimos en la explicación del funcionamiento de *WordCount*, es decir, creamos un archivo llamado "*WordCount.java*" con el mismo código del *Apartado 6.1. Prueba de Funcionamiento del Entorno*.

Incluimos el archivo en el directorio */hadoop/tmp/codigo* y creamos un directorio en el que almacenaremos las clases que son creadas en la compilación de la siguiente forma:

```
hduser@nodomaster:/usr/local/hadoop/tmp/wordcount$ mkdir
wordcount_classes
```

Ya tenemos el fichero "*.java*" con el código del programa y un directorio para la compilación, por tanto, realizamos la compilación:

```
hduser@nodomaster:/usr/local/hadoop/tmp/wordcount$ javac -classpath /usr/local/hadoop-0.20.2-core.jar -d wordcount_classes WordCount.java
```

El siguiente paso, es crear un ejecutable para poder ejecutar el programa en todas las pruebas de rendimiento que realizaremos a continuación, lo hacemos con el siguiente comando:

```
hduser@nodomaster:/usr/local/hadoop/tmp/wordcount$ jar -cvf wordcount.jar -C wordcount_classes/.
```

Con esto, disponemos de nuestro ejecutable propio con código *WordCount* y así ya estamos preparados para realizar las pruebas de rendimiento.

Se han realizado trece pruebas sobre el ejecutable creado, variando en cada una de ellas el número de libros que forman parte de la biblioteca de entrada. A continuación se incluyen tres de ellas, las más representativas. Del resto de pruebas realizadas, solo recogemos los datos, que serán estudiados en el apartado *6.3. Resultados Obtenidos en las Pruebas de Rendimiento MapReduce*.

6.2.1. Ejecución *WordCount* sobre diez libros

La primera de las pruebas de rendimiento de nuestro clúster multinodo *Apache Hadoop* supone la ejecución del programa que contabiliza el número de apariciones de cada una de las palabras que componen una biblioteca de entrada, que en este caso cuenta con diez libros.

Como hemos comentado anteriormente, el primer paso es medir el tiempo que tarda el entorno en incluir los diez libros que componen la biblioteca de entrada en el sistema de ficheros distribuido.

Esto se hace con el siguiente comando:

```
>> time bin/hadoop dfs -copyFromLocal tmp/Gutenberg10 /home/hduser/
```

Así indicamos al *framework Hadoop* que ha de incluir el contenido del directorio *tmp/Gutenberg10* al directorio raíz del sistema de archivos HDFS. Escribimos en línea de comandos la instrucción acompañada de la variable "*time*" y vemos el resultado.

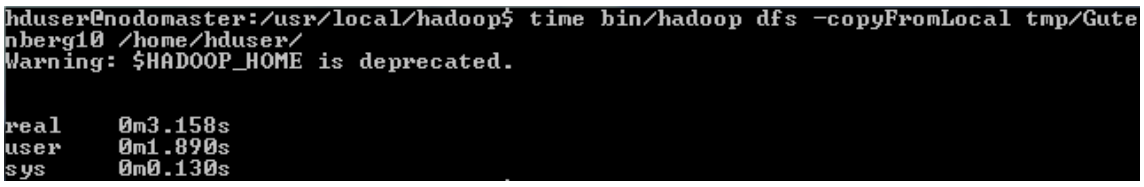


Figura 6.12. Incluir Biblioteca 10 en HDFS

El tiempo real de la instrucción ha sido de **3,158 segundos**.

Comprobamos con la instrucción,

```
>> bin/hadoop dfs -ls /home/hduser/
```

que el directorio ha sido creado en el HDFS y por tanto los libros ya están repartidos entre los nodos que componen el clúster.

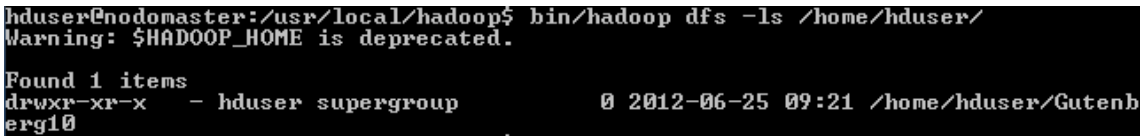


Figura 6.13. Contenido HDFS Prueba 10 Entrada

Efectivamente, ya disponemos de los archivos de entrada para el programa. Procedemos a su ejecución mediante la siguiente instrucción, teniendo en cuenta que en esta ocasión utilizaremos para ello el ejecutable anteriormente creado.

```
>> time bin/hadoop jar tmp/wordcount/wordcount.jar
org.myorg.WordCount /home/hduser/Gutenberg10
/home/hduser/Gutenberg10-salida
```

Hemos incluido la orden "*time*" para contabilizar el tiempo de ejecución, vemos el resultado:

```
hduser@nodomaster:/usr/local/hadoop$ time bin/hadoop jar tmp/wordcount/wordcount
.jar org.myorg.WordCount /home/hduser/Gutenberg10 /home/hduser/Gutenberg10-salid
a
Warning: $HADOOP_HOME is deprecated.

12/06/25 09:54:21 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
12/06/25 09:54:22 INFO util.NativeCodeLoader: Loaded the native-hadoop library
12/06/25 09:54:22 WARN snappy.LoadSnappy: Snappy native library not loaded
12/06/25 09:54:22 INFO mapred.FileInputFormat: Total input paths to process : 10
12/06/25 09:54:22 INFO mapred.JobClient: Running job: job_201206210904_0011
12/06/25 09:54:23 INFO mapred.JobClient:  map 0% reduce 0%
12/06/25 09:54:42 INFO mapred.JobClient:  map 20% reduce 0%
12/06/25 09:54:43 INFO mapred.JobClient:  map 60% reduce 0%
12/06/25 09:54:44 INFO mapred.JobClient:  map 100% reduce 0%
12/06/25 09:55:00 INFO mapred.JobClient:  map 100% reduce 100%
12/06/25 09:55:05 INFO mapred.JobClient: Job complete: job_201206210904_0011
12/06/25 09:55:05 INFO mapred.JobClient: Counters: 31
12/06/25 09:55:05 INFO mapred.JobClient:   Job Counters
12/06/25 09:55:05 INFO mapred.JobClient:     Launched reduce tasks=1
12/06/25 09:55:05 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=128951
12/06/25 09:55:05 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/06/25 09:55:05 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/06/25 09:55:05 INFO mapred.JobClient:     Rack-local map tasks=1
12/06/25 09:55:05 INFO mapred.JobClient:     Launched map tasks=10
12/06/25 09:55:05 INFO mapred.JobClient:     Data-local map tasks=9
12/06/25 09:55:05 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=17670
12/06/25 09:55:05 INFO mapred.JobClient:   File Input Format Counters
12/06/25 09:55:05 INFO mapred.JobClient:     Bytes Read=10528276
12/06/25 09:55:05 INFO mapred.JobClient:   File Output Format Counters
12/06/25 09:55:05 INFO mapred.JobClient:     Bytes Written=2701122
12/06/25 09:55:05 INFO mapred.JobClient:   FileSystemCounters
12/06/25 09:55:05 INFO mapred.JobClient:     FILE_BYTES_READ=6371351
12/06/25 09:55:05 INFO mapred.JobClient:     HDFS_BYTES_READ=10529343
12/06/25 09:55:05 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=11432399
12/06/25 09:55:05 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=2701122
12/06/25 09:55:05 INFO mapred.JobClient:   Map-Reduce Framework
12/06/25 09:55:05 INFO mapred.JobClient:     Map output materialized bytes=48238
86
12/06/25 09:55:05 INFO mapred.JobClient:     Map input records=208120
12/06/25 09:55:05 INFO mapred.JobClient:     Reduce shuffle bytes=4823886
12/06/25 09:55:05 INFO mapred.JobClient:     Spilled Records=644383
12/06/25 09:55:05 INFO mapred.JobClient:     Map output bytes=17156700
12/06/25 09:55:05 INFO mapred.JobClient:     Total committed heap usage (bytes)=
1348575232
12/06/25 09:55:05 INFO mapred.JobClient:     CPU time spent (ms)=25670
12/06/25 09:55:05 INFO mapred.JobClient:     Map input bytes=10528276
12/06/25 09:55:05 INFO mapred.JobClient:     SPLIT_RAW_BYTES=1067
12/06/25 09:55:05 INFO mapred.JobClient:     Combine input records=1678026
12/06/25 09:55:05 INFO mapred.JobClient:     Reduce input records=269273
12/06/25 09:55:05 INFO mapred.JobClient:     Reduce input groups=161463
12/06/25 09:55:05 INFO mapred.JobClient:     Combine output records=269273
12/06/25 09:55:05 INFO mapred.JobClient:     Physical memory (bytes) snapshot=18
69381632
```

Figura 6.14. Prueba *WordCount* 10 (1 de 2)

En la *Figura 6.14*, vemos como reconoce los diez libros como entrada y a continuación comienza a realizar las tareas *Map* para, tras su finalización, ejecutar las tareas *Reduce*. En la *Figura 6.15* vemos la finalización de la ejecución y el tiempo de duración de la misma.

```
12/06/25 09:55:05 INFO mapred.JobClient:     Reduce output records=161463
12/06/25 09:55:05 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=540
6617600
12/06/25 09:55:05 INFO mapred.JobClient:     Map output records=1678026

real    0m45.321s
user    0m1.950s
sys     0m0.110s
hduser@nodomaster:/usr/local/hadoop$
```

Figura 6.15. Prueba *WordCount* 10 (2 de 2)

Termina por tanto la ejecución y nos devuelve, gracias a la orden "*time*", el tiempo de que ha tardado en contabilizar todas las palabras de la biblioteca de entrada que ha sido de 45,321 segundos.

Para asegurarnos de que la ejecución ha sido exitosa, revisamos que se han creado los ficheros de salida, gracias al comando:

```
>> bin/hadoop dfs -ls /home/hduser/
```

Lo escribimos en línea de comandos y comprobamos el resultado.

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -ls /home/hduser/
Warning: $HADOOP_HOME is deprecated.

Found 12 items
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:21 /home/hduser/Gutenb
erg10
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:55 /home/hduser/Gutenb
erg10-salida
```

Figura 6.16. Contenido HDFS Prueba 10 Salida

Efectivamente, se han creado los ficheros de salida, por ello aparentemente la ejecución ha sido exitosa. No obstante, vamos a obtener una copia del fichero de salida que almacenaremos dentro del directorio temporal de *Hadoop* con el comando:

```
>> bin/hadoop dfs -getmerge /home/hduser/Gutenberg10-salida
tmp/Gutenberg10-Salida/
```

Así podemos abrir el fichero y comprobar si en él están los vocablos de cada uno de los libros ordenados junto con su número de apariciones.

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -getmerge /home/hduser/Guten
berg10-salida tmp/Gutenberg10-Salida/
Warning: $HADOOP_HOME is deprecated.

12/06/26 08:31:45 INFO util.NativeCodeLoader: Loaded the native-hadoop library
hduser@nodomaster:/usr/local/hadoop$
```

Figura 6.17. Importar Salida *WordCount* 10

Seguidamente hacemos una vista del inicio del fichero y nos aseguramos de que la ejecución ha sido correcta.

```
hduser@nodomaster:/usr/local/hadoop/tmp/Gutenberg10-Salida$ head Gutenberg10-sal
ida
"       34
"'A     2
"'About 1
"'Absolute      1
"'Ah!'  3
"'Ah,   2
"'All   1
"'Among 2
"'Ample.'       1
"'And   11
hduser@nodomaster:/usr/local/hadoop/tmp/Gutenberg10-Salida$
```

Figura 6.18. Resultado *WordCount* 10

Para concluir, haremos un resumen de los resultados obtenidos. Se han tomado dos mediciones, la primera de ellas contabiliza el tiempo que ha tardado el entorno *Apache Hadoop* en incluir los archivos de entrada en el HDFS tal y como se especificó en la configuración (repartir y replicar entre los nodos). La segunda de ellas, mide el tiempo de ejecución del programa.

	Incluir Archivos en HDFS (mm:ss,d)	Ejecución (mm:ss,d)
***WordCount* 10 Libros**	00:03,2	00:45,3

Tabla 6.1. Resultado *WordCount 10*

6.2.2. Ejecución *WordCount* sobre cien libros

La siguiente prueba de rendimiento de nuestro entorno consistirá en realizar un *wordcount* sobre una biblioteca de entrada que aumentará el número de libros de la del apartado anterior, llegando a cien unidades.

Comenzamos, como en los casos anteriores, distribuyendo los libros que forman parte de la entrada del programa e incluyéndolos dentro del sistema de archivos distribuido de *Hadoop*. Lo hacemos escribiendo en línea de comandos la siguiente instrucción:

```
>> time bin/hadoop dfs -copyFromLocal tmp/Gutenberg100 /home/hduser/
```

Hemos incluido la orden *"time"* para obtener la duración de esta acción. Vemos el resultado en la *Figura 6.19*.

```
hduser@nodomaster:/usr/local/hadoop$ time bin/hadoop dfs -copyFromLocal tmp/Gute
nberg100 /home/hduser/
Warning: $HADOOP_HOME is deprecated.


real    0m25.172s
user    0m3.090s
sys     0m0.700s
```

Figura 6.19. Incluir Biblioteca 100 en HDFS

Ha tardado algo más de veinticinco segundos en copiar los libros y distribuirlos entre todos los nodos del clúster. Vamos a asegurarnos de que se ha creado el directorio con la información enviada, de la misma forma que en todos los casos.

```
>> bin/hadoop dfs -ls /home/hduser/
```

Comprobamos a continuación que se crea el directorio mencionado.

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -ls /home/hduser/
Warning: $HADOOP_HOME is deprecated.

Found 3 items
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:21 /home/hduser/Gutenb
erg10
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:23 /home/hduser/Gutenb
erg100
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:22 /home/hduser/Gutenb
erg50
```

Figura 6.20. Contenido HDFS Prueba 100 Entrada

Teniendo ya los datos de entrada en nuestro *framework*, pasamos a ejecutar el programa. Hemos de ejecutar la siguiente orden:

```
>> time bin/hadoop jar tmp/wordcount/wordcount.jar
org.myorg.WordCount /home/hduser/Gutenberg100
/home/hduser/Gutenberg100-salida
```

Como en todos los casos, incluimos "*time*" en la orden y así nos devuelve el tiempo de ejecución.

```
hduser@nodomaster:/usr/local/hadoop$ time bin/hadoop jar tmp/wordcount/wordcount
.jar org.myorg.WordCount /home/hduser/Gutenberg100 /home/hduser/Gutenberg100-sal
ida
Warning: $HADOOP_HOME is deprecated.

12/06/25 10:03:13 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
12/06/25 10:03:14 INFO util.NativeCodeLoader: Loaded the native-hadoop library
12/06/25 10:03:14 WARN snappy.LoadSnappy: Snappy native library not loaded
12/06/25 10:03:14 INFO mapred.FileInputFormat: Total input paths to process : 10
0
12/06/25 10:03:14 INFO mapred.JobClient: Running job: job_201206210904_0013
12/06/25 10:03:15 INFO mapred.JobClient:  map 0% reduce 0%
12/06/25 10:03:34 INFO mapred.JobClient:  map 1% reduce 0%
12/06/25 10:03:35 INFO mapred.JobClient:  map 13% reduce 0%
12/06/25 10:03:36 INFO mapred.JobClient:  map 17% reduce 0%
12/06/25 10:03:37 INFO mapred.JobClient:  map 18% reduce 0%
12/06/25 10:03:44 INFO mapred.JobClient:  map 20% reduce 0%
12/06/25 10:03:47 INFO mapred.JobClient:  map 28% reduce 0%
12/06/25 10:03:48 INFO mapred.JobClient:  map 31% reduce 0%
12/06/25 10:03:50 INFO mapred.JobClient:  map 35% reduce 0%
12/06/25 10:03:56 INFO mapred.JobClient:  map 38% reduce 9%
12/06/25 10:03:59 INFO mapred.JobClient:  map 45% reduce 9%
12/06/25 10:04:00 INFO mapred.JobClient:  map 49% reduce 9%
12/06/25 10:04:02 INFO mapred.JobClient:  map 54% reduce 12%
12/06/25 10:04:08 INFO mapred.JobClient:  map 59% reduce 12%
12/06/25 10:04:11 INFO mapred.JobClient:  map 65% reduce 18%
12/06/25 10:04:12 INFO mapred.JobClient:  map 69% reduce 18%
12/06/25 10:04:14 INFO mapred.JobClient:  map 73% reduce 18%
12/06/25 10:04:17 INFO mapred.JobClient:  map 78% reduce 19%
12/06/25 10:04:20 INFO mapred.JobClient:  map 80% reduce 19%
12/06/25 10:04:23 INFO mapred.JobClient:  map 84% reduce 24%
12/06/25 10:04:24 INFO mapred.JobClient:  map 88% reduce 24%
12/06/25 10:04:26 INFO mapred.JobClient:  map 98% reduce 26%
12/06/25 10:04:29 INFO mapred.JobClient:  map 98% reduce 29%
12/06/25 10:04:32 INFO mapred.JobClient:  map 100% reduce 29%
12/06/25 10:04:38 INFO mapred.JobClient:  map 100% reduce 32%
12/06/25 10:04:48 INFO mapred.JobClient:  map 100% reduce 100%
12/06/25 10:04:53 INFO mapred.JobClient: Job complete: job_201206210904_0013
12/06/25 10:04:53 INFO mapred.JobClient: Counters: 31
12/06/25 10:04:53 INFO mapred.JobClient:   Job Counters
12/06/25 10:04:53 INFO mapred.JobClient:     Launched reduce tasks=1
12/06/25 10:04:53 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=1024689
12/06/25 10:04:53 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/06/25 10:04:53 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/06/25 10:04:53 INFO mapred.JobClient:     Rack-local map tasks=4
12/06/25 10:04:53 INFO mapred.JobClient:     Launched map tasks=109
12/06/25 10:04:53 INFO mapred.JobClient:     Data-local map tasks=105
12/06/25 10:04:53 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=70841
12/06/25 10:04:53 INFO mapred.JobClient:   File Input Format Counters
12/06/25 10:04:53 INFO mapred.JobClient:     Bytes Read=66568148
12/06/25 10:04:53 INFO mapred.JobClient:   File Output Format Counters
12/06/25 10:04:53 INFO mapred.JobClient:     Bytes Written=7291988
12/06/25 10:04:53 INFO mapred.JobClient:   FileSystemCounters
12/06/25 10:04:53 INFO mapred.JobClient:     FILE_BYTES_READ=30535636
```

Figura 6.21. Prueba *WordCount* 100 (1 de 2)

Lo primero es asegurarnos de que el número de libros de entrada es cien, que efectivamente lo es. Posteriormente vemos como se inician las tareas *Map*, comenzando las tareas *Reduce* antes de que la ejecución de los *Mappers* llegue al cuarenta por ciento.

Cuando finaliza la ejecución obtenemos la duración de la misma.

```
12/06/25 10:04:53 INFO mapred.JobClient:     HDFS_BYTES_READ=66578848
12/06/25 10:04:53 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=57396603
12/06/25 10:04:53 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=7291988
12/06/25 10:04:53 INFO mapred.JobClient:   Map-Reduce Framework
12/06/25 10:04:53 INFO mapred.JobClient:     Map output materialized bytes=24682
641
12/06/25 10:04:53 INFO mapred.JobClient:     Map input records=1372410
12/06/25 10:04:53 INFO mapred.JobClient:     Reduce shuffle bytes=24562050
12/06/25 10:04:53 INFO mapred.JobClient:     Spilled Records=3687296
12/06/25 10:04:53 INFO mapred.JobClient:     Map output bytes=110055208
12/06/25 10:04:53 INFO mapred.JobClient:     Total committed heap usage (bytes)=
13221957632
12/06/25 10:04:53 INFO mapred.JobClient:     CPU time spent (ms)=180790
12/06/25 10:04:53 INFO mapred.JobClient:     Map input bytes=66568148
12/06/25 10:04:53 INFO mapred.JobClient:     SPLIT_RAW_BYTES=10700
12/06/25 10:04:53 INFO mapred.JobClient:     Combine input records=11562904
12/06/25 10:04:53 INFO mapred.JobClient:     Reduce input records=1644597
12/06/25 10:04:53 INFO mapred.JobClient:     Reduce input groups=559143
12/06/25 10:04:53 INFO mapred.JobClient:     Combine output records=1949557
12/06/25 10:04:53 INFO mapred.JobClient:     Physical memory (bytes) snapshot=18
057048064
12/06/25 10:04:53 INFO mapred.JobClient:     Reduce output records=559143
12/06/25 10:04:53 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=497
53808896
12/06/25 10:04:53 INFO mapred.JobClient:     Map output records=11257944

real    1m41.006s
user    0m2.140s
sys     0m0.110s
hduser@nodomaster:/usr/local/hadoop$
```

Figura 6.22. Prueba *WordCount* 100 (2 de 2)

En este caso, observamos que está por encima del minuto y medio. Antes de dar por finalizada la prueba vamos a cerciorarnos de que se han creado los ficheros de salida de la misma forma que lo hacemos para los de entrada.

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -ls /home/hduser/
Warning: $HADOOP_HOME is deprecated.

Found 14 items
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:21 /home/hduser/Gutenb
erg10
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:55 /home/hduser/Gutenb
erg10-salida
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:23 /home/hduser/Gutenb
erg100
drwxr-xr-x   - hduser supergroup          0 2012-06-25 10:04 /home/hduser/Gutenb
erg100-salida
drwxr-xr-x   - hduser supergroup          0 2012-06-25 09:22 /home/hduser/Gutenb
erg50
drwxr-xr-x   - hduser supergroup          0 2012-06-25 10:00 /home/hduser/Gutenb
erg50-salida
```

Figura 6.23. Contenido HDFS Prueba 100 Salida

Para terminar, mostramos en la *Tabla 6.2* con los tiempos de ejecución de esta prueba.

	Incluir Archivos en HDFS (mm:ss,d)	Ejecución (mm:ss,d)
***WordCount* 100 Libros**	00:25,2	01:41,0

Tabla 6.2. Resultado *WordCount* 100

6.2.3. Ejecución *WordCount* sobre mil libros

Para finalizar las pruebas de rendimiento del entorno *Apache Hadoop MapReduce* con el programa de conteo de vocablos de una librería de entrada *WordCount*, se va a llevar a cabo el último de los trabajos con un millar de textos.

Al igual que en los casos anteriores, la primera labor es dar la orden al *framework* de que distribuya, a partir del *NameNode,* todos los libros que forman parte de la biblioteca que tenemos preparada en nuestro sistema de archivos local entre todos los *DataNodes* que conforman nuestro HDFS. Para ello, ejecutamos el siguiente comando:

```
>> time bin/hadoop dfs -copyFromLocal tmp/Gutenberg1000
/home/hduser/
```

Veamos a continuación la duración de este proceso en la *Figura 6.24*.

```
hduser@nodomaster:/usr/local/hadoop$ time bin/hadoop dfs -copyFromLocal tmp/Gute
nberg1000/ /home/hduser/
Warning: $HADOOP_HOME is deprecated.


real    4m12.411s
user    0m8.800s
sys     0m3.620s
hduser@nodomaster:/usr/local/hadoop$
```

Figura 6.24. Incluir Biblioteca 1000 en HDFS

Es destacable el hecho de que este *framework* es capaz de recibir mil archivos de texto y dividirlos en bloques que reparte entre todos los nodos de almacenamiento generando réplicas de los mismos para mantener la seguridad y metadatos para saber en todo momento dónde se almacena la información, todo ello en poco más de cuatro minutos.

Teniendo ya los datos correctamente distribuidos en nuestro sistema de ficheros, el siguiente paso consiste en realizar la ejecución del programa que trabaja sobre ellos. Para ello, hemos de emitir la siguiente orden:

```
>> time bin/hadoop jar tmp/wordcount/wordcount.jar
org.myorg.WordCount /home/hduser/Gutenberg1000
/home/hduser/Gutenberg1000-salida
```

Así se inicia el trabajo *WordCount*, lo vemos en las siguientes figuras.

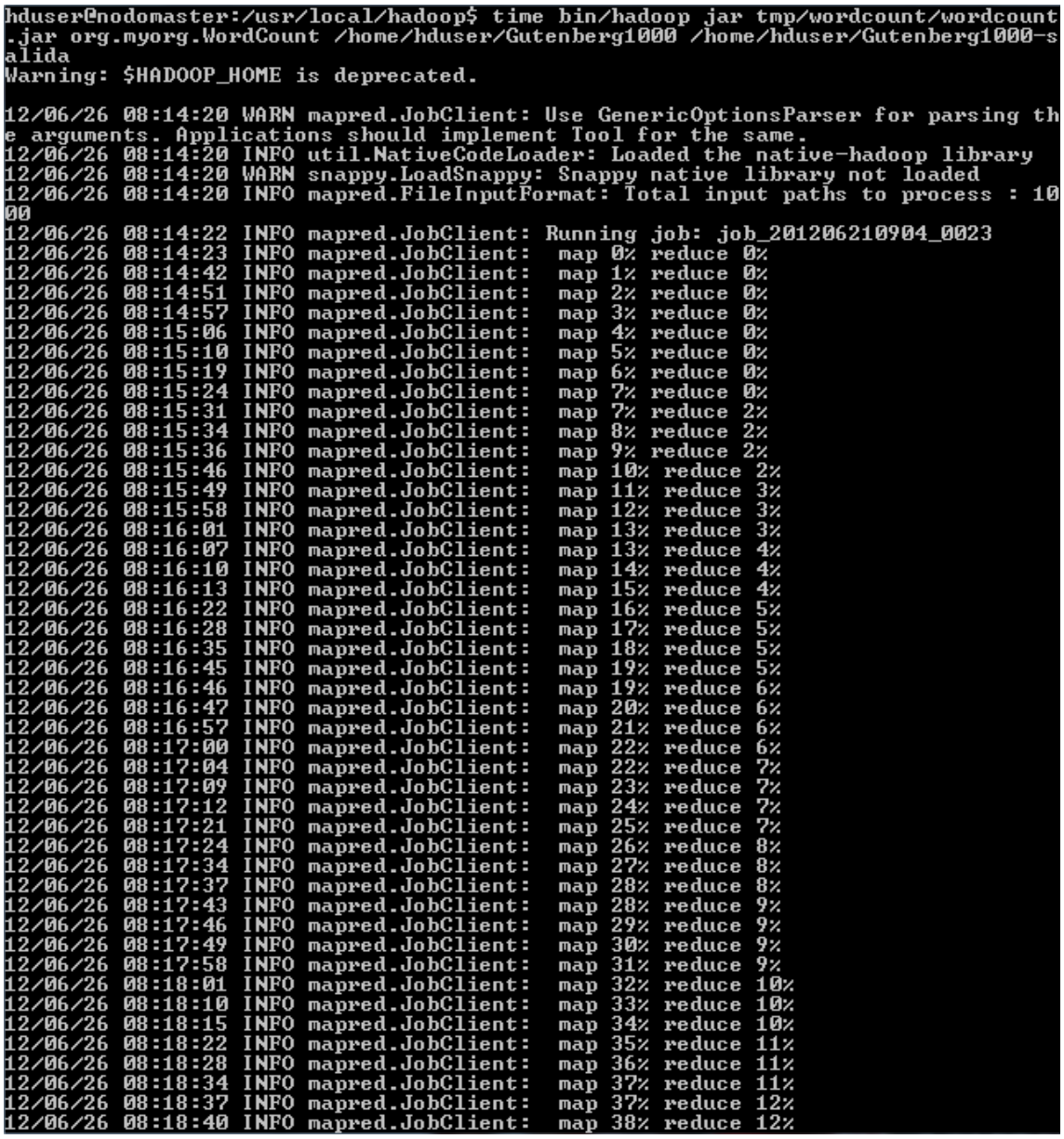


Figura 6.25. Prueba *WordCount* 1000 (1 de 4)

Reconoce los mil libros que enviamos como entrada del trabajo, emite un identificador del trabajo y comienzan a ejecutarse las taras *Map*. Cuando ya hay alguna tarea *Reduce* pendiente y *TaskTrackers* liberados de trabajo, empiezan también estas instrucciones.

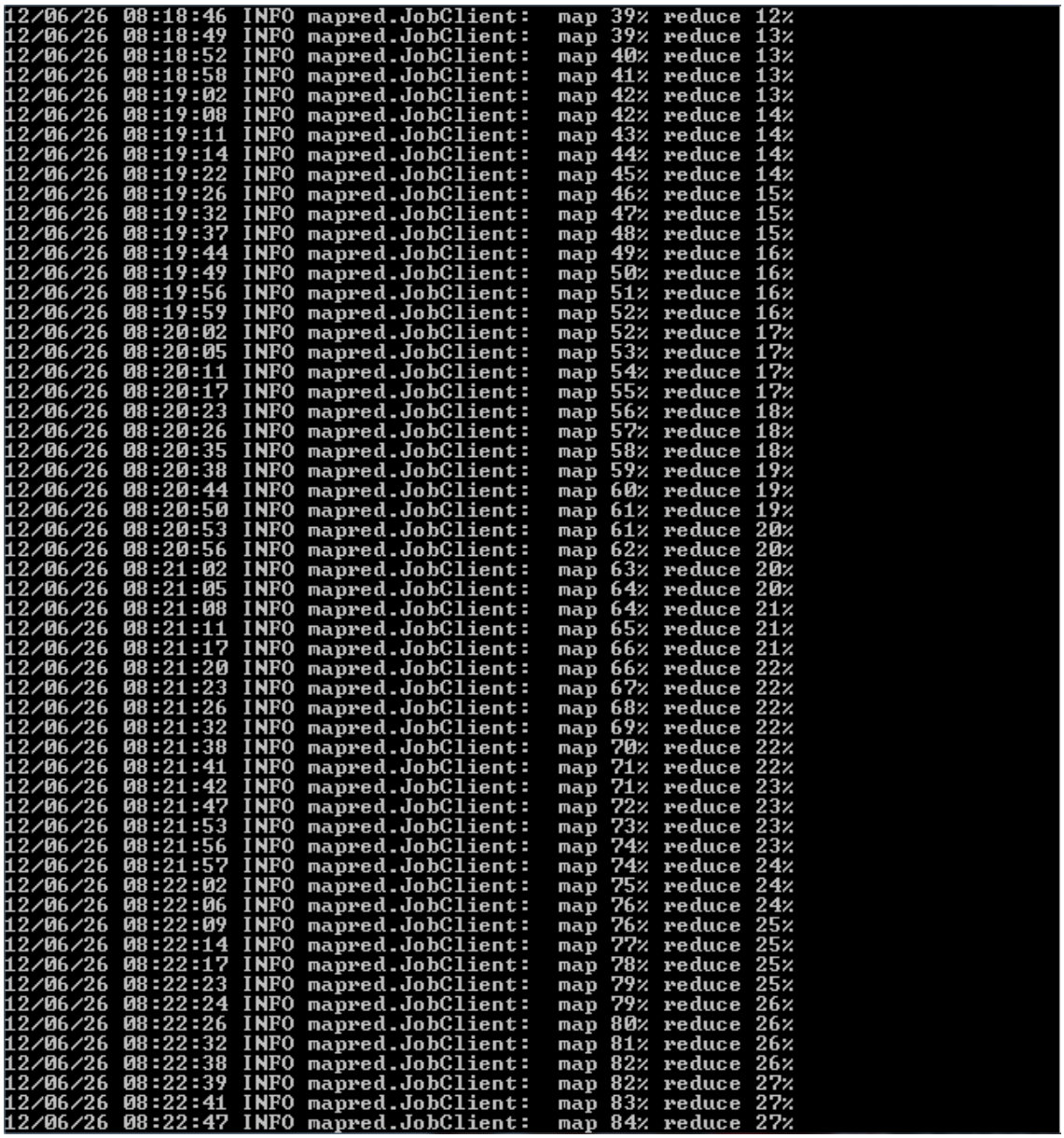


Figura 6.26. Prueba *WordCount* 1000 (2 de 4)

Continúa el trabajo avanzando en paralelo ambas tareas, con mayor progresión de las que son llevadas a cabo en los *Mappers*.

```
12/06/26 08:22:51 INFO mapred.JobClient:  map 85% reduce 27%
12/06/26 08:22:54 INFO mapred.JobClient:  map 85% reduce 28%
12/06/26 08:22:56 INFO mapred.JobClient:  map 86% reduce 28%
12/06/26 08:23:02 INFO mapred.JobClient:  map 87% reduce 28%
12/06/26 08:23:05 INFO mapred.JobClient:  map 88% reduce 28%
12/06/26 08:23:09 INFO mapred.JobClient:  map 88% reduce 29%
12/06/26 08:23:11 INFO mapred.JobClient:  map 89% reduce 29%
12/06/26 08:23:14 INFO mapred.JobClient:  map 90% reduce 29%
12/06/26 08:23:20 INFO mapred.JobClient:  map 91% reduce 29%
12/06/26 08:23:21 INFO mapred.JobClient:  map 91% reduce 30%
12/06/26 08:23:26 INFO mapred.JobClient:  map 92% reduce 30%
12/06/26 08:23:29 INFO mapred.JobClient:  map 93% reduce 30%
12/06/26 08:23:35 INFO mapred.JobClient:  map 94% reduce 30%
12/06/26 08:23:36 INFO mapred.JobClient:  map 94% reduce 31%
12/06/26 08:23:38 INFO mapred.JobClient:  map 95% reduce 31%
12/06/26 08:23:44 INFO mapred.JobClient:  map 96% reduce 31%
12/06/26 08:23:48 INFO mapred.JobClient:  map 97% reduce 31%
12/06/26 08:23:51 INFO mapred.JobClient:  map 97% reduce 32%
12/06/26 08:23:53 INFO mapred.JobClient:  map 98% reduce 32%
12/06/26 08:23:59 INFO mapred.JobClient:  map 99% reduce 32%
12/06/26 08:24:02 INFO mapred.JobClient:  map 100% reduce 32%
12/06/26 08:24:03 INFO mapred.JobClient:  map 100% reduce 33%
12/06/26 08:24:21 INFO mapred.JobClient:  map 100% reduce 66%
12/06/26 08:24:28 INFO mapred.JobClient:  map 100% reduce 69%
12/06/26 08:24:31 INFO mapred.JobClient:  map 100% reduce 74%
12/06/26 08:24:34 INFO mapred.JobClient:  map 100% reduce 85%
12/06/26 08:24:43 INFO mapred.JobClient:  map 100% reduce 100%
12/06/26 08:24:48 INFO mapred.JobClient: Job complete: job_201206210904_0023
12/06/26 08:24:48 INFO mapred.JobClient: Counters: 31
12/06/26 08:24:48 INFO mapred.JobClient:   Job Counters
12/06/26 08:24:48 INFO mapred.JobClient:     Launched reduce tasks=1
12/06/26 08:24:48 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=8628231
12/06/26 08:24:48 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/06/26 08:24:48 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/06/26 08:24:48 INFO mapred.JobClient:     Rack-local map tasks=5
12/06/26 08:24:48 INFO mapred.JobClient:     Launched map tasks=1010
12/06/26 08:24:48 INFO mapred.JobClient:     Data-local map tasks=1005
12/06/26 08:24:48 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=571484
12/06/26 08:24:48 INFO mapred.JobClient:   File Input Format Counters
12/06/26 08:24:48 INFO mapred.JobClient:     Bytes Read=454640882
12/06/26 08:24:48 INFO mapred.JobClient:   File Output Format Counters
12/06/26 08:24:48 INFO mapred.JobClient:     Bytes Written=28909284
12/06/26 08:24:48 INFO mapred.JobClient:   FileSystemCounters
12/06/26 08:24:48 INFO mapred.JobClient:     FILE_BYTES_READ=147581614
12/06/26 08:24:48 INFO mapred.JobClient:     HDFS_BYTES_READ=454749225
12/06/26 08:24:48 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=353878764
12/06/26 08:24:48 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=28909284
12/06/26 08:24:48 INFO mapred.JobClient:   Map-Reduce Framework
12/06/26 08:24:48 INFO mapred.JobClient:     Map output materialized bytes=18470
2720
12/06/26 08:24:48 INFO mapred.JobClient:     Map input records=9450758
12/06/26 08:24:48 INFO mapred.JobClient:     Reduce shuffle bytes=184702720
12/06/26 08:24:48 INFO mapred.JobClient:     Spilled Records=22522925
12/06/26 08:24:48 INFO mapred.JobClient:     Map output bytes=757959147
12/06/26 08:24:48 INFO mapred.JobClient:     Total committed heap usage (bytes)=
```

Figura 6.27. Prueba *WordCount* 1000 (3 de 4)

Finaliza el proceso de trabajo, comprobando una vez más cómo cuando finalizan las tareas *Map,* las *Reduce* avanzan rápidamente, finalizando en poco tiempo.

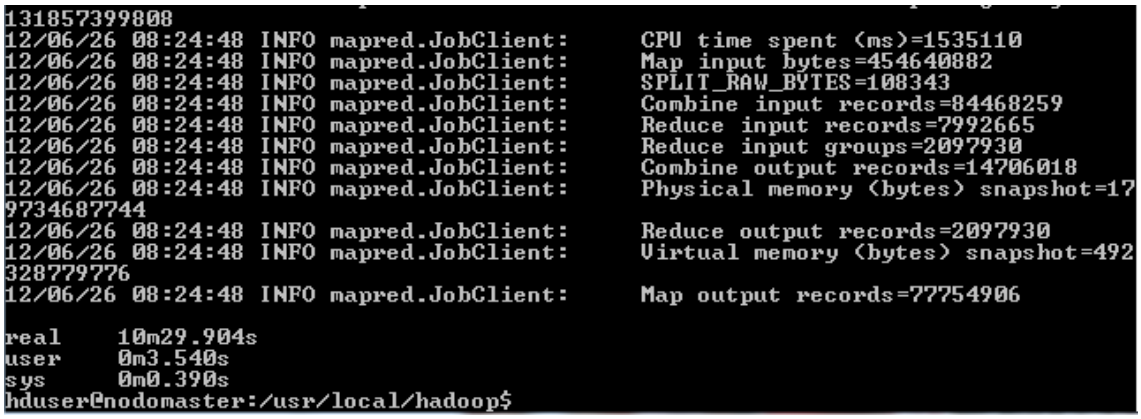


Figura 6.28. Prueba *WordCount* 1000 (4 de 4)

Llegamos al final de la última prueba de rendimiento de nuestro entorno, alcanzando un tiempo de cómputo de diez minutos y medio, realmente ajustado si pensamos en que la labor realizada implica contabilizar todos los vocablos de mil textos y generar un fichero con cada uno de ellos acompañado con su número de apariciones.

Finalizamos el apartado, realizando una síntesis de la prueba en una tabla con los tiempos de ejecución registrados.

	Incluir Archivos en HDFS (mm:ss,d)	Ejecución (mm:ss,d)
***WordCount* 1000 Libros**	04:12,4	10:29,9

Tabla 6.3. Resultado *WordCount* 1000

6.3. Resultados Obtenidos en las Pruebas de Rendimiento *MapReduce*

En el *Apartado 6.2. Prueba de Rendimiento MapReduce* compilamos y creamos un ejecutable de un código *Java* que lleva a cabo, de manera más eficiente que el código de ejemplo del entorno *Apache Hadoop*, el conteo de palabras de una biblioteca de entrada y el número de ocasiones que aparece cada uno de los vocablos. Elegimos este programa, llamado *WordCount*¸ debido a que nos permite testear el rendimiento de nuestro clúster midiendo variables de tiempo en las dos tareas básicas que se pueden realizar en este *framework*.

- Carga de Archivos en el sistema de ficheros distribuido HDFS.
- Ejecución de trabajos *MapReduce*.

Por un lado, realizamos cargas con ficheros de entrada que aumentan en número de archivos desde diez libros a mil; y por otro, ejecutamos el programa sobre dichos conjuntos de entrada. Así tomamos, para cada uno de los casos, dos mediciones de tiempo, cada una correspondiente a una de las tareas anteriores. En la siguiente tabla, podemos ver recogidos conjuntamente todos los datos de forma que nos sea más sencillo obtener resultados y conclusiones de las pruebas realizadas.

Test *WordCount* (Libros)	Incluir Archivos en HDFS (mm:ss,d)	Ejecución (mm:ss,d)
10	00:03,2	00:45,3
50	00:13,1	01:12,4
100	00:25,2	01:41,0
150	00:42,6	02:18,8
200	01:04,6	03:02,9
300	01:50,8	03:58,1
400	02:25,5	05:04,7
500	02:53,8	05:54,7
600	02:59,8	06:46,5
700	03:21,6	07:43,7
800	03:27,1	08:18,0
900	03:48,3	09:44,7
1000	04:12,4	10:29,9

Tabla 6.4. Resumen General Mediciones *WordCount*

Si reparamos en los datos que aparecen en la tabla, podemos distinguir tres columnas. En la primera de ellas (izquierda) apreciamos el número de textos de entrada para cada una de las pruebas de rendimiento; en la segunda (central) se observa la evolución, medida en tiempo, de la carga de los datos de la columna anterior; en la última columna (derecha) aparece la duración de la ejecución del programa *WordCount* sobre los conjuntos de entrada correspondientes por fila.

Con esta tabla podemos obtener resultados, pero sí, con los datos que contiene, generamos un gráfico es más sencillo alcanzar conclusiones que nos permitan corroborar las características del entorno estudiadas en capítulos anteriores, así como llegar a obtener nuevas valoraciones del *framework* propias, derivadas de nuestras pruebas.

En el gráfico siguiente, vemos la representación de los datos de la *Tabla 6.4. Resumen General Mediciones WordCount*.

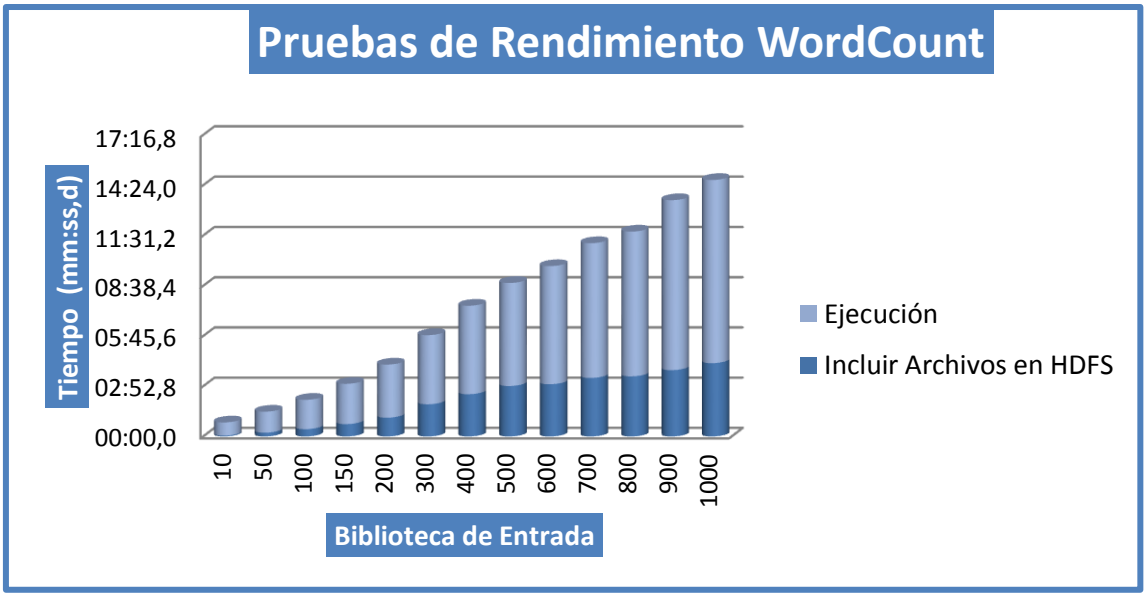


Gráfico 6.1. Ejecución y Carga HDFS

Este gráfico representa, mediante columnas, la suma de los datos de tiempo obtenidos en las dos tareas realizadas en las pruebas de rendimiento. El eje de abscisas (horizontal) contiene el número de libros de la biblioteca de entrada y el de ordenadas (vertical) contiene la duración total de ambas tareas, la suma.

De este modo, con la ayuda del gráfico, podemos apreciar que las distintas ejecuciones siguen cierta linealidad en cuanto al incremento de tiempo entre unas y otras pruebas. En los primeros intentos, la diferencia en tiempo de cómputo es menor que en el resto de test, algo lógico, debido a que el aumento en número de textos de entrada es mayor a partir de la prueba con doscientos libros.

Las mediciones están divididas en dos valores de tiempo, una de ellas toma el tiempo de carga de los archivos y la segunda el tiempo de ejecución del programa. Para obtener conclusiones más efectivas, es preferible separar el estudio de estas magnitudes, por ello, a continuación vamos a incluir dos apartados para hacer un análisis más exhaustivo de cada una de las mediciones. En cualquier caso, a primera vista, observamos que el tiempo de carga de ficheros no mantiene un crecimiento tan constante como el tiempo de ejecución, es decir, para conjuntos de archivos grandes, el tiempo de carga no crece tanto como en las primeras pruebas. En la siguiente sección estudiaremos el porqué de este hecho. Por otro lado, el tiempo de ejecución de la aplicación sí parece que sigue una linealidad constante en cuanto al incremento de tiempo entre una y otra prueba.

Como hemos comentado, pasamos a estudiar de forma más concreta cada una de las mediciones por separado.

6.3.1. Resultados Carga de Ficheros en el HDFS

En las pruebas realizadas en el *Apartado 6.2. Prueba de Rendimiento MapReduce* hemos destacado un aspecto que, si bien no es el núcleo de la ejecución del programa que supone la mayor parte del trabajo, conviene tener en cuenta como es el hecho de incluir en el sistema de ficheros distribuido HDFS los datos que formen parte del test.

El *framework Apache Hadoop MapReduce*, como ya se expuso en los *Capítulos 3. Estudio del entorno Apache Hadoop* y *4. Estudio de la programación MapReduce* es idóneo para aplicaciones que computen sobre grandes volúmenes de datos. Por ello el tiempo que precisa el entorno en obtener y distribuir la información entre todos los nodos del clúster puede ser importante en la ejecución. Aun más conociendo el hecho de que es necesario dividir esos datos en bloques, generar metadatos que ayuden a la localización de los mismos y que, según la configuración *Apache Hadoop* desplegada, suele ser preciso replicar la información para mantener la seguridad.

En entornos en los que los datos sean introducidos una vez en el sistema y se realicen continuas ejecuciones de aplicaciones sobre el mismo bloque de información, el tiempo de incluir los datos en el sistema va perdiendo peso según crece el número de trabajos. Es decir, para un entorno en el que se trabaje sobre un contenedor de información de cien gigabytes, es obvio que la vez que ingresemos los datos en el sistema nos va a llevar un tiempo elevado (variable según el clúster). Por ello, el primer trabajo que llevemos a cabo en dicho entorno, tendrá un coste muy elevado si tenemos en cuenta la duración de la carga de los archivos. Sin embargo, la segunda aplicación y sucesivas, no tendrán ese costo de tiempo, ya que si computan sobre los mismos datos ya cargados, solo se contabilizará tiempo de ejecución puro.

En el caso que nos ocupa, hemos buscado un trabajo *Hadoop MapReduce* en el que la información de entrada cambia para cada una de las pruebas, ya que no sólo añadíamos nuevos libros en la biblioteca de entrada por cada prueba, sino que además esos libros difieren entre uno y otro trabajo. Es decir, entre la prueba con cien libros y la de ciento cincuenta textos no incluimos cincuenta nuevos libros, sino que incluimos por completo una librería nueva. Así conseguimos unas pruebas de rendimiento en las que las condiciones son cambiantes. Es interesante para nosotros ya que queremos evaluar el comportamiento del entorno *Apache Hadoop MapReduce* en todas las condiciones posibles, dadas las características de nuestro clúster y hacer pruebas con conjuntos de datos de entrada tan cambiantes arroja una mayor cantidad de información.

Por tanto, los datos que hemos recogido en las pruebas del apartado anterior y que vamos a estudiar en este punto, recogen el tiempo que emplea el entorno en incluir las diferentes librerías de entrada para cada una de las ejecuciones del programa *WordCount* utilizado. En la tabla que incluimos a continuación podemos ver estas mediciones.

Test *WordCount* (Libros)	Incluir Archivos en HDFS
10	00:03,2
50	00:13,1
100	00:25,2
150	00:42,6
200	01:04,6
300	01:50,8
400	02:25,5
500	02:53,8
600	02:59,8
700	03:21,6
800	03:27,1
900	03:48,3
1000	04:12,4

Tabla 6.5. Mediciones Carga HDFS

En una primera vista, podemos obtener conclusiones que resultan bastante interesantes. Si pensamos en las operaciones que se realizan en cada uno de las pruebas con los conjuntos de textos, la duración de cada una de las tareas de carga de los datos en el HDFS es realmente ajustada. Es decir, si nos fijamos en el último de los trabajos, incluir un millar de libros en el sistema precisa una duración de poco más de cuatro minutos. Esta labor supone los siguientes pasos:

- Copiar los datos de un sistema de archivos local y enviarlos al sistema de ficheros distribuido *Hadoop Distributed File System* configurado en nuestro entorno.
- En *NameNode* ha de dividir la información en bloques (el tamaño difiere según la configuración).
- Posteriormente el mismo nodo decide dónde se almacena cada uno de los bloques y genera metadatos que guardan el estado de los mismos.
- Generar réplicas de la información según dicte la configuración del clúster y almacenarlas y generar metadatos que vinculen estas copias de datos con los originales y dónde se encuentran almacenados.

Si pensamos que nuestro entorno *Apache Hadoop MapReduce* es capaz de realizar este trabajo en poco más de cuatro minutos y contando para ello con un clúster de **nueve máquinas virtuales (un nodo maestro y ocho nodos esclavos)**, parece que **podemos afirmar**, sin temor a equivocarnos, **que lo hace con un rendimiento muy superior al esperado**.

Para tratar de obtener nuevas conclusiones, vamos a dibujar la evolución de la duración de esta tarea en cada una de las pruebas en un gráfico. Así podremos ver de forma más clara el comportamiento en cada uno de los test realizados sobre nuestro entorno.

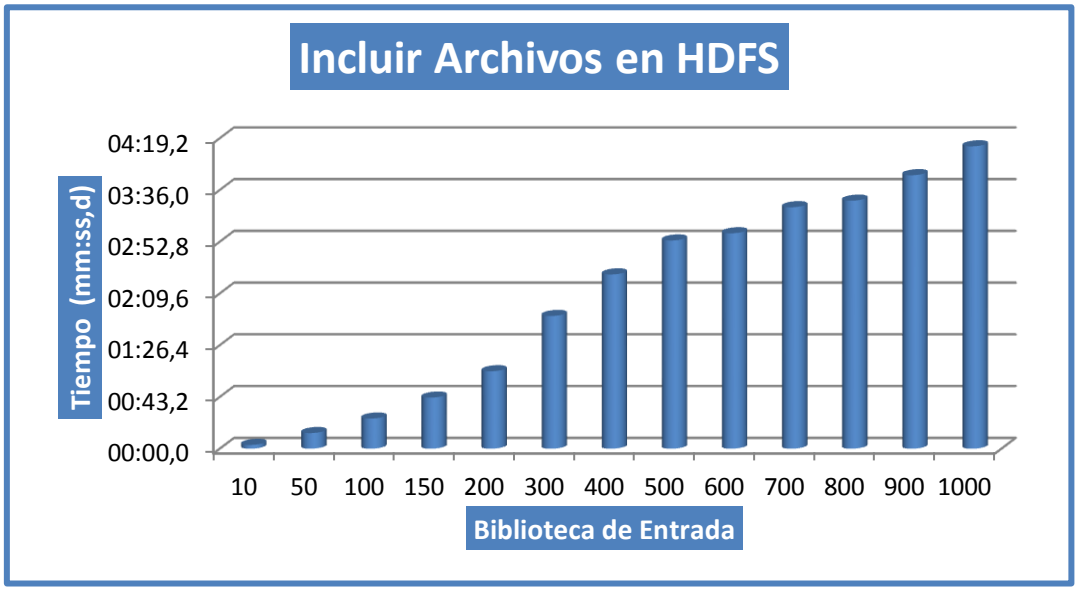


Gráfico 6.2. Carga Bibliotecas en HDFS

Si analizamos el gráfico obtenido con la duración de la carga de los libros en el sistema de ficheros distribuido podemos obtener ciertas conclusiones que afianzan las características del *framework Apache Hadoop MapReduce* estudiadas en los Capítulos 3 y 4 de esta memoria.

Los dos aspectos fundamentales que podemos abstraer son:

- Cuanto mayor son los conjuntos de datos de entrada, mejor es el comportamiento.
- Las diferencias en tiempo de carga para grandes conjuntos de datos, son menos significativas que entre grupos menores.

Las mayores diferencias en la duración de la tarea se producen para conjuntos de datos pequeños, dos casos claros son el aumento en tiempo que se aprecia entre el test con cincuenta y cien libros y la prueba con doscientos textos y el de trescientos. En ambos casos la diferencia en el número de archivos de entrada es menor que en las pruebas posteriores, pero también es cierto que la proporción de aumento es mayor. Además a este incremento de la duración afecta el hecho del que el propio entorno invierte parte del tiempo total en tareas de gestión y generación de metadatos. Cuanto menor son los conjuntos de datos, más significativas son estas tareas del entorno. Por tanto, con conjuntos menores, se obtienen peores rendimientos, ya que afectan en mayor medida tanto las labores de gestión del entorno como el incremento en el volumen de los datos de entrada. Por otro lado, si nos centramos en las pruebas a partir de quinientos libros, aparecen casos entre los que la diferencia de tiempo es casi inapreciable. Esto ocurre entre la tarea para quinientos y seiscientos ficheros o la tarea entre setecientos y ochocientos. Con esto reafirmamos lo anteriormente comentado, ya que para conjuntos de datos mayores, el incremento en el

volumen de datos y las tareas de generación de metadatos no suponen un aumento en el tiempo global significativo.

Cuando realizamos el estudio del entorno *Apache Hadoop*, de su sistema de ficheros distribuido *Hadoop Distributed File System* y de la programación *MapReduce*, una de las características principales obtenidas fue que el comportamiento de este sistema de archivos distribuido es más eficiente cuanto mayor es el volumen de datos con el que trabaja la aplicación *MapReduce*. Sabíamos que este *framework* estaba especialmente indicado para trabajos sobre conjuntos de datos de gran peso, recordamos que en la descripción oficial de *Apache Hadoop* hablan de que el entorno está especialmente indicado para hasta petabytes de datos, y esto lo hemos podido comprobar según íbamos aumentando el número de libros en los test de rendimiento realizados.

Si bien es cierto, este entorno de trabajo está diseñado para sistemas en los que se produzcan cuantas menos cargas de archivos mejor, es decir, el tipo de sistema ideal sería aquel en el que se produce una primera gran carga de archivos y posteriormente si realizan ejecuciones de trabajos *MapReduce* sobre ese conjunto de datos. Por ello, para obtener el mejor rendimiento en los test de *WordCount* lo idóneo habría sido realizar en primer lugar la carga de mil libros y sobre ella realizar todos los test, empezando desde diez textos hasta el millar. Sin embargo, dado que el objetivo de este proyecto es estudiar las propiedades de este *framework* y probar si son ciertas las características expuestas en los capítulos anteriores, decidimos realizar una a una las cargas de los conjuntos de datos.

Para concluir, podemos afirmar que **el comportamiento del sistema de ficheros *Hadoop Distributed File System* expuesto de manera teórica ha sido confirmado empíricamente** en la parte dedicada a incluir los archivos en el HDFS de las pruebas de rendimiento *MapReduce* con el programa *WordCount*, ya que **para grandes volúmenes de datos el funcionamiento del HDFS es mucho más eficiente** que para conjuntos más pequeños.

6.3.2. Resultados Ejecución *WordCount*

Tras estudiar el comportamiento del sistema de ficheros distribuido para las cargas de los conjuntos de datos de entrada, llega el momento de analizar los datos obtenidos en las mediciones del tiempo de ejecución del test *WordCount*.

Al igual que en el caso anterior, se realizan pruebas sobre bibliotecas de entrada que van desde los diez libros hasta el millar, avanzando de cincuenta en cincuenta inicialmente para acabar aumentando el número de textos de cien en cien. Cada prueba consiste en compilar el ejecutable del programa *WordCount* sobre los conjuntos de datos de entrada ya cargados en el sistema.

Si nos trasladamos al estudio teórico, la ventaja en tiempo de cómputo de este entorno viene dada por el sistema de ficheros distribuido. En cuanto a la ejecución del programa, se obtiene ventaja con respecto a otros entornos gracias a la interacción de la programación *MapReduce* y el sistema de ficheros distribuido HDFS.

Con los datos que hemos obtenido, vamos a estudiar el incremento de tiempo entre las distintas pruebas, si se mantiene constante el crecimiento, podremos decir que efectivamente, tal y como se planteó en el estudio teórico, el entorno de computación distribuida *Apache Hadoop* está diseñado para entornos en que se trabaje con grandes volúmenes de datos, dado que la ejecución del programa no se ve afectada por los conjuntos de datos con los que trabaja. Hemos de suponer que este hecho tendrá un límite marcado por recursos del clúster.

Para poder comenzar el análisis de las mediciones obtenidas, lo primero es tener esos valores de tiempo recogidos, en nuestro caso, en la tabla.

Test *WordCount* (Libros)	Tiempo de Ejecución (mm:ss,d)
10	00:45,3
50	01:12,4
100	01:41,0
150	02:18,8
200	03:02,9
300	03:58,1
400	05:04,7
500	05:54,7
600	06:46,5
700	07:43,7
800	08:18,0
900	09:44,7
1000	10:29,9

Tabla 6.6. Mediciones Ejecución *WordCount*

Si estudiamos los datos de la tabla, podemos ver que la evolución es casi constante entre cada una de las pruebas. En los primeros test en que la diferencia de textos de entrada es de cincuenta, el incremento de tiempo ronda los treinta segundos; en las siguientes ejecuciones en que las se dobla el aumento de libros de entrada a cien archivos, el tiempo de ejecución también duplica su acrecentamiento a un minuto.

Por tanto, parece que se comprueba lo que estábamos buscando con estas pruebas, que para grandes conjuntos de datos de entrada el comportamiento del entorno es el mismo que para menores volúmenes de datos. Esto, **en otros entornos de computación distribuida** no es así, ya que lo que suele ocurrir es que, **al aumentar progresivamente el volumen de datos con los que trabajar el tiempo de ejecución se dispara y no crece de forma constante**.

En cualquier caso conviene representar las mediciones obtenidas para terminar de asegurar las conclusiones obtenidas, lo hacemos en el gráfico que se presenta a continuación.

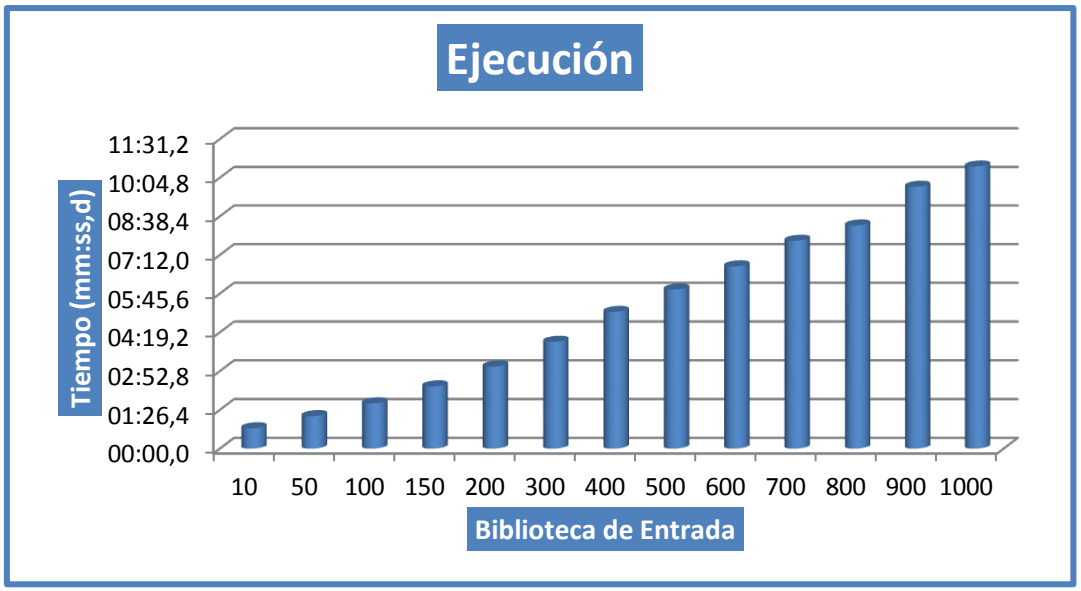


Gráfico 6.3. Ejecuciones *WordCount*

Vemos como el crecimiento es lineal y constante, el aumento de tiempo entre una y otra prueba es fijo, dejando claro que el incremento de los libros en la biblioteca de entrada afecta de forma normal, dado que existen más datos con los que trabajar, pero en ningún momento el clúster se ve desbordado por los archivos de entrada.

Otra variable que podemos estudiar para conocer cómo se comporta el entorno, es la relación de los datos estudiados, es decir, por un lado tenemos los libros que conforman la biblioteca de entrada y por otro, los segundos invertidos en cada una de las pruebas. De esa relación, podemos obtener el tiempo que invierte el entorno en analizar cada libro;

$$Si\ para\ X\ libros\ se\ invierte\ un\ tiempo\ \alpha\ \rightarrow Para\ 1\ libro\ el\ tiempo\ es\ \beta$$

$$\beta = \frac{1L * \alpha"}{XL}$$

Por tanto, vamos a crear una nueva tabla con los datos de tiempo en segundos y con el tiempo que se invierte en analizar un libro.

Libros	Segundos Totales	Segundos por Libro
10	45”	4,5”
50	72”	1,44”
100	101”	1,01”
150	139”	0,93”
200	183”	0,91”
300	238”	0,79”
400	305”	0,76”
500	355”	0,71”
600	407”	0,68”
700	464”	0,66”
800	498”	0,62”
900	585”	0,65”
1000	630”	0,63”

Tabla 6.7. Relación Tiempo por Libro

Al calcular la relación entre el tiempo de procesamiento y los ficheros de entrada de la prueba, obtenemos cuánto tarda el entorno en analizar un texto en cada prueba, es decir, para cada libro en la primera prueba se emplean cuatro segundos y medio y para el test de mil libros se revisa cada uno de ellos en poco más de medio segundo.

En a la *Tabla 6.7. Relación Tiempo por Libro* podemos corroborar que **el *framework Apache Hadoop* junto con la programación *MapReduce* obtiene mejor rendimiento cuanto mayor son los conjuntos de datos de entrada**. Esto es así debido a que para agrupaciones pequeñas de datos, el tiempo que emplea el entorno en la creación y gestión de tareas afecta directamente al tiempo de cómputo. Sin embargo, **conforme el volumen de los datos de entrada aumenta, menos se ve afectado el tiempo de cómputo por el tiempo de gestión de tareas *MapReduce***. Además, también apreciamos algo anteriormente descubierto, y es que para los mayores conjuntos de datos, **el tiempo de ejecución observa un crecimiento casi constante entre pruebas**. Así vemos como el **coeficiente de segundos por libro se mantiene prácticamente invariable en las últimas pruebas**.

Otra conclusión que podemos sacar de esta síntesis de datos es que **el tiempo que emplea el clúster en analizar los grandes volúmenes de datos es realmente ajustado**. Comprobar que un millar de libros son analizados, palabra por palabra, anotando en un fichero cada uno de los distintos vocablos y contabilizando sus apariciones en menos de diez minutos y medio puede parecer un muy buen resultado. Pero es que si analizamos un poco más en profundidad, vemos que el clúster es capaz de analizar cada libro en poco más de medio segundo, teniendo en cuenta que en ese tiempo van incluidas todas las tareas que genera el *framework Hadoop MapReduce* de comunicación entre los nodos, de asignación de trabajos, etc., así como todas las labores de apertura de ficheros, creación de archivos y escritura.

6.4. Conclusiones de las Pruebas de Rendimiento *MapReduce*

Tras llevar a cabo los distintos test de rendimiento con los que pusimos a prueba nuestro clúster y tras analizar los resultados obtenidos, se han extraído una serie de conclusiones que nos han llevado a confirmar las características de entorno estudiadas en los capítulos teóricos anteriores.

A pesar de que ya se han ido citando según iban apareciendo las conclusiones obtenidas, pasamos a recogerlas en este apartado para así mostrarlas de manera esquemática.

El objetivo principal de las pruebas de rendimiento es someter a nuestro clúster a trabajos en los que el volumen de datos es elevado y así poder comprobar su eficacia. Por un lado llevamos a cabo los test sobre el sistema de ficheros distribuido y por otro realizamos la ejecución de un programa *Java MapReduce*. En ambos casos, ambas tareas arrojan los siguientes resultados:

- **Para grandes volúmenes de datos el funcionamiento del HDFS es mucho más eficiente.**
- **Un clúster *Hadoop* junto con la programación *MapReduce* obtiene mejor rendimiento cuanto mayor son los conjuntos de datos de entrada**.
- **El *framework Apache Hadoop* está diseñado para sistemas en los que los datos se carguen una vez y sobre ellos se realicen múltiples trabajos.**

En ambos casos, los recursos del clúster marcan el límite de estas dos conclusiones. Sin embargo, nosotros no hemos llegado a alcanzar ese tope dado que no hemos realizado ninguna prueba en la que el conjunto de entrada fuera tan grande como para colapsar el clúster.

6.5. Uso de la Librería HIPI

Realizando el estudio teórico del entorno, apreciamos la dificultad que el *framework Apache Hadoop* junto con la programación *MapReduce* tiene a la hora de trabajar con imágenes. El lenguaje *MapReduce* posee sus propios formatos de archivos y en ninguno de ellos se contemplan instrucciones para trabajar con imágenes, ni tan siquiera las más comunes, como pueden ser abrir o cerrar, acceso a pixel, etc. Si pensamos en los beneficios de este marco de trabajo, que en general tienen que ver con ventajas a la hora de trabajar con volúmenes de datos grandes, un sistema de computación en los que los datos sean imágenes, parece muy adecuado dado el peso que pueden alcanzar estos archivos. Además dado el mundo multimedia en el que vivimos, en el que todo el mundo tiene un *Smartphone* o una cámara de fotos digital y conexión a Internet, todos compartimos, editamos y comentamos imágenes. Surgen muchas aplicaciones de entornos distribuidos para trabajar con grandes volúmenes de imágenes, como pueden ser redes sociales (recordamos que *Facebook* utiliza *Hadoop*), sistemas de edición de imágenes distribuidos en el que se preste el software como servicio, etc.

El principal problema, es poder trabajar con imágenes en este entorno de trabajo. Sin embargo, la librería *Hadoop Image Processing Interface* contiene las clases *Java* que otorgan al modelo de programación *MapReduce* la funcionalidad de manejo de imágenes como las de las clases *Image*, *ImageIcon* o *ImageIO* de *Java*. Recordamos que en el *Apartado 4.4. Librería HIPI* podemos encontrar toda la información sobre esta librería, para poder comprender los pasos siguientes es recomendable repasar ese punto de la memoria.

El objetivo de este apartado es probar esta librería, ejecutar programas que trabajen con imágenes siguiendo las directrices marcadas por HIPI. Vamos a llevar a cabo dos aplicaciones, la primera de ellas crea un contenedor de imágenes *Hipi Image Bundle* (HIB) de una lista de enlaces de descarga. El segundo código, toma ese contenedor, lo abre y crea un fichero de salida en el que almacena propiedades de cada una de las imágenes.

6.5.1. Creación de un contenedor HIB (*Hipi Image Bundle*)

En el *Capítulo 4. Estudio de la Programación MapReduce*, se expuso de manera teórica el funcionamiento de esta librería y dimos a conocer el hecho de que para trabajar con imágenes utilizando HIPI es preciso crear un contenedor que las almacena. Este contenedor denominado HIB, es un almacén de imágenes conjuntadas en un solo archivo similar a los ficheros ".tar" en *GNU/Linux* y que está diseñado para poder tratar con aplicaciones que utilicen imágenes en entornos *Hadoop MapReduce*.

Para aprender a manejar estos contenedores, vamos a ejecutar un código llamado ***"HIPI Distributed Downloader"*** que crea un HIB a partir de una lista de URL's. Estas direcciones de descarga han de contener imágenes y pueden utilizarse links como los de la red social *Flickr* o de *Google Images*. La lista de enlaces para descargar ha de estar en un archivo de texto ASCII con una URL por línea. Descargar un gran conjunto de imágenes en una sola máquina puede alcanzar un tiempo prohibitivo ya que puede llegar a saturarse el ancho de banda de la misma. Sin embargo llevar a cabo esta tarea en un clúster no supone un esfuerzo muy elevado en cuanto a tiempo de ejecución.

La aplicación *Distributed Downloader* aprovecha el clúster *Hadoop* para, con un fichero de entrada de texto con los enlaces, realizar las descargas de manera distribuida como tarea *Map* entre todos los nodos. En la *Figura 6.29* podemos ver el procedimiento seguido por el programa.

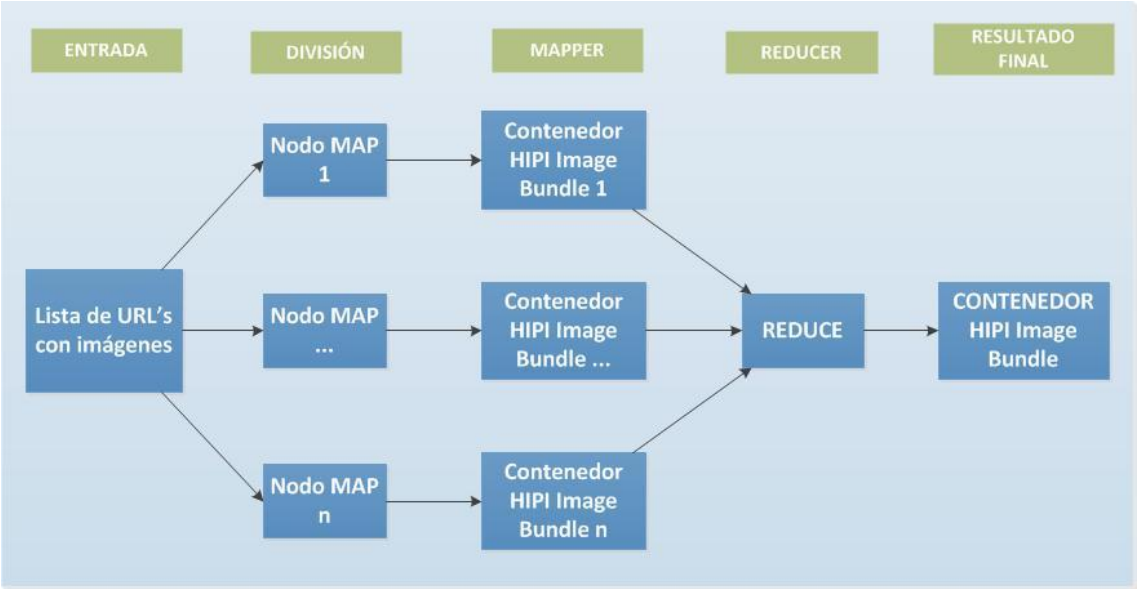


Figura 6.29. Diagrama Creación Contenedor HIB

Esta ejecución sigue la misma estructura de todos los programas *MapReduce* que ya hemos expuesto y utilizado anteriormente. Todos los programas de este entorno se inician con un contenedor, fichero o listado de entrada; en este caso, recibimos como entrada un fichero de texto con una lista de enlaces de descarga de imágenes. Posteriormente se produce la división y generación de tareas *Map*. Acto seguido, se ejecutan las tareas *Mapper* que, en este caso, generan como pares intermedios, contenedores con la imagen descargada. Tras las tareas *Map*, las tareas *Reduce*, que reciben los contenedores intermedios, los aglutinan y generan como salida el contenedor HIP definitivo con todas las imágenes descargadas.

El código de este programa se encuentra recogido en el *Anexo I. Pruebas HIPI* y en él podemos encontrar todas las clases y métodos que forman parte de la aplicación.

El fichero de entrada que vamos a utilizar, va a contener enlaces de descarga de la red social *Flickr* y cada uno de ellos en una línea. En la *Figura 6.30* podemos ver su formato:

```
http://farm3.static.flickr.com/2413/1637516180_7d7aac7831_b.jpg
http://farm3.static.flickr.com/2386/2016921042_c351deb735_b.jpg
http://farm3.static.flickr.com/2395/2016921570_5935e4cb07_b.jpg
http://farm2.static.flickr.com/1054/692713810_8bc84408c1_b.jpg
...
```

Figura 6.30. Fichero *Flickr*

El programa, utiliza una clase llamada `DownloaderInputFormat` propia del *Distributed Downloader* que es una versión especial de la clase estándar de *Hadoop* `FileInputFormat` y que es capaz de forzar a un determinado número de nodos de un clúster *Hadoop* a procesar las líneas de un fichero de manera distribuida. Es decir, repartir las líneas del fichero de entrada para que sean procesadas entre todos los nodos. El procedimiento es dividir la cuantía de imágenes entre el número de nodos, para que cada uno de los nodos descargue la misma cantidad de ellas y las almacene después en un HIB. Cuando finalice esta tarea, la fase *Reduce* ordena todas las imágenes en el HIB definitivo.

DownloaderInputFormat

Bajo un funcionamiento normal, *Hadoop* envía el código ejecutable de un trabajo a los nodos del clúster que contienen los datos que van a ser operados. Dado que los datos de entrada son simples enlaces de descarga, el *NameNode* del sistema de ficheros, en la mayoría de los casos, distribuirá el fichero por completo con los enlaces en un único nodo. Por ello, para este trabajo, ha de configurar el comportamiento de *Hadoop* para que actúe del modo adecuado, es decir que no tome el fichero como un archivo único, sino que lo interprete como un conjunto de descargas que son distribuidas entre los nodos del clúster.

La clase `DownloaderInputFormat` actúa de forma que crea archivos temporales en el sistema de ficheros distribuido HDFS con cada uno de los enlaces de descarga y envía la orden de ejecución de dichas descargas entre los diversos nodos de manera balanceada. El proceso es el siguiente:

1. Cree un fichero temporal en una localización aleatoria del HDFS.

```
...
ArrayList<String> hosts = new ArrayList<String>(0);
FileSystem fileSystem = FileSystem.get(conf);
String tempOutputPath = conf.get("downloader.outpath") + "_tmp";
Path tempOutputDir = new Path(tempOutputPath);
int i = 0;
while( hosts.size() < nodes && i < 2*nodes)
{
  String tempFileString = tempOutputPath + "/" + i;
  Path tempFile = new Path(tempFileString);
  FSDataOutputStream os = fileSystem.create(tempFile);
  os.write(i);
  os.close();
```

Figura 6.31. Fichero Temporal

2. Determine en cuál de los nodos fue guardado el fichero temporal.

```
FileStatus match = fileSystem.getFileStatus(tempFile);
long length = match.getLen();
BlockLocation[] blocks = fileSystem.getFileBlockLocations(match, 0, length);
```

Figura 6.32. Localización Fichero Temporal

3. Asegúrese de que este nodo es único. En caso contrario, comience de nuevo.

```
boolean save = true;
for (int j = 0; j < hosts.size(); j++)
{
    if (blocks[0].getHosts()[0].compareTo(hosts.get(j)) == 0)
    {
        save = false;
        break;
    }
}
```

Figura 6.33. Nodo Único

4. Si el nodo es único, se guarda en una lista de nodos y se repiten los pasos hasta completar la lista con todos los nodos del clúster.

```
if (save)
{
  hosts.add(blocks[0].getHosts()[0]);
}
i++;
```

Figura 6.34. Guardar en Lista de Nodos

La lista resultante de hosts se utiliza para generar un *FileSplit* (división de archivo) en cada nodo. El *FileSplit*, en este caso, es utilizado para indicar a los nodos, de que URLs serán responsables. El constructor de *FileSplit*, normalmente toma cuatro argumentos: la ruta del archivo, el byte inicial del *Split* o división, la longitud en bytes del *Split* y la lista de hosts donde el *Split* se encuentra almacenado. En lugar de utilizar esta forma convencional, el desplazamiento inicial y los parámetros de longitud se sustituyen con el desplazamiento de línea y el número de líneas a procesar, siguiendo así el formato del archivo que contiene los enlaces de descarga.

```
int span = (int) Math.ceil(((float) (num_lines)) / ((float) hosts.size()));
int last = num_lines - span * (hosts.size()-1);

FileSplit[] f = new FileSplit[hosts.size()];
for (int j = 0; j < f.length; j++)
{
    String[] host = new String[1];
    host[0] = hosts.get(j);
    if (j < f.length - 1)
    {
        splits.add( new FileSplit(path , (j*span) , span, host));
    } else
    {
        splits.add( new FileSplit(path , (j*span) , last, host));
    }
}
```

Figura 6.35. *Downloader FileSplit*

DownloaderRecordReader

El objeto *RecordReader* se encarga de emitir un conjunto de registros para cada *InputSplit* que recibe de un *InputFormat* especificado. En este caso, el *DownloaderRecordReader* es responsable de comunicar al *Mapper* las líneas de la lista de entrada de URLs de las que es responsable de descargar. Como se mencionó anteriormente, la clase *DownloaderInputFormat* utiliza el objeto *FileSplit* para guardar esta información.

Para cada *FileSplit*, el *DownloaderRecordReader* emite exactamente un registro cuya clave indica el número de la línea de partida y cuyo valor contiene la lista de URLs a procesar.

```
...
public void initialize(InputSplit split, TaskAttemptContext context)
  throws IOException, InterruptedException {

  FileSplit f = (FileSplit) split;
  Path path = f.getPath();

  start_line = f.getStart();
  long num_lines = f.getLength();
```

Figura 6.36. *DownloaderRecordReader* (1 de 3)

Las URLs correspondientes al nodo en cuestión, se determinan mediante la lectura del archivo leyendo las líneas "*num_lines*" desde el valor de inicio "*start_line*".

```
BufferedReader reader = new BufferedReader(new InputStreamReader(fs.open(path)));
int i = 0;
while(i < start_line && reader.readLine() != null){
  i++;
}

urls = "";
String line;
for(i = 0; i < num_lines && (line = reader.readLine()) != null; i++){
  urls += line + '\n';
}
reader.close();
```

Figura 6.37. *DownloaderRecordReader* (2 de 3)

Los Mappers reciben sus registros llamando repetidamente al `RecordReader::NextKeyValue ()`, `RecordReader::getCurrentKey ()`, y `RecordReader::GetCurrentValue ()`. Por lo tanto, en este caso, sólo existe un registro cuya clave y valor corresponden con un valor de inicio "start_line" y la lista de URLs respectivamente:

```
public IntWritable getCurrentKey() throws IOException, InterruptedException {
    return new IntWritable((int)start_line);
}

public Text getCurrentValue() throws IOException, InterruptedException {
    return new Text(urls);
}

public boolean nextKeyValue() throws IOException, InterruptedException {
    if(singletonEmit == false){
        singletonEmit = true;
        return true;
    }
    else
        return false;
}
```

Figura 6.38. *DownloaderRecordReader* (3 de 3)

Mapper

La clase RecordReader emite registros que envía a los ejecutores de tareas Map (Mappers) donde las claves son la línea de inicio de lectura de la lista de entrada de URLs y los valores son las URLs a ser descargadas (separadas por caracteres de nueva línea). La tarea Mapper descarga cada una de las imágenes contenidas en los enlaces Web en un HipiImageBundle local (HIB). Al finalizar, el contenedor local es enviado al reductor donde se combina con el resto de HIBs locales.

Los HIBs temporales se crean en cada uno de los nodos Mappers para completar el conjunto de imágenes a descargar:

```
public void map(IntWritable key, Text value, Context context)
  throws IOException, InterruptedException
{
  String temp_path = conf.get("downloader.outpath") + key.get() + ".hib.tmp";

  HipiImageBundle hib = new HipiImageBundle(new Path(temp_path), conf);
  hib.open(HipiImageBundle.FILE_MODE_WRITE, true);
  ...
```

Figura 6.39. *Downloader Mapper* Iniciación

Cada uno de los nodos procede a iterar sobre el conjunto de direcciones URL y para llevar a cabo la descarga real de las imágenes, se crear una *URLConnection* que genera una corriente de datos (*stream*) que puede ser escrita directamente en el HIB.

```
String word = value.toString();
BufferedReader reader = new BufferedReader(new StringReader(word));
String uri;

while((uri = reader.readLine()) != null)
{
  ...
  String type = "";
  URLConnection conn;

  try {
    URL link = new URL(uri);
    conn = link.openConnection();
    conn.connect();
    type = conn.getContentType();
  } catch (Exception e)
  {
    continue;
  }
  if (type != null && type.compareTo("image/jpeg") == 0)
    hib.addImage(conn.getInputStream(), ImageType.JPEG_IMAGE);
  ...
}
```

Figura 0.40. *Downloader Mapper* Procedimiento

Una vez que las descargas de las URLs con las imágenes han sido procesadas, el contenedor HIB puede ser cerrado indicando la localización del mismo en el HDFS al *Reducer*.

```
hib.close();
context.write(new BooleanWritable(true), new Text(hib.getPath().toString()));
```

Figura 6.41. *Downloader Mapper* Finalización

Reducer

El trabajo del *Reducer* consiste en cargar los contenedores intermedios procedentes de los *Mappers*, ordenarlos y combinarlos en un gran HIB con todas las imágenes.

```
public void reduce(BooleanWritable key, Iterable<Text> values, Context context)
  throws IOException, InterruptedException
{
  if(key.get()) {
    FileSystem fileSystem = FileSystem.get(conf);
    HipiImageBundle hib = new HipiImageBundle(new Path(conf.get("downloader.outfile")), conf);
    hib.open(HipiImageBundle.FILE_MODE_WRITE, true);
    for (Text temp_string : values) {
      Path temp_path = new Path(temp_string.toString());
      HipiImageBundle input_bundle = new HipiImageBundle(temp_path, conf);
      hib.append(input_bundle);

      Path index_path = input_bundle.getPath();
      Path data_path = new Path(index_path.toString() + ".dat");

      context.write(new BooleanWritable(true), new Text(input_bundle.getPath().toString()));
      context.progress();
    }
    hib.close();
  }
}
```

Figura 6.42. *Downloader Reducer*

6.5.2. Ejecución *Distributed Downloader*

El código que vamos a ejecutar para comprender la utilización de los contenedores de imágenes HIB propios de la librería HIPI, toma como entrada una lista de enlaces de descarga de la red social *Flickr* que contienen imágenes. El procedimiento del mismo es tomar los enlaces, descargarlos de forma distribuida entre los nodos del clúster, creando contenedores intermedios y posteriormente en la fase *Reduce* crear un contenedor definitivo ordenando los generados por los *Mappers*.

Compilar el código

Antes de poder compilar un programa que utilice la librería HIPI, es necesario configurar el *script* que contiene la información sobre donde reside la instalación de HIPI.

Para ello, hemos de acceder al directorio raíz donde instalamos la librería y abrir el fichero "*build.xml*". En este archivo hay dos propiedades que están sin completar y que hacen referencia a la versión de *Hadoop* del clúster (*hadoop.version*) y a la ubicación de la instalación (*hadoop.home*). Hemos de rellenar estos dos campos, quedando un fichero similar a la *Figura 6.43*.

```
<project basedir="." default="all">

<target name="setup">
<property name="hadoop.home" value="/hadoop/hadoop-0.20.1" />
<property name="hadoop.version" value="0.20.1" />
<property name="hadoop.classpath" value="${hadoop.home}/hadoop-${hadoop.version}-core.jar" />
<property name="metadata.jar" value="3rdparty/metadata-extractor-2.3.1.jar" />
</target>
...
```

Figura 6.43. Editar *build.xml*

Marcadas con el cuadro azul vemos ambas propiedades y como quedarían después de modificar los valores.

Tras esta modificación, ya es posible compilar el código. En el fichero anteriormente modificado, están configuradas las instrucciones necesarias para compilar el programa.

```
<!-- The distributed downloader, which takes a database of URL's and creates a HIB -->
<target name="downloader">
  <antcall target="compile">
    <param name="srcdir" value="examples/hipi/examples/downloader" />
    <param name="files" value="DownloaderInputFormat.java,
        DownloaderRecordReader.java, Downloader.java" />
    <param name="jarfilename" value="downloader.jar" />
    <param name="jardir" value="examples" />
    <param name="mainclass" value="hipi.examples.downloader.Downloader" />
  </antcall>
</target>
```

Figura 6.44. *build.xml Downloader*

Como vemos, realiza la llamada a un *target* llamado "*compile*" que es donde están configurados de manera genérica los pasos de la compilación.

```
<target name="compile" depends="setup, test_settings">
  <mkdir dir="bin" />
  <!-- Compile -->
  <javac target="1.6" nowarn="on" srcdir="${srcdir}:./src:./examples:./experiments"
     includes="${files}" destdir="./bin" classpath="${hadoop.classpath}:${metadata.jar}" />
  <!-- Create the jar -->
  <jar destfile="${jardir}/${jarfilename}" basedir="./bin">
    <zipfileset src="${metadata.jar}" />
    <manifest>
  <attribute name="Main-Class" value="${mainclass}" />
    </manifest>
  </jar>
</target>
```

Figura 6.45. *build.xml Tarjet compile*

Por tanto, para realizar la compilación, simplemente hemos de ejecutar el siguiente comando:

```
$> ant downloader
```

Sin embargo, disponemos de un script que lleva a cabo tanto la compilación como la ejecución del programa. Se llama "*runDownloader.sh*" y contiene el siguiente código:

```
#!/bin/bash

ant -f ../build.xml downloader

hadoop jar downloader.jar $1 $2 $3
```

Si ponemos en marcha este *script*, en el mismo paso realizamos la compilación que genera las clases y el ejecutable del programa y se ejecuta la aplicación.

Este *script* precisa tres parámetros al ejecutarse en línea de comandos. El primero de ellos es la ruta en la que se encuentra el fichero de texto con los enlaces de descarga. El segundo es el directorio de salida donde queremos que se almacene el contenedor que se va a crear. El tercero corresponde con el número de nodos entre los que queremos que se distribuya la ejecución, que siempre ha de ser igual o inferior al número de nodos de nuestro clúster. Podría ejecutarse con el siguiente comando:

```
$>            ./runDownloader.sh            /home/hduser/lista.txt
/home/hduser/contenedor.hib 9
```

Una vez que conocemos las herramientas para compilar y ejecutar el código, llega el momento de probar el programa. En nuestro caso, vamos a realizar tres pruebas, aumentando el número de descargas progresivamente.

Como ya conocemos de las ejecuciones anteriores, el primer paso es incluir los archivos de entrada en el sistema de ficheros distribuido HDFS. Para el caso de *WordCount*, tomamos mediciones de la duración de esta tarea para así poder analizar los resultados posteriormente. En este caso, no es necesario tomarlas, ya que simplemente es un archivo de texto. En la *Figura 6.46* podemos ver cómo añadimos las listas con los enlaces de descarga en el sistema de ficheros, utilizando el comando:

```
$> bin/hadoop dfs -copyFromLocal tmp/ListaN /home/hduser
```

Donde N se corresponde con el número de enlaces de descarga de cada una de las imágenes.

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -copyFromLocal tmp/Lista10 /
home/hduser/
Warning: $HADOOP_HOME is deprecated.

hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -copyFromLocal tmp/Lista50 /
home/hduser/
Warning: $HADOOP_HOME is deprecated.

hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -copyFromLocal tmp/Lista100
/home/hduser/
Warning: $HADOOP_HOME is deprecated.
```

Figura 6.46. Incluir Listas en HDFS

Comprobamos que efectivamente han sido cargadas de forma satisfactoria utilizando el comando:

```
$> bin/hadoop dfs -ls /home/hduser
```

Esto nos muestra el contenido de la raíz del sistema de ficheros HDFS, donde deben aparecer las tres listas.

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -ls /home/hduser/
Warning: $HADOOP_HOME is deprecated.

Found 13 items
drwxr-xr-x   - hduser supergroup          0 2012-06-26 08:35 /home/hduser/Lista1
0
drwxr-xr-x   - hduser supergroup          0 2012-06-26 08:36 /home/hduser/Lista5
0
drwxr-xr-x   - hduser supergroup          0 2012-06-26 08:36 /home/hduser/Lista1
00
```

Figura 6.47. Contenido Raíz HDFS

Ya disponemos de las listas con los enlaces de descarga para las tres pruebas en nuestro HDFS, con lo que podemos comenzar a ejecutar el código del programa para cada una de ellas.

Ejecución Downloader con diez imágenes

La primera de las pruebas, se realiza sobre una lista de descargas de diez imágenes. Vamos a ejecutar el *script* que ejecuta el programa sobre esta lista, midiendo el tiempo que transcurre hasta que se genera el contenedor HIB.

Como ya hemos dado a conocer, el *script* que se va a utilizar compila el código del programa genera las clases y los ejecutables necesarios. La instrucción que ejecuta este conjunto de comandos es:

```
$>   time   ./runDownloader.sh   /home/hduser/Lista10/list10.txt
/home/hduser/Lista10/salida10.hib 9
```

Como argumentos de esta instrucción incluimos la ruta de la lista de enlaces que utilizamos como entrada, el directorio de salida en el que se almacena el contenedor y el número de nodos que queremos que participen en la ejecución (en nuestro caso 9).

Ejecutamos la instrucción y vemos el resultado de la misma en la *Figura 6.48*.

```
hduser@nodomaster:/usr/local/hadoop/hipi/examples$ time ./runDownloader.sh /home
/hduser/Lista10/list10.txt /home/hduser/Lista10/salida10.hib 9
Buildfile: ../build.xml

downloader:

setup:

test_settings:

compile:
    [javac] Compiling 3 source files to /usr/local/hadoop/hipi/bin
      [jar] Building jar: /usr/local/hadoop/hipi/examples/downloader.jar

BUILD SUCCESSFUL
Total time: 2 seconds
Warning: $HADOOP_HOME is deprecated.

Output HIB: /home/hduser/Lista10/
12/06/26 08:42:00 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
Found host successfully: 9
Tried to get 9 nodes, got 9
12/06/26 08:42:02 INFO input.FileInputFormat: Total input paths to process : 1
First n-1 nodes responsible for 10 images
Last node responsible for 10 images
12/06/26 08:42:02 INFO mapred.JobClient: Running job: job_201206210904_0025
12/06/26 08:42:03 INFO mapred.JobClient:  map 0% reduce 0%
12/06/26 08:42:19 INFO mapred.JobClient:  map 100% reduce 0%
12/06/26 08:42:31 INFO mapred.JobClient:  map 100% reduce 100%
12/06/26 08:42:36 INFO mapred.JobClient: Job complete: job_201206210904_0025
12/06/26 08:42:36 INFO mapred.JobClient: Counters: 29
12/06/26 08:42:36 INFO mapred.JobClient:   Job Counters
12/06/26 08:42:36 INFO mapred.JobClient:     Launched reduce tasks=1
12/06/26 08:42:36 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=18177
12/06/26 08:42:36 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/06/26 08:42:36 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/06/26 08:42:36 INFO mapred.JobClient:     Rack-local map tasks=1
12/06/26 08:42:36 INFO mapred.JobClient:     Launched map tasks=1
12/06/26 08:42:36 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=11181
12/06/26 08:42:36 INFO mapred.JobClient:   File Output Format Counters
12/06/26 08:42:36 INFO mapred.JobClient:     Bytes Written=36
12/06/26 08:42:36 INFO mapred.JobClient:   FileSystemCounters
12/06/26 08:42:36 INFO mapred.JobClient:     FILE_BYTES_READ=40
12/06/26 08:42:36 INFO mapred.JobClient:     HDFS_BYTES_READ=1479908
12/06/26 08:42:36 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=43981
12/06/26 08:42:36 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=2958401
12/06/26 08:42:36 INFO mapred.JobClient:   File Input Format Counters
12/06/26 08:42:36 INFO mapred.JobClient:     Bytes Read=612
12/06/26 08:42:36 INFO mapred.JobClient:   Map-Reduce Framework
12/06/26 08:42:36 INFO mapred.JobClient:     Map output materialized bytes=40
12/06/26 08:42:36 INFO mapred.JobClient:     Map input records=1
12/06/26 08:42:36 INFO mapred.JobClient:     Reduce shuffle bytes=0
12/06/26 08:42:36 INFO mapred.JobClient:     Spilled Records=2
12/06/26 08:42:36 INFO mapred.JobClient:     Map output bytes=32
12/06/26 08:42:36 INFO mapred.JobClient:     CPU time spent (ms)=2440
12/06/26 08:42:36 INFO mapred.JobClient:     Total committed heap usage (bytes)=
163581952
12/06/26 08:42:36 INFO mapred.JobClient:     Combine input records=0
12/06/26 08:42:36 INFO mapred.JobClient:     SPLIT_RAW_BYTES=115
12/06/26 08:42:36 INFO mapred.JobClient:     Reduce input records=1
12/06/26 08:42:36 INFO mapred.JobClient:     Reduce input groups=1
12/06/26 08:42:36 INFO mapred.JobClient:     Combine output records=0
12/06/26 08:42:36 INFO mapred.JobClient:     Physical memory (bytes) snapshot=25
5737856
12/06/26 08:42:36 INFO mapred.JobClient:     Reduce output records=1
12/06/26 08:42:36 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=986
230784
12/06/26 08:42:36 INFO mapred.JobClient:     Map output records=1

real    0m40.422s
user    0m4.580s
sys     0m0.190s
hduser@nodomaster:/usr/local/hadoop/hipi/examples$
```

Figura 6.48. Ejecución *Downloader* Lista 10

Las primeras líneas del resultado obtenido son referentes a la compilación del código que finaliza con la creación del ejecutable, que se denomina "*downloader.jar*". Posteriormente, intenta la conexión con los nueve nodos indicados en la entrada, resultando exitosa la misma. Tras estos primeros pasos, comienza la ejecución de las tareas *Map*, que descargan las

imágenes y generan los contenedores intermedios. Cuando éstas finalizan, toma el control el *Reducer* que ordena y combina estos contenedores, generando el HIB definitivo.

El siguiente paso es asegurarnos de que la ejecución del código ha sido exitosa, generándose tras la finalización de ésta el contenedor. Procedemos entonces a mostrar el contenido del directorio fijado para la prueba con diez imágenes, usando para ello el siguiente comando:

```
$> bin/hadoop dfs -ls /home/hduser/Lista10
```

El resultado que arroja el mismo es el siguiente:

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -ls /home/hduser/Lista10
Warning: $HADOOP_HOME is deprecated.

Found 4 items
-rw-r--r--   3 hduser supergroup        612 2012-06-26 08:41 /home/hduser/Lista1
0/list10.txt
-rw-r--r--   3 hduser supergroup        130 2012-06-26 08:42 /home/hduser/Lista1
0/salida10.hib
-rw-r--r--   3 hduser supergroup    1479054 2012-06-26 08:42 /home/hduser/Lista1
0/salida10.hib.dat
drwxr-xr-x   - hduser supergroup          0 2012-06-26 08:42 /home/hduser/Lista1
0/salida10.hib_output
```

Figura 6.49. Contenido HDFS *Downloader* Lista 10

Se ha generado el contenedor, así que podemos afirmar que la ejecución ha sido satisfactoria, descargándose todas las imágenes de la lista y almacenándose éstas de forma conjunta en un HIB.

Ejecución Downloader con cincuenta imágenes

Procedemos a continuación a realizar una nueva prueba del programa de generación de contenedores HIBs a partir de la descarga de imágenes, aumentando en este caso la lista de enlaces que utilizamos como entrada del programa a cincuenta URLs.

Los pasos a seguir para completar la ejecución son los mismos que en el caso anterior. Sabiendo que disponemos de la lista de enlaces en el sistema de ficheros distribuido HDFS, podemos ejecutar el *script* que compila el código y poner en marcha el programa. Para ello ejecutamos:

```
$>  time  ./runDownloader.sh  /home/hduser/Lista50/list50.txt
/home/hduser/Lista50/salida50.hib 9
```

Ejecutamos la instrucción por línea de comandos y el resultado de la misma lo vemos a continuación en la *Figura 6.50*.

```
hduser@nodomaster:/usr/local/hadoop/hipi/examples$ time ./runDownloader.sh /home
/hduser/Lista50/list50.txt /home/hduser/Lista50/salida50.hib 9
Buildfile: ../build.xml

downloader:

setup:

test_settings:

compile:
    [javac] Compiling 3 source files to /usr/local/hadoop/hipi/bin
      [jar] Building jar: /usr/local/hadoop/hipi/examples/downloader.jar

BUILD SUCCESSFUL
Total time: 2 seconds
Warning: $HADOOP_HOME is deprecated.

Output HIB: /home/hduser/Lista50/
12/06/26 08:44:46 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
Found host successfully: 9
Tried to get 9 nodes, got 9
12/06/26 08:44:47 INFO input.FileInputFormat: Total input paths to process : 1
First n-1 nodes responsible for 50 images
Last node responsible for 50 images
12/06/26 08:44:47 INFO mapred.JobClient: Running job: job_201206210904_0026
12/06/26 08:44:48 INFO mapred.JobClient:  map 0% reduce 0%
12/06/26 08:45:04 INFO mapred.JobClient:  map 100% reduce 0%
12/06/26 08:45:41 INFO mapred.JobClient:  map 100% reduce 100%
12/06/26 08:45:46 INFO mapred.JobClient: Job complete: job_201206210904_0026
12/06/26 08:45:46 INFO mapred.JobClient: Counters: 29
12/06/26 08:45:46 INFO mapred.JobClient:   Job Counters
12/06/26 08:45:46 INFO mapred.JobClient:     Launched reduce tasks=1
12/06/26 08:45:46 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=40775
12/06/26 08:45:46 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/06/26 08:45:46 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/06/26 08:45:46 INFO mapred.JobClient:     Rack-local map tasks=1
12/06/26 08:45:46 INFO mapred.JobClient:     Launched map tasks=1
12/06/26 08:45:46 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=14857
12/06/26 08:45:46 INFO mapred.JobClient:   File Output Format Counters
12/06/26 08:45:46 INFO mapred.JobClient:     Bytes Written=36
12/06/26 08:45:46 INFO mapred.JobClient:   FileSystemCounters
12/06/26 08:45:46 INFO mapred.JobClient:     FILE_BYTES_READ=40
12/06/26 08:45:46 INFO mapred.JobClient:     HDFS_BYTES_READ=4883547
12/06/26 08:45:46 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=43981
12/06/26 08:45:46 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=9760781
12/06/26 08:45:46 INFO mapred.JobClient:   File Input Format Counters
12/06/26 08:45:46 INFO mapred.JobClient:     Bytes Read=3061
12/06/26 08:45:46 INFO mapred.JobClient:   Map-Reduce Framework
12/06/26 08:45:46 INFO mapred.JobClient:     Map output materialized bytes=40
12/06/26 08:45:46 INFO mapred.JobClient:     Map input records=1
12/06/26 08:45:46 INFO mapred.JobClient:     Reduce shuffle bytes=0
12/06/26 08:45:46 INFO mapred.JobClient:     Spilled Records=2
12/06/26 08:45:46 INFO mapred.JobClient:     Map output bytes=32
12/06/26 08:45:46 INFO mapred.JobClient:     CPU time spent (ms)=5390
12/06/26 08:45:46 INFO mapred.JobClient:     Total committed heap usage (bytes)=
163581952
12/06/26 08:45:46 INFO mapred.JobClient:     Combine input records=0
12/06/26 08:45:46 INFO mapred.JobClient:     SPLIT_RAW_BYTES=115
12/06/26 08:45:46 INFO mapred.JobClient:     Reduce input records=1
12/06/26 08:45:46 INFO mapred.JobClient:     Reduce input groups=1
12/06/26 08:45:46 INFO mapred.JobClient:     Combine output records=0
12/06/26 08:45:46 INFO mapred.JobClient:     Physical memory (bytes) snapshot=25
7286144
12/06/26 08:45:46 INFO mapred.JobClient:     Reduce output records=1
12/06/26 08:45:46 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=958
652416
12/06/26 08:45:46 INFO mapred.JobClient:     Map output records=1

real    1m4.225s
user    0m4.470s
sys     0m0.330s
hduser@nodomaster:/usr/local/hadoop/hipi/examples$
```

Figura 6.50. Ejecución *Downloader* Lista 50

La rutina de actuación que se sigue es la misma que en la prueba con diez imágenes, compilación del código y generación del ejecutable, comprobación del estado activo de los nodos y ejecución del programa. Por tanto, el siguiente paso ha de ser comprobar que ha sido generado el contenedor HIB que es el resultado que esperamos que devuelva el programa.

Para ello, utilizamos la instrucción que nos devuelve el contenido de un directorio sobre la ruta que fijamos para la salida de la aplicación.

```
$> bin/hadoop dfs -ls /home/hduser/Lista50
```

Si el resultado de este comando nos devuelve el contenedor HIB, podemos dar por válida la prueba.

En la *Figura 6.51* podemos apreciar que el contenedor HIB ha sido generado con éxito, de manera que damos por finalizada esta prueba con cincuenta imágenes.

```
hduser@nodomaster:/usr/local/hadoop$ bin/hadoop dfs -ls /home/hduser/Lista50
Warning: $HADOOP_HOME is deprecated.

Found 4 items
-rw-r--r--   3 hduser supergroup        652 2012-06-26 08:41 /home/hduser/Lista5
0/list50.txt
-rw-r--r--   3 hduser supergroup        179 2012-06-26 08:45 /home/hduser/Lista5
0/salida50.hib
-rw-r--r--   3 hduser supergroup    1675422 2012-06-26 08:45 /home/hduser/Lista5
0/salida50.hib.dat
drwxr-xr-x   - hduser supergroup          0 2012-06-26 08:45 /home/hduser/Lista5
0/salida50.hib_output
```

Figura 6.51. Contenido HDFS *Downloader* Lista 50

Ejecución Downloader con cien imágenes

Para dar por terminado el estudio práctico sobre la creación de contenedores HIB útiles para poder utilizar archivos de entrada de imágenes en aplicaciones *Apache Hadoop MapReduce*, vamos a ejecutar de nuevo el programa que venimos usando en los apartados anteriores pero, en este caso utilizando como entrada una lista de enlaces con cien URLs.

El procedimiento va a ser el mismo que hemos venido realizando. Tras haber grabado en el HDFS la lista con los cien enlaces de descarga, podemos ejecutar el *script* que compila el código del programa, crea el ejecutable y poner en marcha el trabajo *MapReduce*. Para ello escribimos la siguiente instrucción en línea de comandos.

```
$>  time  ./runDownloader.sh  /home/hduser/Lista100/list100.txt
/home/hduser/Lista100/salida100.hib 9
```

Ejecutamos esta orden y esperamos a que el entorno realice las operaciones necesarias. En la *Figura 6.52* podemos apreciar este procedimiento.

```
hduser@nodomaster:/usr/local/hadoop/hipi/examples$ time ./runDownloader.sh /home
/hduser/Lista100/list100.txt /home/hduser/Lista100/salida100.hib 9
Buildfile: ../build.xml

downloader:

setup:

test_settings:

compile:
    [javac] Compiling 3 source files to /usr/local/hadoop/hipi/bin
      [jar] Building jar: /usr/local/hadoop/hipi/examples/downloader.jar

BUILD SUCCESSFUL
Total time: 2 seconds
Warning: $HADOOP_HOME is deprecated.

Output HIB: /home/hduser/Lista100/
12/06/26 08:48:28 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
Found host successfully: 9
Tried to get 9 nodes, got 9
12/06/26 08:48:29 INFO input.FileInputFormat: Total input paths to process : 1
First n-1 nodes responsible for 100 images
Last node responsible for 100 images
12/06/26 08:48:30 INFO mapred.JobClient: Running job: job_201206210904_0027
12/06/26 08:48:31 INFO mapred.JobClient:  map 0% reduce 0%
12/06/26 08:48:48 INFO mapred.JobClient:  map 100% reduce 0%
12/06/26 08:49:33 INFO mapred.JobClient:  map 100% reduce 100%
12/06/26 08:49:38 INFO mapred.JobClient: Job complete: job_201206210904_0027
12/06/26 08:49:38 INFO mapred.JobClient: Counters: 29
12/06/26 08:49:38 INFO mapred.JobClient:   Job Counters
12/06/26 08:49:38 INFO mapred.JobClient:     Launched reduce tasks=1
12/06/26 08:49:38 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=48426
12/06/26 08:49:38 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/06/26 08:49:38 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/06/26 08:49:38 INFO mapred.JobClient:     Launched map tasks=1
12/06/26 08:49:38 INFO mapred.JobClient:     Data-local map tasks=1
12/06/26 08:49:38 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=14213
12/06/26 08:49:38 INFO mapred.JobClient:   File Output Format Counters
12/06/26 08:49:38 INFO mapred.JobClient:     Bytes Written=37
12/06/26 08:49:38 INFO mapred.JobClient:   FileSystemCounters
12/06/26 08:49:38 INFO mapred.JobClient:     FILE_BYTES_READ=41
12/06/26 08:49:38 INFO mapred.JobClient:     HDFS_BYTES_READ=9825329
12/06/26 08:49:38 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=43997
12/06/26 08:49:38 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=19638243
12/06/26 08:49:38 INFO mapred.JobClient:   File Input Format Counters
12/06/26 08:49:38 INFO mapred.JobClient:     Bytes Read=6111
12/06/26 08:49:38 INFO mapred.JobClient:   Map-Reduce Framework
12/06/26 08:49:38 INFO mapred.JobClient:     Map output materialized bytes=41
12/06/26 08:49:38 INFO mapred.JobClient:     Map input records=1
12/06/26 08:49:38 INFO mapred.JobClient:     Reduce shuffle bytes=0
12/06/26 08:49:38 INFO mapred.JobClient:     Spilled Records=2
12/06/26 08:49:38 INFO mapred.JobClient:     Map output bytes=33
12/06/26 08:49:38 INFO mapred.JobClient:     CPU time spent (ms)=4390
12/06/26 08:49:38 INFO mapred.JobClient:     Total committed heap usage (bytes)=
163581952
12/06/26 08:49:38 INFO mapred.JobClient:     Combine input records=0
12/06/26 08:49:38 INFO mapred.JobClient:     SPLIT_RAW_BYTES=117
12/06/26 08:49:38 INFO mapred.JobClient:     Reduce input records=1
12/06/26 08:49:38 INFO mapred.JobClient:     Reduce input groups=1
12/06/26 08:49:38 INFO mapred.JobClient:     Combine output records=0
12/06/26 08:49:38 INFO mapred.JobClient:     Physical memory (bytes) snapshot=25
6761856
12/06/26 08:49:38 INFO mapred.JobClient:     Reduce output records=1
12/06/26 08:49:38 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=919
048192
12/06/26 08:49:38 INFO mapred.JobClient:     Map output records=1

real    1m14.466s
user    0m4.620s
sys     0m0.200s
hduser@nodomaster:/usr/local/hadoop/hipi/examples$
```

Figura 6.52. Ejecución *Downloader* Lista 100

Se ha completado el proceso como esperábamos realizando, por este orden, compilación, generación del ejecutable, comunicación con los nodos, trabajo *MapReduce* y creación del HIB. Comprobamos que el contenedor ha sido generado de forma correcta, usando el comando que vemos a continuación:

```
$> bin/hadoop dfs -ls /home/hduser/Lista100
```

Este comando nos muestra el contenido del directorio que hemos fijado como ruta de salida de la ejecución del trabajo *MapReduce*.

Si observamos la *Figura 6.53*, apreciamos que disponemos del contenedor HIB con las cien imágenes descargadas.

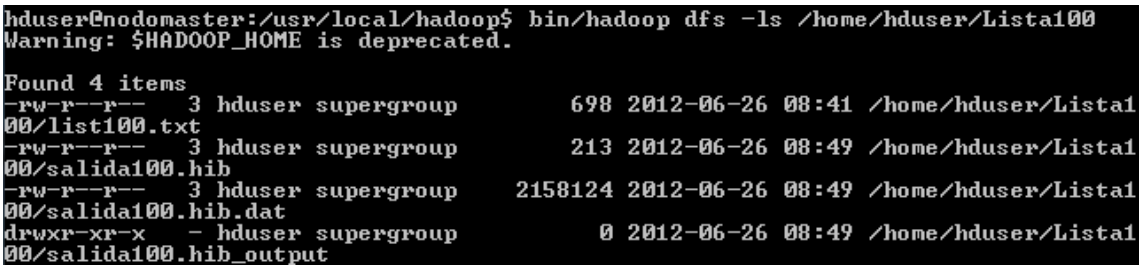


Figura 6.53. Contenido HDFS *Downloader* Lista 100

Tras llevar a cabo los tres trabajos *MapReduce* anteriores, disponemos en nuestro entorno de tres contendores HIB con imágenes que nos van a servir de entrada para la siguiente tarea que vamos a tratar. Dado que ya conocemos el procedimiento para crear HIBs, el siguiente paso ha de ser aprender a manejarlos, abrirlos y poder acceder al contenido de los mismos, que son las imágenes en si con las que queremos trabajar.

6.5.3. Abrir y utilizar un contenedor HIB

Hasta ahora solamente hemos presentado un requisito que la librería HIPI precisa para poder incluir este tipo de archivos en el entorno y que así puedan formar parte de la entrada de una aplicación, el cual es crear un contenedor donde presentar de manera conjunta todas las imágenes. Una vez que disponemos de contenedores HIB con imágenes para poder procesar en un trabajo *MapReduce*, podemos llevar a cabo el objetivo que perseguimos.

Se va a poner en práctica un programa sencillo cuya finalidad será tomar los contenedores creados con anterioridad y devolver un fichero de texto de salida en el que se recojan algunas propiedades de las imágenes, así demostraremos cómo la librería HIPI es capaz de procesar imágenes en entornos *Apache Hadoop MapReduce*. La aplicación que vamos a utilizar se llama ***DumpHib*** y toma un *Hipi Image Bundle* como entrada, lo descomprime y accede a cada una de las imágenes. Finalmente, genera un fichero de texto en el que aparece cada una de las fotografías con su número de apariciones en la lista, el tamaño en pixeles y el identificador que esta imagen tiene dentro de la red social Flickr de donde proviene cada una de ellas.

Al igual que en la prueba anterior, el código de este programa está incluido en el *Anexo I. Pruebas HIPI* y es recomendable consultarlo para comprender en la totalidad lo que a continuación se expone.

Este programa únicamente comprende una sola clase en la que se integra el método principal, el *Mapper* y el *Reducer*. El método principal se encarga de realizar todos los pasos de

configuración del trabajo MapReduce. La clase *Mapper*, llamada *DumpHibMapper*, realiza la tarea distribuida que consiste en, recibiendo como entrada pares <ImageHeader, FloatImage>, inspecciona cada una de las imágenes y devuelve un valor intermedio compuesto por las características de cada una de ellas anteriormente comentadas. Estos pares son enviados al reductor, llamado *DumpHibReducer*, cuya función es ordenar y almacenar estos valores en un fichero de texto que es la salida del programa, presentando cada imagen en una línea.

Método Principal

El procedimiento controlador de este programa es el método `run()` de la clase *DumpHIB* invocado directamente desde el principal o `main()`.

```
public static void main(String[] args) throws Exception {
  int exitCode = ToolRunner.run(new DumpHib(), args);
}
```

Figura 6.54. Principal *DumpHib* (1 de 2)

Este método establece los parámetros de configuración que un trabajo *Hadoop MapReduce* requiere (véase la *Figura 6.55*).

```
...
Job job = new Job(conf, "dumphib");
job.setJarByClass(DumpHib.class);
job.setMapperClass(DumpHibMapper.class);
job.setReducerClass(DumpHibReducer.class);

// Set formats
job.setOutputKeyClass(IntWritable.class);
job.setOutputValueClass(Text.class);
job.setInputFormatClass(ImageBundleInputFormat.class);

// Set out/in paths
removeDir(outputPath, conf);
FileOutputFormat.setOutputPath(job, new Path(outputPath));
FileInputFormat.setInputPaths(job, new Path(inputPath));
...
```

Figura 6.55. Principal *DumpHib* (2 de 2)

Principalmente, se configuran los formatos y las rutas de entrada y salida y las clases *Map* y *Reduce*. El formato de entrada de este trabajo es una *ImageBundleInputFormat* (HIB) y la salida son registros en los que la clave es *IntWritable* (entero) y el valor es una cadena de caracteres (*String*).

Clase Mapper

El procedimiento que siguen las tareas *Map* de esta aplicación es realmente sencillo ya que se reciben registros generados a partir del *Hipi Image Bundle* cuya clave es un *ImageHeader* y el valor es una *FloatImage*.

```
public static class DumpHibMapper extends Mapper<ImageHeader, FloatImage, IntWritable, Text> {

    @Override
    public void map(ImageHeader key, FloatImage value, Context context)
    throws IOException, InterruptedException {
        if (value != null) {
            int imageWidth = value.getWidth();
            int imageHeight = value.getHeight();
            String hexHash = ByteUtils.asHex(ByteUtils.FloatArraytoByteArray(value.getData()));
            String camera = key.getEXIFInformation("Model");
            String output = imageWidth + "x" + imageHeight + "\t(" + hexHash + ")\t " + camera;

            context.write(new IntWritable(1), new Text(output));
        }
    }

}
```

Figura 6.56. *DumpHib Mapper*

Simplemente se obtiene de cada uno de los registros de entrada la anchura y la altura así como el identificador de la misma y la cámara desde la que fue tomada y se concatenan todos los valores en una cadena que es devuelta como par intermedio.

Clase Reducer

Esta clase contiene el método reductor que es aún más sencillo que el *Mapper*, ya que únicamente toma los pares intermedios generados por estos y genera el fichero de salida con todos los valores de estos pares.

```
public static class DumpHibReducer extends Reducer<IntWritable, Text, IntWritable, Text> {
    public void reduce(IntWritable key, Iterable<Text> values, Context context)
    throws IOException, InterruptedException {
        for (Text value : values) {
            context.write(key, value);
        }
    }
}
```

Figura 6.57. *DumpHib Reducer*

6.5.4. Ejecución *DumpHip*

Una vez que conocemos el funcionamiento de la aplicación *MapReduce* que vamos a llevar a cabo para hacer uso de la API de HIPI y aprovecharnos de su funcionalidad de trabajo con imágenes, vamos a proceder a ejecutarla sobre los contenedores creados en los apartados anteriores.

Compilar el código

Para poder llevar a cabo los trabajos *MapReduce* anteriores en los que creamos los contenedores HIB, modificamos en el fichero de configuración de HIPI dos propiedades: la versión de la instalación *Apache Hadoop* y al directorio en la que ésta se localiza. En la *Figura 6.43* podemos recordar estas variables, sería bueno asegurarnos de que se encuentran bien modificadas en nuestro clúster, antes de realizar las pruebas siguientes.

Una vez que las variables están bien configuradas en nuestro clúster, podemos acudir, dentro del mismo fichero "*build.xml*", a la parte en la que se encuentran las instrucciones que compilan y generan el ejecutable del código del programa que vamos a poner en práctica en las pruebas. En la *Figura 6.58* podemos apreciar dichas líneas de código.

```
<!-- Dump some information about all of the information in a HIB -->
<target name="dumphib">
  <antcall target="compile">
    <param name="srcdir" value="examples/hipi/examples/dumphib" />
    <param name="files" value="DumpHib.java" />
    <param name="jarfilename" value="dumphib.jar" />
    <param name="jardir" value="examples" />
    <param name="mainclass" value="hipi.examples.dumphib.DumpHib" />
  </antcall>
</target>
```

Figura 6.58. *build.xml DumpHib*

Ya se explicó anteriormente que desde este *target* se invoca a otro denominado "*compile*" que es el encargado de la compilación del código en sí. En la *Figura 6.58* podemos observar las líneas de código que conforman dicho *target*.

Usando el siguiente comando es posible compilar el código:

```
$> ant dumphib
```

Sin embargo, al igual que en el caso del *Downloader*, disponemos de un script que lleva a cabo tanto la compilación como la generación del ejecutable y la ejecución del programa. Se llama "*runDumpHIB.sh*" y contiene el siguiente código:

```
#!/bin/bash

ant -f ../build.xml dumphib

hadoop jar dumphib.jar $1 $2
```

Al poenr en marcha este *script* es necesario incluir dos parámetros que se corresponden con el contenedor HIB de entrada y con el directorio de salida en el que queremos que se guarde el

archivo de texto que genera el programa. Teniendo esto en cuenta, podemos ejecutar el *script* con el siguiente comando:

```
$>          ./runDumpHIB.sh          /home/hduser/contenedor.hib
/home/hduser/salida
```

Para llevar a cabo cualquier trabajo *Hadoop MapReduce*, es preciso disponer de los archivos de entrada del mismo distribuidos en el sistema de ficheros HDFS. Para este caso, sabemos que estos archivos, los contenedores de imágenes HIB, se encuentran dentro del HDFS ya que se crearon como salida de otro trabajo *MapReduce*. Por tanto, ya podemos comenzar a realizar las pruebas.

Contenedor de diez imágenes

Comenzamos a ejecutar trabajos *MapReduce* con la aplicación *DumpHIB* que toma un contenedor HIB de entrada, accede a las imágenes de su interior y genera un fichero de texto de salida con ciertas propiedades de cada una de fotografías.

En esta primera prueba, vamos a llevar a cabo la ejecución del código sobre el contenedor HIB de diez imágenes. Para ello emplearemos el siguiente comando:

```
$>        ./runDumpHIB.sh        /home/hduser/Salida10/salida.hib
/home/hduser/resultadoDump10/
```

Podemos apreciar que, como primer parámetro de entrada incluimos el contenedor generado como salida del trabajo *Downloader* con diez imágenes y como segundo parámetro un directorio de nuestro HDFS que se utiliza como salida de esta tarea *MapReduce* y en el que guarda el fichero de texto generado por la misma. Comprobamos el resultado en la siguiente *Figura 6.59*.

```
hduser@nodomaster:/usr/local/hadoop/hipi/examples$ time ./runDumpHIB.sh /home/hd
user/Salida10/salida.hib /home/hduser/resultadoDump10/
Buildfile: ../build.xml

dumphib:

setup:

test_settings:

compile:
    [javac] Compiling 1 source file to /usr/local/hadoop/hipi/bin
      [jar] Building jar: /usr/local/hadoop/hipi/examples/dumphib.jar

BUILD SUCCESSFUL
Total time: 2 seconds
Warning: $HADOOP_HOME is deprecated.

12/07/06 08:56:10 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
12/07/06 08:56:11 INFO input.FileInputFormat: Total input paths to process : 1
1 tasks spawned
12/07/06 08:56:11 INFO mapred.JobClient: Running job: job_201207031001_0014
12/07/06 08:56:12 INFO mapred.JobClient:  map 0% reduce 0%
12/07/06 08:56:26 INFO mapred.JobClient:  map 100% reduce 0%
12/07/06 08:56:41 INFO mapred.JobClient:  map 100% reduce 100%
12/07/06 08:56:46 INFO mapred.JobClient: Job complete: job_201207031001_0014
12/07/06 08:56:46 INFO mapred.JobClient: Counters: 29
12/07/06 08:56:46 INFO mapred.JobClient:   Job Counters
12/07/06 08:56:46 INFO mapred.JobClient:     Launched reduce tasks=1
12/07/06 08:56:46 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=16546
12/07/06 08:56:46 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/07/06 08:56:46 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/07/06 08:56:46 INFO mapred.JobClient:     Rack-local map tasks=1
12/07/06 08:56:46 INFO mapred.JobClient:     Launched map tasks=1
12/07/06 08:56:46 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=14653
12/07/06 08:56:46 INFO mapred.JobClient:   File Output Format Counters
12/07/06 08:56:46 INFO mapred.JobClient:     Bytes Written=538
12/07/06 08:56:46 INFO mapred.JobClient:   FileSystemCounters
12/07/06 08:56:46 INFO mapred.JobClient:     FILE_BYTES_READ=584
12/07/06 08:56:46 INFO mapred.JobClient:     HDFS_BYTES_READ=1479174
12/07/06 08:56:46 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=44051
12/07/06 08:56:46 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=538
12/07/06 08:56:46 INFO mapred.JobClient:   File Input Format Counters
12/07/06 08:56:46 INFO mapred.JobClient:     Bytes Read=1479054
12/07/06 08:56:46 INFO mapred.JobClient:   Map-Reduce Framework
12/07/06 08:56:46 INFO mapred.JobClient:     Map output materialized bytes=584
12/07/06 08:56:46 INFO mapred.JobClient:     Map input records=10
12/07/06 08:56:46 INFO mapred.JobClient:     Reduce shuffle bytes=0
12/07/06 08:56:46 INFO mapred.JobClient:     Spilled Records=20
12/07/06 08:56:46 INFO mapred.JobClient:     Map output bytes=558
12/07/06 08:56:46 INFO mapred.JobClient:     CPU time spent (ms)=3440
12/07/06 08:56:46 INFO mapred.JobClient:     Total committed heap usage (bytes)=
163581952
12/07/06 08:56:46 INFO mapred.JobClient:     Combine input records=0
12/07/06 08:56:46 INFO mapred.JobClient:     SPLIT_RAW_BYTES=120
12/07/06 08:56:46 INFO mapred.JobClient:     Reduce input records=10
12/07/06 08:56:46 INFO mapred.JobClient:     Reduce input groups=1
12/07/06 08:56:46 INFO mapred.JobClient:     Combine output records=0
12/07/06 08:56:46 INFO mapred.JobClient:     Physical memory (bytes) snapshot=26
3274496
12/07/06 08:56:46 INFO mapred.JobClient:     Reduce output records=10
12/07/06 08:56:46 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=102
9709824
12/07/06 08:56:46 INFO mapred.JobClient:     Map output records=10

real    0m36.066s
user    0m4.270s
sys     0m0.250s
hduser@nodomaster:/usr/local/hadoop/hipi/examples$
```

Figura 6.59. Ejecución *DumpHib* Lista 10

Este trabajo *MapReduce* sigue un procedimiento similar al de cualquier aplicación de este entorno. En primer lugar, compilamos el código y generamos el ejecutable. A continuación se inician las tareas *Map* que reciben imágenes de los contenedores y acceden a sus propiedades para generar pares intermedios que envían al *Reducer*. Finalmente, éste último crea un fichero de salida en el que inserta en línea a línea los pares intermedios recibidos de los *Mappers*.

Para comprobar que el trabajo ha resultado exitoso, vamos a mostrar el fichero de texto generado. Puede verse en la *Figura 6.60*.

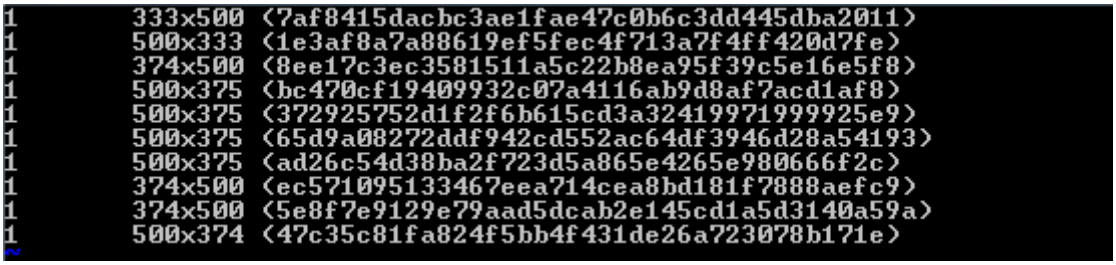


Figura 6.60. Resultado *DumpHib* Lista 10

En la *Figura 6.60* se muestra un fichero de salida en el que vemos el indicador de cada imagen a la izquierda, en la columna central las dimensiones de la fotografía en cuestión y su identificador en la red social *Flickr* a continuación.

Contenedor de cincuenta imágenes

Continuamos las pruebas, eligiendo ahora como fichero de entrada el contenedor *Hipi Image Bundle* que contiene cincuenta imágenes. El procedimiento es el mismo que para el caso de diez imágenes, por tanto el comando que ejecuta el *script* que compila y pone en marcha el trabajo *MapReduce*, es el siguiente:

```
$>          ./runDumpHIB.sh          /home/hduser/Salida50/salida.hib
/home/hduser/resultadoDump50/
```

En la *Figura 6.61* puede ver los pasos que sigue la ejecución de este trabajo.

```
hduser@nodomaster:/usr/local/hadoop/hipi/examples$ time ./runDumpHIB.sh /home/hd
user/Salida50/salida.hib /home/hduser/resultadoDump50/
Buildfile: ../build.xml

dumphib:

setup:

test_settings:

compile:
    [javac] Compiling 1 source file to /usr/local/hadoop/hipi/bin
      [jar] Building jar: /usr/local/hadoop/hipi/examples/dumphib.jar

BUILD SUCCESSFUL
Total time: 2 seconds
Warning: $HADOOP_HOME is deprecated.

12/07/06 09:00:28 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
12/07/06 09:00:28 INFO input.FileInputFormat: Total input paths to process : 1
1 tasks spawned
12/07/06 09:00:29 INFO mapred.JobClient: Running job: job_201207031001_0015
12/07/06 09:00:30 INFO mapred.JobClient:  map 0% reduce 0%
12/07/06 09:00:43 INFO mapred.JobClient:  map 100% reduce 0%
12/07/06 09:00:55 INFO mapred.JobClient:  map 100% reduce 100%
12/07/06 09:01:00 INFO mapred.JobClient: Job complete: job_201207031001_0015
12/07/06 09:01:00 INFO mapred.JobClient: Counters: 29
12/07/06 09:01:00 INFO mapred.JobClient:   Job Counters
12/07/06 09:01:00 INFO mapred.JobClient:     Launched reduce tasks=1
12/07/06 09:01:00 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=15123
12/07/06 09:01:00 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/07/06 09:01:00 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/07/06 09:01:00 INFO mapred.JobClient:     Rack-local map tasks=1
12/07/06 09:01:00 INFO mapred.JobClient:     Launched map tasks=1
12/07/06 09:01:00 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=11123
12/07/06 09:01:00 INFO mapred.JobClient:   File Output Format Counters
12/07/06 09:01:00 INFO mapred.JobClient:     Bytes Written=2700
12/07/06 09:01:00 INFO mapred.JobClient:   FileSystemCounters
12/07/06 09:01:00 INFO mapred.JobClient:     FILE_BYTES_READ=2906
12/07/06 09:01:00 INFO mapred.JobClient:     HDFS_BYTES_READ=4880044
12/07/06 09:01:00 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=48695
12/07/06 09:01:00 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=2700
12/07/06 09:01:00 INFO mapred.JobClient:   File Input Format Counters
12/07/06 09:01:00 INFO mapred.JobClient:     Bytes Read=4879924
12/07/06 09:01:00 INFO mapred.JobClient:   Map-Reduce Framework
12/07/06 09:01:00 INFO mapred.JobClient:     Map output materialized bytes=2906
12/07/06 09:01:00 INFO mapred.JobClient:     Map input records=50
12/07/06 09:01:00 INFO mapred.JobClient:     Reduce shuffle bytes=2906
12/07/06 09:01:00 INFO mapred.JobClient:     Spilled Records=100
12/07/06 09:01:00 INFO mapred.JobClient:     Map output bytes=2800
12/07/06 09:01:00 INFO mapred.JobClient:     CPU time spent (ms)=4370
12/07/06 09:01:00 INFO mapred.JobClient:     Total committed heap usage (bytes)=
234684416
12/07/06 09:01:00 INFO mapred.JobClient:     Combine input records=0
12/07/06 09:01:00 INFO mapred.JobClient:     SPLIT_RAW_BYTES=120
12/07/06 09:01:00 INFO mapred.JobClient:     Reduce input records=50
12/07/06 09:01:00 INFO mapred.JobClient:     Reduce input groups=1
12/07/06 09:01:00 INFO mapred.JobClient:     Combine output records=0
12/07/06 09:01:00 INFO mapred.JobClient:     Physical memory (bytes) snapshot=31
4986496
12/07/06 09:01:00 INFO mapred.JobClient:     Reduce output records=50
12/07/06 09:01:00 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=102
4679936
12/07/06 09:01:00 INFO mapred.JobClient:     Map output records=50

real    0m39.000s
user    0m4.330s
sys     0m0.190s
hduser@nodomaster:/usr/local/hadoop/hipi/examples$
```

Figura 6.61. Ejecución *DumpHib* Lista 50

Al igual que en el caso anterior, este trabajo sigue la misma secuencia de tareas de la mayoría de las aplicaciones *MapReduce*. Comienza con la compilación del código y seguidamente se inician las tareas *Map* que toman los valores de cada una de las imágenes que son enviados al *Reducer* en forma de pares intermedios. Éste es el encargado de generar el fichero de salida con las propiedades de las imágenes y que podemos ver a continuación.

Figura 6.62. Resultado *DumpHib* Lista 50

Podemos comprobar cómo han sido examinadas cada una de las cincuenta imágenes, almacenándose de todas ellas las propiedades ya comentadas.

Contenedor de cien imágenes

Pasamos finalmente a ejecutar el trabajo *MapReduce DumpHIB* que descomprime contenedores Hipi Image Bundle y envía las imágenes de su contenido a los *Mappers* del clúster que actúan sobre ellas obteniendo información de las mismas por última vez, tomando el contenedor de cien imágenes.

El comando que inicia este programa a partir del *script* es prácticamente el mismo que en los casos anteriores y lo vemos a continuación.

```
$>          ./runDumpHIB.sh          /home/hduser/Salida50/salida.hib
/home/hduser/resultadoDump50/
```

El resultado de este trabajo podemos verlo en la *Figura 6.63*.

```
hduser@nodomaster:/usr/local/hadoop/hipi/examples$ time ./runDumpHIB.sh /home/hd
user/Salida100/salida.hib /home/hduser/resultadoDump100/
Buildfile: ../build.xml

dumphib:

setup:

test_settings:

compile:
    [javac] Compiling 1 source file to /usr/local/hadoop/hipi/bin
      [jar] Building jar: /usr/local/hadoop/hipi/examples/dumphib.jar

BUILD SUCCESSFUL
Total time: 2 seconds
Warning: $HADOOP_HOME is deprecated.

12/07/06 09:03:52 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
12/07/06 09:03:52 INFO input.FileInputFormat: Total input paths to process : 1
1 tasks spawned
12/07/06 09:03:52 INFO mapred.JobClient: Running job: job_201207031001_0016
12/07/06 09:03:53 INFO mapred.JobClient:  map 0% reduce 0%
12/07/06 09:04:10 INFO mapred.JobClient:  map 100% reduce 0%
12/07/06 09:04:22 INFO mapred.JobClient:  map 100% reduce 100%
12/07/06 09:04:27 INFO mapred.JobClient: Job complete: job_201207031001_0016
12/07/06 09:04:27 INFO mapred.JobClient: Counters: 29
12/07/06 09:04:27 INFO mapred.JobClient:   Job Counters
12/07/06 09:04:27 INFO mapred.JobClient:     Launched reduce tasks=1
12/07/06 09:04:27 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=18096
12/07/06 09:04:27 INFO mapred.JobClient:     Total time spent by all reduces wai
ting after reserving slots (ms)=0
12/07/06 09:04:27 INFO mapred.JobClient:     Total time spent by all maps waitin
g after reserving slots (ms)=0
12/07/06 09:04:27 INFO mapred.JobClient:     Rack-local map tasks=1
12/07/06 09:04:27 INFO mapred.JobClient:     Launched map tasks=1
12/07/06 09:04:27 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=11132
12/07/06 09:04:27 INFO mapred.JobClient:   File Output Format Counters
12/07/06 09:04:27 INFO mapred.JobClient:     Bytes Written=5283
12/07/06 09:04:27 INFO mapred.JobClient:   FileSystemCounters
12/07/06 09:04:27 INFO mapred.JobClient:     FILE_BYTES_READ=5681
12/07/06 09:04:27 INFO mapred.JobClient:     HDFS_BYTES_READ=9723510
12/07/06 09:04:27 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=54249
12/07/06 09:04:27 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=5283
12/07/06 09:04:27 INFO mapred.JobClient:   File Input Format Counters
12/07/06 09:04:27 INFO mapred.JobClient:     Bytes Read=9723389
12/07/06 09:04:27 INFO mapred.JobClient:   Map-Reduce Framework
12/07/06 09:04:27 INFO mapred.JobClient:     Map output materialized bytes=5681
12/07/06 09:04:27 INFO mapred.JobClient:     Map input records=98
12/07/06 09:04:27 INFO mapred.JobClient:     Reduce shuffle bytes=5681
12/07/06 09:04:27 INFO mapred.JobClient:     Spilled Records=196
12/07/06 09:04:27 INFO mapred.JobClient:     Map output bytes=5479
12/07/06 09:04:27 INFO mapred.JobClient:     CPU time spent (ms)=5750
12/07/06 09:04:27 INFO mapred.JobClient:     Total committed heap usage (bytes)=
234684416
12/07/06 09:04:27 INFO mapred.JobClient:     Combine input records=0
12/07/06 09:04:27 INFO mapred.JobClient:     SPLIT_RAW_BYTES=121
12/07/06 09:04:27 INFO mapred.JobClient:     Reduce input records=98
12/07/06 09:04:27 INFO mapred.JobClient:     Reduce input groups=1
12/07/06 09:04:27 INFO mapred.JobClient:     Combine output records=0
12/07/06 09:04:27 INFO mapred.JobClient:     Physical memory (bytes) snapshot=33
5814656
12/07/06 09:04:27 INFO mapred.JobClient:     Reduce output records=98
12/07/06 09:04:27 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=982
159360
12/07/06 09:04:27 INFO mapred.JobClient:     Map output records=98

real    0m40.161s
user    0m4.530s
sys     0m0.150s
hduser@nodomaster:/usr/local/hadoop/hipi/examples$
```

Figura 6.63. Ejecución *DumpHib* Lista 100

De forma análoga a las dos pruebas anteriores, este trabajo accede a la totalidad de las imágenes del contenedor HIB generando el fichero de texto con algunas propiedades de las mismas que podemos ver en la *Figura 6.64*.

Figura 6.64. Resultado *DumpHib* Lista 100

Con este trabajo, damos por finalizadas las pruebas de la API de la librería *Hadoop Image Processing Interface* con las que hemos dado a conocer una sencilla forma de poder incluir como archivos de entrada de nuestras aplicaciones *MapReduce* archivos de imagen.

6.5.5. Resultados de las pruebas con la Librería HIPI

Las ejecuciones de programas *MapReduce* que hacen uso de la API de HIPI que se han llevado a cabo en los apartados anteriores han sido principalmente con el objetivo de poner en práctica un modo de poder llevar a cabo trabajos en este entorno que puedan incluir como ficheros de entrada archivos de imagen, algo que con la API de *MapReduce* no es posible. Además, hemos aprovechado estas pruebas para tomar mediciones de tiempo que puedan ayudarnos a alcanzar alguna conclusión sobre el uso de esta librería.

Dada la sencillez de las pruebas realizadas y que solamente se han llevado a cabo tres pruebas por programa, las mediciones tomadas no podrán ser del todo lo concluyentes que fueron las de las pruebas de rendimiento realizadas en el *Apartado 6.2. Prueba de Rendimiento MapReduce* pero si nos darán una pequeña idea general del comportamiento de las aplicaciones basadas en esta librería.

Han sido tomados, para los trabajos *MapReduce* de los programas *Downloader* y *DumpHIB* el tiempo de compilación y ejecución para conjuntos de imágenes de diez, cincuenta y cien unidades. En la tabla que aparece a continuación podemos ver las medidas tomadas.

Imágenes	Downloader	DumpHIB
10	40”	36”
50	1’04”	39”
100	1’14”	40”

Tabla 6.8. Mediciones HIPI

Si evaluamos los resultados anteriores, podemos deducir que en ambas aplicaciones el número de imágenes gestionadas afecta realmente poco al tiempo total de ejecución. Es decir, la mayoría del gasto en tiempo ha sido, para el caso del *DumpHIB* en tareas de gestión del entorno y para el caso del *Downloader* igual, añadiendo el tiempo de duración de las descargas. Esta conclusión la obtenemos debido a que la diferencia en tiempo de ejecución entre los diferentes trabajos es casi nula. En el caso del *Downloader*, es algo mayor que en el *DumpHIB* debido a que se realizan descargas en las que también influye la velocidad de descarga.

Por tanto, podemos concluir que **utilizar la librería HIPI para el trabajo con imágenes en el entorno *Apache Hadoop MapReduce* mantiene las ventajas del mismo, siendo aún más destacable sus ventajas cuanto mayor sean los conjuntos de datos de entrada**. Cuantas más imágenes incluyamos en los conjuntos de entrada, menos influencia en el tiempo de ejecución final tendrán las tareas de gestión que el entorno *Hadoop MapReduce* ha de llevar a cabo para cada trabajo.

6.6. Conclusiones de las Aplicaciones del Entorno *Hadoop MapReduce*

En este capítulo hemos llevado a cabo pruebas de nuestro clúster *Apache Hadoop MapReduce* con dos objetivos; el primero de ellos obtener valores del rendimiento del mismo y el segundo conseguir llevar a cabo aplicaciones que utilicen imágenes como entrada, algo que la API del lenguaje *MapReduce* no contempla.

Los primeros valores tomados fueron sobre el tiempo que precisa el entorno para incluir en el sistema de ficheros *Hadoop Distributed File System* conjuntos de datos que iban desde los diez al millar de libros. La segunda medición fue tomada sobre el tiempo de cómputo de las pruebas de rendimiento del programa *WordCount* que utiliza la librería *Java MapReduce*. Finalmente, las últimas medidas son las que se tomaron sobre las pruebas con la API de *Hadoop Image Processing Interface.*

La conclusión general, obtenida tanto en las mediciones sobre el sistema de ficheros distribuido HDFS, como en los valores tomados en las ejecuciones del programa *WordCount* o en los programas de la API de HIPI es que **este entorno alcanza sus mejores prestaciones cuanto mayores sean los conjuntos de datos de entrada**, siempre y cuando no lleguemos al límite en cuanto a recursos del clúster. Esto es así porque este entorno conlleva un gasto en tiempo para cada ejecución derivado de tareas propias. Así, con conjuntos de entrada de mayor volumen, menor es la influencia de estas labores en el tiempo de cómputo global.

Apache Hadoop MapReduce es un entorno de computación distribuido orientado a programas que computen sobre grandes volúmenes de datos y que, de forma conjunta con la librería *Hadoop Image Processing Interface*, permite que estas aplicaciones puedan estar orientadas a sistemas que usen conjuntos de entrada de casi cualquier formato.

En el siguiente capítulo vamos a realizar una síntesis global de proyecto, incluyendo conclusiones tomadas de todos los capítulos que en esta memoria se desarrollan. Desde las tecnologías *Cloud Computing,* a las aplicaciones *Hadoop MapReduce* tratadas en este capítulo, pasando por aspectos fundamentales que se han estudiado en este proyecto como son los sistemas de ficheros distribuidos, los clústeres de máquinas virtuales, etc., así como la interacción conjunta de todas estas tecnologías estudiadas en el clúster generado en el proyecto basado en una nube *OpenStack* en el que máquinas virtuales *Apache Hadoop* trabajan de manera conjunta y distribuida con aplicaciones *MapReduce* y añadiendo la librería *Hadoop Image Processing Interface* que se apoyan en el sistema de ficheros distribuido *Hadoop Distributed File System* para alcanzar un óptimo rendimiento sobre grandes volúmenes de datos.

7. Análisis de Resultados, Conclusiones y Trabajo Futuro

En este proyecto, se ha buscado la **integración** de dos entornos de computación; se ha integrado el *framework* de computación distribuida ***Apache Hadoop MapReduce*** en un entorno ***Cloud Computing*** *OpenStack*. Con esto, se han tratado de obtener conjuntamente los beneficios que ambos entornos aportan en un marco de trabajo que combina estas dos tecnologías.

Comenzamos dando a conocer el entorno *Cloud Computing*, estudiando a fondo sus características, los servicios que ofrece, ventajas y desventajas y las distintas tecnologías que ofrece el mercado. Este estudio nos ha llevado a tomar la decisión de elegir como base de nuestro proyecto la **distribución *OpenStack***, ya que, siendo de **licencia GPL**, cuenta con un **amplio soporte** debido a que son muchas las empresas que están integradas en su desarrollo. Además es **altamente flexible** aportando **diversas topologías de desarrollo** que la hacen **adaptable a casi cualquier entorno** con un **elevado nivel de escalabilidad**.

Por ello, al decidirnos por este entorno *Cloud Computing*, el siguiente paso fue llevar a cabo la instalación y configuración del mismo. Para llevar esta acción a la práctica, descubrimos la **distribución baremetal *StackOps*** que permite **configurar** cualquier tipo de nube ***OpenStack* de forma sencilla**. Para decidir que topología de nube utilizar, realizamos el estudio de los servidores disponibles y de las arquitecturas que nos ofrece el entorno *Cloud* elegido. En base a lo asimilado, nos hemos decidido por la **arquitectura *Dual Node*** en la que **un Nodo Controlador gestiona** al resto de **Nodos de Cómputo** llevando también el **control de la red** y el **almacenamiento** en la nube, así como siendo el **portal de acceso** de los **usuarios** de la nube.

Tras la configuración del entorno *Cloud Computing* ya disponemos de la primera mitad del proyecto en marcha. El siguiente paso ha sido la realización de un estudio teórico sobre el ***framework*** de computación distribuida ***Apache Hadoop***, del que hemos dado a conocer que es de **licencia libre** y recibe **soporte de multitud de grandes empresas** entre las que destaca como mayor colaborador Yahoo!, sin olvidarnos de otras como Facebook, eBay, Google, Twitter, etc. **Creado** por **Doug Cutting**, está **basado en *Java*** y destaca su capacidad de gestionar **clústeres de** hasta **miles de nodos** y poder **computar sobre petabytes de datos**. Por ello podemos decir que es **altamente flexible y escalable**, ya que es capaz de **adaptarse a cualquier tipo de entorno** (**incluso** entornos **virtuales**) y puede **formar clústeres que van desde un solo nodo a miles** de ellos. Este *framework* **se crea para** ser instalado sobre **hardware de bajo coste**, por lo que **no precisa grandes servidores** ni equipos especialmente potentes.

Uno de los aspectos más destacables de este entorno es su **sistema de ficheros** distribuido ***Hadoop Distributed File System***. Por ello, hemos realizado un estudio de los sistemas de

ficheros distribuidos, dando a conocer su funcionamiento, evolución histórica y presentando los más utilizados y destacados del mercado. Esto nos ha permitido conocer la potencia real del sistema de ficheros que la ***Apache Software Fundation*** proporciona para trabajar conjuntamente con el entorno de computación distribuida *Hadoop*. Dado que está **diseñado para funcionar bajo** el entorno de computación ***Apache Hadoop***, este sistema de ficheros es **flexible y escalable**, pudiendo **adaptarse a las distintas topologías de clúster** del *framwork*. Diseñado para **trabajar sobre miles de nodos**, soporta aplicaciones que utilizan **volúmenes de datos realmente elevados**, alcanza su mejor rendimiento con sistemas que realicen mínimas escrituras y múltiples accesos a los datos. Está **compuesto por** el controlador o ***NameNode*** que maneja el sistema de ficheros y por los nodos de almacenamiento o ***DataNode***.

Una vez estudiado de forma teórica el marco de trabajo *Apache Hadoop* y su sistema de ficheros distribuido HDFS, procedemos a **decidir** cuál es **la topología de clúster** que se va a poner en práctica. Este *framework* **propone** dos arquitecturas, ***Single Node*** que **aloja todos los servicios en un nodo** y ***Multinode*** que lo hace **repartido en múltiples** hosts. Estos **servicios** están **divididos en dos capas**; **la primera** es la capa ***MapReduce* con** los servicios ***JobTracker*, *TaskTracker*** y **la segunda**, la capa **HDFS con** los servicios ***DataNode* y *NameNode***. **El rendimiento** de la segunda de las topologías (***Multinode***) es obviamente **superior a** la que aloja todos los servicios en un sola equipo (***Single Node***), sin embargo, la configuración *Apache Hadoop* recomienda **configurar inicialmente la topología más sencilla** y, **a partir de ésta**, **configurar** el clúster con **múltiples nodos**.

Es en este punto donde **la relación *Cloud Computing – Apache Hadoop*** alcanza uno de los puntos **más destacados**, ya que a la hora de **formar el clúster de computación distribuida**, creamos una **instancia virtual con la totalidad de los servicios *Hadoop*** configurados y a partir de ella **generamos un patrón** que nos ha permitido ir añadiendo esclavos al clúster. Así **hemos aprovechado algunas de las ventajas que aportan estos dos entornos**; por un lado, nos hemos servido de la **facilidad con la que los entornos *Clouds* permiten crear instancias virtuales que, tras ser configuradas**, pueden ser **almacenadas como** imagen que sirve de **patrón**; y por otro lado, hemos **promocionado la flexibilidad y la escalabilidad** del entorno ***Apache Hadoop***, así como su **adaptabilidad a cualquier sistema** (**incluso virtuales**) lo que permite **crear un clúster con una sola instancia virtual y a partir de ahí ir añadiendo** tantas como se deseen.

Llegados a este punto, se ha configurado una nube *OpenStack Dual Node* en la que hemos desplegado un clúster *Apache Hadoop Multinode*. Sin embargo, **para obtener rendimiento de este sistema de computación distribuida** que hemos conformado, hemos de utilizar **aplicaciones *MapReduce***, que es un **lenguaje basado en Java** creado para **aprovechar** al máximo **las ventajas del entorno *Apache Hadoop* y de** su sistema de ficheros distribuido **HDFS**. Por ello, se ha llevado a cabo el estudio sobre **este lenguaje**, dando a conocer sus principales características, como son **licencia libre**, **programación funcional** (**función *Map* y *Reduce***) y orientado al **cómputo sobre grandes cantidades de datos en entornos distribuidos**. **Las aplicaciones** escritas en el lenguaje de programación ***MapReduce* toman como entrada un conjunto de datos que, de manera distribuida, es procesado en los *Mappers* que generan pares intermedios clave-valor que son ordenados, combinados y reducidos en un archivo de salida final por el *Reducer***. En este entorno de programación son funda**mentales los datos de entrada**, por ello es muy importante **conocer en profundidad los formatos de archivo** que la

API de *MapReduce* incluye. Al dar a conocer estos formatos, **aparece la necesidad de un tipo de archivo que permita** el uso de aplicaciones que computen directamente sobre **imágenes**. Es aquí donde **aparece la librería *Hadoop Image Processing Interface*** con la que la ***University of Virginia*** aporta una ***API* para la realización de tareas de procesamiento de imágenes** en este entorno. Su funcionamiento se basa en la **creación de un contenedor *Hipi Image Bundle*** (HIB) en el que **se almacenan de manera conjunta las imágenes** y que sí permite servir de **entrada para un trabajo *MapReduce***. Una vez que **el entorno de programación recibe el contenedor HIB**, basta con **descomprimirlo** y **ya pueden utilizarse las imágenes** de forma similar a cualquier aplicación *Java*.

Conociendo el lenguaje de programación que vamos a utilizar y teniendo correctamente configurado el clúster *Apache Hadoop*, **procedemos a llevar a cabo trabajos *MapReduce*** para:

- **Probar el entorno**.
- **Obtener el máximo rendimiento** del clúster.
- **Conocer el funcionamiento de la librería** de trabajo con imágenes **HIPI**.

Habiendo puesto en práctica programas *MapReduce* sobre nuestro clúster *Apache Hadoop*, damos por finalizado el estudio de este proyecto, con lo que procedemos a la extracción de conclusiones generales.

7.1. Conclusiones

En el apartado anterior, hemos llevado a cabo un recordatorio general de todo lo estudiado y puesto en práctica a lo largo de este proyecto. Tras las configuraciones y pruebas realizadas y habiendo hecho un análisis de los resultados obtenidos, hemos podido alcanzar conclusiones que afianzan en el trabajo realizado.

7.1.1. Conclusiones particulares

Para que sea más sencillo obtener conclusiones a modo general del proyecto, a continuación vamos a exponer las ideas principales que hemos ido adquiriendo en el estudio de cada una de las tecnologías, entornos o librerías de programación estudiadas. Teniendo claro que ha aportado cada una de ellas al proyecto global, nos será más sencillo sintetizar conclusiones generales.

Hemos dividido el estudio global del proyecto en cinco apartados:

- Paradigma de computación *Cloud Computing.*
- Entorno de computación distribuida *Apache Hadoop*.
- Sistema de ficheros *Hadoop Distributed File System*.
- Integración *Cloud Computing – Apache Hadoop*.
- Lenguaje *MapReduce* y librería *Hadoop Image Processing Interface*.

En los siguientes apartados, van a exponerse las ideas, prerrogativas y desventajas que hemos obtenido de cada uno de ellos y que nos afectan al proyecto.

Cloud Computing

Configurar una nube *OpenStack*, como base de nuestro proyecto nos ha proporcionado las siguientes ventajas:

- Un entorno *Cloud Computing* conlleva **ahorro generalizado en costes** de administración, mantenimiento, monitorización, recuperación ante desastres, políticas de seguridad, etc. En concreto, ***OpenStack*** al ser de **licencia libre**, aporta todas estas características a **coste cero**.
- *OpenStack* es un sistema *Cloud Computing*, con un **amplio soporte y** en constante **desarrollo** con grandes empresas implicadas.
- **Abstracción del hardware,** permitiéndonos integrar en un mismo sistema servidores de distintas características.
- **Simplicidad** en la **instalación** y **mantenimiento** por parte del usuario, sobre todo gracias a la distribución baremetal ***StackOps***.
- **Escalabilidad** del sistema. Hemos podido crear instancias virtuales, grabarlas como imágenes y desplegar tantas como nuestros recursos nos permiten.
- **Agilidad** en la producción y puesta en marcha de servicios.
- **Alta disponibilidad** de todos los recursos de nuestro sistema **y alto rendimiento** gracias al aprovechamiento al máximo de los mismos.

Como se puede apreciar, son bastantes las prerrogativas que nos ha aportado este entorno *Cloud Computing* que si bien posee **algunas desventajas** como pueden ser **gastos de recursos para gestión** del sistema o **dependencia de la red**, son mínimos los efectos que poseen sobre el rendimiento final del entorno. Al desplegar una **nube privada**, **eliminamos** la **dependencia** de cualquier **proveedor de servicios**, con lo que también **desaparece** cualquier **sentimiento de inseguridad o vulnerabilidad** debido a que disponemos de todos los datos dentro de nuestros servidores y nuestra red privada.

Por tanto, son más las ventajas que hemos obtenido de este entorno que los inconvenientes que nos podría haber ocasionado.

Apache Hadoop

El objetivo principal de este proyecto es encontrar un entorno de computación distribuida capaz de procesar grandes volúmenes de datos alcanzando el máximo rendimiento. Es precisamente eso lo que el *framework Apache Hadoop* aporta, siendo su adaptabilidad a nuestro entorno el principal hándicap que nos podríamos haber encontrado. Tras conocer que este entorno es adaptable a casi cualquier sistema en el que se pretenda configurar, podemos concluir que utilizar este marco de trabajo para procesar grandes volúmenes de datos nos ha aportado las siguientes prerrogativas:

- **Adaptabilidad** a cualquier entorno, **incluso virtuales** y **compatible** con **todo tipo** de **equipos**, lo que hace que un mismo clúster puedan conformarlo computadoras totalmente distintas.
- **Escalabilidad**, pudiendo formar clústeres de hasta miles de nodos.
- **Amplio soporte y desarrollo** de múltiples empresas.
- Realmente **sencillo de configurar y agilidad en el despliegue de nuevos nodos**.
- Capacidad de **procesar grandes volúmenes de datos**.
- Coste cero, **licencia libre**.
- Gracias a que es adaptable a cualquier servidor y que pueden conformarse clústeres de tantos nodos como queramos aunque no sean hosts iguales, se da un muy buen **aprovechamiento de recursos**.

Obviamente, no todo son ventajas en este entorno, pero los perjuicios que afectan a nuestro clúster son asumibles. El principal inconveniente es que, como todo *framework*, acarrea cierta pérdida de rendimiento por tareas de gestión del sistema, así como cierto requerimiento de recursos. Además, a la hora de la configuración requiere algunos servicios como Java o SSH, algo que tampoco es un inconveniente real, dado que ambos son servicios gratuitos.

Sistema de Ficheros Hadoop Distributed File System (HDFS)

Diseñado ex profeso para el entorno de computación distribuida *Apache Hadoop*, las ventajas de utilizar este sistema de ficheros distribuido en nuestro entorno son muchas, ya que éste aporta todas las características de los mejores y más utilizados sistemas de ficheros. Por tanto, configurar este sistema de ficheros HDFS proporciona:

- **Diseño específico** para el clúster ***Apache Hadoop***, **adaptabilidad máxima.**
- Incluido en la instalación de *Hadoop*, es de **licencia libre**.
- Muy **sencillo de configurar** ya que se hace a la vez que el clúster.
- Preparado para el **procesamiento de grandes volúmenes** de archivos.
- **Tolerante a fallos** con posibilidad de **réplica de nodos.**
- Diseñado para **computación paralela**.
- Permite la **concurrencia de clientes**.
- **Seguridad de acceso de usuarios** por clave RSA.
- Mantiene todas las ventajas de *Apache Hadoop* anteriormente comentadas (escalabilidad, flexibilidad…)

Uno de los **inconvenientes** de este sistema de ficheros es que, **con la configuración típica, no proporciona alta disponibilidad**. Sin embargo, es **sencillo incluir** esta característica en nuestro clúster, básicamente **replicando los nodos de control y aplicando** unos sencillos cambios en la **configuración**.

Integración Cloud Computing – Apache Hadoop

Como ya mencionamos anteriormente, en este proyecto hemos tratado de encontrar un entorno de computación distribuida capaz de procesar grandes volúmenes de datos. El hecho de **integrar** el entorno ***Apache Hadoop*** **en un sistema** ***Cloud Computing*** **podía provocar** tanto **incompatibilidad** de ambos sistemas como una gran **pérdida en el rendimiento**.

Conocemos el hecho de que el *framework* de computación distribuida ***Apache Hadoop*** **es integrable en cualquier tipo de sistema**, **incluyendo** los entornos **virtuales**. Además la sencillez de configuración e instalación que **permite ir añadiendo nodos al clúster** *Hadoop* cuando se desee, unido a que **en un entorno** ***Cloud Computing*** **podemos grabar instancias** ya configuradas para posteriormente **desplegar copias de las mismas**, hace que **el trabajo conjunto en un sistema** de ambos marcos de trabajo sea **especialmente adecuado**.

Otros inconvenientes que podemos encontrar en la integración de ambos entornos en un sistema de computación conjunto vienen dados por la **pérdida de rendimiento que las tareas de gestión** que uno y otro *framework* **generen y** porque es posible **que se desaprovechen recursos** de los servidores que conforman el clúster. La segunda de las desventajas, **la pérdida de recursos, es inapreciable ya que** la nube ***OpenStack*** es capaz de **crear instancias virtuales con los requisitos hardware que se deseen y en un clúster** ***Hadoop*** **pueden formar parte del**

mismo instancias de cualquier tipo, aunque a nivel hardware sean diferentes. Así, el **aprovechamiento de recursos es máximo**. En cuanto al primer inconveniente planteado, el hecho de **la pérdida de rendimiento por tareas de gestión** del entorno, podemos comentar que en el entorno de computación ***Apache Hadoop* no aumentan las tareas de gestión** porque el clúster esté formado **por instancias virtuales**. Por otro lado, en cuanto a **los sistemas *Cloud Computing***, gracias a proyectos como "Clúster de Alto Rendimiento en un Cloud: Ejemplo de Aplicación en Criptoanálisis de Funciones Hash" realizado por Gabriel Molero Escobar y dirigido por Julio Gómez López y Eugenio Eduardo Villar Fernández para la Universidad de Almería en septiembre de 2011, sabemos que **las pérdidas de rendimiento por tareas de control son más que asumibles**, teniendo en cuenta las ventajas que esta tecnología aporta al proyecto y que ya han sido mencionadas.

Programación MapReduce y librería Hadoop Image Processing Interface (HIPI)

Al igual que el sistema de ficheros HDFS, el lenguaje de programación *MapReduce* obtiene su máximo rendimiento en entornos de computación distribuidos y que procesen grandes volúmenes de datos, como es el caso de nuestro clúster *Apache Hadoop*. Por esto, es un lenguaje de programación especialmente indicado para este entorno y que aporta:

- **Basado en *Java*, soporte y desarrollo** están más que **garantizados**, dada la popularidad de este lenguaje.
- **Licencia libre**.
- **Orientado para procesar grandes volúmenes de datos**.
- Indicado para **entornos distribuidos**.
- **Programación funcional** que **divide los procesos en tareas *Map*** que se distribuyen por todos los nodos de trabajo del clúster y generan resultados intermedios. El HDFS distribuye los datos entre los nodos de almacenamiento y *MapReduce* al trabajar de manera conjunta con el sistema de ficheros, envía las tareas a los nodos en los que se encuentran los datos sobre los que van a computar. Los pares intermedios son procesados para obtener el resultado final en el *Reducer*.

Un **hándicap** de este entorno es el hecho de **depender de los formatos de entrada** de los archivos. **En la API de *MapReduce*** se incluyen varios formatos de archivos y entre ellos **no se encuentra** un **tipo** que puede suponerse como básico y que es el de **imagen**. Sin embargo, este contratiempo se soluciona **gracias a la librería *Hadoop Image Processing Interface*** que aporta **una sencilla forma de incluir grandes cantidades de imágenes** como entrada de un programa. Como hemos visto en capítulos anteriores, este procedimiento implica generar un contenedor de imágenes que será incluido en el HDFS y posteriormente servirá de entrada del trabajo *MapReduce*. Si atendemos a las pruebas realizadas, cuanto mayor sea el número de imágenes mejor es el comportamiento de este procedimiento.

7.1.2. Conclusiones generales

Una vez finalizada la revisión de cada uno de los aspectos fundamentales del proyecto, podemos enumerar las conclusiones que hemos obtenido fruto del estudio, configuraciones y pruebas realizadas:

- Se ha cumplido con el objetivo principal del proyecto, obteniendo un sistema de computación distribuida de alto rendimiento y sencilla configuración de licencia libre.
- Hemos logrado la integración *Cloud Computing* con el *framework* de procesamiento distribuido *Apache Hadoop*.
- *OpenStack* junto con la distribución baremetal *StackOps* proporciona una sencilla forma de desplegar sistemas *Cloud Computing* con los que aprovechar al máximo los recursos hardware disponibles.
- Tanto el sistema *Cloud Computing OpenStack* como el entorno distribuido *Apache Hadoop* conforman un sistema altamente adaptable, escalable, sencillo de configurar y que a su vez ofrece alto rendimiento en el procesamiento de elevados volúmenes de datos.
- El entorno de computación distribuida *Apache Hadoop* ha demostrado ser adaptable a cualquier tipo de sistema incluso virtuales alcanzando alto rendimiento en el procesamiento de grandes volúmenes de datos.
- El lenguaje de programación *MapReduce* en trabajo conjunto con la librería *Hadoop Image Processing Interface* ha demostrado ser un *framework* de procesamiento de archivos en entornos distribuidos muy potente y versátil.
- El sistema de ficheros distribuido *Hadoop Distributed File System* es soporte fundamental del entorno *Apache Hadoop MapReduce* proporcionando la gestión completa de los archivos utilizados por los programas *MapReduce,* distribuyendo en bloques los datos por los nodos del clúster.
- La programación *MapReduce* y el HDFS están adaptados en su totalidad para que el sistema de ficheros distribuya los archivos y las tareas *Map* se lleven a cabo en los nodos donde los bloques de datos se localicen.
- Las aplicaciones en el entorno *Apache Hadoop MapReduce* alcanzan el mayor rendimiento cuanto mayor son los conjuntos de datos de entrada reduciéndose, en función del aumento del tamaño de los mismos, la influencia de las tareas de gestión del entorno en el resultado final.

7.2. Trabajo Futuro

Este proyecto, al combinar varias tecnologías en un sistema común, resulta muy atractivo para desarrollar nuevas investigaciones sobre él. Pueden establecerse varias vías de investigación independientes; a nivel de programación *MapReduce*, desarrollo del entorno *Apache Hadoop* o trabajo con nuevos entornos *Cloud Computing*.

El entorno de computación *Apache Hadoop* junto con la **programación *MapReduce*** desplegado en este proyecto dentro de un sistema *Cloud Computing* puede servir como *framework* para multitud de **aplicaciones**, entre las que podemos valorar las siguientes:

1. Crear un **programa** que incluya la librería HIPI que **tome imágenes** (por ejemplo de satélite) **y detecte figuras o patrones** incluidos en ellas. Este tipo de aplicaciones son las que la empresa *Facebook* lleva a cabo en clústeres *Hadoop* detectando caras en imágenes. En nuestro caso, podría servir para detectar campos de cultivos, invernaderos, edificaciones, etc.

2. Este clúster dado que es distribuido, que está orientado a trabajar con grandes volúmenes de datos y que permite el acceso multiusuario concurrente a su sistema de ficheros, puede ser **utilizado para alojar cualquier tipo de red social o sistema de gestión de cualquier organización pública o privada** que suelen almacenar grandes cantidades de datos y sobre los que es preciso realizar gran cantidad de operaciones. Estas características han llevado a grandes empresas como *Yahoo!, eBay, Google, Facebook o Twitter* a migrar parte de sus servicios a clúster *Hadoop*.

3. Otra línea de investigación a nivel de aplicaciones es realizar un **sistema de gestión de bases de datos** en este entorno. En este proyecto se ha dado a conocer el hecho de que este entorno está especialmente indicado para trabajar con volúmenes de datos elevados, con lo que parece idóneo plantear su aplicación a las grandes bases de datos.

Otra vía de investigación está orientada al **desarrollo del clúster *Apache Hadoop***:

1. Uno de los inconvenientes originados tras el estudio de este entorno, es que **no posee alta disponibilidad**. Sin embargo, **es posible configurar esta característica**, incluyendo en un nuevo nodo o bien en uno de los ya existentes un *NameNode* secundario inactivo en un principio y que toma el control en caso de pérdida o error del primario. Por supuesto, este nodo ha de disponer de la misma información replicada del nodo de control activo, aunque también es necesario un mecanismo de control de errores, para que información errónea que pueda acceder al primario no acceda al secundario.

2. Otra posible vía es **crear clústeres en los que se integren máquinas virtuales de distintas características**. Podría plantearse un clúster en el que **los recursos se repartieran de manera óptima**. Es decir, el Nodo Controlador puede utilizarse solamente para tareas de control, en vez de control más cómputo y almacenamiento (como en nuestro clúster) con lo que ésta máquina debería de disponer de mayor capacidad de procesamiento y menor almacenamiento que se podría aprovechar para los Nodos Esclavos ejecutores de tareas *MapReduce*. En cualquier caso, gracias a la

flexibilidad de este entorno y a su adaptabilidad, lo más indicado es crear cada clúster en función de las necesidades del usuario y así alcanzar el rendimiento máximo.

Finalmente el último campo de trabajo en el que podría realizarse investigación sería el de la integración de este entorno en el paradigma de computación *Cloud Computing*. Gracias al estudio realizado, conocemos la existencia tanto de nubes públicas como privadas e híbridas. Ya se ha mencionado en varias ocasiones que el entorno de computación distribuida *Apache Hadoop* permite, gracias a su sistema de ficheros HDFS¸ acceso concurrente de usuarios. Por ello, puede ser **interesante crear un sistema basado en una nube pública en el que se ofrezca como servicio un clúster *Apache Hadoop MapReduce*** configurado para que empresas o usuarios particulares lo utilicen en bruto para integrar sus sistemas o bien ofrecer aplicaciones ya configuradas en el clúster. Es decir, podría ofrecerse cualquier aplicación, bien de las que se han utilizado en este proyecto o bien cualquiera de las planteadas a lo largo de este apartado como servicio para que usuarios externos puedan utilizarlas sin tener que configurar ningún sistema propio. También podría ofrecerse clústeres de distinta configuración en cuanto a capacidad de almacenamiento, número de servidores, etc. para que empresas o particulares realizasen investigaciones o implantasen su negocio sin necesidad de adquirir ningún tipo de equipo ni de configurar ningún sistema base.

Anexo I. Pruebas HIPI

Este anexo de la memoria recoge el código completo en *Java* de los dos programas utilizados para realizar las pruebas de la librería *Hadoop Image Processing Interface*.

El primer apartado recoge las clases que conforman el paquete que ejecuta el programa *Downloader* y en el segundo podemos ver la clase única que ejecuta la prueba *DumpHib* con la que se expanden los contenedores HIB.

II.1. Código Prueba *Downloader*

El programa *Downloader.java* se compone de tres ficheros *Java* que contienen todas las clases que son necesarias para la ejecución. Estos archivos son *Downloader.java, DownloaderInputFormat.java* y *DownloaderRecordReader* que están explicados por partes en el *Capítulo 6. Ejemplos de Aplicación del Entorno Apache Hadoop MapReduce* y que aquí podemos ver al completo.

Clase Downloader.java

A continuación podemos ver la clase principal del paquete *Downloader* que contiene tanto el método `main()` que controla la ejecución como los dos métodos más importantes que son el *Map* y el *Reduce*.

```
package hipi.examples.downloader;

import hipi.image.ImageHeader.ImageType;
import hipi.imagebundle.HipiImageBundle;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.StringReader;
import java.net.URL;
import java.net.URLConnection;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BooleanWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
```

```
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * A utility MapReduce program that takes a list of image URL's,
downloads them, and creates
 * a {@link hipi.imagebundle.HipiImageBundle} from them.
 *
 * When running this program, the user must specify 3
parameters. The first is the location
 * of the list of URL's (one URL per line), the second is the
output path for the HIB that will
 * be generated, and the third is the number of nodes that
should be used during the
 * program's execution. This final parameter should be chosen
with respect to the total
 * bandwidth your particular cluster is able to handle. An
example usage would be:
 * <br /><br />
 * downloader.jar /path/to/urls.txt /path/to/output.hib 10
 * <br /><br />
 * This program will automatically force 10 nodes to download
the set of URL's contained in
 * the input list, thus if your list contains 100,000 images,
each node in this example will
 * be responsible for downloading 10,000 images.
 *
 */
public class Downloader extends Configured implements Tool{


     public static class DownloaderMapper extends
Mapper<IntWritable, Text, BooleanWritable, Text>
     {
          private static Configuration conf;
          // This method is called on every node
          public void setup(Context jc) throws IOException
          {
               conf = jc.getConfiguration();
          }

          public void map(IntWritable key, Text value, Context
context)
          throws IOException, InterruptedException
          {
               String temp_path =
conf.get("downloader.outpath") + key.get() + ".hib.tmp";
               System.out.println("Temp path: " + temp_path);

               HipiImageBundle hib = new HipiImageBundle(new
Path(temp_path), conf);
               hib.open(HipiImageBundle.FILE_MODE_WRITE, true);

               String word = value.toString();
```

```
                BufferedReader reader = new BufferedReader(new
StringReader(word));
                String uri;
                int i = key.get();
                int iprev = i;
                while((uri = reader.readLine()) != null)

                {
                     if(i >= iprev+100) {
                          hib.close();
                          context.write(new
BooleanWritable(true), new Text(hib.getPath().toString()));
                          temp_path =
conf.get("downloader.outpath") + i + ".hib.tmp";
                          hib = new HipiImageBundle(new
Path(temp_path), conf);

     hib.open(HipiImageBundle.FILE_MODE_WRITE, true);
                          iprev = i;
                     }
                     long startT=0;
                     long stopT=0;
                     startT = System.currentTimeMillis();

                     try {
                          String type = "";
                          URLConnection conn;
                          // Attempt to download
                          context.progress();

                          try {
                               URL link = new URL(uri);
                               System.err.println("Downloading
" + link.toString());
                               conn = link.openConnection();
                               conn.connect();
                               type = conn.getContentType();
                          } catch (Exception e)
                          {
                               System.err.println("Connection
error to image : " + uri);
                               continue;
                          }

                          if (type == null)
                               continue;

                          if (type.compareTo("image/gif") == 0)
                               continue;

                          if (type != null &&
type.compareTo("image/jpeg") == 0)

     hib.addImage(conn.getInputStream(), ImageType.JPEG_IMAGE);

                     } catch(Exception e)
```

```
                        {
                              e.printStackTrace();
                              System.err.println("Error... probably
cluster downtime");
                              try
                              {
                                    Thread.sleep(1000);
                              } catch (InterruptedException e1)
                              {
                                    e1.printStackTrace();
                              }
                        }

                        i++;

                        // Emit success
                        stopT = System.currentTimeMillis();
                        float el = (float)(stopT-startT)/1000.0f;
                        System.err.println("> Took " + el + "
seconds\n");
                  }


                  try
                  {
                        reader.close();
                        hib.close();
                        context.write(new BooleanWritable(true),
new Text(hib.getPath().toString()));
                  } catch (Exception e)
                  {
                        e.printStackTrace();
                  }

            }
      }

      public static class DownloaderReducer extends
Reducer<BooleanWritable, Text, BooleanWritable, Text> {

            private static Configuration conf;
            public void setup(Context jc) throws IOException
            {
                  conf = jc.getConfiguration();
            }

            public void reduce(BooleanWritable key,
Iterable<Text> values, Context context)
            throws IOException, InterruptedException
            {
                  if(key.get()){
                        FileSystem fileSystem =
FileSystem.get(conf);
                        HipiImageBundle hib = new
HipiImageBundle(new Path(conf.get("downloader.outfile")), conf);
```

```
                        hib.open(HipiImageBundle.FILE_MODE_WRITE,
true);
                        for (Text temp_string : values) {
                              Path temp_path = new
Path(temp_string.toString());
                              HipiImageBundle input_bundle = new
HipiImageBundle(temp_path, conf);
                              hib.append(input_bundle);

                              Path index_path =
input_bundle.getPath();
                              Path data_path = new
Path(index_path.toString() + ".dat");
                              System.out.println("Deleting: " +
data_path.toString());
                              fileSystem.delete(index_path, false);
                              fileSystem.delete(data_path, false);

                              context.write(new
BooleanWritable(true), new
Text(input_bundle.getPath().toString()));
                              context.progress();
                        }
                        hib.close();
                  }
            }
      }


      public int run(String[] args) throws Exception
      {

            // Read in the configuration file
            if (args.length < 3)
            {
                  System.out.println("Usage: downloader <input
file> <output file> <nodes>");
                  System.exit(0);
            }

            // Setup configuration
            Configuration conf = new Configuration();

            String inputFile = args[0];
            String outputFile = args[1];
            int nodes = Integer.parseInt(args[2]);

            String outputPath = outputFile.substring(0,
outputFile.lastIndexOf('/')+1);
            System.out.println("Output HIB: " + outputPath);


            conf.setInt("downloader.nodes", nodes);
            conf.setStrings("downloader.outfile", outputFile);
            conf.setStrings("downloader.outpath", outputPath);
```

```
            Job job = new Job(conf, "downloader");
            job.setJarByClass(Downloader.class);
            job.setMapperClass(DownloaderMapper.class);
            job.setReducerClass(DownloaderReducer.class);

            // Set formats
            job.setOutputKeyClass(BooleanWritable.class);
            job.setOutputValueClass(Text.class);
            job.setInputFormatClass(DownloaderInputFormat.class);

            //*************** IMPORTANT ***************\\
            job.setMapOutputKeyClass(BooleanWritable.class);
            job.setMapOutputValueClass(Text.class);
            FileOutputFormat.setOutputPath(job, new
Path(outputFile + "_output"));

            DownloaderInputFormat.setInputPaths(job, new
Path(inputFile));

            job.setNumReduceTasks(1);
            System.exit(job.waitForCompletion(true) ? 0 : 1);
            return 0;
      }

      public static void createDir(String path, Configuration
conf) throws IOException {
            Path output_path = new Path(path);

            FileSystem fs = FileSystem.get(conf);

            if (!fs.exists(output_path)) {
                  fs.mkdirs(output_path);
            }
      }

      public static void main(String[] args) throws Exception {
            int res = ToolRunner.run(new Downloader(), args);
            System.exit(res);
      }
}
```

Clase DownloaderInputFormat.java

Esta clase genera un formato de entrada específico para esta aplicación cuya funcionalidad reside en que es capaz de forzar a un número determinado de nodos de un clúster a procesar ciertas líneas de un fichero de texto.

```
package hipi.examples.downloader;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.BlockLocation;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

public class DownloaderInputFormat extends
FileInputFormat<IntWritable, Text>
{
	/**
	 * Returns an object that can be used to read records of
type ImageInputFormat
	 */
	@Override
	public RecordReader<IntWritable, Text>
createRecordReader(InputSplit genericSplit, TaskAttemptContext
context)
	throws IOException, InterruptedException
	{
		return new DownloaderRecordReader();
	}


	@Override
	public List<InputSplit> getSplits(JobContext job) throws
IOException
	{
		Configuration conf = job.getConfiguration();
		int nodes = conf.getInt("downloader.nodes", 10);


		ArrayList<String> hosts = new ArrayList<String>(0);
```

```
          List<InputSplit> splits = new
ArrayList<InputSplit>();

          FileSystem fileSystem = FileSystem.get(conf);
          String tempOutputPath =
conf.get("downloader.outpath") + "_tmp";
          Path tempOutputDir = new Path(tempOutputPath);

          if (fileSystem.exists(tempOutputDir))
          {
               fileSystem.delete(tempOutputDir, true);
          }
          fileSystem.mkdirs(tempOutputDir);

          int i = 0;
          while( hosts.size() < nodes && i < 2*nodes)
          {
               String tempFileString = tempOutputPath + "/" +
i;
               Path tempFile = new Path(tempFileString);
               FSDataOutputStream os =
fileSystem.create(tempFile);
               os.write(i);
               os.close();

               FileStatus match =
fileSystem.getFileStatus(tempFile);
               long length = match.getLen();
               BlockLocation[] blocks =
fileSystem.getFileBlockLocations(match, 0, length);

               boolean save = true;
               for (int j = 0; j < hosts.size(); j++)
               {
                    if
(blocks[0].getHosts()[0].compareTo(hosts.get(j)) == 0)
                    {
                         save = false;
                         System.out.println("Repeated host: "
+ i);
                         break;
                    }
               }

               if (save)
               {
                    hosts.add(blocks[0].getHosts()[0]);
                    System.out.println("Found host
successfully: " + i);
               }
               i++;
          }

          System.out.println("Tried to get " + nodes + " nodes,
got " + hosts.size());
```

```
          FileStatus file = listStatus(job).get(0);
          Path path = file.getPath();
          BufferedReader reader = new BufferedReader(new
InputStreamReader(fileSystem.open(path)));
          int num_lines = 0;
          while(reader.readLine() != null)
          {
               num_lines++;
          }
          reader.close();

          int span = (int) Math.ceil(((float) (num_lines)) /
((float) hosts.size()));
          int last = num_lines - span * (hosts.size()-1);
          System.out.println("First n-1 nodes responsible for "
+ span + " images");
          System.out.println("Last node responsible for " +
last + " images");

          FileSplit[] f = new FileSplit[hosts.size()];
          for (int j = 0; j < f.length; j++)
          {
               String[] host = new String[1];
               host[0] = hosts.get(j);
               if (j < f.length - 1)
               {
                    splits.add( new FileSplit(path , (j*span)
, span, host));
               } else
               {
                    splits.add( new FileSplit(path , (j*span)
, last, host));
               }
          }

          if (fileSystem.exists(tempOutputDir))
          {
               fileSystem.delete(tempOutputDir, true);
          }
          return splits;
     }
}
```

Clase DownloaderRecordReader.java

Es utilizada por el *Downloader* para generar un registro que sirve de entrada a los *Mappers*. Cada *Mapper* recibe la lista de URLs que ha de descargar del fichero de entrada de la ejecución. Esta clase genera un registro que contiene el número de la línea del fichero del que parte la descarga y la lista de URLs a descargar.

```
package hipi.examples.downloader;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

/**
 * Treats keys as index into training array and value as the
training vector.
 */
public class DownloaderRecordReader extends
RecordReader<IntWritable, Text>
{
	private boolean singletonEmit;
	private String urls;
	private long start_line;

	public void initialize(InputSplit split, TaskAttemptContext
context)
	throws IOException, InterruptedException {
		FileSplit f = (FileSplit) split;
		Path path = f.getPath();
		Configuration conf = context.getConfiguration();
		FileSystem fs = path.getFileSystem(conf);

		start_line = f.getStart();
		long num_lines = f.getLength();

		singletonEmit = false;

		BufferedReader reader = new BufferedReader(new
InputStreamReader(fs.open(path)));
		int i = 0;
		while(i < start_line && reader.readLine() != null){
			i++;
		}

		urls = "";
		String line;
```

```
          for(i = 0; i < num_lines && (line =
reader.readLine()) != null; i++){
               urls += line + '\n';
          }
          reader.close();
     }


     /**
      * Get the progress within the split
      */
     public float getProgress()
     {
          if (singletonEmit) {
               return 1.0f;
          } else {
               return 0.0f;
          }
     }

     public void close() throws IOException
     {
          return;
     }

     @Override
     public IntWritable getCurrentKey() throws IOException,
     InterruptedException {
          return new IntWritable((int)start_line);
     }

     @Override
     public Text getCurrentValue() throws IOException,
     InterruptedException {
          return new Text(urls);
     }

     @Override
     public boolean nextKeyValue() throws IOException,
InterruptedException {
          if(singletonEmit == false){
               singletonEmit = true;
               return true;
          }
          else
               return false;
     }
}
```

II.2. Código Prueba *DumpHib*

A continuación, podemos ver el código fuente escrito en *Java* que se utiliza en la segunda de las pruebas realizadas con la librería HIPI, que toma como entrada un contenedor HIB, lo descomprime y realiza algunas operaciones sobre cada una de las imágenes, creando para finalizar un archivo de texto con ciertas propiedades de cada una de las imágenes.

```
package hipi.examples.dumphib;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import hipi.image.FloatImage;
import hipi.image.ImageHeader;
import hipi.imagebundle.mapreduce.ImageBundleInputFormat;
import hipi.util.ByteUtils;

public class DumpHib extends Configured implements Tool {

     public static class DumpHibMapper extends
Mapper<ImageHeader, FloatImage, IntWritable, Text> {

          @Override
          public void map(ImageHeader key, FloatImage value,
Context context)
          throws IOException, InterruptedException {
               if (value != null) {
                    int imageWidth = value.getWidth();
                    int imageHeight = value.getHeight();
                    String hexHash =
ByteUtils.asHex(ByteUtils.FloatArraytoByteArray(value.getData())
);
                    String camera =
key.getEXIFInformation("Model");
                    String output = imageWidth + "x" +
imageHeight + "\t(" + hexHash + ")\t  " + camera;

                    context.write(new IntWritable(1), new
Text(output));
               }
          }
     }
```

```
     public static class DumpHibReducer extends
Reducer<IntWritable, Text, IntWritable, Text> {
          public void reduce(IntWritable key, Iterable<Text>
values, Context context)
          throws IOException, InterruptedException {
               for (Text value : values) {
                    context.write(key, value);
               }
          }
     }

     public int run(String[] args) throws Exception {
          Configuration conf = new Configuration();
          if (args.length < 2) {
               System.out.println("Usage: dumphib <input hib>
<outputdir>");
               System.exit(0);
          }
          String inputPath = args[0];
          String outputPath = args[1];

          Job job = new Job(conf, "dumphib");
          job.setJarByClass(DumpHib.class);
          job.setMapperClass(DumpHibMapper.class);
          job.setReducerClass(DumpHibReducer.class);

          // Set formats
          job.setOutputKeyClass(IntWritable.class);
          job.setOutputValueClass(Text.class);

     job.setInputFormatClass(ImageBundleInputFormat.class);

          // Set out/in paths
          removeDir(outputPath, conf);
          FileOutputFormat.setOutputPath(job, new
Path(outputPath));
          FileInputFormat.setInputPaths(job, new
Path(inputPath));

          job.setNumReduceTasks(1);
          System.exit(job.waitForCompletion(true) ? 0 : 1);
          return 0;
     }

     public static void main(String[] args) throws Exception {
          int exitCode = ToolRunner.run(new DumpHib(), args);
          System.exit(exitCode);
     }

     public static void removeDir(String path, Configuration
conf) throws IOException {
          Path output_path = new Path(path);

          FileSystem fs = FileSystem.get(conf);

          if (fs.exists(output_path)) {
```

```
                fs.delete(output_path, true);
            }
        }
}
```

Apéndice A. Enlaces de Interés.

A continuación se van a listar todas las URLs que han sido utilizadas para la documentación del proyecto y que es recomendable visitar. Estos enlaces aparecen listados por grupos, según temática.

Cloud Computing

- *OpenStack*: http://www.openstack.org
- *StackOps*: http://www.stackops.com
- *Abiquo*: http://www.abiquo.com/
- *Bitnami*: http://bitnami.org/
- *CloudStack*: http://www.cloudstack.com
- *CA 3Tera AppLogic*: http://www.ca.com/us/cloud-platform.aspx
- *Cloud.com CloudStack*: http://www.cloudstack.com/
- *Collectd*: http://collectd.org/
- *ControlTier:* http://doc36.controltier.org/wiki/Main_Page
- *Convirture ConVirt*: http://www.convirture.com/index.php
- *Enomaly plataforma Elastic Computing (ECP)*: http://src.enomaly.com/
- *Eucalyptus*: http://www.eucalyptus.com/.
- *Extility Flexiant*: http://www.flexiant.com/2011/11/18/extility-1-5-2-release/
- *HP CloudSystem*: http://www8.hp.com/us/en/business-solutions/solution.html?compURI=1079455#.URFMZvLjE5Z
- *IBM CloudBurst*: http://www-01.ibm.com/software/tivoli/products/cloudburst/
- *Incontinuum CloudController*: http://www.incontinuum.com/
- *Nimbula Director*: http://nimbula.com/
- *OnApp*: http://onapp.com/cloud/pricing/
- *OpenNebula*: http://opennebula.org/
- *OpenQRM*: http://www.openqrm-enterprise.com/community/
- *Opscode Chef*: http://wiki.opscode.com/display/chef/Home
- *Parallels Automation Cloud Infrastructure (CI)*: http://www.parallels.com/es/products/iaas/
- *Puppet*: https://puppetlabs.com/
- *VMware vCloud*: http://www.vmware.com/products/datacenter-virtualization/vcloud-suite/overview.html
- *Xen plataforma de nube (XCP)*: http://www.xen.org/products/cloudxen.html
- *Zenoss*: http://community.zenoss.org/index.jspa

- 21 Plataformas Cloud (Iaas Software): http://www.jorgedelacruz.es/2011/12/04/21-plataformas-cloud-iaas-software/

OpenStack y StackOps

- Contruyendo un Cloud Privado: http://www.jorgedelacruz.es/category/stackops/
- Implementación de *OpenStack* con *StackOps*: http://es.scribd.com/doc/77975915/Implementacion-de-StackOps
- *OpenStack Compute Starter Guide*: http://docs.openstack.org/trunk/openstack-compute/starter/content/Introduction_to_OpenStack_and_its_components-d1e59.html
- *OpenStack Compute Administration Manual* (Obtener imagines de máquinas virtuales): http://docs.openstack.org/trunk/openstack-compute/admin/content/starting-images.html
- *OpenStack Compute Administration Manual* (Crear imagines de instancias en ejecución con KVM y XEN): http://docs.openstack.org/trunk/openstack-compute/admin/content/creating-images-from-running-instances.html
- *Fedora Virtualizatión Guide* (Uso de qemu-img): https://docs.fedoraproject.org/es-ES/Fedora/12/html/Virtualization_Guide/sect-Virtualization_Guide-Tips_and_tricks-Using_qemu_img.html
- Documentación *OpenStack*: http://docs.openstack.org/

Apache Hadoop

- *Apache Hadoop*, web oficial: http://hadoop.apache.org
- *Apache Hadoop Main 2.0.2-alpha API*: http://hadoop.apache.org/docs/current/api/overview-summary.html
- *Apache Download Mirrors*: http://www.apache.org/dyn/closer.cgi/hadoop/core
- *Getting Started with Hadoop: http://wiki.apache.org/hadoop/GettingStartedWithHadoop*
- *Hadoop Powered by*: http://wiki.apache.org/hadoop/PoweredBy
- Cómo instalar *Hadoop* en *Ubuntu (Single Node)*: http://www.binaridev.es/2011/02/como-instalar-hadoop-en-ubuntu-modo.html
- Cómo instalar *Hadoop* en *Ubuntu (multinode)*: http://www.binaridev.es/2010/12/como-instalar-hadoop-en-ubuntu-modo.html
- *Running Hadoop on Ubuntu Linux (Single-Node Cluster)*: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/
- *Running Hadoop on Ubuntu Linux (Multi-Node Cluster)*: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/
- Computación Distribuida con *Hadoop* II: http://www.forondarena.net/2010/02/computacion-distribuida-con-hadoop-ii/
- Instalación de *Hadoop* (Parte I): http://www.undercloud.org/?p=18
- Instalación de *Hadoop* (Parte II): http://www.undercloud.org/?p=32
- Instalación de *Hadoop* (Parte III Multi-Node): http://www.undercloud.org/?p=93

- Tecnologías Grid, Capítulo 6.- *Hadoop*: http://ocw.uniovi.es/file.php/27/MSISEINF-1-002/practicas/datagrid_hadoop.html
- Introducción a *Apache Hadoop*: http://www.slideshare.net/paradigmatecnologico/introduccion-apache-hadoop
- *Hadoop* en Acción: www.slideshare.net/campuspartycolombia/hadoop-en-accion
- *Hadoop MapReduce* para Procesar Grandes Cantidades de Datos: http://www.slideshare.net/rochoa/hadoop-mapreduce-para-procesar-grandes-cantidades-de-datos
- Intalación de *Hadoop* sobre *Solaris 10*: http://sparcki.blogspot.com.es/2009/07/instalacion-de-hadoop-sobre-solaris-10.html
- *Hadoop on Windows with Eclipse*: http://v-lad.org/Tutorials/Hadoop/00%20-%20Intro.html
- *Big Data – Hadoop – BIDOOP*: http://bigdata-hadoop.pragsis.com/pages/2/big_data_hadoop_bidoop
- *Running Hadoop on Windows*: http://hayesdavis.net/2008/06/14/running-hadoop-on-windows/
- *Bitvise Tunnelier:* http://dl.bitvise.com/BvSshClient-Inst.exe

Hadoop Distributed File System

- *Hadoop Distributed File System*: http://hadoop.apache.org/hdfs
- *The Small Files Problem*: http://blog.cloudera.com/blog/2009/02/the-small-files-problem/
- *Hadoop Shell Commands*: http://hadoop.apache.org/docs/r0.17.2/hdfs_shell.html
- *Hadoop DFS User Guide:* *http://hadoop.apache.org/docs/r0.17.2/hdfs_user_guide.html*
- Alta Disponibilidad para el Sistema de Archivos Distribuidos *Hadoop* HDFS: http://bigdata-hadoop.pragsis.com/blog/2012/03/22/alta_disponibilidad_para_el_sistema_de_archivos_distribuidos_hadoop_hdfs
- HDFS: *Hadoop Distributed File System*: http://www.enriquedominguez.com/hdfs-hadoop-distributed-file-system/
- Herramientas *Backup NameNode* para HDFS (*Hadoop Distributed File System*): http://bigdata-hadoop.pragsis.com/blog/2012/05/30/herramientas_backup_namenode_para_hdfs_hadoop_distributed_file_system
- *Andrew File System*: http://www.openafs.org/
- CODA *File System*: http://coda.cs.cmu.edu/
- *Encrypting File System* (EFS): http://technet.microsoft.com/en-us/library/cc721923%28WS.10%29.aspx

MapReduce

- *MapReduce*: http://www.mapreduce.org
- *MapReduce Wikipedia*: http://es.wikipedia.org/wiki/MapReduce
- *Apache Hadoop MapReduce Tutorial*: http://hadoop.apache.org/docs/r0.20.2/mapred_tutorial.html
- *How to Include Third-Party Libraries in Your MapReduce Job*: http://blog.cloudera.com/blog/2011/01/how-to-include-third-party-libraries-in-your-map-reduce-job/
- *Indexing Files via Solr and Java MapReduce*: http://blog.cloudera.com/blog/2012/03/indexing-files-via-solr-and-java-mapreduce/
- *Building Hadoop MapReduce Jobs In Java*: http://www.higherpass.com/java/Tutorials/Building-Hadoop-Mapreduce-Jobs-In-Java/
- *Writing your first map-reduce program on hadoop*: http://jayant7k.blogspot.com.es/2010/06/writing-your-first-map-reduce-program.html
- *MapReduce Part II*: http://www.theserverside.com/news/1321218/MapReduce-Part-II
- Pensando en paralelo - Introduccion a Map/Reduce: http://cesar.la/pensando-en-paralelo-introduccion-a-mapreduce.html
- Uso de MapReduce y equilibrio de la carga en nube: http://www.ibm.com/developerworks/ssa/cloud/library/cl-mapreduce/index.html
- *Cloud Computing* y *MapReduce*: http://www.slideshare.net/jelabra/cloud-computing-y-mapreduce
- *MapReduce*: http://www.slideshare.net/vicente.ordonez/mapreduce
- API *MapReduce JobConf*: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapred/JobConf.html
- API *MapReduce Mapper*: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapred/Mapper.html
- API *MapReduce Reducer*: http://hadoop.apache.org/docs/r0.20.0/api/org/apache/hadoop/mapred/Reducer.html
- API *MapReduce InputFormat*: http://hadoop.apache.org/docs/r0.20.2/api/org/apache/hadoop/mapred/InputFormat.html
- API *MapReduce OutputFormat*: http://hadoop.apache.org/docs/r0.20.2/api/org/apache/hadoop/mapred/OutputFormat.html
- *MapReduce: The Programming Model and Practice*: http://www.cslu.ogi.edu/~zak/cs506-pslc/mapred_slides.pdf
- *Using the Java Mapper Framework for App Engine*: http://ikaisays.com/2010/07/09/using-the-java-mapper-framework-for-app-engine/
- *Amazon Elastic MapReduce*: http://aws.amazon.com/es/elasticmapreduce/

- *Teradata Aster*: http://www.asterdata.com/product/advanced-analytics.php http://www.asterdata.com/resources/mapreduce.php
- *Greenplum MapReduce*: http://www.greenplum.com/technology/mapreduce
- *Disco Project*: http://discoproject.org/
- *Holumbus*: http://holumbus.fh-wedel.de/trac/wiki/MapReduce
- *Gutenberg*: http://www.gutenberg.org/

Hadoop Image Processing Interface

- *Hadoop Image Processing Interface*: http://hipi.cs.virginia.edu
- *Hadoop binary files processing introduced by image duplicates finder*: http://eldadlevy.wordpress.com/2011/02/05/hadoop-binary-files-processing-entroduced-by-image-duplicates-finder/
- *Keith Wiley: Astronomical Image Processing with Hadoop*: http://escience.washington.edu/get-help-now/astronomical-image-processing-hadoop
- *Using Amazon MapReduce/Hadoop for Image Processing*: http://stackoverflow.com/questions/7816334/using-amazon-mapreduce-hadoop-for-image-processing
- *HIPI: A Hadoop Image Processing Interface for Image-based MapReduce Tasks*: http://cs.ucsb.edu/~cmsweeney/papers/undergrad_thesis.pdf
- *University of Virginia*: http://www.virginia.edu/
- API HIPI: http://hipi.cs.virginia.edu/doc/api/index.html
- API *Hadoop Configuration*: http://hadoop.apache.org/docs/current/api/org/apache/hadoop/conf/Configuration.html
- API *HipiImageBundle*: http://hipi.cs.virginia.edu/doc/api/hipi/imagebundle/HipiImageBundle.html
    - *FileReader*: http://hipi.cs.virginia.edu/doc/api/hipi/imagebundle/HipiImageBundle.FileReader.html
- API HIPI *FloatImage*: http://hipi.cs.virginia.edu/doc/api/hipi/image/FloatImage.html
- API HIPI *CullMapper*: http://hipi.cs.virginia.edu/doc/api/hipi/imagebundle/mapreduce/CullMapper.html
- API HIPI HipiJob: http://hipi.cs.virginia.edu/doc/api/hipi/imagebundle/mapreduce/HipiJob.html
- HIPI Downloads: http://hipi.cs.virginia.edu/downloads.html
- *Flickr*: http://www.flickr.com/

Bibliografía

- S. Zhang, S. Zhang, X. Chen y X. Huo. *Cloud Computing Research and Development Trend*. *Second International Conference on Future Networks*. 2010.

- A. C. Guerrero y E. K. Mena. *Implementación de un Prototipo de Cloud Computing de Modelo Privado para ofrecer Infraestructura como Servicio*. Escuela Politecnica Nacional Quito. Febrero 2011.

- Sun Microsystems. *Introduction to Cloud Computing Architecture*. *White Paper 1st Edition*. Junio 2009. http://www.sun.com/offers/docs/cloud_computing_primer.pdf

- T. Jones. *Virtual Linux, An Overview of virtualization methods, architectures, and implementations*. *Consultant Engineer, Emulex*. IBM, 2006.

- E. P. Montanera. *Gestión sostenible de clústers de recursos virtuales*. Universitat Autonoma de Barcelona. Septiembre 2009.

- B. Furht, A. Escalante. *Handbook of Cloud Computing*. Springer, 2010.

- R. Buyya. *High Performance Cluster Computing: Architectures and Systems*. Volumen 1. Prentice-Hall PRT. 1999.

- R. Buyya. *High Performance Cluster Computing: Programing and applications*. Volumen 2. Prentice-Hall PRT. 1999.

- B. Sotomayor, R. S. Montero, I. M. Llorente y I. Foster. *Virtual infrastructure management in private and hybrid clouds. IEEE Internet Computing*. 2009.

- D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youse y D. Zagorodnov. *The Eucalyptus Open-Dource Cloud-Computing System*. *9th IEEE International Symposium on Cluster Computing and the Grid*, Shanghai, China, 2008.

- K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. S. Harvey, J. Wasserman y N. J. Wright. *Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud*. *2nd IEEE International Conference on Cloud Computing Technology and Science.* 2010.

- C. Vecchiola, S. Pandey y R. Buyya. *High Performance Cloud Computing: A View of Scientific Applications*. *10th International Symposium on Pervasive Systems, Algorithms, and Networks*. 2009.

- G. Mateescu, W. Gentzsch y C. J. Ribbens. *Hybrid Computing – Where HPC meets grid and Cloud Computing*. Elsevir B.V. 2010.

- A. S. Tanenbaum. *Sistemas Operativos Modernos*. Pearson Educación, 2003.

- R. Pressman. *Ingeniería del software: un enfoque práctico*. McGraw-Hill, 2006.

- L. Gillam. *Cloud Computing: Principles, Systems and Applications*. Springer, 2010.

- T. Velte, A. Velte, T. J. Velte y Robert C. Elsenpeter. *Cloud Computing: A Practical Approach*. McGraw Hill Professional, 2009.

- J. Gómez, F. Gil, E. Villar y F. Méndez. *Administración avanzada de sistemas informáticos*. RA-MA Editorial 2010.

- R. Lämmel. *Google's MapReduce Programming Model. Data Programmability Team Microsoft Corp*. Redmond, WA, USA,

- Cheng-Tao Chu. *Map-Reduce for Machine Learning on Multicore.* Stanford University, Stanford CA.

- J. Dean y S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters.* Google Inc.

- A. Luengo. *Desarrollo de un entorno basado en MapReduce para la ejecución distribuida de algoritmos genéticos paralelos.* Proyecto Fin de Carrera, Universidad de La Coruña. 2010

- G. Molero, J. Gómez y E. Villar. *Clúster de Alto Rendimiento en un Cloud: Ejemplo de Aplicación en Criptoanálisis de Funciones Hash.* Proyecto Fin de Carrera, Universidad de Almería. 2011.

- J. Seoane. *Ejemplos de Sistemas de Ficheros Distribuidos.* Departamento de Ingeniería de Sistemas Telemáticos Universidad Politécnica de Madrid. 2009.

- J. Zhao. *MapReduce, the Programming Model and Practice*. Google Inc. 2009.

- D. N. Peek. *Consumer Distributed File Systems*. University of Michigan. 2009.

- M. Addison. *Caching in Large-Scale Distributed File Systems*. Princeton University, Dept of Computer Science. 1993.
- *Distributed File Systems*. General Books LLC, 2010
- T. White. *Hadoop, the Definitive Guide.* O'Reilly. 2010
- J. Venner. *Pro Hadoop.* Apress. 2009.
- C. Lam. *Hadoop in Action*. O'Reilly Media. 2010.
- C. Eaton y P. Zikopoulos. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw Hill Professional. 2011.
- A. Holmes. *Hadoop in Practice*. Manning Publications Company.2012.
- S. Perera y T.Gunarathne. *Hadoop MapReduce Cookbook*. Packt Publinning. 2013.
- J. Lin y C. Dyer. *Data-Intensive Text Processing With MapReduce*. Graeme Hirst, Series Editor. 2010.
- D. Miner y A. Shook. *MapReduce Design Patterns.* O'Reilly. 2012.
- K. Pepple. *Deploying OpenStack*. O'Reilly. 2011. Packt Publinning. 2012.
- K. Jackson. OpenStack Cloud Computing Cookbook.
- MapReduce Tutorial, Apache Hadoop, http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html