



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Máster en Ingeniería de Computadores y Redes  
Trabajo Fin de Máster

# Instalación, configuración y evaluación de un clúster de cómputo

**Autor:** *Jordi Climent Andrés*

**Director(es):** *Pedro Juan López Rodríguez, María Elvira Baydal Cardona*

*Septiembre 2016*



# Resumen

Este proyecto trata sobre la instalación, configuración y evaluación de un clúster de cómputo de alta disponibilidad, es decir, un clúster orientado al cálculo científico, mediante la utilización de máquinas virtuales con la distribución de Linux Ubuntu, poniendo especial hincapié en las tecnologías que nos van a proporcionar la alta disponibilidad y la distribución de la carga de cómputo.



# Tabla de contenidos

1	Introducción .....	7
2	Definición de la estructura del clúster .....	8
2.1	Redes del clúster .....	9
2.2	Nodos maestro.....	10
2.3	Nodos servidor .....	10
2.4	Nodo NFS .....	11
2.5	Nodos SAN .....	11
3	Software utilizado .....	12
3.1	dnsmasq.....	12
3.2	PXE .....	12
3.3	Servidor NFS.....	13
3.4	NIS .....	14
3.5	GlusterFS.....	15
3.6	MOSIX .....	15
3.7	Condor.....	16
3.8	Keepalived.....	17
3.9	HAProxy.....	17
4	Instalación y configuración del clúster.....	19
4.1	Creación de las máquinas virtuales .....	19
4.2	Instalación del sistema operativo .....	21
4.3	Clonación de los servidores .....	26
4.3.1	Scripts para facilitar la clonación.....	26
4.3.2	Preparación de la imagen a clonar .....	27
4.3.3	Instalación del servidor PXE .....	31
4.4	Sistema de almacenamiento para clústeres.....	37
4.4.1	GlusterFS.....	37
4.5	Sistema de alta disponibilidad en el nodo director .....	40
4.5.1	Keepalived .....	40
4.5.2	HAProxy .....	42

4.6	Sistema de gestión de trabajos de usuario.....	44
4.6.1	Instalación y configuración de NIS .....	44
4.6.2	Instalación y configuración de MOSIX.....	47
4.6.3	Instalación y configuración de Condor .....	49
5	Pruebas realizadas.....	53
5.1	Sistema de almacenamiento SAN .....	53
5.2	Channel bonding .....	55
5.3	Sistema de alta disponibilidad .....	56
5.4	Gestión de trabajos de usuario.....	59
5.4.1	MOSIX .....	59
5.4.2	CONDOR.....	62
6	Conclusiones.....	69
7	Referencias bibliográficas .....	70
8	Índice de imágenes y tablas .....	72
8.1	Tablas .....	72
8.2	Figuras.....	72
8.3	Imágenes.....	72

# 1 Introducción

Un clúster de computadores está compuesto por un conjunto de computadores convencionales interconectados mediante una red, que combinan sus capacidades y prestaciones para ofrecer a los usuarios la imagen de un único sistema de gran potencia de cómputo, más potente que los ordenadores comunes de escritorio.

Los clústeres ofrecen una serie de ventajas, como son alto rendimiento, alta disponibilidad, alta eficiencia y escalabilidad, lo que los convierte en el candidato perfecto para que las empresas puedan disponer de servidores que ofrezcan sus servicios sin interrupción o supercomputadores especializados para la realización de cálculos científicos complejos.

La configuración que suelen adoptar estos sistemas es la de un nodo maestro o director que permite la entrada al sistema y realiza el reparto de la carga entre los nodos servidores.

En este Proyecto de Fin de Máster se ha instalado, configurado y evaluado un clúster orientado al cálculo científico, ofreciendo también alta disponibilidad. Para ello, el sistema incorpora dos nodos directores que garantizan el funcionamiento continuo del sistema, así como mecanismos que gestionan la ejecución de las aplicaciones de los usuarios y el almacenamiento de sus datos. Todo esto se ha realizado en un entorno de máquinas virtuales, utilizando el sistema operativo Ubuntu.

A lo largo de este trabajo podremos ver cómo ha sido el proceso de la instalación y la configuración del sistema, así como de las tecnologías utilizadas para conseguir nuestro propósito.

Por último evaluaremos el sistema mediante una serie de pruebas y reflexionaremos sobre los resultados.

## 2 Definición de la estructura del clúster

El clúster que vamos a crear en este proyecto va a estar compuesto de 10 máquinas virtuales con Ubuntu Server 14.04.3 de 64 bits funcionando bajo la versión 5.0.20 del software de virtualización VirtualBox.

El equipo con el que se ha trabajado para el montaje del sistema cuenta con las siguientes características:

- Windows 10 64 bits.
- 4 GB de memoria RAM.
- Procesador Intel i5.
- Tarjeta gráfica NVIDIA GeForce GT520M.

El propósito de nuestro clúster va a ser el cálculo científico y la alta disponibilidad, por lo que el sistema incorporará dos nodos directores para garantizar un funcionamiento continuo, así como mecanismos para gestionar la ejecución de las aplicaciones de los usuarios y el almacenamiento de sus datos.

En el siguiente diagrama podemos observar cómo va a ser la estructura del clúster que vamos a montar.

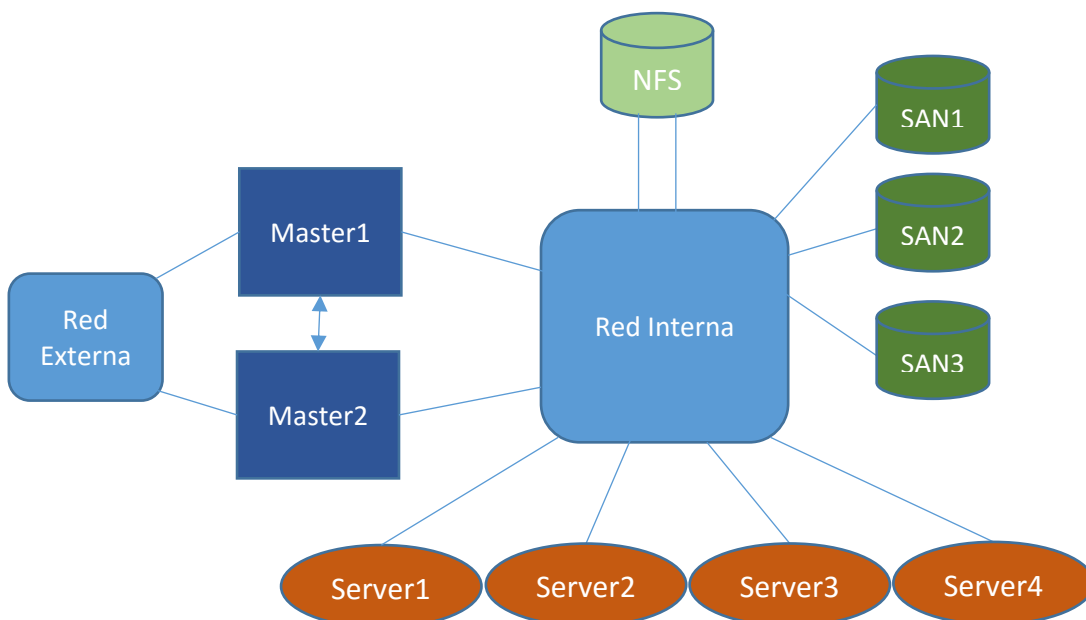


Figura 1: Estructura del clúster



## 2.1 Redes del clúster

El clúster cuenta con una **red interna** a través de la cual se comunican los nodos maestros y los servidores. Los nodos maestros también se comunican entre sí para monitorizar su funcionamiento. Asimismo, permite el acceso al servidor de almacenamiento.

La **red externa** permite el acceso de las máquinas del clúster a la Internet real. Este tipo de red también permite la comunicación entre las máquinas virtuales y la máquina anfitrión, y viceversa.

Como el nodo maestro va a estar replicado para que en caso de fallo otro nodo pueda encargarse de las tareas del maestro, utilizaremos un par de direcciones IP virtuales, una por cada interfaz de red, de manera que el nodo que este activo será el que las tendrá configuradas.

El servidor NFS cuenta con dos interfaces de red para poder realizar **channel bonding**, que consiste en conectar las dos interfaces para incrementar el ancho de banda.

En la siguiente tabla podemos ver un resumen de cómo quedará la distribución de las direcciones IP en nuestro clúster.

	Nombre	Dirección IP	
Director Master	cluster1 (red externa)	DIP: 192.168.1.101	VIP: 192.168.1.200
	cluster1 (red interna)	DIP: 10.0.100.1	VIP: 10.0.100.200
Servidores Reales	cluster2	DIP: 10.0.100.2	
	cluster3	DIP: 10.0.100.3	
	cluster4	DIP: 10.0.100.4	
	cluster5	DIP: 10.0.100.5	
Director Master2	master2 (red externa)	DIP: 192.168.1.102	VIP: 192.168.1.200
	master2 (red interna)	DIP: 10.0.100.50	VIP: 100.0.100.200
Servidores Almacenamiento	NFS	DIP: 10.0.100.100	
	SAN1	DIP: 10.0.100.6	
	SAN2	DIP: 10.0.100.7	
	SAN3	DIP: 10.0.100.8	

Tabla 1: Distribución direcciones IP

## 2.2 Nodos maestro

### Master1 y Master2

Son los nodos más importantes del clúster. Están duplicados para conseguir alta disponibilidad y forman el *front-end* del sistema. Además de las funciones propias del nodo maestro de un clúster, también se ocupará de distribuir la carga, repartiendo las peticiones SSH recibidas entre los servidores reales del *back-end*.

Inicialmente, el maestro activo es master1. Cuando el nodo master2 detecta la caída de master1 o del proceso de distribución de carga que allí se ejecuta, toma el relevo. Para este propósito instalaremos y configuraremos el software **HAProxy** y **Keepalived** en los nodos maestros.

Cuentan con las siguientes características:

- Sistema operativo: Linux Ubuntu 64 bits.
- Memoria 512 MB.
- Almacenamiento:
  - 1 disco SATA de 8GB.
- Interfaces de red:
  - Red NAT.
  - Red interna.

## 2.3 Nodos servidor

### Server1, Server2, Server3 y Server4

Son los nodos encargados de realizar las tareas de cómputo del clúster, por lo que solo van a tener instalado lo imprescindible para realizar este propósito. Principalmente instalaremos **MOSIX** y **Condor**, que se trata del software que se encarga de la gestión de tareas de usuario.

Cuentan con las siguientes características:

- Sistema operativo: Linux Ubuntu 64 bits.
- Memoria 256 MB.
- Almacenamiento:
  - 1 disco SATA de 8GB.
- Interfaces de red:
  - Red interna.

## 2.4 Nodo NFS

Se trata de un nodo de almacenamiento que incluye un sistema de archivos en red, por lo que cualquier otro nodo podrá acceder a una partición del disco que será compartida entre todos.

Como ya hemos comentado anteriormente, cuenta con dos interfaces de red para realizar una conexión **channel bonding** que le permitirá doblar el ancho de banda.

Cuenta con las siguientes características:

- Sistema operativo: Linux Ubuntu 64 bits.
- Memoria 256 MB.
- Almacenamiento:
  - 1 disco SATA de 10GB.
- Interfaces de red:
  - 2 interfaces de red interna.

## 2.5 Nodos SAN

SAN1, SAN2 y SAN3

Son los nodos que forman la red de área de almacenamiento. Se trata de nodos dedicados al almacenamiento, por lo que contarán con dos discos adicionales que se utilizarán para el montaje de la red de almacenamiento mediante el sistema **GlusterFS**.

Cuentan con las siguientes características:

- Sistema operativo: Linux Ubuntu 64 bits.
- Memoria 128 MB.
- Almacenamiento:
  - 1 disco SATA de 8GB.
  - 2 discos SATA de 1 GB.
- Interfaces de red:
  - Red interna.

## 3 Software utilizado

En este capítulo vamos a describir el software o las tecnologías que hemos empleado en nuestro proyecto, gracias a las cuales ha sido posible el correcto funcionamiento del clúster.

### 3.1 dnsmasq

El paquete dnsmasq permite poner en marcha un servidor **DNS** y un servidor **DHCP** de una forma muy sencilla: simplemente instalando y arrancando el servicio dnsmasq, sin realizar ningún tipo de configuración adicional, nuestro PC se convertirá en un servidor caché DNS y además, resolverá los nombres que tengamos configurados en el archivo */etc/hosts* de nuestro servidor. (1)

Adicionalmente, dnsmasq dispone de servidor DHCP y permite resolver los nombres de los PCs a los que les ha asignado dirección IP dinámica. Es posible configurar el servidor DHCP añadiendo simplemente una única línea al archivo de configuración, para indicar el rango de cesión.

DHCP (Dynamic Host Configuration Protocol, protocolo de configuración de host dinámico), permite que un equipo conectado a una red pueda obtener su configuración de red en forma dinámica, a través de la propia red a la que se encuentra conectado.

Los objetivos principales de DHCP son simplificar la administración de la red, evitar errores respecto a la configuración IP e incluso disminuir el desperdicio de direcciones IP en la red.

En nuestro caso, la función de servidor DHCP y DNS la realizará el nodo maestro, con la instalación del paquete dnsmasq.

### 3.2 PXE

El entorno de ejecución de prearranque de Linux (PXE) se puede utilizar para arrancar el servidor desde una interfaz de red en lugar de un almacenamiento local. Para instalar el sistema operativo, es lo mismo arrancar el servidor de destino desde una imagen de distribución del SO basado en PXE que arrancarlo desde un DVD, salvo que en el primer caso el medio de instalación se encuentra en la red. Para utilizar PXE, se debe configurar la infraestructura de red necesaria:

- Un servidor de DHCP que ejecute Linux y configurado para el arranque de PXE. En el ejemplo que aparece en este apartado, el servidor DHCP también será el servidor PXE.

- Servidor TFTP que admite el arranque de PXE. Las imágenes de arranque de PXE estarán ubicadas en el servidor TFTP. En el ejemplo que aparece en este apartado, el servidor DHCP actuará como servidor PXE con la ejecución de TFTP en el mismo como servicio.
- Utilidad PXELINUX instalada en el servidor PXE.
- Imagen de PXE en el servidor PXE. La imagen será el medio de instalación de distribución del SO de Linux utilizado para realizar una instalación remota del SO en el cliente PXE.
- Cliente PXE (también llamado "sistema de destino") con una tarjeta de interfaz de red que admite el arranque desde la red. El cliente se arrancará a través de la red mediante una imagen PXE. (2)

En nuestro trabajo instalaremos el servidor PXE en el nodo maestro y lo utilizaremos para cargar un entorno de ejecución de prearranque que nos permita replicar el sistema en cada uno de los nodos servidores

### 3.3 Servidor NFS

**Network File System** (Sistema de archivos de red), o NFS, es un protocolo de nivel de aplicación. Es utilizado para sistemas de archivos distribuidos en un entorno de red de ordenadores de área local. Posibilita que distintos sistemas conectados a una misma red accedan a ficheros remotos como si se tratara de locales. Originalmente fue desarrollado en 1984 por Sun Microsystems, con el objetivo de que sea independiente de la máquina, el sistema operativo y el protocolo de transporte. El protocolo NFS está incluido por defecto en los Sistemas Operativos UNIX y la mayoría de distribuciones Linux.

El sistema NFS está dividido al menos en dos partes principales: un servidor y uno o más clientes. Los clientes acceden de forma remota a los datos que se encuentran almacenados en el servidor. Entre las ventajas de utilizar este tipo de sistemas de archivos podemos citar:

- Las estaciones de trabajo locales utilizan menos espacio de disco debido a que los datos se encuentran centralizados en un único lugar pero pueden ser accedidos y modificados por varios usuarios, de tal forma que no es necesario replicar la información.
- Los usuarios no necesitan disponer de un directorio "home" en cada una de las máquinas de la organización. Los directorios "home" pueden crearse en el servidor de NFS para posteriormente poder acceder a ellos desde cualquier máquina a través de la infraestructura de red. (3)

Todas las operaciones sobre ficheros son síncronas. Esto significa que la operación sólo retorna cuando el servidor ha completado todo el trabajo asociado para esa operación. En caso de una solicitud de escritura, el servidor escribirá físicamente los datos en el disco, y si es necesario, actualizará la estructura de directorios, antes de devolver una respuesta al cliente. Esto garantiza la integridad de los ficheros.

En nuestro sistema el servidor NFS nos permitirá que los usuarios dispongan del directorio home accesible desde todas las máquinas así como nos servirá de ayuda para realizar la clonación de las particiones en los servidores.

### 3.4 NIS

**Network Information Service** (conocido por su acrónimo NIS, que en español significa **Servicio de Información de Red**), es el nombre de un protocolo de servicios de directorios cliente-servidor desarrollado por Sun Microsystems para el envío de datos de configuración en sistemas distribuidos tales como nombres de usuarios y hosts entre computadoras sobre una red.

Se trata de un sistema cliente servidor basado en llamadas RPC que permite a un grupo de máquinas que se encuentran definidas dentro de un dominio administrativo NIS compartir un conjunto de ficheros de configuración. Esto permite al administrador de sistemas por un lado configurar clientes NIS de forma minimalista y por otro lado centralizar la gestión de los ficheros de configuración en una única ubicación (una sola máquina). (4)

Existen tres tipos de máquinas dentro del entorno NIS: los servidores maestros, los servidores esclavos y los clientes de NIS. Los servidores actúan como repositorios centrales para almacenamiento de información de configuración. Los servidores maestros mantienen una copia maestra de dicha información, mientras que los servidores esclavos mantienen copias de la información maestra por motivos de redundancia. Los servidores se encargan de transmitir la información necesaria a los clientes a petición de estos últimos.

De esta forma se puede compartir mucha información contenida en varios archivos. Los ficheros *master.passwd*, *group* y *hosts* normalmente se comparten a través de NIS. Siempre que un proceso en un cliente necesita información que, en caso de no utilizar NIS, se podría recuperar de ficheros locales, en este caso se envía una solicitud al servidor NIS con el que nos encontramos asociados.

### 3.5 GlusterFS

Un inconveniente del sistema NFS explicado anteriormente es que no es escalable. Para un número de clientes relativamente alto, puede convertirse en un cuello de botella. Por ello hemos añadido a nuestro clúster una red de almacenamiento.

GlusterFS se trata de un sistema multiescalable de archivos para NAS desarrollado inicialmente por Gluster Inc. Permite agregar varios servidores de archivos sobre Ethernet o interconexiones Infiniband RDMA en un gran entorno de archivos de red en paralelo. El diseño del GlusterFS se basa en la utilización del espacio de usuario y de esta manera no compromete el rendimiento. Se utiliza en una gran variedad de entornos y aplicaciones como computación en la nube, ciencias biomédicas y almacenamiento de archivos.

GlusterFS se basa en la interacción de componentes cliente y servidor. Los servidores normalmente se implementan como almacenamiento en bloques. En cada servidor el proceso daemon `glusterfsd` exporta un sistema de archivos local como un volumen. El proceso cliente `glusterfs`, se conecta a los servidores a través de algún protocolo TCP/IP, InfiniBand o SDP, compone volúmenes virtuales a partir de los múltiples servidores remotos, mediante el uso de traductores. Por defecto, los archivos son almacenados enteros, pero también puede configurarse que se fragmente en múltiples porciones en cada servidor.

Algunas de las características más destacadas de GlusterFS son:

- Espejado y replicación de archivos.
- Fragmentación de los archivos o Data striping.
- Equilibrado de carga para la lectura y escritura de archivos.
- Volúmenes con tolerancia a fallos.
- Planificación de E/S y almacenamiento en caché de disco.
- Cuotas de almacenamiento

Los volúmenes pueden ser agregados, eliminados o migrados en forma dinámica. Esto ayuda a prever problemas de consistencia, y permite que GlusterFS pueda ser escalado a varios petabytes sobre hardware de bajo coste, evitando así los cuellos de botella que normalmente afectan a muchos sistemas de archivos distribuidos con múltiple concurrencia. (5)

### 3.6 MOSIX

Mosix es un paquete de software diseñado para añadir a Linux la capacidad de procesamiento clúster. Éste incluye balanceo de carga, *memory ushering* (subsistema que se encarga de migrar las tareas que superan la memoria disponible en el nodo en el

que se ejecutan) y algoritmos de optimización de E/S que responden a las variaciones del uso de los recursos del clúster, trabaja silenciosamente y sus operaciones son transparentes a las aplicaciones. (6)

Los usuarios que usan Mosix para lanzar las tareas pueden ejecutarlas tanto en secuencial como en paralelo, pero no conocen donde se ejecutan sus aplicaciones y no son conscientes de lo que otros usuarios están haciendo.

Cuando se crea un proceso Mosix, el sistema intenta asignar el proceso al nodo menos cargado en ese instante de tiempo. Para conseguir la migración de procesos realiza una monitorización de todos los procesos sin que afecte al funcionamiento de Linux.

Presenta una serie de ventajas como:

- No se requieren paquetes extra.
- No son necesarias modificaciones en el código.

Pero también tiene desventajas:

- Es dependiente del kernel.
- No migra todos los procesos siempre, tiene limitaciones de funcionamiento.
- Problemas con memoria compartida.

### 3.7 Condor

Es un proyecto de la Universidad de Wisconsin-Madison. Está ideado para facilitar la utilización de un clúster de computadores. Es un sistema que permite gestionar la ejecución de múltiples trabajos sobre un clúster. Sin la ayuda de este sistema, habría que lanzar manualmente cada trabajo sobre cada uno de los nodos servidores del clúster.

Condor nos permite ejecutar nuestros trabajos en tantas máquinas como haya disponibles, poniendo así a nuestra disposición toda la capacidad de cálculo del clúster. Nos será útil siempre que necesitemos ejecutar múltiples trabajos sobre el clúster. (7)

Además, nos permite:

- Conocer el estado de nuestros trabajos en cada momento Implementar nuestras propias políticas de orden de ejecución
- Mantener un registro de la actividad de nuestros trabajos
- Anadir tolerancia a fallos en la ejecución trabajos

La tolerancia a fallos la consigue a través de un mecanismo de **checkpoints** que le permite reanudar la ejecución de una tarea aunque esta se haya cerrado de forma forzosa por algún problema del sistema.



En nuestro clúster configuraremos un servidor de checkpoints para que los archivos temporales creados se guarden en un servidor de almacenamiento.

### 3.8 Keepalived

Keepalived nos ofrece una solución de alta disponibilidad mediante el uso del protocolo VRRP. Este protocolo, permite utilizar una dirección IP "virtual" contra el que van dirigidas las peticiones, enrutando las peticiones sobre uno de los nodos físicos que prestan servicio, de manera totalmente transparente para el usuario. En caso de caída del nodo, se negocia el paso del servicio a otro nodo sin que se aprecie pérdida de servicio. Keepalived también nos proporciona equilibrado de carga a nivel de transporte basado en LVS. (8)

El punto fuerte de Keepalived es posiblemente su sencillez. La configuración se basa únicamente en un archivo de configuración (keepalived.conf) donde se incluyen todas las opciones necesarias para su funcionamiento, y los scripts de arranque y parada típicos.

### 3.9 HAProxy

HAProxy es una herramienta gratuita, rápida y fiable y que ofrece a los usuarios un proxy TCP y HTTP de alta disponibilidad con control de equilibrado de carga a nivel de aplicación. Este tipo de tecnología es imprescindible para los sitios web con una alta carga de proceso o que generan un gran tráfico. Esta herramienta está catalogada como "código abierto" y cada vez es más utilizada en todo tipo de servidores Linux. (8)

Gracias a la combinación de HAProxy y Keepalived en nuestro clúster hemos conseguido ofrecer un sistema de alta disponibilidad mediante la introducción de dos servidores maestros y distribución de la carga de las peticiones ssh que hacen los usuarios para trabajar en el clúster.

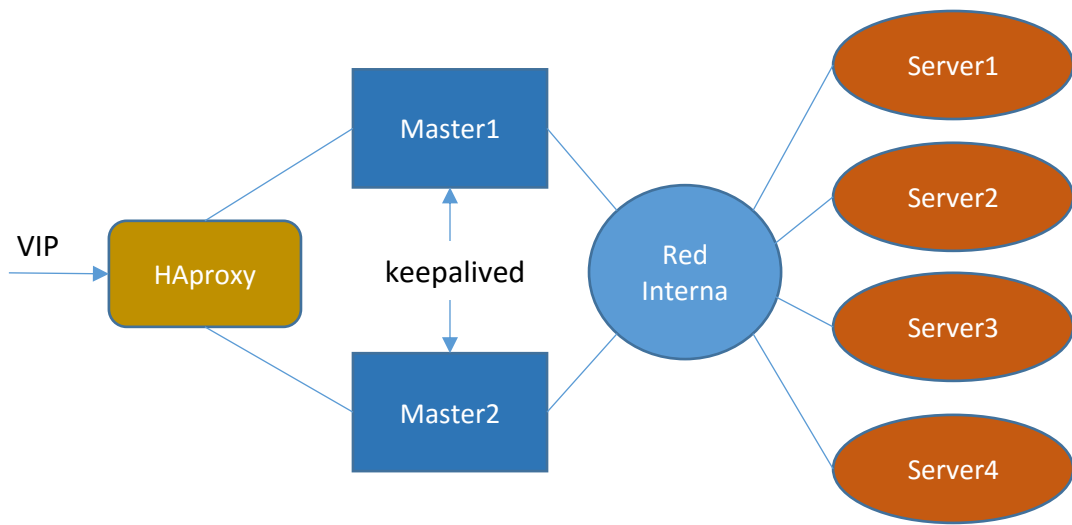


Figura 2: Esquema Keepalived y HAProxy

## 4 Instalación y configuración del clúster

### 4.1 Creación de las máquinas virtuales

La creación de las máquinas virtuales con VirtualBox (9) es muy sencilla, solo consiste en pulsar en el botón “nuevo” y seleccionar el tipo de máquina virtual que queremos crear, indicándole el nombre que queremos.

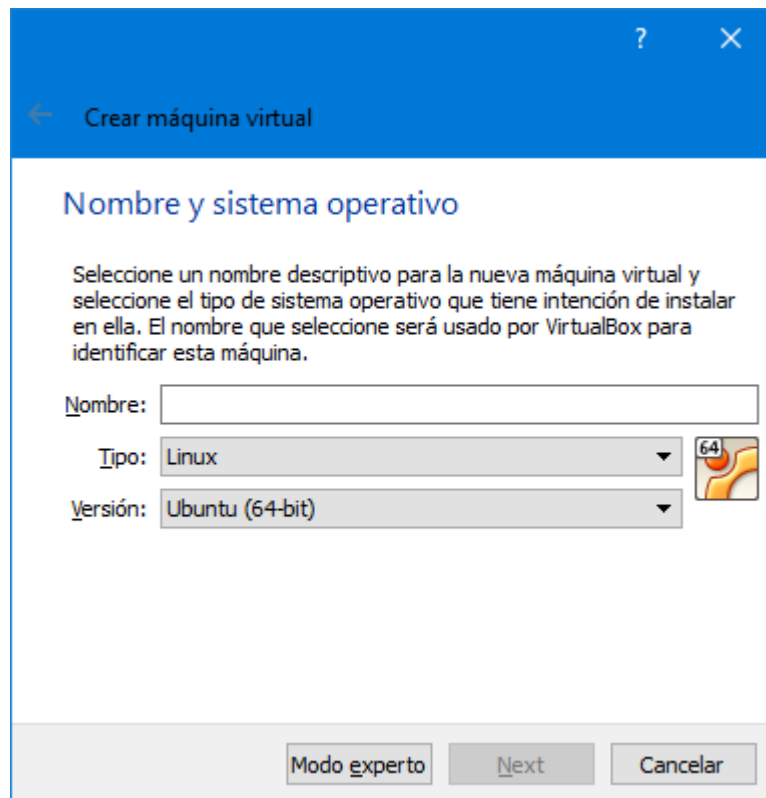


Imagen 1: Creación máquina virtual

Luego nos pedirá que seleccionemos el tamaño del disco y la memoria RAM que le vamos a asignar, que lo haremos acorde a las características indicadas en el apartado anterior de cada una de las máquinas.

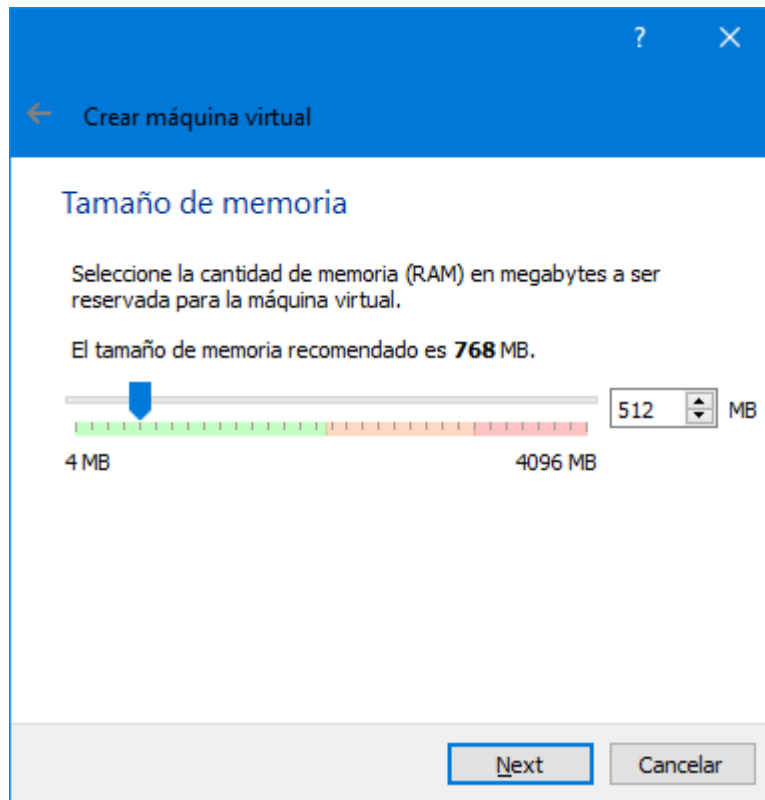


Imagen 2: Tamaño de la memoria

Por último, solo queda entrar en la configuración de cada una de las máquinas y ajustar las interfaces de red según lo especificado en la estructura de nuestro clúster.

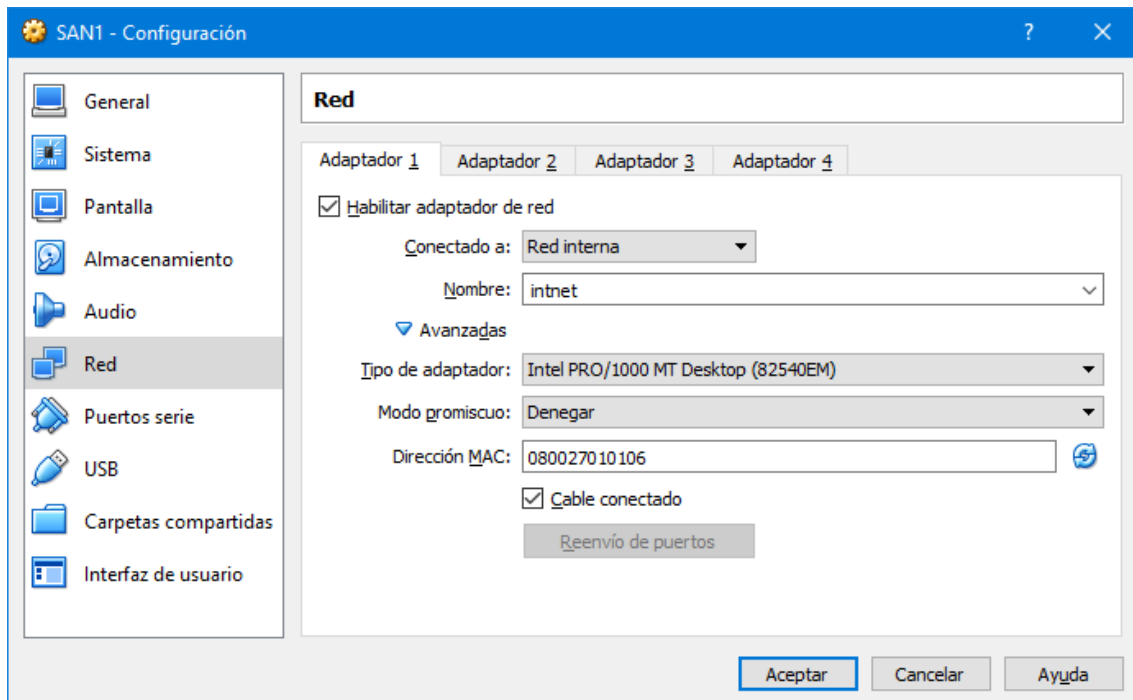


Imagen 3: Configuración del adaptador de red

## 4.2 Instalación del sistema operativo

Una vez creadas las máquinas el siguiente paso es instalar el sistema operativo, lo instalaremos tanto en el servidor de almacenamiento NFS como en el nodo maestro. En nuestro caso hemos utilizado la distribución *Ubuntu 14.04.3 LTS Server Edition* de 64 bits que se puede conseguir en la página oficial de Ubuntu. (10)

De manera general, el proceso de instalación que hemos seguido consiste en lo siguiente:

1. Instalación del Sistema Operativo en el nodo NFS.
2. Instalación del Sistema Operativo en el nodo Master.
3. Instalación, configuración y puesta en marcha del Servidor PXE en el nodo Master.
4. Inicio de los servidores de cómputo (server1 al server4) mediante la red (PXE).
5. Clonación de la partición principal del nodo Master en la partición principal de los nodos Esclavos.
6. Configuración adicional en el nodo Master y los nodos servidores.
7. Apagado del Servidor PXE y arranque del servidor en modo DHCP.
8. Reinicio de los nodos servidores.

Con la imagen ya descargada, la cargamos en la unidad virtual de la máquina para que arranque desde la imagen y poder realizar la instalación del sistema operativo.

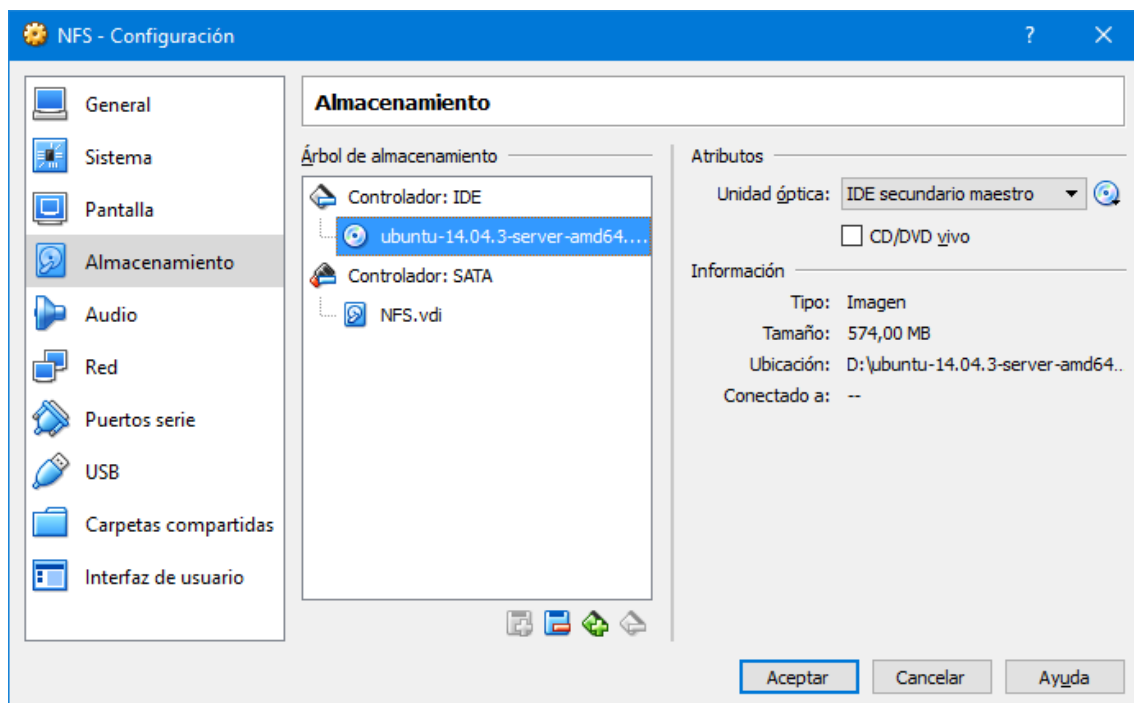


Imagen 4: Selección de la ISO del SO

Primero empezaremos instalando el sistema operativo en el nodo NFS.

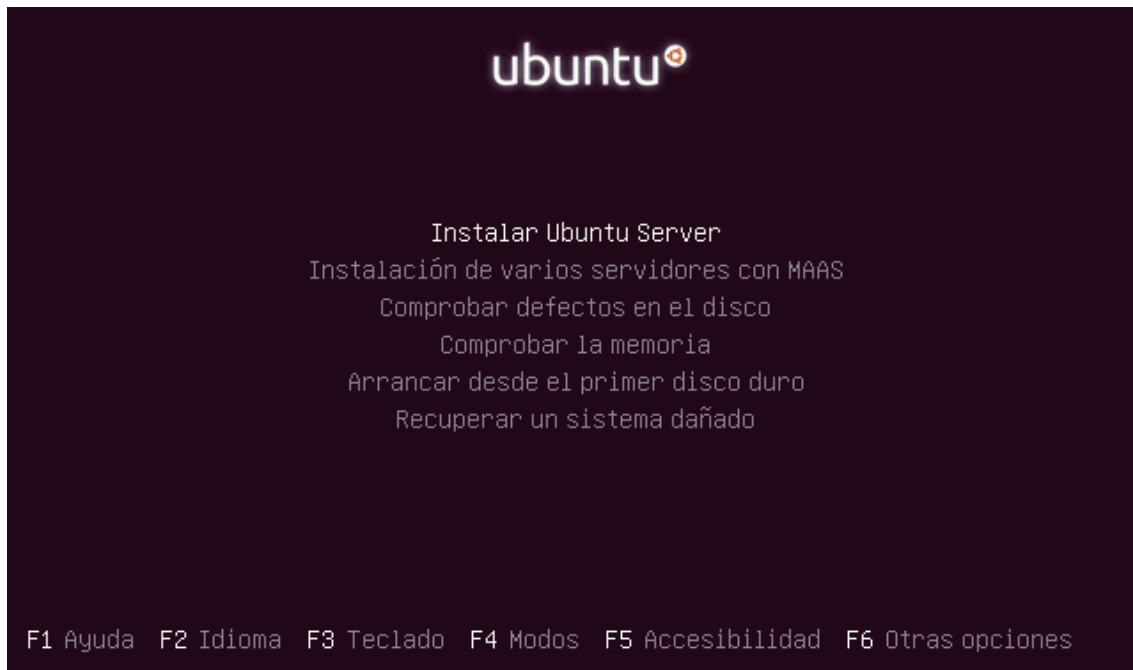


Imagen 5: Comienzo instalación Ubuntu

Tras pulsar en “Instalar Ubuntu Server” nos aparecerán una serie de preguntas relativas al idioma que tendremos que ir seleccionando las opciones deseadas.

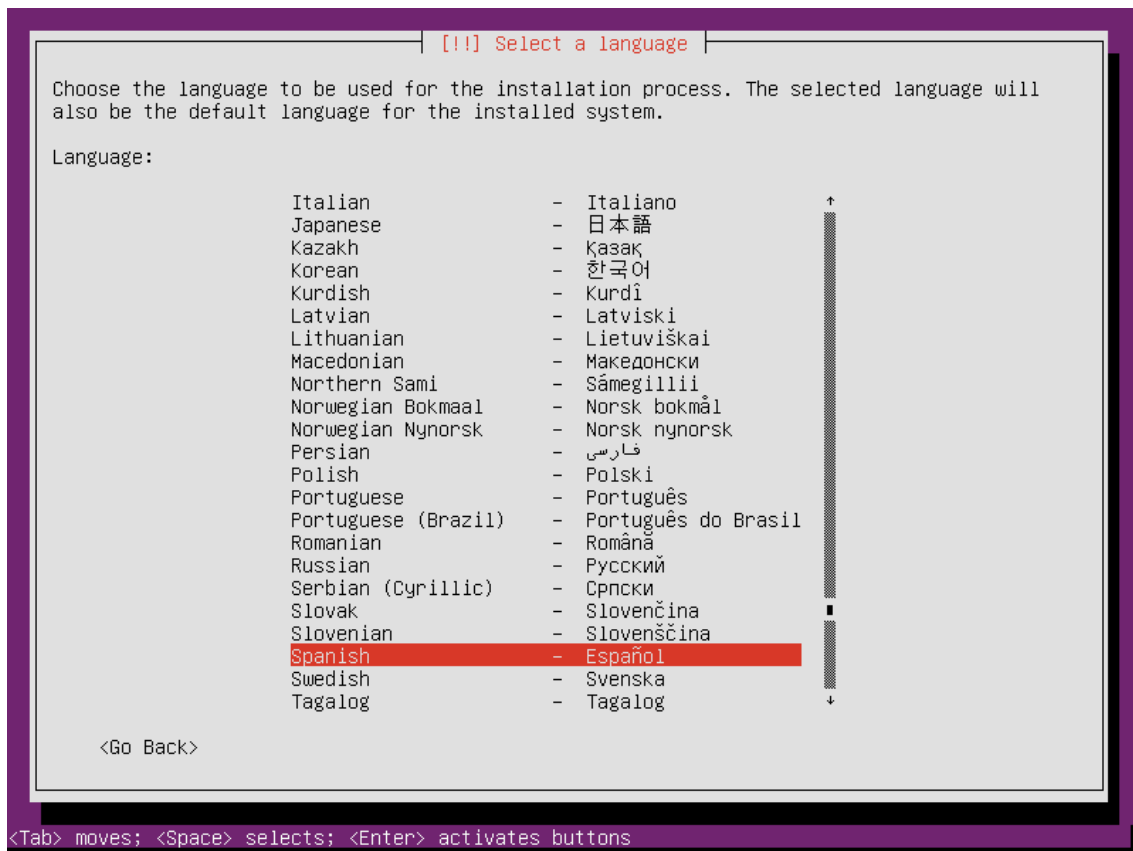


Imagen 6: Selección del idioma

Luego elegiremos un nombre para la máquina, en este caso elegimos nas y crearemos un usuario por defecto y configuramos las particiones del sistema.

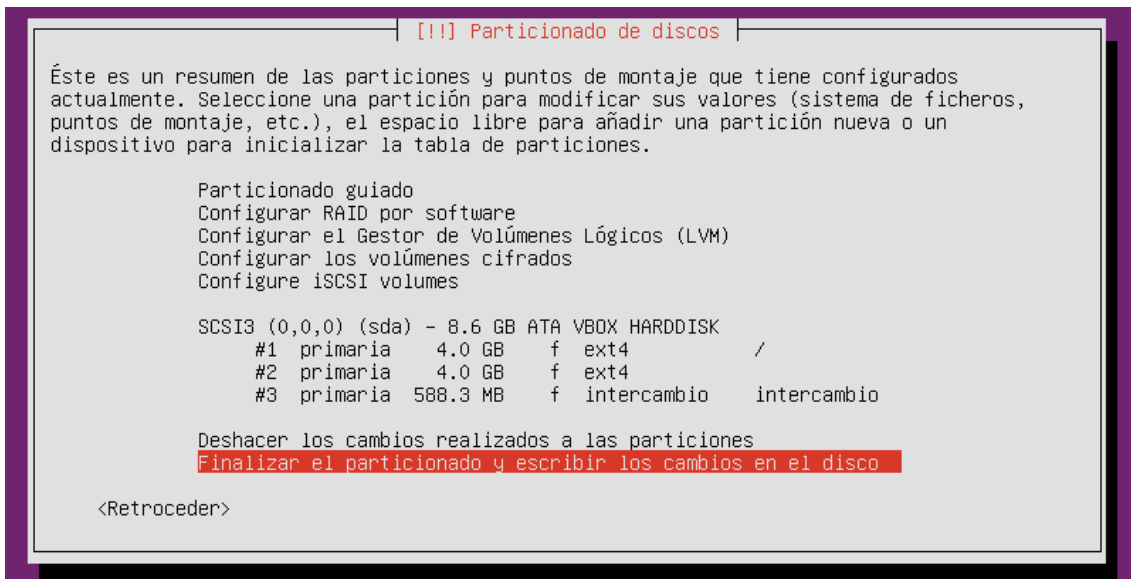


Imagen 7: Particionado del disco

Y por último indicamos que queremos instalar OpenSSH-server



Imagen 8: Instalación OpenSSH

Iniciamos sesión con el nombre de usuario elegido durante la instalación. Seguidamente, activaremos el usuario root mediante la orden:

```
sudo passwd root
```

Indicando la contraseña deseada. Cerraremos la sesión actual e iniciaremos la sesión como root.

Ahora realizaremos algunos ajustes en el sistema recién instalado.

Primero instalaremos algunos paquetes adicionales:

```
apt-get update
apt-get install rpcbind nfs-kernel-server
```

Configuramos el servidor ssh para que acepte conexiones remotas a la cuenta root. Para ello modificamos el archivo de configuración `/etc/ssh/sshd_config`:

```
# Authentication:
LoginGraceTime 120
# PermitRootLogin without-password
PermitRootLogin yes
StrictModes yes
```

Imagen 9: `/etc/ssh/sshd_config`

Y lo reiniciamos para que los cambios tengan efecto:

```
service ssh restart
```

Ahora vamos a configurar la red. Recordemos que el servidor NFS tiene 2 interfaces de red para realizar *channel bonding*, por lo que el archivo `/etc/network/interfaces` lo configuramos de la siguiente manera:

```
GNU nano 2.2.6 Archivo: /etc/network/interfaces
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual
bond-master bond0

auto eth1
iface eth1 inet manual
bond-master bond0

auto bond0
iface bond0 inet static
    address 10.0.100.100
    netmask 255.255.255.0
    gateway 10.0.100.1
    mtu 9000
    dns-nameservers 10.0.100.1
    bond-mode 4
    bond-miimon 100
    bond-lacp-rate 1
    bond-slaves eth0 eth1
```

Imagen 10: `/etc/network/interfaces`

Utilizamos el **bond-mode 4** para que el *channel bonding* funcione en modo **link aggregation**, como ya mencionamos en la planificación de nuestro clúster. (11)

Ya configurada la red, ahora procedemos a configurar el servidor de archivos NFS. Se desea exportar el directorio `/home` con la apariencia del directorio `/nfs` para las máquinas de la red interna (10.0.100.0/24), y con permisos de lectura/escritura (rw).



Para ello añadimos las siguientes líneas en el archivo `/etc/fstab`:

```
GNU nano 2.2.6 Archivo: /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point> <type> <options> <dump> <pass>
# / was on /dev/sda1 during installation
UUID=febcae86-713a-400a-a0cf-6660df175c9e / ext4 errors=remoun$
# /home was on /dev/sda2 during installation
UUID=e150a23a-9a53-4d1a-878f-355b38b5a392 /home ext4 defaults $
# swap was on /dev/sda3 during installation
UUID=f4cbc0a9-ad3d-47a8-a7df-9ddb6f3424c1 none swap sw $
/home /nfs none bind 0 0
```

Imagen 11: `/etc/fstab`

Y la siguiente línea en el archivo `/etc/exports`:

```
GNU nano 2.2.6 Archivo: /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
# to NFS clients. See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_sub$
#
# Example for NFSv4:
# /srv/nfs4 gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
#
/nfs 10.0.100.0/24(fsid=0,rw,sync,no_subtree_check,no_root_squash)
```

Imagen 12: `/etc/exports`

Luego procederemos a montarlo manualmente.

```
mkdir /nfs
mount /nfs
```

Finalmente iniciaremos el servidor NFS.

```
/etc/init.d/nfs-kernel-server restart
```

Una vez acabada la instalación y configuración básica del NFS, realizaremos la instalación del nodo maestro.

Para la instalación de Ubuntu en el nodo maestro solo hay que repetir el mismo proceso que hemos realizado con el servidor NFS pero ajustando los parámetros de almacenamiento propuestos para el nodo maestro.

Las configuraciones específicas las detallaremos en el apartado de clonación de los servidores ya que la configuración del nodo maestro va a ser la base del sistema que se va a clonar en los demás nodos servidores.

### 4.3 Clonación de los servidores

En este apartado vamos a exponer el proceso seguido para la preparación y la clonación de la imagen desde el nodo maestro a los nodos servidores.

#### 4.3.1 Scripts para facilitar la clonación

Antes de empezar el proceso de clonar el sistema operativo en los distintos nodos hemos realizado unos scripts que nos van a ayudar en la tarea de instalar, configurar y mantener los nodos de cómputo del clúster.

Estos scripts nos permiten lanzar órdenes y copiar ficheros mediante ssh desde el nodo Master hacia los nodos servidores.

##### *psh*

Este script nos sirve para lanzar órdenes a todos los nodos servidor del clúster, lo hace de forma secuencial.

```
echo "psh $1"
echo "======"
for i in 2 3 4 5
do echo cluster$i
echo "-----"
ssh cluster$i $1
echo "======"
done
```

##### *ppsh*

Tiene la misma función que el script anterior pero para lanzar las órdenes de forma paralela, muy útil cuando se requiera hacer alguna operación que pueda tardar cierto tiempo en terminar.

```
echo "ppsh $1"
for i in 2 3 4 5
do
ssh cluster$i $1 &
done
```

##### *pscp*

Sirve para copiar archivos desde un nodo al resto de servidores del clúster.

```
for i in 2 3 4 5
do echo cluster$i
scp $1 cluster$i:$2
done
```

## *npsh*

Se trata de una versión avanzada del script psh donde podemos lanzar una orden a determinados nodos del clúster, utiliza las siguientes opciones:

- m: Nodo maestro.
- a: Todos los nodos
- f x: Desde el nodo con índice x
- t y: Hasta el nodo con índice y.

```
while getopts "maf:t:" opt; do
case $opt in
m) ssh cluster1 $2 ;;
a) for i in 2 3 4 5
do echo cluster$i
ssh cluster$i $2
done
;;
f) from=$OPTARG ;;
t) to=$OPTARG
for ((x = $from ; x <= $to ; x++));
do
echo cluster$x
ssh cluster$x $5
done
;;
esac
done
```

### 4.3.2 Preparación de la imagen a clonar

Lo primero de todo es preparar la imagen para la clonación, para ello vamos a realizar las siguientes configuraciones.

Primero instalaremos el cargador de arranque GRUBv2 en el MBR de nuestro disco */dev/sda*

```
apt-get update
apt-get install grub2
grub-install /dev/sda
```

Y lo configuraremos para que no busque automáticamente los sistemas instalados, indicándole explícitamente los sistemas a arrancar, para ello desactivamos la ejecución de algunos scripts.

```
chmod -x /etc/grub.d/10_linux
chmod -x /etc/grub.d/20_memtest86+
chmod -x /etc/grub.d/30_os-prober
```

Por último modificaremos el archivo */etc/grub.d/40\_custom* para indicarle las opciones de inicio deseadas.

```
GNU nano 2.2.6 Archivo: /etc/grub.d/40_custom Modificado
#!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries.  Simply type the
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.

menuentry "Ubuntu-Part1" {
    set root=(hd0,1)
    linux /vmlinuz root=/dev/sda1
    initrd /initrd.img
}_
```

Imagen 13: /etc/grub.d/40\_custom

Finalmente, tras modificar los archivos de configuración, para que los cambios surtan efecto, ejecutamos la orden:

```
update-grub
```

Configuramos la red del nodo maestro para que obtenga una dirección IP por DHCP (eth0) la interfaz de la red externa modificando el archivo [/etc/network/interfaces](#). También configuramos estáticamente la red interna (eth1).

```
GNU nano 2.2.6 Archivo: /etc/network/interfaces Modificado
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet static
    address 10.0.100.1
    netmask 255.255.255.0
    mtu 9000_
```

Imagen 14: /etc/network/interfaces DHCP

Generamos las claves y copiamos la clave pública del nodo maestro sobre el archivo que contiene las claves autorizadas, y que más tarde distribuiremos a los servidores:

```
ssh-keygen
cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys
```

Al generar las claves obtenemos una salida como la que se muestra a continuación.

```
root@cluster1:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
cf:36:eb:3b:39:cd:d5:7d:b9:d8:a9:48:38:1b:2e:87 root@cluster1
The key's randomart image is:
+--[ RSA 2048 ]-----+
|
|          S          .o|
|         o.         ..+|
|        .+==       .o +|
|       E.oOo+.    + |
|      o++=       .. |
+-----+
root@cluster1:~# cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys
root@cluster1:~#
```

Imagen 15: ssh-keygen

Y modificamos el archivo `/etc/hosts`, añadiendo las parejas dirección IP-nombre para la resolución de nombres.

```
127.0.0.1    localhost

10.0.100.100 nas.cluster          nas
10.0.100.1   cluster1.cluster    cluster1       master lb1
10.0.100.2   cluster2.cluster    cluster2       server1
10.0.100.3   cluster3.cluster    cluster3       server2
10.0.100.4   cluster4.cluster    cluster4       server3
10.0.100.5   cluster5.cluster    cluster5       server4

# The following lines are desirable for IPv6 capable hosts
::1        localhost ip6-localhost ip6-loopback
ff02::1   ip6-allnodes
ff02::2   ip6-allrouters
```

Imagen 16: /etc/hosts

Una vez hecho esto ya podemos copiar las claves y el archivo `/etc/hosts` al servidor NFS para poder acceder por ssh mediante clave pública.

```
ssh nas "mkdir /root/.ssh"
scp /root/.ssh/id_rsa.pub nas:/root/.ssh/authorized_keys
scp /etc/hosts nas:/etc
```

Por último vamos a instalar el paquete cliente NFS para poder acceder al almacenamiento del servidor NFS. Para ello tan solo hay que lanzar la orden:

```
apt-get install nfs-common
```

Y crear una entrada en el archivo [/etc/fstab](#) para que monte el directorio compartido automáticamente en el arranque, lo que haremos mediante la siguiente orden:

```
echo "nas:/nfs /nfs nfs auto,rsize=8192,wsiz=8192 0 0" >>
/etc/fstab
```

Ahora ya podremos montarlo manualmente

```
mkdir /nfs
mount /nfs
```

Una vez ya tenemos preparada la imagen del sistema que vamos a copiar en los servidores lo copiamos en el servidor NFS. Para ello, utilizaremos la orden **cp** con los parámetros **-a** para que mantenga usuarios, permisos y mantenga los enlaces simbólicos y **-x** para que no cruce los límites del sistema de archivos montado:

```
mkdir /nfs/srv
cp -ax / /nfs/srv/
```

Como ya hemos copiado la imagen base de la partición que va a ser replicada en los demás servidores vamos a realizar una serie de configuraciones en el nodo maestro para que realice el reenvío de paquetes entre la red externa y la interna de nuestro clúster, ya que va a ser la única salida al exterior de todo el sistema. Para ello, debemos activar el reenvío de paquetes en el nodo maestro (*ip forwarding*) y configurar correctamente el cortafuegos mediante iptables.

Esto se puede conseguir insertando la siguiente secuencia de órdenes en el archivo [/etc/rc.local](#), que se ejecuta automáticamente en el arranque:

```
sysctl -w net.ipv4.ip_forward=1
iptables -P FORWARD ACCEPT
iptables --table nat -A POSTROUTING -o eth0 -j MASQUERADE
exit 0
```

*Imagen 17: /etc/rc.local*

Lo que estamos haciendo con este comando de iptables es manipular la tabla NAT (*--table nat*), para que modifique los paquetes IP que deban salir del por el interfaz eth0 (*-o eth0*) cambiando su dirección IP origen por la que tenga la interfaz de salida (*-j MASQUERADE*). La sustitución debe realizarse tras decidir por dónde encaminar el paquete (*-A POSTROUTING*).

Una vez modificado lo ejecutamos manualmente, las próximas veces no será necesario ya que se lanza automáticamente en el arranque del sistema:

```
chmod +x /etc/rc.local
/etc/rc.local
```

Para terminar con la configuración del nodo maestro instalaremos el entorno gráfico lxde que es liviano y consume pocos recursos y hace más amigable el uso de los nodos maestros que serán la puerta de enlace para que los usuarios trabajen con el clúster.

```
apt-get install lxde
```

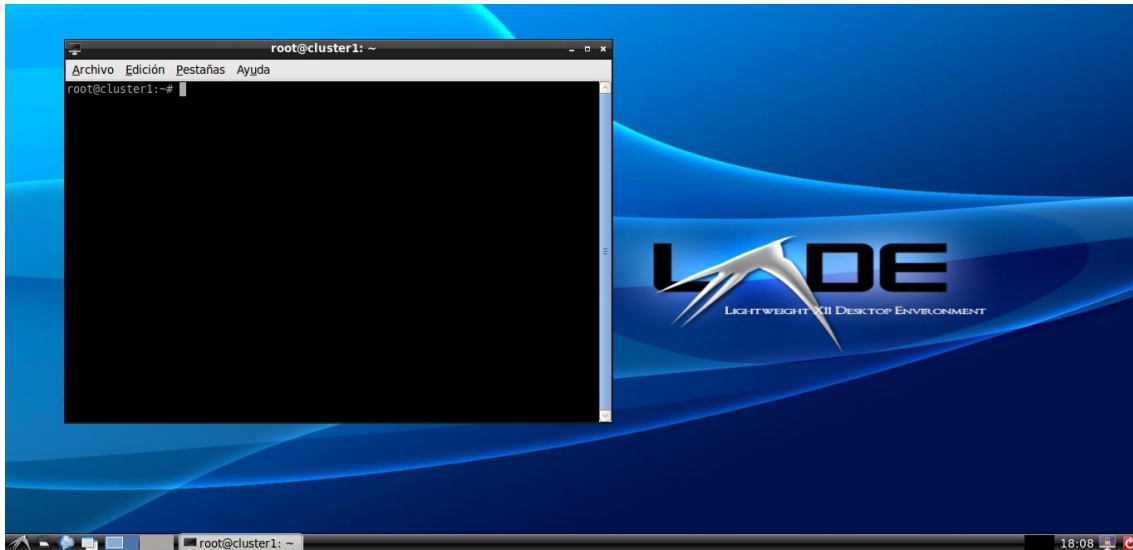


Imagen 18: LXDE

#### 4.3.3 Instalación del servidor PXE

Como ya hemos explicado anteriormente el servidor PXE nos permitirá que los nodos servidores puedan cargar mediante la red un entorno de ejecución de prearranque que nos permita replicar el sistema en cada una de las particiones primarias.

En primer lugar, instalaremos el paquete que contiene el cargador a través de red:

```
apt-get install syslinux
```

Y lo ubicaremos en la carpeta `/boot`

```
cp /usr/lib/syslinux/pxelinux.0 /boot/  
mkdir /boot/pxelinux.cfg
```

También generamos un archivo de configuración `/boot/pxelinux.cfg/default` que indique la ubicación de los archivos que contienen el núcleo y `ramdisk` inicial. Por otra parte, una vez arrancado por red, los nodos montarán su sistema de archivos raíz en el servidor NFS.

```
GNU nano 2.2.6 Archivo: /boot/pxelinux.cfg/default Modificado
DEFAULT Linux
LABEL Linux
KERNEL vmlinuz
APPEND root=/dev/nfs initrd=/initrd_netboot nfsroot=10.0.100.100:/nfs/srv ip=dhcp rw
```

Imagen 19: /boot/pxelinux.cfg/default

Seguidamente, prepararemos un *initial ramdisk* que monte el sistema de archivos raíz en el servidor NFS.

Para ello, editaremos el archivo `/etc/initramfs-tools/initramfs.conf` y buscaremos la línea que indica desde dónde se inicia el sistema (BOOT=), indicando que debe ser desde NFS. Previamente, sacaremos una copia:

```
cp /etc/initramfs-tools/initramfs.conf /etc/initramfs-
tools/initramfs.conf.m
```

Y el contenido del archivo debe ser el siguiente:

```
GNU nano 2.2.6 Archivo: /etc/initramfs-tools/initramfs.conf Modificado
COMPRESS=gzip
#
# NFS Section of the config.
#
#
# BOOT: [ local | nfs ]
#
# local - Boot off of local media (harddrive, USB stick).
#
# nfs - Boot using an NFS drive as the root of the drive.
#
BOOT=nfs
#
# DEVICE: ...
#
```

Imagen 20: /etc/initramfs-tools/initramfs.conf

Generamos un nuevo *initial ramdisk* con esta configuración, y, después, restauraremos el archivo de configuración inicial:

```
/usr/sbin/mkinitramfs -o /boot/initrd_netboot
cp /etc/initramfs-tools/initramfs.conf.m /etc/initramfs-
tools/initramfs.conf
ln -s /boot/vmlinuz-* /boot/vmlinuz
chmod 644 /boot/vmlinuz-*
```



Con estos comandos hemos puesto al *ramdisk* inicial el mismo nombre *initrd\_netboot* que utilizamos en el archivo de configuración de PXE */boot/pxelinux.cfg/default*, y también hemos generado un enlace simbólico al archivo del kernel con el mismo nombre que hemos utilizado en dicho archivo. Además hemos cambiado los permisos para que los nodos que arrancan por PXE puedan cargarlo .

### *Instalación dnsmasq*

Como ya mencionamos anteriormente utilizaremos la utilidad dnsmasq, que incorpora un servidor DHCP, un servidor DNS y un servidor tftp necesarios para el arranque por PXE. En primer lugar, instalaremos el paquete correspondiente, y detendremos el servicio para configurarlo:

```
apt-get install dnsmasq
/etc/init.d/dnsmasq stop
```

Vamos a preparar dos configuraciones. La primera, que utilizaremos en esta etapa de instalación del clúster, incorporará los servicios DNS, DHCP y PXE, mientras que la segunda sólo incorporará DNS y DHCP. Por lo tanto, crearemos dos archivos de configuración:

### */etc/dnsmasq-dhcp-pxe.conf*

```
enable-tftp
tftp-root=/boot
dhcp-boot=/pxelinux.0
dhcp-range=10.0.100.2,10.0.100.50,infinite
log-dhcp
dhcp-option=26,9000
dhcp-option=3,10.0.100.1
dhcp-option=6,10.0.100.1
interface=eth1
read-ethers
```

### */etc/dnsmasq-dhcp.conf*

```
dhcp-range=10.0.100.2,10.0.100.50,infinite
log-dhcp
dhcp-option=26,9000
dhcp-option=3,10.0.100.1
dhcp-option=6,10.0.100.1
interface=eth1
read-ethers
```

La opción `interface=eth1` indica que sólo debe escuchar y aceptar peticiones que lleguen por la red interna. Las opciones `dhcp-option=26, 3 y 6` fijan la mtu, la puerta de enlace y el servidor DNS, respectivamente. La opción `read-ethers` nos permite tener control sobre las direcciones IP que proporciona el servidor DHCP a través del archivo `/etc/ethers`, que incluye las parejas MAC (las de los servidores)-dirección IP para preasignar las direcciones IP.

En nuestro caso creamos el siguiente archivo `/etc/ethers`:

```
08:00:27:01:01:02 10.0.100.2
08:00:27:01:01:03 10.0.100.3
08:00:27:01:01:04 10.0.100.4
...
08:00:27:01:01:08 10.0.100.8
```

Para poder optar entre una y otra configuración no hay más que copiar el archivo deseado sobre el de configuración de dnsmasq `/etc/dnsmasq.conf`. Por lo que hemos creado los siguientes scripts para realizar dicha tarea.

`/root/start-dhcp-pxe`

```
echo "Iniciando dnsmasq en modo DHCP + PXE"
cp /etc/dnsmasq-dhcp-pxe.conf /etc/dnsmasq.conf
/etc/init.d/dnsmasq restart
```

`/root/start-dhcp`

```
echo "Iniciando dnsmasq en modo DHCP"
cp /etc/dnsmasq-dhcp.conf /etc/dnsmasq.conf
/etc/init.d/dnsmasq restart
```

Como en este momento estamos instalando, lo iniciaremos en modo DHCP + PXE:

```
./start-dhcp-pxe
```

Seguidamente, iniciaremos los servidores a clonar. Como los hemos configurado para que arranquen por red como primera opción y van a encontrar un servidor PXE activo, vemos cómo obtienen una dirección IP, el cargador (pxelinux.0), el kernel, etc.

```
ipXE 1.0.0+ -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS TFTP HTTP PXE PXEXT Menu

net0: 08:00:27:01:01:02 using 82540em on PCI00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
DHCP (net0 08:00:27:01:01:02)..... ok
net0: 10.0.100.2/255.255.255.0 gw 10.0.100.1
Next server: 10.0.100.1
Filename: /pxelinux.0
tftp://10.0.100.1/pxelinux.0... ok
!PXE entry point found (we hope) at 9CB7:03F0 via plan A
UNDI code segment at 9CB7 len 0694
UNDI data segment at 9D21 len 25F0
Getting cached packet 01 02 03
My IP address seems to be 0A006402 10.0.100.2
ip=10.0.100.2:10.0.100.1:10.0.100.1:255.255.255.0
BOOTIF=01-08-00-27-01-01-02
SYSUUID=60922055-8199-4008-89a2-2ca73423231f
TFTP prefix: /
Trying to load: pxelinux.cfg/default                               ok
Loading vmlinuz.....
Loading /initrd_netboot.....
```

Imagen 21: Carga mediante PXE

Una vez han arrancado todos los servidores habrán iniciado un sistema cuya raíz está compartida por todos y reside en el servidor NFS. Además, cada uno de ellos habrá obtenido la dirección IP que les habíamos asignado en el archivo ethers.

Como los discos de los servidores son idénticos al del maestro, vamos a volcar la tabla de particiones del nodo maestro en un archivo y lo propagaremos a los servidores. Para obtener una copia de la tabla de particiones del nodo maestro, teclearemos:

```
sfdisk -d /dev/sda > /nfs/srv/root/sda.out
```

Y Creamos la tabla de particiones en los servidores:

```
ppsh "sfdisk -f /dev/sda < /root/sda.out"
```

La opción `-f` de la orden `sfdisk` permite leer de un archivo la tabla de particiones a escribir en el disco.

También cabe destacar que, como los nodos servidores tienen todos montado su sistema raíz en el NFS, el directorio `/root` está compartido por todos ellos.

Formateamos la partición de arranque:

```
ppsh "mkfs -t ext4 /dev/sda1"
```

Nótese que utilizamos el script `ppsh`, para ejecutar la orden en todos los nodos simultáneamente.

Preparamos e iniciamos el área de intercambio (swap):

```
ppsh "mkswap /dev/sda3"
ppsh "swapon /dev/sda3"
```

Montamos los discos locales de los servidores y copiamos el sistema desde el raíz ubicado en el servidor NFS.

```
ppsh "mkdir /mnt/sda1"  
ppsh "mount /dev/sda1 /mnt/sda1"  
ppsh "cp -ax / /mnt/sda1"
```

Luego instalamos el cargador de arranque en los servidores. En cada uno de los servidores hay que ejecutar la orden:

```
psh "grub-install --root-directory=/mnt/sda1 /dev/sda"
```

La opción `--root-directory=/mnt/sda1` le indica a la aplicación `grub-install` dónde puede encontrar los archivos que contienen el cargador de arranque.

Y para regenerar correctamente los archivos de configuración, montaremos los directorios necesarios en la partición que estamos instalando:

```
psh "for i in /dev /dev/pts /proc /sys; do mount -B \${i} /mnt/sda1\${i}; done"
```

Ahora ejecutaremos las órdenes que consolidan la configuración, con la ayuda de `chroot`, que permite cambiar el punto de montaje del sistema de archivos raíz:

```
psh "chroot /mnt/sda1 dpkg-reconfigure grub2"  
psh "chroot /mnt/sda1 update-grub"
```

Asignamos nuevos nombres de maquina a los servidores. En cada nodo hay que generar el archivo `/etc/hostname` correspondiente.

```
ssh cluster2 "echo cluster2 > /mnt/sda1/etc/hostname"  
ssh cluster3 "echo cluster3 > /mnt/sda1/etc/hostname"  
ssh cluster4 "echo cluster4 > /mnt/sda1/etc/hostname"  
ssh cluster5 "echo cluster5 > /mnt/sda1/etc/hostname"
```

Lanzamos la utilidad `dnsmasq` en modo DHCP para que los nodos ya no arranquen por PXE:

```
start-dhcp
```

Y reiniciamos los servidores, desmontando previamente su disco:

```
psh "umount /mnt/sda1"  
psh reboot
```

Como hemos clonado el sistema desde el nodo maestro, todos los nodos tienen una copia de las claves del maestro (son réplicas del maestro). Podemos regenerarlas haciendo uso de la orden *dpkg-reconfigure* que vuelve a configurar un paquete después de su instalación:

```
psch "rm /etc/ssh/ssh_host_*; dpkg-reconfigure openssh-server"
```

Como consecuencia, hay que actualizar las claves públicas de los servidores en el nodo maestro. Para ello, las borramos y conforme lancemos ssh sobre los servidores se irán almacenando.

```
rm /root/.ssh/known_hosts
```

Y por fin después de este largo proceso de instalación y configuración tenemos nuestro clúster funcionando con los nodos servidores y de almacenamiento creados, pero todavía falta instalar y configurar el software necesario para que realicen las tareas para las que están previstos. Lo cual veremos en los siguientes apartados.

## 4.4 Sistema de almacenamiento para clústeres

Aunque ya tenemos un servidor NFS en nuestro clúster, cuya unidad de red puede montarse en cualquiera de los demás nodos y acceder a los archivos comunes, hemos decidido instalar un sistema más avanzado multiescalable para servidores de ficheros, **GlusterFS**.

Para ello hemos creado tres nodos nuevos, que actuarán como servidores de almacenamiento SAN. En concreto, crearemos dos unidades, una distribuida, donde los archivos se distribuyen de manera equitativa entre los tres nodos del servidor de almacenamiento, y una replicada, donde todos los archivos se replican en los tres nodos.

Estos nodos han seguido el mismo proceso de clonación que los nodos servidores por lo que tienen instalado un sistema Linux configurado y funcionando, simplemente procederemos a instalar y configurar glusterFS. (12)

### 4.4.1 GlusterFS

Cada nodo SAN tiene dos discos dedicados al servidor de ficheros por lo que el primer paso va a consistir en crear una partición en cada uno de ellos con la utilidad **fdisk**.

Seguidamente, los formatearemos mediante la orden **mkfs**. En este caso, utilizaremos el sistema de archivos **xfs**.

```
mkfs.xfs /dev/sdb1  
mkfs.xfs /dev/sdc1
```

Crearemos los puntos de montaje:

```
mkdir -p /export/brick1
mkdir -p /export/brick2
```

Y procedemos a montarlos. En este caso, cada máquina ofrecerá dos *bricks*, donde cada uno de ellos estará asociado con uno de los nuevos dispositivos.

```
mount /dev/sdc1 /export/brick1
mount /dev/sdd1 /export/brick2
```

Para consolidar la configuración, solo hay que crear entradas correspondientes en el archivo [/etc/fstab](#)

```
/dev/sda1      /          ext4      defaults    0          0
/dev/sda3      none       swap      auto,defaults 0          0
nas:/nfs       /nfs       nfs       auto,defaults 0          0
/dev/sdb1      /export/brick1 xfs      defaults    0          0
/dev/sdc1      /export/brick2 xfs      defaults    0          0
```

Imagen 22: /etc/fstab

Una vez montado, creamos el subdirectorio que contendrá los datos:

```
mkdir /export/brick1/data
mkdir /export/brick2/data
```

En este momento, cada máquina ofrece sendos bricks en las rutas [/export/brick1/data](#) y [/export/brick2/data](#)

Con la ayuda de la herramienta de instalación de paquetes de ubuntu, instalaremos el software necesario:

```
apt-get install glusterfs-server
```

Desde uno de los nodos del grupo, crearemos un volumen replicado con el disco de 1GB mediante la orden:

```
gluster volume create gv0 replica 3
san1:/export/brick1/data \san2:/export/brick1/data
san3:/export/brick1/data
```

Y lo iniciamos con:

```
gluster volume start gv0
```

También creamos otro volumen, en este caso distribuido, con el otro disco de 1GB mediante la orden:

```
gluster volume create gv1 san1:/export/brick2/data \
san2:/export/brick2/data san3:/export/brick2/data
```

Y lo iniciamos con:

```
gluster volume start gv1
```

Podemos obtener las características de los volúmenes disponibles con:

```
gluster volume info
```

```
root@san1:~#  
root@san1:~# gluster volume info  
  
Volume Name: gv1  
Type: Distribute  
Volume ID: dcd3735e-f6f0-4f79-aa9d-bf4781ce2de2  
Status: Started  
Number of Bricks: 3  
Transport-type: tcp  
Bricks:  
Brick1: san1:/export/brick2/data  
Brick2: san2:/export/brick2/data  
Brick3: san3:/export/brick2/data  
  
Volume Name: gv0  
Type: Replicate  
Volume ID: e0a59236-1e3c-499e-9fd5-7a2bcc8a40ea  
Status: Started  
Number of Bricks: 1 x 3 = 3  
Transport-type: tcp  
Bricks:  
Brick1: san1:/export/brick1/data  
Brick2: san2:/export/brick1/data  
Brick3: san3:/export/brick1/data  
root@san1:~# _
```

*Imagen 23: Información volúmenes gluster*

Los dos volúmenes glusterFS creados los montaremos en todos los nodos del clúster para que se puede acceder a ellos desde cualquier nodo. Para ello, tan solo tenemos que instalar el paquete:

```
apt-get install glusterfs-client
```

Ahora, crearemos sendos puntos de montaje:

```
mkdir -p /mnt/gfs-r  
mkdir -p /mnt/gfs-d
```

Y montamos los dos volúmenes, replicado y distribuido, respectivamente:

```
mount -t glusterfs san1:/gv0 /mnt/gfs-r  
mount -t glusterfs san1:/gv1 /mnt/gfs-d
```

Si queremos que se monte automáticamente en el arranque solo hay que añadirlo al archivo */etc/fstab*, y como hemos mencionado anteriormente este proceso hay que repetirlo en todos los nodos del clúster para que tengan acceso al almacenamiento del servidor SAN.

## 4.5 Sistema de alta disponibilidad en el nodo director

En un sistema informático la posibilidad de que algún nodo falle cuando menos lo esperamos siempre está presente, por eso es importante incluir un sistema que ofrezca alta disponibilidad en nuestro sistema.

Esto se consigue principalmente mediante la redundancia, de tal forma que si alguna máquina falla otra la pueda sustituir en sus funciones.

En nuestro caso la solución empleada ha consistido en agregar un nodo adicional al clúster que hemos llamado master2 y consiste en una réplica exacta del nodo Master para que pueda realizar las mismas funciones en caso de que el primero falle.

Para ello hemos empleado la combinación de software Keepalived y HAProxy. (13)

### 4.5.1 Keepalived

Lo primero que realizamos es instalar el paquete del software mediante la orden:

```
apt-get install keepalived
```

Luego solo hace falta crear el archivo de configuración, que quedara como se puede ver a continuación:

[/etc/keepalived/keepalived.conf](#)

```
vrp_script chk_haproxy {                # Requires keepalived-1.1.13
    #script "killall -0 haproxy"         # cheaper than pidof
    script "pidof haproxy"
    interval 2                          # check every 2 seconds
    weight 2                             # add 2 points of prio if OK
}

vrp_sync_group VG1 {
    group {
        VI_1
        VE_1
    }
}

vrp_instance VI_1 {
    interface eth1
    state MASTER                         # en el standby ponemos BACKUP
    virtual_router_id 52                 # 101 on master, 100 on backup
    priority 101
    virtual_ipaddress {
        10.0.100.200
    }
    track_script {
        chk_haproxy
    }
}
```



```

vrp_instance VE_1 {
    interface eth0
    state MASTER # en el standby ponemos BACKUP
    virtual_router_id 51
    priority 101 # 101 on master, 100 on backup
    virtual_ipaddress {
        192.168.1.200
    }
    track_script {
        chk_haproxy
    }
}

```

En él empezamos con la definición de un chequeo de salud para nuestro servicio HAProxy mediante la apertura de un bloque **vrp\_script**. Esto permitirá a keepalived monitorear nuestro equilibrador de carga ante los posibles fallos, de modo que pueda indicar que el proceso está caído y comenzar las medidas de recuperación.

El sistema de chequeo es muy simple. Cada dos segundos se comprueba que el proceso llamado haproxy sigue teniendo un pid válido.

A continuación, vamos a abrir dos bloques **vrp\_instance**, uno para la red interna y otro para la externa. Esta es la sección principal de la configuración que define la forma en que Keepalived va a implementar la alta disponibilidad.

Vamos a empezar diciéndole a keepalived que se comunique sobre la interfaz eth1, nuestra interfaz de red interna o eth0 en el caso de la red externa. Dado que estamos configurando nuestro servidor principal, vamos a definir la configuración de *state* a "MASTER". En el nodo master2 que funciona de respaldo en caso de fallo se configura con el valor "BACKUP".

Con la opción **virtual\_ipaddress** definimos la dirección IP virtual, que en nuestro caso la hemos configurado como 192.168.1.200. Esta dirección IP virtual permite apuntar a ambos nodos maestro, así si alguno de los dos falla, la dirección IP redirige automáticamente el tráfico a la interfaz del nodo que está activo.

La opción de prioridad se utiliza para decidir qué nodo es elegido en caso de que los dos estén activos. La decisión se basa simplemente en qué servidor tiene el número más alto para este ajuste. Por ello vamos a utilizar "101" para nuestro servidor principal y "100" para el servidor de respaldo.

A continuación, le diremos a keepalived que utilice la comprobación de salud que hemos creado en la parte inicial de la configuración, **chk\_haproxy**. (14)

## 4.5.2 HAProxy

Instalamos el paquete con la orden:

```
apt-get install haproxy
```

Y creamos el archivo de configuración siguiente:

*/etc/haproxy/haproxy.cfg*

```
global
    log /dev/log      local0
    log /dev/log      local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

    # Default SSL material locations
    ca-base /etc/ssl/certs
    crt-base /etc/ssl/private

    # Default ciphers to use on SSL-enabled listening sockets.
    # For more information, see ciphers(1SSL). This list is from:
    # https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
    ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL:!MD5:!DSS
    ssl-default-bind-options no-sslv3

defaults
    log global
    mode http
    option httplog
    option dontlognull
        timeout connect 5000
        timeout client 50000
        timeout server 50000
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

frontend sshfarm 192.168.1.200:2000
    mode tcp
    option tcplog
    default_backend serverfarm

backend serverfarm
    mode tcp
    balance roundrobin
    option tcp-check
```

```
server webA 10.0.100.2:22 check
server webB 10.0.100.3:22 check
server webC 10.0.100.4:22 check
server webD 10.0.100.5:22 check

listen stats 0.0.0.0:9000
mode http
stats show-desc Master1
stats refresh 5s
stats uri /haproxy?stats
stats realm HAProxy\ Statistics
stats auth admin:admin
stats admin if TRUE
```

En la configuración de HAProxy tenemos las siguientes partes a destacar:

- **frontend**

Es dónde HAProxy realiza la escucha para las conexiones. Le hemos puesto el nombre sshfarm y va a escuchar en la IP virtual de nuestro sistema, 192.168.1.200 y el puerto 2000. Hemos establecido el modo tcp ya que en nuestro caso el sistema va a distribuir las conexiones ssh que funcionan a través del protocolo tcp.

Por último se le indica el backend que hemos llamado serverfarm y configuraremos a continuación.

- **backend**

Especifica dónde HAProxy envía las conexiones entrantes. Lo configuramos con el modo tcp al igual que el frontend. Con la opción **balance roundrobin** definimos la estrategia de distribución entre los servidores y por último configuramos la opción **tcp-check** para que compruebe el estado de salud de los 4 servidores.

- **stats**

Configuramos la herramienta web de monitorización de HAProxy para que esté accesible en la dirección 192.168.1.200:9000/haproxy?stats (15)

## HAProxy version 1.5.14, released 2015/07/02

### Statistics Report for pid 1842: Master1

#### > General process information

pid = 1842 (process #1, nproc = 1)  
 uptime = 0d 0h20m38s  
 system limits: memmax = unlimited; ulimit-n = 4036  
 maxsock = 4036; maxconn = 2000; maxpipes = 0  
 current conns = 1; current pipes = 0; conn rate = 0/sec  
 Running tasks: 1/10; idle = 100 %

active UP  
 active UP, going down  
 active DOWN, going up  
 active or backup DOWN  
 active or backup DOWN for maintenance (MAINT)  
 active or backup SOFT STOPPED for maintenance

backup UP  
 backup UP, going down  
 backup DOWN, going up  
 not checked  
 active or backup DOWN for maintenance (MAINT)  
 Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:  
 • Scope:   
 • Hide DOWN servers  
 • Disable refresh  
 • Refresh now  
 • CSV export

External resources:  
 • Primary site  
 • Updates (v1.5)  
 • Online manual

	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server												
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle		
Frontend	0	0	-	0	0	-	0	0	2 000	0	0	0	0	0	0	0	0	0	0	0	0	OPEN										

	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
<input type="checkbox"/> webA	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	0	0	20m38s UP	L4OK in 2ms	1	Y	-	0	0	0	0s	-
<input type="checkbox"/> webB	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	0	0	20m38s UP	L4OK in 1ms	1	Y	-	0	0	0	0s	-
<input type="checkbox"/> webC	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	0	0	20m38s UP	L4OK in 2ms	1	Y	-	0	0	0	0s	-
<input type="checkbox"/> webD	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	0	0	20m38s UP	L4OK in 1ms	1	Y	-	0	0	0	0s	-
Backend	0	0	-	0	0	-	0	0	200	0	0	?	0	0	0	0	0	0	0	0	0	20m38s UP		4	4	0	0	0	0s	-	

Choose the action to perform on the checked servers :  Apply

	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server												
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle		
Frontend	0	2	-	1	1	-	2 000	14	0	0s	63 695	3 101 826	0	0	0	0	0	0	0	0	0	OPEN										
Backend	0	2	-	0	1	-	200	12	0	0s	63 695	3 101 826	0	0	0	0	12	0	0	0	0	20m38s UP		0	0	0	0	0	0	0	0s	-

Imagen 24: Monitorización HAProxy

Una vez ya tenemos configurados los paquetes HAProxy y Keepalived ya podemos reiniciar los servicios y tendremos en funcionamiento nuestro repartidor de carga con alta disponibilidad.

```
service haproxy restart
service keepalived restart
```

## 4.6 Sistema de gestión de trabajos de usuario

Una vez ya tenemos en funcionamiento el sistema que nos va a proporcionar alta disponibilidad en el clúster, es hora de configurar el software que va a permitir a los usuarios lanzar aplicaciones y que el propio sistema se encargue de repartir la carga.

### 4.6.1 Instalación y configuración de NIS

En primer lugar instalamos el software NIS en el nodo NFS que como ya hemos explicado nos permitirá que los perfiles de usuario se creen en todas las máquinas y tengan sus documentos accesibles desde cualquier nodo. (16)

Primero instalaremos el paquete nis con la siguiente orden:

```
apt-get -y install nis
```

Se nos solicitará el nombre del dominio NIS, en nuestro caso hemos utilizado "clúster" y tras 1 minuto de espera, finalizará la instalación.

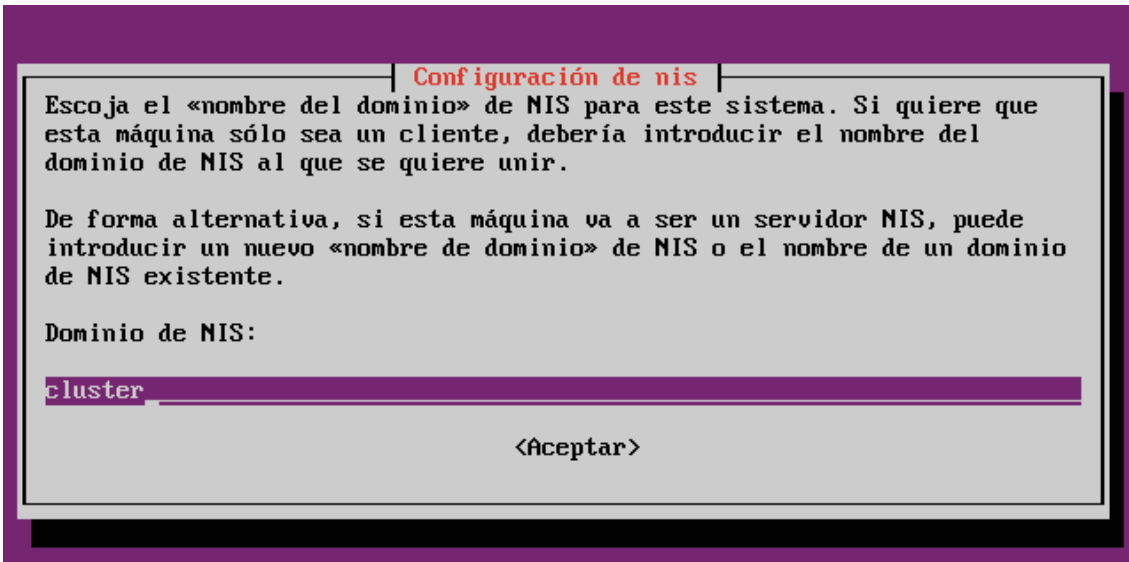


Imagen 25: Configuración dominio NIS

Seguidamente, modificamos el archivo `/etc/default/nis` para indicarle que se trata de un servidor NIS maestro (también se pueden definir también servidores esclavos con propósitos de tolerancia a fallos). También mantenemos el valor de la variable `NISCLIENT` a `true`, lo que indica que el propio nodo es un cliente NIS.

`/etc/default/nis`

```

GNU nano 2.2.6      Archivo: /etc/default/nis      Modificado
#
# /etc/default/nis      Configuration settings for the NIS daemons.
#
# Are we a NIS server and if so what kind (values: false, slave, master)?
NISSERVER=master
# Are we a NIS client?
NISCLIENT=true
# Location of the master NIS password file (for yppasswdd).
# If you change this make sure it matches with /var/yp/Makefile.
YPPWDDIR=/etc
# Do we allow the user to use ypchsh and/or ypchfn ? The YPCHANGEOK
# fields are passed with -e to yppasswdd, see it's manpage.
# Possible values: "chsh", "chfn", "chsh,chfn"
YPCHANGEOK=chsh
# NIS master server.  If this is configured on a slave server then ypinit
# will be run each time NIS is started.
NISMASTER=
# Additional options to be given to ypserv when it is started.
YPSERVARGS=

```

Imagen 26: Configuración NIS

Inicializamos la base de datos con la orden:

```
/usr/lib/yp/ypinit -m
```

Reiniciamos los servicios implicados:

```
restart rpcbind
start ypserv
start ypbind
```

Finalmente, añadiremos al archivo `/etc/passwd` la cadena “+:::”; al archivo `/etc/group` la cadena “+:::” y al archivo `/etc/shadow` la cadena “+:::”, lo que indica al sistema que se debe completar el contenido de esos archivos consultando al servidor NIS.

```
echo +::: >> /etc/passwd
echo +::: >> /etc/group
echo +::: >> /etc/shadow
```

Ahora procederemos con la instalación y configuración de los clientes NIS. Esto hay que realizarlo en todos los nodos del clúster para que los usuarios puedan tener acceso a todos ellos. La instalación de los clientes NIS es idéntica a la del maestro salvo que estos no actúan como servidores NIS. Por tanto, no tenemos que modificar el archivo `/etc/default/nis` para indicarle que trata del maestro.

Instalamos el paquete “nis”:

```
apt-get -y install nis
```

Se nos solicitará el nombre del dominio NIS, a lo que responderemos con el mismo nombre utilizado en el nodo maestro (dominio “clúster”) y finalizará la instalación.

Igual que antes, añadiremos al archivo `/etc/passwd` la cadena “+:::”; al archivo `/etc/group` la cadena “+:::” y al archivo `/etc/shadow` la cadena “+:::”.

```
echo +::: >> /etc/passwd
echo +::: >> /etc/group
echo +::: >> /etc/shadow
```

Y por último reiniciaremos los servicios implicados:

```
restart rpcbind
start ypbind
```

Una vez instalado el NIS ya podemos crear usuarios. Para ello utilizaremos la orden `useradd`. Por ejemplo, creamos el usuario “jordi”, desde el servidor NIS:

```
useradd jordi -s /bin/bash -b /nfs -m
```

La opción `-m` indica que debe crearse el directorio de trabajo si no existe, la opción `-s` el shell por defecto, y la opción `-b` indica dónde se ubicará el directorio de trabajo.

También hay que asignarle una contraseña. Para ello usamos el comando:

```
passwd jordi
```

Cada vez que se realice una modificación en los archivos gestionados por NIS, debe regenerarse la base de datos lanzando la siguiente orden en el servidor:

```
make -C /var/yp
```

A partir de este momento, los cambios se han difundido al resto de nodos, por lo que el usuario creado puede acceder a cualquiera de las máquinas del clúster.

Para eliminar usuarios, se utilizaría la orden *userdel*.

```
userdel jordi -r
```

La opción `-r` indica que debe borrarse el directorio de trabajo.

#### 4.6.2 Instalación y configuración de MOSIX

Como ya hemos explicado, MOSIX es un paquete de software diseñado para añadir a Linux la capacidad de procesamiento clúster, permitiendo a los usuarios mandar tareas que se ejecutaran automáticamente en los servidores

Este paquete no está incluido en la distribución por lo que hay que descargar el archivo de instalación desde su página web:

```
wget http://www.mosix.cs.huji.ac.il/mos4/MOSIX-4.4.0.tbz
bunzip2 MOSIX-4.4.0.tbz
tar -xvf MOSIX-4.4.0.tar
```

Tras descomprimir el archivo, se creará el directorio `mosix-4.4.0`, con todos los archivos necesarios para la instalación. Para proceder a la instalación, lanzamos el script correspondiente:

```
cd mosix-4.4.0
./mosix.install
```

Seguidamente, se nos mostrará un menú de opciones como el que muestra la siguiente imagen.

```
To protect your MOSIX cluster from abuse, preventing
unauthorised persons from gaining control over your computers,
you need to set up a secret cluster protection key.
This key can include any echo characters, but must be
identical throughout your cluster.
```

```
Your secret cluster protection key: toor
Your key is 4 characters long.
(in the future, please consider a longer one)
```

```
What would you like to configure next?
```

```
=====
1. Which nodes are in this cluster
2. Authentication
3. Logical node numbering
4. Processor speed (recommended)
5. Freezing policies
6. Miscellaneous policies
7. Become part of a multi-cluster private cloud
8. Parameters of 'mosrun'
q. Exit
```

Imagen 27: Instalación MOSIX

Procederemos a definir los nodos que componen el clúster, seleccionando la opción 1. Tras pulsar *n*, escribiremos el nombre o dirección de la primera de las máquinas (*cluster1* en nuestro caso) e indicaremos que hay 5 máquinas en total.

Pulsando *q*, terminamos la definición. El script nos preguntará si procede a la generación de nodos lógicos, a lo que responderemos afirmativamente.

A continuación, procederemos a definir la contraseña para la autenticación de los nodos, seleccionando la opción 2. Podemos escribir la tira de caracteres que deseemos.

Pulsando *q* saldremos del script, y se nos preguntará si deseamos iniciar MOSIX, a lo que respondemos afirmativamente.

Una vez terminado con la primera máquina, tenemos que instalarlo en los otros nodos servidores del clúster. Para ello, utilizaremos el mismo script de instalación interactivo que hemos empleado en el nodo maestro.

Primero procederemos a copiar la carpeta de instalación a los nodos:

```
scp -r /root/mosix-4.4.0 cluster2:
scp -r /root/mosix-4.4.0 cluster3:
scp -r /root/mosix-4.4.0 cluster4:
scp -r /root/mosix-4.4.0 cluster5:
```



Y utilizaremos la siguiente orden desde el nodo maestro, para lanzar en cada uno de los nodos el script de instalación:

```
psh "cd /root/mosix-4.4.0; ./mosix.install"
```

Conforme vayan apareciendo en la consola la configuración, iremos respondiendo las preguntas y utilizando las mismas opciones que anteriormente.

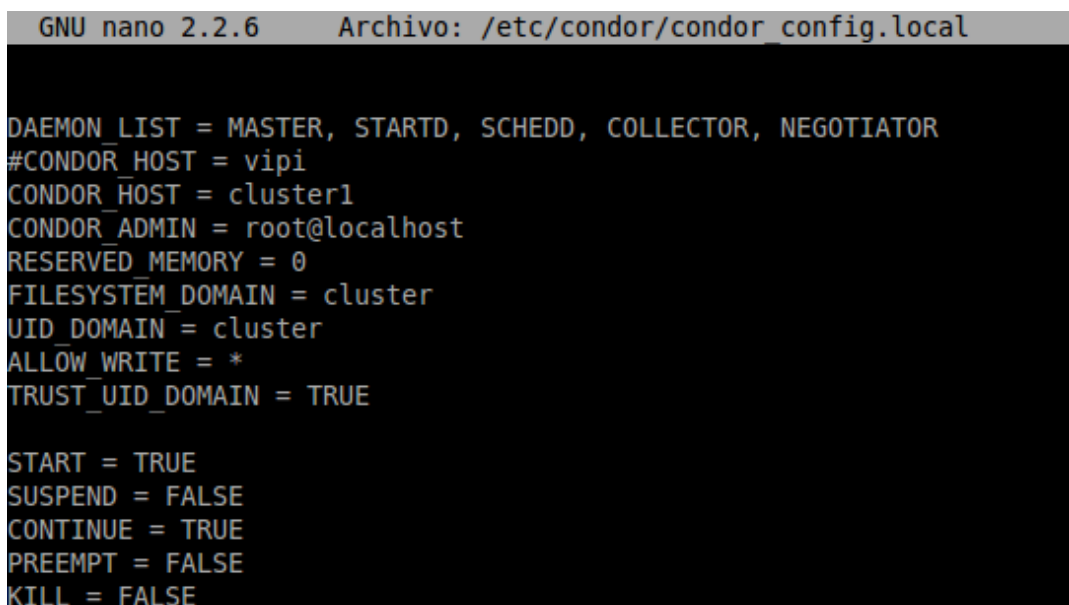
Como se puede observar la instalación de MOSIX es muy sencilla y apenas requiere de configuración, más adelante haremos una prueba del funcionamiento donde veremos cómo se utiliza MOSIX.

### 4.6.3 Instalación y configuración de Condor

La versión que hay en los repositorios de Ubuntu es antigua y limitada para las pruebas que queremos realizar en nuestro proyecto por lo que para instalarlo nos descargamos el paquete "deb" de la versión 8.5.5 para Ubuntu desde la página web del proyecto Condor y lo instalaremos manualmente.

Ahora procederemos a modificar los archivos de configuración para que el nodo maestro actúe de "central manager" al tiempo que puede lanzar y ejecutar trabajos. Para ello, modificaremos los archivos de configuración. (17)

*/etc/condor/condor\_config.local*



```
GNU nano 2.2.6 Archivo: /etc/condor/condor_config.local
DAEMON_LIST = MASTER, STARTD, SCHEDD, COLLECTOR, NEGOTIATOR
#CONDOR_HOST = vipi
CONDOR_HOST = cluster1
CONDOR_ADMIN = root@localhost
RESERVED_MEMORY = 0
FILESYSTEM_DOMAIN = cluster
UID_DOMAIN = cluster
ALLOW_WRITE = *
TRUST_UID_DOMAIN = TRUE

START = TRUE
SUSPEND = FALSE
CONTINUE = TRUE
PREEMPT = FALSE
KILL = FALSE
```

*Imagen 28: Configuración CONDOR maestro*

Y reiniciamos el servicio:

```
service condor restart
```

La instalación en los servidores es similar a la que hemos realizado en el nodo maestro. La diferencia está en que todas las órdenes las vamos a lanzar desde el nodo maestro, utilizando nuestros scripts *psh*, que envían una orden a todos los servidores y *pscp*, que difunde un archivo.

Copiamos el archivo de configuración que hemos preparado:

```
cp /etc/condor/condor_config.local
/root/condor_config.local.servers
```

Y lo modificamos para que los nodos servidores sólo puedan lanzar y ejecutar trabajos:

[/root/condor\\_config.local.servers](#)



```
GNU nano 2.2.6 Archivo: /etc/condor/condor_config.local
DAEMON_LIST = MASTER, STARTD, SCHEDD
#CONDOR_HOST = vipi
CONDOR_HOST = cluster1
CONDOR_ADMIN = root@localhost
RESERVED_MEMORY = 0
FILESYSTEM_DOMAIN = cluster
UID_DOMAIN = cluster
ALLOW_WRITE = *
TRUST_UID_DOMAIN = TRUE
START = TRUE
SUSPEND = FALSE
CONTINUE = TRUE
PREEMPT = FALSE
KILL = FALSE
```

Imagen 29: Configuración CONDOR clientes

Finalmente, lo difundimos a los servidores:

```
pscp /root/condor_config.local.servers
/etc/condor/condor_config.local
```

Y reiniciamos el servicio:

```
psh "service condor restart"
```

Podemos comprobar que Condor está correctamente configurado y arrancado sin más que lanzar la orden en el nodo maestro:

```
condor_status
```

Comprobando que están disponibles todas las máquinas de nuestro clúster.

```

root@cluster1:~# condor_status
Name                OpSys      Arch      State      Activity LoadAv Mem      ActvtyTime
cluster1.cluster    LINUX      X86_64    Unclaimed Idle       0.080  489  0+00:09:39
cluster2.cluster    LINUX      X86_64    Unclaimed Idle       0.030  237  0+00:54:44
cluster3.cluster    LINUX      X86_64    Unclaimed Idle       0.010  237  0+00:54:37
cluster4.cluster    LINUX      X86_64    Unclaimed Idle       0.000  237  0+00:49:43
cluster5.cluster    LINUX      X86_64    Unclaimed Idle       0.010  237  0+00:49:05

                                Total Owner Claimed Unclaimed Matched Preempting Backfill
Drain
  X86_64/LINUX      5      0      0      5      0      0      0
  0
                                Total      5      0      0      5      0      0
  0
root@cluster1:~# █

```

Imagen 30: condor\_status

#### 4.6.3.1 Configuración del servidor de checkpoints

Como ya mencionamos anteriormente vamos a crear un servidor de checkpoints. Se trata de un repositorio de archivos de checkpoint que reducirá los requisitos de tamaño de disco de las máquinas del clúster ya que no tendrán que almacenar los checkpoints de manera local. Este servidor lo vamos a montar en el servidor NFS.

Primero instalaremos la misma versión de condor que instalamos en el servidor maestro.

Una vez instalada tendremos que crear el siguiente archivo de configuración:

*/etc/condor/condor\_config.local*

```

DAEMON_LIST = MASTER, CKPT_SERVER
CKPT_SERVER_DIR = /nfs/ckpt_server
CONDOR_HOST = cluster1
CONDOR_ADMIN = root@localhost
RESERVED_MEMORY = 0
FILESYSTEM_DOMAIN = cluster
UID_DOMAIN = cluster
ALLOW_WRITE = *
TRUST_UID_DOMAIN = TRUE
START = TRUE
SUSPEND = FALSE
CONTINUE = TRUE
PREEMPT = FALSE
KILL = FALSE

```

Imagen 31: Configuración CONDOR servidor checkpoints

Y añadiremos las siguientes líneas en los archivos de configuración del master y de los clientes para que utilicen el servidor de checkpoints.

```
USE_CKPT_SERVER = TRUE
CKPT_SERVER_HOST = nas
```

Una vez terminada la configuración ya podemos reiniciar el servicio y tendremos el servidor de checkpoints de Condor en funcionamiento.

## 5 Pruebas realizadas

Una vez configurado el clúster, vamos a realizar una serie de pruebas para comprobar el funcionamiento correcto de nuestro clúster y más concretamente de los sistemas de almacenamiento, equilibrado de la carga y de gestión de trabajos de usuario.

### 5.1 Sistema de almacenamiento SAN

Para probar el funcionamiento de los volúmenes de GlusterFS vamos a generar unos archivos de forma artificial. La siguiente orden crea 100 copias de un archivo (en nuestro caso `/etc/hosts`) en el volumen replicado:

```
for i in `seq 1 100`; do cp /etc/hosts /mnt/gfs-  
r/replica$i; done
```

Seguidamente obtenemos un listado del contenido del *brick* correspondiente en cada una de las máquinas.

```
ls -la /export/brick1/data
```

```
root@san1:~# ls /export/brick1/data  
replica1    replica22  replica36  replica5    replica63  replica77  replica90  
replica10   replica23  replica37  replica50   replica64  replica78  replica91  
replica100  replica24  replica38  replica51   replica65  replica79  replica92  
replica11   replica25  replica39  replica52   replica66  replica8   replica93  
replica12   replica26  replica4    replica53   replica67  replica80  replica94  
replica13   replica27  replica40  replica54   replica68  replica81  replica95  
replica14   replica28  replica41  replica55   replica69  replica82  replica96  
replica15   replica29  replica42  replica56   replica7    replica83  replica97  
replica16   replica3   replica43  replica57   replica70  replica84  replica98  
replica17   replica30  replica44  replica58   replica71  replica85  replica99  
replica18   replica31  replica45  replica59   replica72  replica86  
replica19   replica32  replica46  replica6    replica73  replica87  
replica2    replica33  replica47  replica60   replica74  replica88  
replica20   replica34  replica48  replica61   replica75  replica89  
replica21   replica35  replica49  replica62   replica76  replica9
```

Imagen 32: Brick replicado san1

```
root@san2:~# ls /export/brick1/data/  
replica1    replica22  replica36  replica5    replica63  replica77  replica90  
replica10   replica23  replica37  replica50   replica64  replica78  replica91  
replica100  replica24  replica38  replica51   replica65  replica79  replica92  
replica11   replica25  replica39  replica52   replica66  replica8   replica93  
replica12   replica26  replica4    replica53   replica67  replica80  replica94  
replica13   replica27  replica40  replica54   replica68  replica81  replica95  
replica14   replica28  replica41  replica55   replica69  replica82  replica96  
replica15   replica29  replica42  replica56   replica7    replica83  replica97  
replica16   replica3   replica43  replica57   replica70  replica84  replica98  
replica17   replica30  replica44  replica58   replica71  replica85  replica99  
replica18   replica31  replica45  replica59   replica72  replica86  
replica19   replica32  replica46  replica6    replica73  replica87  
replica2    replica33  replica47  replica60   replica74  replica88  
replica20   replica34  replica48  replica61   replica75  replica89  
replica21   replica35  replica49  replica62   replica76  replica9
```

Imagen 33: Brick replicado san2

```

root@san3:~# ls /export/brick1/data/
replica1  replica22  replica36  replica5   replica63  replica77  replica90
replica10 replica23  replica37  replica50  replica64  replica78  replica91
replica100 replica24  replica38  replica51  replica65  replica79  replica92
replica11  replica25  replica39  replica52  replica66  replica8   replica93
replica12  replica26  replica4   replica53  replica67  replica80  replica94
replica13  replica27  replica40  replica54  replica68  replica81  replica95
replica14  replica28  replica41  replica55  replica69  replica82  replica96
replica15  replica29  replica42  replica56  replica7   replica83  replica97
replica16  replica3   replica43  replica57  replica70  replica84  replica98
replica17  replica30  replica44  replica58  replica71  replica85  replica99
replica18  replica31  replica45  replica59  replica72  replica86
replica19  replica32  replica46  replica6   replica73  replica87
replica2   replica33  replica47  replica60  replica74  replica88
replica20  replica34  replica48  replica61  replica75  replica89
replica21  replica35  replica49  replica62  replica76  replica9
root@san3:~#

```

Imagen 34: Brick replicado san3

Observamos que todas tienen los 100 archivos creados, puesto que el volumen funciona como replicado. Por eso en todos los nodos del servidor SAN aparecen los mismos archivos dentro del volumen replicado, una buena opción cuando queremos tener varias copias de seguridad de los archivos.

Ahora vamos a probar a realizar el mismo proceso en el volumen distribuido:

```

for i in `seq 1 100`; do cp /etc/hosts /mnt/gfs-d/distr$i;
done

```

Obtenemos un listado del contenido del *brick* correspondiente en cada una de las máquinas.

```

ls -la /export/brick2/data

```

```

root@san1:~# ls /export/brick2/data
distr10  distr24  distr38  distr58  distr67  distr75  distr88
distr100 distr25  distr45  distr59  distr68  distr76  distr90
distr14  distr26  distr47  distr6   distr7   distr78  distr91
distr15  distr31  distr48  distr61  distr70  distr79  distr96
distr16  distr32  distr52  distr62  distr73  distr8   distr98
distr19  distr33  distr56  distr63  distr74  distr82
root@san1:~#

```

Imagen 35: Brick distribuido san1

```

root@san2:~# ls /export/brick2/data/
distr1  distr2  distr35  distr44  distr54  distr72  distr87  distr99
distr11 distr21  distr36  distr5   distr55  distr81  distr9   distr9
distr12 distr29  distr39  distr50  distr64  distr84  distr94
distr18 distr3   distr43  distr53  distr71  distr86  distr95
root@san2:~#

```

Imagen 36: Brick distribuido san2

```

root@san3:~# ls /export/brick2/data/
distr13  distr23  distr34  distr41  distr51  distr66  distr83  distr93
distr17  distr27  distr37  distr42  distr57  distr69  distr85  distr97
distr20  distr28  distr4   distr46  distr60  distr77  distr89
distr22  distr30  distr40  distr49  distr65  distr80  distr92
root@san3:~#

```

Imagen 37: Brick distribuido san3

Observamos que los archivos se han repartido entre los tres bricks que componen el volumen, san1, san2 y san3, Con este modo de funcionamiento se pueden conseguir volúmenes de gran capacidad combinando las unidades de los distintos nodos.

## 5.2 Channel bonding

Como ya explicamos en el apartado de configuración de nuestro clúster, el servidor NFS está configurado con dos interfaces de red en formando un **channel bonding** en modo **link aggregation**.

Si comprobamos el archivo de configuración [/proc/net/bonding/bond0](#)

```
GNU nano 2.2.6 Archivo: /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)
Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: fast
Min links: 0
Aggregator selection policy (ad_select): stable
Active Aggregator Info:
  Aggregator ID: 1
  Number of ports: 1
  Actor Key: 9
  Partner Key: 1
  Partner Mac Address: 00:00:00:00:00:00

Slave Interface: eth1
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 08:00:27:da:c3:b1
Aggregator ID: 1
Slave queue ID: 0

Slave Interface: eth0
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 08:00:27:d7:6c:4c
Aggregator ID: 2
Slave queue ID: 0
```

Imagen 38: bond0

Confirmamos que la configuración de la interfaz **bond0** es la correcta ya que está en modo **Dynamic link aggregation**.

Para comprobar que está funcionando correctamente lanzamos un ping desde el servidor Master1 al NFS:

```
ping 10.0.100.100
```

Seguidamente, en el servidor NFS usamos el siguiente comando para ver el tráfico de paquetes:

```
watch -d -n1 netstat -i
```

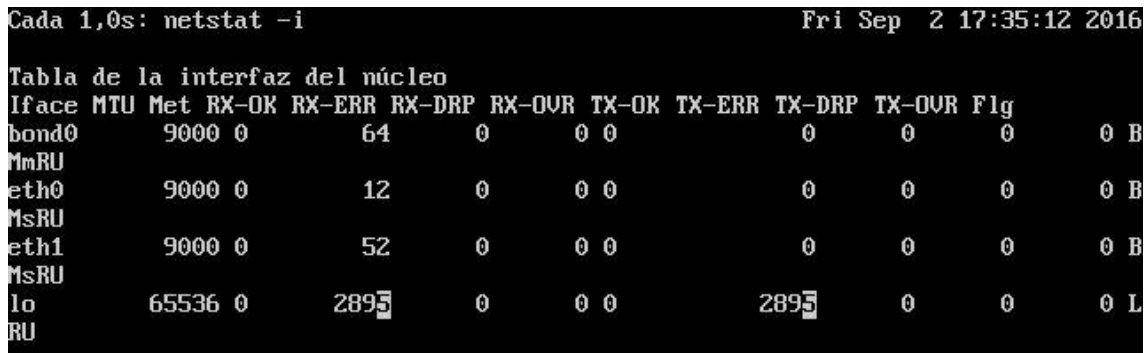


Imagen 39: Prueba channel bonding

Se puede observar como ambas interfaces eth0 y eth1 están recibiendo paquetes de forma correcta (RX-OK).

### 5.3 Sistema de alta disponibilidad

A continuación vamos a probar el sistema de alta disponibilidad

Si nos conectamos a la IP Virtual 192.168.1.200 en el puerto 9000 y la url haproxy?stats, accedemos a la interfaz visual de monitorización de HAProxy.

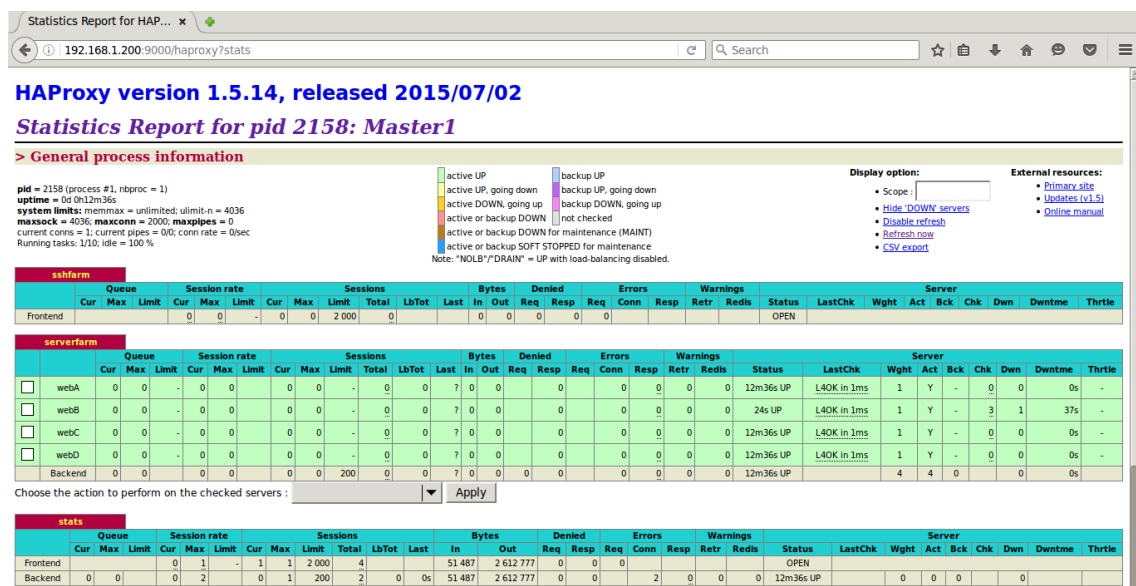


Imagen 40: Pruebas HAProxy 1



Podemos observar que el host activo que está realizando las tareas de gestión es Master1, también nos muestra que están todos los nodos del servidor en funcionamiento ya que aparecen de color verde

Ahora desconectamos el adaptador de red de uno de los nodos servidores (el servidor cluster1) para simular un fallo en el sistema que impida su correcto funcionamiento o comunicación con el host.

**HAProxy version 1.5.14, released 2015/07/02**  
**Statistics Report for pid 2158: Master1**

> **General process information**

pid = 2158 (process #1, nbproc = 1)  
 uptime = 00:0h23m12s  
 system limits: memmax = unlimited; ulimit-n = 4036  
 maxsock = 4036; maxconn = 2000; maxpipes = 0  
 current conns = 1; current pipes = 0/0; conn rate = 0/sec  
 running tasks: 1/0; idle = 100 %

Legend:  
 active UP (green)  
 active UP, going down (yellow)  
 active DOWN, going up (orange)  
 active or backup DOWN (red)  
 active or backup SOFT STOPPED for maintenance (purple)  
 backup UP (blue)  
 backup UP, going down (light blue)  
 backup DOWN, going up (light orange)  
 not checked (grey)

Display option:  
 Scope: [ ]  
 Hide 'DOWN' servers  
 Disable refresh  
 Refresh now  
 CSV export

External resources:  
 Primary site  
 Issues (v1.5)  
 Online manual

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

sshfarm		Queue		Session rate		Sessions				Bytes		Denied		Errors		Warnings		Status		Server										
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend	0	1	-	0	1	2 000	5	2	2	42s	7 738	7 230	0	0	0	0	0	0	0	0	0	29% DOWN	L4TOUT in 2003ms	1	Y	-	3	1	29s	-

serverfarm		Queue		Session rate		Sessions				Bytes		Denied		Errors		Warnings		Status		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
<input type="checkbox"/>	webA	0	0	-	0	1	0	1	-	2	2	42s	7 738	7 230	0	0	0	0	0	0	0	0	29% DOWN	L4TOUT in 2003ms	1	Y	-	3	1	29s	-
<input type="checkbox"/>	webB	0	0	-	0	1	0	1	-	1	1	44s	2 059	1 971	0	0	0	0	0	0	0	0	11m UP	L4OK in 1ms	1	Y	-	3	1	37s	-
<input type="checkbox"/>	webC	0	0	-	0	1	0	1	-	1	1	1m12s	2 059	1 971	0	0	0	0	0	0	0	0	23m12s UP	L4OK in 1ms	1	Y	-	0	0	0s	-
<input type="checkbox"/>	webD	0	0	-	0	1	0	1	-	1	1	46s	2 059	1 971	0	0	0	0	0	0	0	0	23m12s UP	L4OK in 2ms	1	Y	-	0	0	0s	-
	Backend	0	0	-	0	1	0	1	200	5	5	42s	13 915	13 143	0	0	0	0	0	0	0	0	23m12s UP		3	3	0	0	0	0s	-

Choose the action to perform on the checked servers: [ ] Apply

stats		Queue		Session rate		Sessions				Bytes		Denied		Errors		Warnings		Status		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend	0	1	-	1	1	2 000	4	2	2	4	0s	97 115	4 993 731	0	0	0	0	0	0	0	0	0	OPEN		0	0	0	0	0	0s	-
Backend	0	0	-	0	2	200	2	2	2	2	0s	97 115	4 993 731	0	0	0	2	0	0	0	0	0	23m12s UP		0	0	0	0	0	0s	-

Imagen 41: Pruebas HAProxy 2

Comprobamos en el monitor de HAProxy que el servidor se queda marcado en rojo, lo que nos indica que ha funcionado correctamente la detección de los nodos activos.

Si realizamos un ssh a la IP virtual y utilizando el puerto 2000 como establecimos en la configuración de HAProxy:

```
ssh -p 2000 192.168.1.200
```

```

root@cluster1:~# ssh -p 2000 192.168.1.200
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Mon Aug 22 20:41:37 CEST 2016

System load:  1.19           Processes:           106
Usage of /:   33.6% of 3.60GB Users logged in:      0
Memory usage: 24%           IP address for eth0: 10.0.100.2
Swap usage:   0%

Graph this data and manage this system at:
https://landscape.canonical.com/

New release '16.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Aug 22 20:40:39 2016 from cluster1.cluster
root@cluster2:~#

```

Imagen 42: ssh Virtual IP

Observamos que nos atiende el nodo servidor cluster2, el primero de los que están activos en este momento. Cada vez que se inicia una sesión ssh sobre la dirección VIP nos atiende un nuevo nodo servidor de forma cíclica, tal y como está configurado.

Seguidamente, volvemos a conectar el adaptador de red del nodo del servidor y ahora desconectamos las interfaces de red del nodo master1 para comprobar si entra en funcionamiento el nodo de respaldo, master2.

Tras desconectar master1 accedemos a la interfaz de monitorización de HAProxy y observamos cómo efectivamente ha entrado en funcionamiento el host Master2 de manera automática, puesto que el interfaz web de monitorización sigue respondiendo y nos indica “Master2”.

**HAProxy version 1.5.14, released 2015/07/02**  
**Statistics Report for pid 1769: Master2**

> **General process information**

pid = 1769 (process #1, nproc = 1)  
 uptime = 01:01:0m21s  
 system limits: memmax = unlimited; ulimit-n = 4036  
 maxsock = 4036; maxconn = 2000; maxpipes = 0  
 current conn = 1; current pipes = 0; conn rate = 0/sec  
 Running tasks: 1/10; idle = 100 %

active UP, active UP, going down, active DOWN, going up, active or backup DOWN, active or backup DOWN for maintenance (MAINT), active or backup SOFT STOPPED for maintenance, backup UP, backup UP, going down, backup DOWN, going up, not checked

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:  External resources:  
 • Scope:   
 • Hide 'DOWN' servers  
 • Disable refresh  
 • Refresh now  
 • CSV export  
 • Primary site  
 • Backup site 1  
 • Online manual

Queue		Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server													
Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
sshfarm																														
Frontend		0	0	-	0	0	0	2 000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OPEN							
serverfarm																														
<input type="checkbox"/>	webA	0	0	-	0	0	0	0	-	0	0	?	0	0	0	0	0	0	0	0	0	3m31s UP	L4OK in 1ms	1	Y	-	3	1	1m31s	-
<input type="checkbox"/>	webB	0	0	-	0	0	0	0	-	0	0	?	0	0	0	0	0	0	0	0	0	15m34s UP	L4OK in 2ms	1	Y	-	3	1	37s	-
<input type="checkbox"/>	webC	0	0	-	0	0	0	0	-	0	0	?	0	0	0	0	0	0	0	0	0	18m21s UP	L4OK in 1ms	1	Y	-	0	0	0s	-
<input type="checkbox"/>	webD	0	0	-	0	0	0	0	-	0	0	?	0	0	0	0	0	0	0	0	0	18m21s UP	L4OK in 1ms	1	Y	-	0	0	0s	-
<input type="checkbox"/>	Backend	0	0	-	0	0	0	0	200	0	0	?	0	0	0	0	0	0	0	0	0	18m21s UP		4	4	0	0	0	0s	

Choose the action to perform on the checked servers:  Apply

Queue		Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server													
Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
stats																														
Frontend		0	1	-	1	1	2 000	4				3 563	135 691	0	0	0	0	0	0	0	0	OPEN								
Backend		0	0	-	0	1	200	2	0	0s		3 563	135 691	0	0	0	2	0	0	0	0	18m21s UP		0	0	0	0	0	0	

Imagen 43: Pruebas HAProxy 3

Tras lanzar repetidamente sesiones ssh, observamos cómo podemos acceder sin problema (a través del master2 en este momento) a los nodos servidores del clúster.

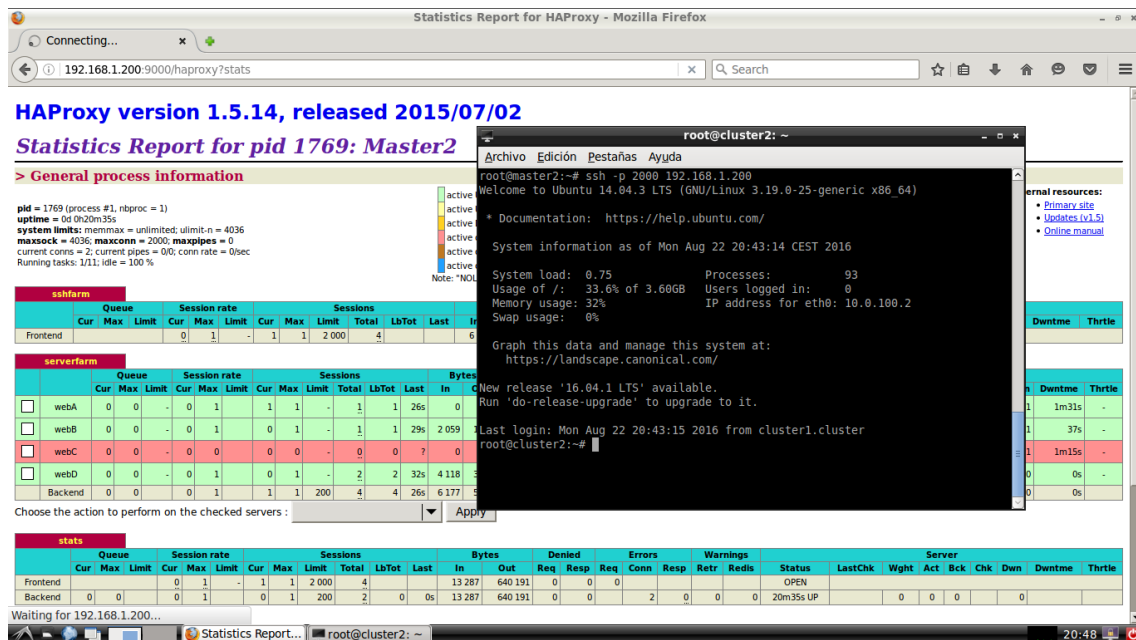


Imagen 44: Pruebas HAProxy 4

Con estas pruebas hemos podido comprobar que nuestro sistema admite el fallo de nodos servidores y de uno de los nodos maestros sin perder funcionalidad, ya que el clúster continua funcionando y asigna los recursos disponibles de forma automática y transparente para el usuario.

## 5.4 Gestión de trabajos de usuario

En esta ocasión vamos a poner a prueba los sistemas de gestión de trabajo de usuario, encargados de repartir las tareas que manden a ejecutar los usuarios entre los nodos del servidor.

### 5.4.1 MOSIX

Para el caso de Mosix vamos a realizar una prueba sencilla que consiste en lanzar a ejecución el programa de test *mostestload*, incluido en el paquete *MOSIX*.

Para ello utilizamos la orden:

```
mosrun mostestload &
```

Podemos monitorizar el estado del clúster MOSIX lanzando, en cualquiera de los nodos del clúster, la orden:

```
mosmon
```

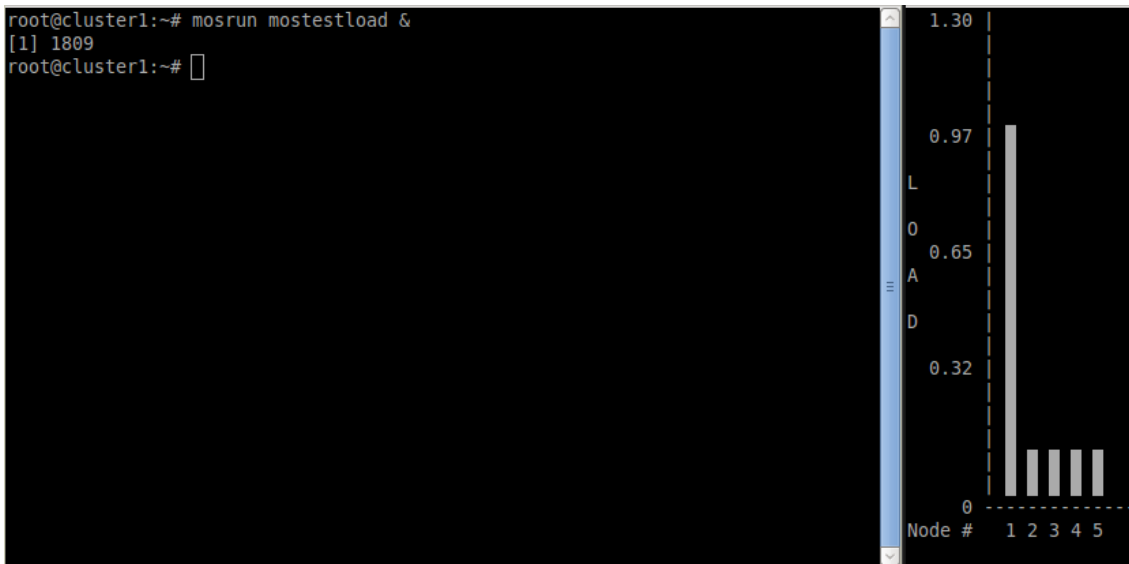


Imagen 45: Pruebas Mosix 1

Aquí observamos un gráfico de barras que varía dinámicamente en función de la carga de los nodos del sistema, como solo hemos lanzado una vez la tarea, la carga solo recae sobre uno de los servidores.

Ahora probamos a lanzar más instancias del programa de prueba para comprobar como el sistema realiza el equilibrio de la carga:

```
mosrun mostestload &
mosrun mostestload &
mosrun mostestload &
mosrun mostestload &
...
```

Con la orden *mosps* nos muestra los procesos lanzados, indicando el nodo que los lanzó y el nodo en el que se está ejecutando:

```
mosps
```

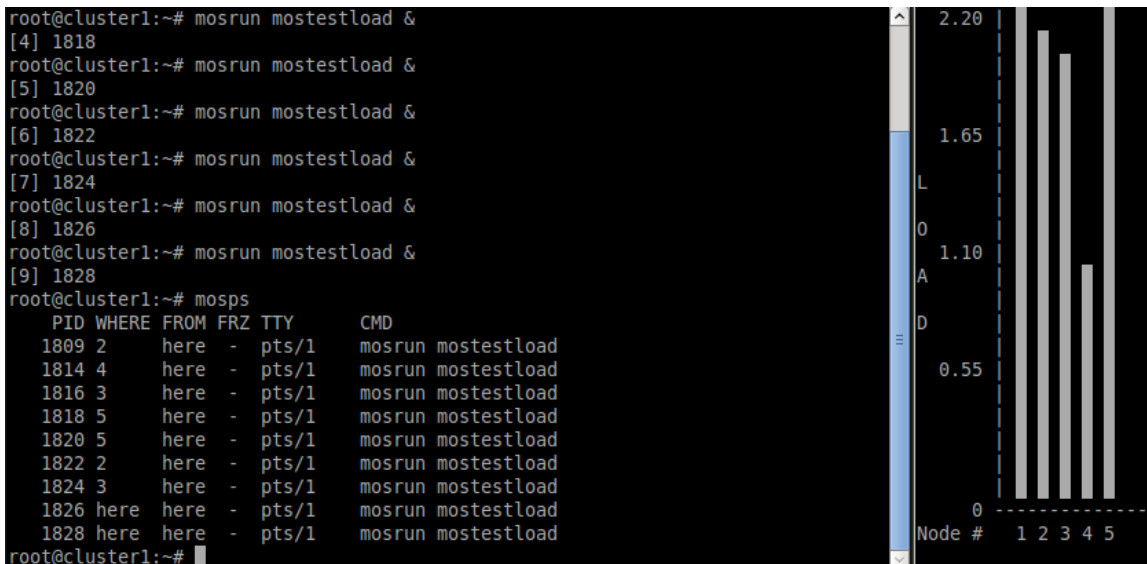


Imagen 46: Pruebas Mosix 2

Podemos comprobamos como la carga se reparte entre los 5 servidores del clúster de forma equitativa. Nótese que todas las instancias del programa se han lanzado desde el mismo nodo.

Finalmente, la orden *moskillall* permite terminar todos los procesos lanzados:

```
moskillall
```

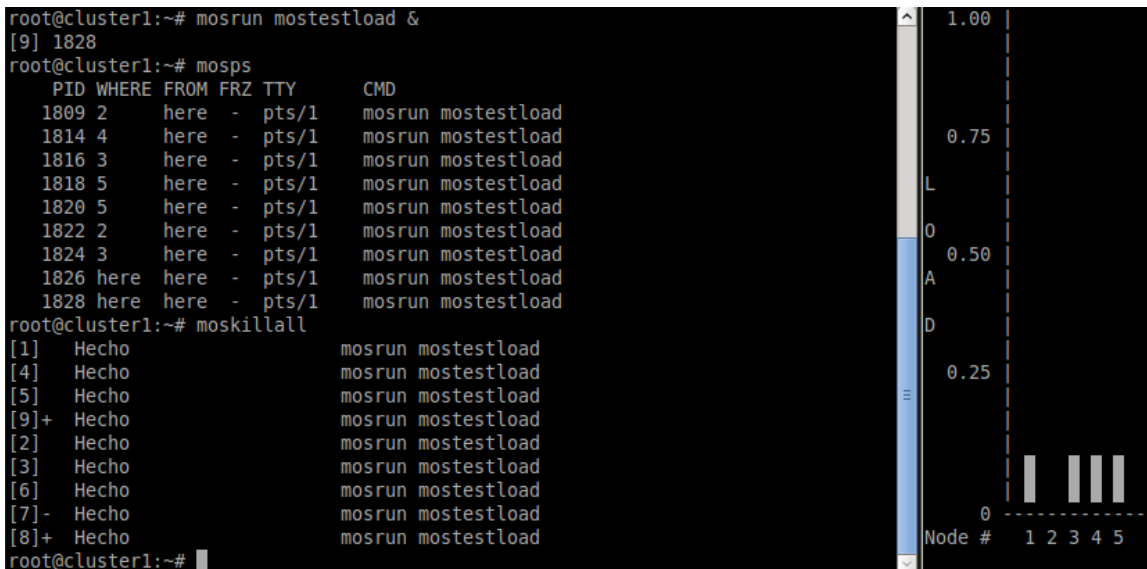


Imagen 47: Pruebas Mosix 3

## 5.4.2 CONDOR

Los universos de condor definen los entornos de ejecución, el **universo vanilla** es el que se ejecuta por defecto si no lo especificamos, proporciona menos servicios pero también tiene menos restricciones y funciona bien con shellscrips. El **universo standard** proporciona migración y fiabilidad, pero tiene algunas restricciones en los programas que se pueden ejecutar, ya que tiene el mecanismo de checkpointing y llamadas remotas al sistema.

Vamos a realizar dos pruebas distintas con CONDOR. La primera va a consistir en crear una tarea sencilla, que la mandaremos a ejecución 100 veces y veremos cómo se crea una cola para ir distribuyendo las tareas entre los servidores, para ello utilizaremos el **universo vanilla**.

La siguiente prueba consistirá en lanzar la ejecución de un programa con el **universo standard**, interrumpirla manualmente en mitad de la ejecución para ver cómo se genera un archivo de respaldo checkpoint y veremos cómo se reanuda la ejecución a partir del checkpoint.

### 5.4.2.1 Prueba cola de trabajos

Como hemos mencionado el universo vanilla funciona bien con shellscrips por eso hemos creado el siguiente script cuya única función es poner a dormir la tarea durante los segundos que le pasemos por parámetro, se trata de un script muy simple pero nos sirve para comprobar cómo se va ejecutando la tarea en los distintos servidores.

#### *myprog*

```
#!/bin/sh
echo "I'm process id $$ on " `hostname`
echo "This is sent to standard error" 1>&2
date
echo "Running as binary $0" "$@"
echo "My name (argument 1) is $1"
echo "My sleep duration (argument 2) is $2"
sleep $2
echo "Sleep of $2 seconds finished. Exiting"
exit 0
```

Luego creamos un trabajo para condor con la siguiente estructura:

#### *myjob.condor*

```
executable=myprog
universe=vanilla
arguments=Example.$(Cluster).$(Process) output=results.output.$(Process)
) 2
error=results.error.$(Process)
log=results.log
notification=never
queue 100
```

En esta le indicamos cual es el programa a ejecutar, el parámetro, que en nuestro caso hemos puesto 2 que indica que va a dejar durmiendo 2 segundos los procesos. También indicamos el universo de ejecución vanilla como ya hemos explicado y con la opción *queue 100* creamos una cola de ejecución del programa de 100 instancias

Utilizamos el siguiente comando para comprobar el estado de los nodos del clúster:

```
condor_status
```

```
pepe@cluster1:~$ condor_status
Name                OpSys    Arch    State    Activity LoadAv Mem    ActvtyTime
cluster1.cluster    LINUX    X86_64 Unclaimed Idle     0.150  489  0+00:00:04
cluster2.cluster    LINUX    X86_64 Unclaimed Idle     0.000  237  0+00:00:04
cluster3.cluster    LINUX    X86_64 Unclaimed Idle     0.110  237  0+00:00:04
cluster4.cluster    LINUX    X86_64 Unclaimed Idle     0.000  237  0+00:00:04
cluster5.cluster    LINUX    X86_64 Unclaimed Idle     0.030  237  0+00:00:04
      Total Owner  Claimed Unclaimed Matched Preempting Backfill
      X86_64/LINUX    5    0    0    5    0    0    0
      Total          5    0    0    5    0    0    0
pepe@cluster1:~$
```

Imagen 48: Pruebas Condor 1

Como todavía no se ha lanzado ninguna tarea observamos que están todos los nodos con el estado “*idle*” que indica que están inactivos.

A continuación lanzamos la tarea con la siguiente orden:

```
condor_submit myjob.condor
```

```
pepe@cluster1:~$ condor_status
Name                OpSys    Arch    State    Activity LoadAv Mem    ActvtyTime
cluster1.cluster    LINUX    X86_64 Unclaimed Idle     0.150  489  0+00:00:04
cluster2.cluster    LINUX    X86_64 Unclaimed Idle     0.000  237  0+00:00:04
cluster3.cluster    LINUX    X86_64 Unclaimed Idle     0.110  237  0+00:00:04
cluster4.cluster    LINUX    X86_64 Unclaimed Idle     0.000  237  0+00:00:04
cluster5.cluster    LINUX    X86_64 Unclaimed Idle     0.030  237  0+00:00:04
      Total Owner  Claimed Unclaimed Matched Preempting Backfill
      X86_64/LINUX    5    0    0    5    0    0    0
      Total          5    0    0    5    0    0    0
pepe@cluster1:~$ condor_submit myjob.condor
Submitting job(s).....
.....
100 job(s) submitted to cluster 5.
```

Imagen 49: Pruebas Condor 2

Comprobamos que se crea una cola con 100 trabajos y que comienza a asignarse a los servidores la ejecución de la tarea.

Si verificamos el estado de los servidores vemos que el estado ha pasado a “Busy” lo que indica que los cinco nodos están ejecutando la aplicación.

```

6.82 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.8
6.83 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.8
6.84 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.8
6.85 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.8
6.86 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.8
6.87 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.8
6.88 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.8
6.89 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.8
6.90 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.91 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.92 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.93 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.94 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.95 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.96 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.97 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.98 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9
6.99 pepe      8/27 13:40 0+00:00:00 I 0 0.0 myprog Example.6.9

90 jobs; 0 completed, 0 removed, 85 idle, 5 running, 0 held, 0 suspended
pepe@cluster1:~$ condor_status
Name                OpSys      Arch      State      Activity LoadAv Mem      ActvtyTime
cluster1.cluster    LINUX      X86_64    Claimed    Busy      0.440 489 0+00:00:02
cluster2.cluster    LINUX      X86_64    Claimed    Busy      0.070 237 0+00:00:02
cluster3.cluster    LINUX      X86_64    Claimed    Busy      0.000 237 0+00:00:02
cluster4.cluster    LINUX      X86_64    Claimed    Busy      0.010 237 0+00:00:02
cluster5.cluster    LINUX      X86_64    Claimed    Busy      0.000 237 0+00:00:02
      Total Owner Claimed Unclaimed Matched Preempting Backfill
      X86_64/LINUX      5      0      5      0      0      0      0
      Total      5      0      5      0      0      0      0
pepe@cluster1:~$ █

```

Imagen 50: Pruebas Condor 3

Las tareas de la cola van desapareciendo a medida que termina la ejecución del shellscript, en la siguiente imagen se puede ver cómo la tarea cambia al estado *R* de *running* cuando está ejecutándose en un nodo.



```

30 jobs; 0 completed, 0 removed, 25 idle, 5 running, 0 held, 0 suspended
pepe@cluster1:~$ condor_q

-- Submitter: cluster1.cluster : <192.168.1.5:56053> : cluster1.cluster
ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD
 6.95   pepe      8/27 13:40     0+00:00:02 R  0  0.0  myprog Example.6.9
 6.96   pepe      8/27 13:40     0+00:00:02 R  0  0.0  myprog Example.6.9
 6.97   pepe      8/27 13:40     0+00:00:02 R  0  0.0  myprog Example.6.9
 6.98   pepe      8/27 13:40     0+00:00:02 R  0  0.0  myprog Example.6.9
 6.99   pepe      8/27 13:40     0+00:00:01 R  0  0.0  myprog Example.6.9

5 jobs; 0 completed, 0 removed, 0 idle, 5 running, 0 held, 0 suspended
pepe@cluster1:~$ condor_q

-- Submitter: cluster1.cluster : <192.168.1.5:56053> : cluster1.cluster
ID      OWNER      SUBMITTED      RUN_TIME ST PRI SIZE CMD

0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
pepe@cluster1:~$ condor_status
Name                OpSys      Arch      State      Activity LoadAv Mem  ActvtyTime
cluster1.cluster    LINUX      X86_64    Unclaimed Idle      0.460 489  0+00:00:04
cluster2.cluster    LINUX      X86_64    Unclaimed Idle      0.040 237  0+00:00:04
cluster3.cluster    LINUX      X86_64    Unclaimed Idle      0.000 237  0+00:00:04
cluster4.cluster    LINUX      X86_64    Unclaimed Idle      0.130 237  0+00:00:04
cluster5.cluster    LINUX      X86_64    Unclaimed Idle      0.000 237  0+00:00:04
Total Owner Claimed Unclaimed Matched Preempting Backfill
X86_64/LINUX      5      0      0      5      0      0      0
Total              5      0      0      5      0      0      0
pepe@cluster1:~$ █

```

Imagen 51: Pruebas Condor 4

Al terminar comprobamos que la cola se queda vacía y que los nodos vuelven al estado *Idle*.

#### 5.4.2.2 Prueba de checkpoints

Para la prueba de checkpoints tenemos que utilizar el universo standard, que no es compatible con shellscrips, por lo que vamos a utilizar el siguiente código del cálculo de la constante PI, al cual le pasamos como argumento el número de iteraciones que realiza la función para calcular el número PI. A mayor número, mayor será la precisión y más tardará la aplicación en ejecutarse.

#### Pi.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){

    double area, pi, x;
    int i,n;

    if (argc != 2) {
        scanf("%d",&n);
    }else{
        n=atoi(argv[1]);
    }
    area = 0.0;

    for(i=0; i<n; i++) {
        x = (i+0.5)/n;
        area += 4.0/(1.0+x*x);
    }
    pi = area/n;
    printf ("%f\n", pi);

    return 0;
}
```

Compilamos la aplicación con la siguiente orden, esto crea un ejecutable con las librerías enlazadas con condor para poder lanzarla en el universo standard:

```
condor_compile gcc pi.c -o pi
```

Una vez compilado lanzamos la tarea con la orden:

```
setarch x86 x86_64 -R ./pi 100000 -_condor_D_ALL
```

La opción *D ALL* hace que nos muestre más información a la hora de la ejecución, sobre todo para comprobar que se está generando correctamente el checkpoint.

Añadimos *setarch x86\_64 -R* para forzar la arquitectura ya que sin esa opción no nos generaba correctamente los checkpoints.

```

pepe@cluster1:~$ setarch x86_64 -R ./pi 100000 -_condor_D_ALL
User Job - $CondorPlatform: x86_64 Ubuntu14 $
User Job - $CondorVersion: 8.5.5 Jun 03 2016 BuildID: 369308 $
Condor: Notice: Will checkpoint to ./pi.ckpt
Condor: Notice: Remote system calls disabled.
3.141593
pepe@cluster1:~$ █

```

Imagen 52: Pruebas Condor 5

Comprobamos que obtenemos el resultado del número PI con relativa rapidez.

Ahora aumentamos el número de iteraciones que tiene que realizar la aplicación

```

setarch x86_64 -R ./pi 1000000000 -_condor_D_ALL

```

La aplicación ahora tarda mucho más en terminar por lo que durante la ejecución simulamos un fallo pulsando la combinación de teclas CTRL+Z.

```

      dups:          1
      open flags:   0x1
      not currently bound to a url.
working dir = /nfs/pepe
Done saving file state
About to update MyImage
Adding a DATA segment: start[0x1], end [0x1]
Image::AddSegment: name=[DATA], start=[7d8000], end=[885000], length=[0x1], prot=[0xad000]
Adding a STACK segment: start[0x1], end [0x1]
Image::AddSegment: name=[STACK], start=[7fffffff5000], end=[7fffffffef], length=[0x1], prot=[0x9fff]
Pos: ld
Pos: ld
Size of ckpt image = ld bytes
About to write checkpoint
Image::Write(): fd -1 file_name ./pi.ckpt
Checkpoint name is "./pi.ckpt"
Tmp name is "./pi.ckpt.tmp"
Wrote headers OK
Wrote all SegMaps OK
write(fd=3,core_loc=0x1,len=0x1)
I wrote 708608 bytes with write...
Wrote Segment[0] of type DATA -> OK
write(fd=3,core_loc=0x1,len=0x1)
I wrote 40959 bytes with write...
Wrote Segment[1] of type STACK -> OK
Wrote all Segments OK
About to close ckpt fd (3)
Closed OK
About to rename "./pi.ckpt.tmp" to "./pi.ckpt"
Renamed OK
USER PROC: CHECKPOINT IMAGE SENT OK
Ckpt exit
Señal definida por el usuario 2
pepe@cluster1:~$ █

```

Imagen 53: Pruebas Condor 6

Observamos en el registro de la ejecución que se genera un archivo de checkpoint.

A continuación verificamos que se ha creado el archivo de checkpoint en el servidor de checkpoints (alojado en el NFS):

```

ls /nfs/ckpt

```

```
pepe@cluster1:~$ ls /nfs/ckpt/  
pi.ckpt  
pepe@cluster1:~$ █
```

Imagen 54: Pruebas Condor 7

Observamos que efectivamente se ha creado un archivo de checkpoint llamado pi.ckpt  
Por último probamos a reanudar la ejecución de la aplicación por donde se había  
quedado, utilizando para ello el archivo de checkpoint, con la orden:

```
setarch x86 x86_64 -R ./pi 1000000000 -_condor_restart  
/nfs/ckpt/pi.ckpt
```

```
About to execute on tmpstack.  
Beginning Execution on TmpStack.  
RestoreStack() Entrance!  
Restoring a STACK segment  
About to overwrite 40959 bytes starting at 0x7fffffff5000(STACK)  
RestoreStack() Exit!  
About to restore file state  
CondorFileTable::resume  
working dir = /nfs/pepe  
  
OPEN FILE TABLE:  
fd 0  
    logical name: default stdin  
    offset:      ld  
    dups:       1  
    open flags: 0x0  
    not currently bound to a url.  
fd 1  
    logical name: default stdout  
    offset:      ld  
    dups:       1  
    open flags: 0x1  
    not currently bound to a url.  
fd 2  
    logical name: default stderr  
    offset:      ld  
    dups:       1  
    open flags: 0x1  
    not currently bound to a url.  
Done restoring file state  
About to restore signal state  
About to return to user code  
3.141593  
pepe@cluster1:~$ █
```

Imagen 55: Pruebas Condor 8

Verificamos que la aplicación termina correctamente reanudando la ejecución desde el  
archivo de checkpoint.

## 6 Conclusiones

Hoy en día se ha convertido en requisito casi imprescindible para cualquier empresa u organización tecnológica contar con su propio clúster, ya sea para tareas de cómputo; para un servidor web; o simplemente como infraestructura central de la organización, como un servidor de directorio activo o de archivos.

En nuestro caso nos hemos centrado en instalar y configurar un clúster para el cálculo científico, utilizados principalmente en universidades y centros de investigación. En este tipo de clúster prima un equilibrado de la carga eficiente y un alto rendimiento para que los complejos cálculos científicos se ejecuten en el menor tiempo posible. También es importante una alta disponibilidad para que el sistema sea capaz de continuar con la ejecución de los programas aunque alguno de los nodos de cómputo falle, (ya que cálculos complejos pueden tardar días en ejecutarse), y un sistema de gestión de las tareas para que los usuarios puedan lanzar todas las aplicaciones que deseen y que se encargue el sistema de gestionarlas y ofrecer los resultados de forma ordenada.

En este proyecto se ha instalado y configurado satisfactoriamente un clúster de cómputo de alta disponibilidad con equilibrado de la carga. Las pruebas realizadas sobre nuestro clúster, que se han centrado en probar las características que hace a nuestro sistema un clúster para el cálculo científico, nos han permitido comprobar el correcto funcionamiento del sistema, por lo que podemos concluir que los objetivos planteados al inicio del proyecto se han llevado a cabo.

Como trabajo futuro se podría configurar y probar otros sistemas de gestión de trabajos de usuario, como podría ser **slurm** o **torque** y realizar una comparativa entre los sistemas probados.

## 7 Referencias bibliográficas

1. **Ministerio de Educación, Cultura y Deporte.** Servidor DNS sencillo en Linux con dnsmasq. [En línea] 15 de Octubre de 2008.  
<http://recursostic.educacion.es/observatorio/web/gl/software/software-general/638-servidor-dns-sencillo-en-linux-con-dnsmasq>.
2. **ORACLE.** Descripción general de PXE. [En línea] <http://docs.oracle.com/cd/E19140-01/821-2239/p44.html>.
3. **Wikipedia.** Network File System. [En línea]  
[https://es.wikipedia.org/wiki/Network\\_File\\_System](https://es.wikipedia.org/wiki/Network_File_System).
4. —. Network Information Service. [En línea]  
[https://es.wikipedia.org/wiki/Network\\_Information\\_Service](https://es.wikipedia.org/wiki/Network_Information_Service).
5. —. Gluster File System. [En línea] [https://es.wikipedia.org/wiki/Gluster\\_File\\_System](https://es.wikipedia.org/wiki/Gluster_File_System).
6. **MOSIX.** MOSIX. [En línea] <http://www.mosix.org/>.
7. **Marrero, Adrián Santos.** Introducción a Condor. [En línea] 17 de Septiembre de 2004. [http://www.iac.es/sieinvens/SINFIN/Condor/iac\\_manual/manual.pdf](http://www.iac.es/sieinvens/SINFIN/Condor/iac_manual/manual.pdf).
8. **iesgn.** Balanceador de carga redundante - HAProxy & KeepAlived. [En línea]  
<http://openstack-olimpo.github.io/contenido/balanceadores/>.
9. **VirtualBox.** VirtualBox. [En línea] <https://www.virtualbox.org/>.
10. **Ubuntu.** Ubuntu. [En línea] <http://www.ubuntu.com/>.
11. —. UbuntuBonding. [En línea]  
<https://help.ubuntu.com/community/UbuntuBonding>.
12. **Gluster.** GLUSTER. [En línea] <https://www.gluster.org/>.
13. **Ellingwood, Justin.** *How To Set Up Highly Available HAProxy Servers with Keepalived and Floating IPs on Ubuntu 14.04.* [En línea] 23 de Octubre de 2015.  
<https://www.digitalocean.com/community/tutorials/how-to-set-up-highly-available-haproxy-servers-with-keepalived-and-floating-ips-on-ubuntu-14-04>.
14. **Keepalived.** *Keepalived.* [En línea] <http://www.keepalived.org/>.
15. **HAProxy.** *HAProxy Configuration Manual.* [En línea]  
<http://cbonte.github.io/haproxy-dconv/configuration-1.7.html>.
16. **The Linux NIS(YP)/NYS/NIS+ HOWTO.** Setting Up the NIS Client. [En línea]  
[http://www.tldp.org/HOWTO/NIS-HOWTO/settingup\\_client.html](http://www.tldp.org/HOWTO/NIS-HOWTO/settingup_client.html).

17. **University of Wisconsin-Madison.** *HTCondorTM Version 8.5.6 Manual.* [En línea]  
<http://research.cs.wisc.edu/htcondor/manual/v8.5/index.html>.

# 8 Índice de imágenes y tablas

## 8.1 Tablas

Tabla 1: Distribución direcciones IP .....	9
--	---

## 8.2 Figuras

Figura 1: Estructura del clúster.....	8
Figura 2: Esquema Keepalived y HAProxy .....	18

## 8.3 Imágenes

Imagen 1: Creación máquina virtual .....	19
Imagen 2: Tamaño de la memoria.....	20
Imagen 3: Configuración del adaptador de red .....	20
Imagen 4: Selección de la ISO del SO .....	21
Imagen 5: Comienzo instalación Ubuntu .....	22
Imagen 6: Selección del idioma .....	22
Imagen 7: Particionado del disco .....	23
Imagen 8: Instalación OpenSSH.....	23
Imagen 9: /etc/ssh/sshd_config.....	24
Imagen 10: /etc/network/interfaces.....	24
Imagen 11: /etc/fstab .....	25
Imagen 12: /etc/exports.....	25
Imagen 13: /etc/grub.d/40_custom .....	28
Imagen 14: /etc/network/interfaces DHCP.....	28
Imagen 15: ssh-keygen .....	29
Imagen 16: /etc/hosts .....	29
Imagen 17: /etc/rc.local .....	30
Imagen 18: LXDE .....	31
Imagen 19: /boot/pxelinux.cfg/default.....	32
Imagen 20: /etc/initramfs-tools/initramfs.conf .....	32
Imagen 21: Carga mediante PXE .....	35
Imagen 22: /etc/fstab .....	38
Imagen 23: Información volúmenes gluster.....	39
Imagen 24: Monitorización HAProxy.....	44
Imagen 25: Configuración dominio NIS.....	45
Imagen 26: Configuración NIS .....	45
Imagen 27: Instalación MOSIX.....	48
Imagen 28: Configuración CONDOR maestro.....	49
Imagen 29: Configuración CONDOR clientes.....	50



Imagen 30: condor_satus .....	51
Imagen 31: Configuración CONDOR servidor checkpoints .....	51
Imagen 32: Brick replicado san1.....	53
Imagen 33: Brick replicado san2.....	53
Imagen 34: Brick replicado san3.....	54
Imagen 35: Brick distribuido san1 .....	54
Imagen 36: Brick distribuido san2 .....	54
Imagen 37: Brick distribuido san3 .....	54
Imagen 38: bond0.....	55
Imagen 39: Prueba channel bonding.....	56
Imagen 40: Pruebas HAProxy 1 .....	56
Imagen 41: Pruebas HAProxy 2 .....	57
Imagen 42: ssh Virtual IP .....	58
Imagen 43: Pruebas HAProxy 3 .....	58
Imagen 44: Pruebas HAProxy 4 .....	59
Imagen 45: Pruebas Mosix 1 .....	60
Imagen 46: Pruebas Mosix 2 .....	61
Imagen 47: Pruebas Mosix 3 .....	61
Imagen 48: Pruebas Condor 1 .....	63
Imagen 49: Pruebas Condor 2 .....	63
Imagen 50: Pruebas Condor 3 .....	64
Imagen 51: Pruebas Condor 4 .....	65
Imagen 52: Pruebas Condor 5 .....	67
Imagen 53: Pruebas Condor 6 .....	67
Imagen 54: Pruebas Condor 7 .....	68
Imagen 55: Pruebas Condor 8 .....	68