UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Título: Diseño e implementación de una solución de clúster para los servidores de aplicación Ruby on Rails.

Autor:

Orisbel Mora Rodríguez

Tutores:

Ing. José Luis Muñoz Suárez.

Ing. Maikel F. Rosabal Alarcón.

La Habana, junio del 2011.

"Año 53 de la Revolución".

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor del presente trabajo de diploma y se reconoce a la Universidad de las Ciencias Informáticas (UCI) de los derechos patrimoniales del mismo con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de junio del 2011.

Ing. Maikel F. Rosabal Alarcón

_____

Firma del Tutor

Ing. José Luis Muñoz Suárez

_____

Firma del Tutor

Orisbel Mora Rodriguez

_____

Firma del Autor

DATOS DE CONTACTO

Ing. Maikel F. Rosabal Alarcón: Ingeniero en Ciencias Informáticas.
Centro de desarrollo: DATEC
Años de experiencia: 2
Correo electrónico: mfrosabal@uci.cu
Universidad de la Ciencias Informáticas, La Habana, Cuba.

Ing. José Luis Muñoz Suárez: Ingeniero en Ciencias Informáticas.
Centro de desarrollo: DATEC
Años de experiencia: 3
Correo electrónico: jlmunoz@uci.cu
Universidad de la Ciencias Informáticas, La Habana, Cuba.

AGRADECIMIENTOS

A mis padres, a mi mujer que siempre ha estado a mi lado apoyándome, a mis amigos que han compartido momentos malos y buenos durante estos cinco años, a mis tutores, a todo aquel que contribuyó de una forma u otra a la realización de este Trabajo de Diploma.

DEDICATORIA

A mi familia por todo el amor y la confianza brindada en todo momento, a mis padres por todos los sacrificios realizados, por ser pacientes y dedicados en mi educación y en mi formación, a mis hermanos, a mi mujer y a mi hijo.

RESUMEN

Actualmente en la Universidad de las Ciencias Informáticas se utiliza el Redmine como gestor de proyectos, el mismo es desplegado por cada centro de producción de forma independiente; previendo el presente crecimiento del personal productivo en los centros de producción de la UCI es necesario elevar el rendimiento de esta aplicación, ya que es cada vez es mayor el número de conexiones que tiene que soportar concurrentemente pero su actual forma de despliegue no la soporta.

El objetivo principal de la presente investigación científica es el diseño e implementación de una solución de clúster para el despliegue de aplicaciones web desarrolladas con el *framework* Ruby on Rails específicamente para el Redmine.

La propuesta de solución diseñada hace uso de Nginx como balanceador de carga, combinado con Memcached para el acelerado de la aplicación, para los servidores de aplicación se utiliza el servidor web Thin, el *software Network File System* como sistema de archivos compartidos y *Distributed Replicated Block Device* para la réplica de datos entre los servidores de archivos compartidos, para la alta disponibilidad de sus servicios se utiliza el *software* Heartbeat.

PALABRAS CLAVE

Alta disponibilidad, balanceo de carga, clúster, Redmine, Ruby on Rails.

ÍNDICE

INTRODUCCIÓN **1**

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS **4**

Introducción 4

1.1. Tecnología Clúster 4

1.1.1. Clasificación de clúster *4*

1.1.2. Balanceo de carga *6*

1.1.3. Software dedicado al balanceo de carga *7*

1.2. Servidor Web a utilizar 8

1.2.1. Apache *9*

1.2.2. Thin *10*

1.2.3. Lighttpd *10*

1.2.4. Ruby Enterprise Edition (REE) *11*

1.3. Otros componentes necesarios para el clúster 11

1.3.1. HEARTBEAT *11*

1.3.2. Network file System *12*

1.3.3. DRBD *12*

1.3.4. Memcached *13*

1.3.1. IPtables *13*

1.4. Pruebas para validar la solución 14

1.4.1. Pruebas de rendimiento *14*

1.4.2. Pruebas de configuración *14*

1.5. Herramientas para las pruebas 14

1.5.1. JMeter *14*

1.5.2. Sysstat *15*

1.6. Métricas para medir el rendimiento 15

1.7. Conclusiones 15

CAPÍTULO 2: DISEÑO DE LA PROPUESTA DE SOLUCIÓN **17**

Introducción 17

2.1. Diseño de la propuesta de solución 17

ÍNDICE

2.1.1. Diseño lógico del clúster ... *17*
2.1.2. Diseño físico del clúster ... *25*
2.1.3. Diseño de interconexión ... *27*
2.1.4. Diseño de seguridad del clúster ... *28*
CONCLUSIONES ... 29
CAPÍTULO 3: VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN ... 30
INTRODUCCIÓN ... 30
3.1. DESCRIPCIÓN DEL PROCESO DE PRUEBAS ... 30
3.2. PRUEBAS REALIZADAS AL DISEÑO DE LA PROPUESTA DE SOLUCIÓN ... 31
3.2.1. Pruebas de configuración realizadas al diseño de la solución ... 32
3.2.2. Pruebas de carga realizadas al diseño de la solución ... 34
3.3. CONCLUSIONES ... 45
CONCLUSIONES ... 46
RECOMENDACIONES ... 47
REFERENCIAS BIBLIOGRÁFICAS ... 48
BIBLIOGRAFÍA ... 50
ANEXOS ... 53
Anexo 1. Documento de diseño de la solución propuesta. ... 53

ÍNDICE DE FIGURAS

Figuras 1 Arquitectura del clúster. 17
Figuras 2 Recorrido de las peticiones en el balanceador de carga. 19
Figuras 3 Representación lógica cuando falla un nodo. 19
Figuras 4 Fallo del balanceador de carga. 20
Figuras 5 Alta disponibilidad en el balanceador de carga. 21
Figuras 6 Recorrido de las peticiones en el servidor NFS. 22
Figuras 7 Falla en el servidor NFS. 23
Figuras 8 Alta disponibilidad en los servidores NFS. 24
Figuras 9 Diseño lógico del clúster. 25
Figuras 10 Diseño físico de los balanceadores de carga. 26
Figuras 11 Diseño físico en los servidores de aplicación. 26
Figuras 12 Diseño físico en los servidores de archivos compartidos. 27
Figuras 13 Diseño de interconexión 28
Figuras 14 Diseño de seguridad. 29
Figuras 15 Tiempos de respuestas promedio. 36
Figuras 16 Consumo de recursos promedio CPU y RAM en Thin y Apache + passenger. 37
Figuras 17 Consumo de recursos promedio tráfico de red en los servidores de aplicaciones. 37
Figuras 18 Tiempos de respuestas promedio contra número de nodos. 39
Figuras 19 Consumo de recursos promedio CPU y RAM en los servidores de aplicaciones. 40
Figuras 20 Consumo de recursos promedio tráfico de red en los servidores de aplicaciones. 40
Figuras 21 Consumo de recursos promedio CPU y RAM en el balanceador de carga. 41
Figuras 22 Consumo de recursos promedio tráfico de red en el balanceador de carga. 41
Figuras 23 Tiempos de respuestas promedio contra niveles de concurrencia 43
Figuras 24 Consumo de recursos promedio CPU y RAM en los servidores de aplicación. 44
Figuras 25 Consumo de recursos promedio de red en los servidores de aplicación 44
Figuras 26 Consumo de recursos promedio CPU y RAM en el balanceador de carga. 45
Figuras 27 Consumo de recursos promedio de red en el balanceador de carga. 45

ÍNDICE DE TABLAS

Tabla 1 Pruebas de configuración realizadas al diseño de solución del clúster de balanceo de carga. 32

ÍNDICE DE FIGURAS Y TABLAS

Tabla 2 Pruebas de carga realizadas al diseño de solución del clúster de balanceo de carga. 34
Tabla 3 Tiempos de Respuesta promedios del sistema. 35
Tabla 4 Consumo de recursos promedio en Thin y Apache + passenger. 36
Tabla 5 Tiempos de Respuesta promedios del sistema. 38
Tabla 6 Consumo de recursos promedio en los servidores de aplicaciones. 38
Tabla 7 Consumo de recursos promedio en el balanceador de carga. 38
Tabla 8 Tiempos de Respuesta promedios del sistema. 42
Tabla 9 Consumo de recursos promedio en los servidores de aplicaciones. 42
Tabla 10 Consumo de recursos promedio en el balanceador de carga. 42
Tabla 11 Consumo de recursos promedio en el servidor de aplicación Thin. 42

INTRODUCCIÓN

En los últimos años, el uso de las llamadas Tecnologías de la Información y las Comunicaciones (TIC) se ha incrementado debido al flujo ininterrumpido de información que estas nos aportan, información que es esencial para nuestro sistema político, para nuestras instituciones económicas, y en muchos casos para los estilos de vida cotidiana de un número considerable de personas a nivel mundial. Esas tecnologías se presentan cada vez más como una necesidad en el contexto de sociedad donde los rápidos cambios y el aumento de los conocimientos constantemente actualizados se convierten en una exigencia permanente. Dentro de estas TIC la industria del software alcanza un papel fundamental por las posibilidades que esta brinda de acelerar el desarrollo de los países en la salud, educación, economía y otras esferas, ya que esta se encuentra fuertemente sujeta al desarrollo económico de cada nación.

Cuba, a pesar del bloqueo económico, comercial y financiero impuesto por el gobierno de Estados Unidos que le impide tener acceso a su mercado y la obliga a invertir varias veces más recursos al tener que recurrir a mercados muy distantes, basándose sobre todo en sus recursos humanos y optimizando sus recursos materiales y financieros, avanza en su informatización, priorizando el uso social y colectivo de las TICs lo que da al país una connotación diferenciada al resto de los países del mundo en cuanto a Tecnologías de la Información se trata. Como parte de este proceso de informatización llevado a cabo en el país se tomaron varias iniciativas para fomentar su desarrollo, ejemplo fehaciente de ello es la creación en el año 2002 del ”Proyecto Futuro” que luego pasó a ser la Universidad de las Ciencias Informáticas (UCI).

A raíz de esto en la UCI se ha desencadenado un gran número de aplicaciones informáticas que permiten automatizar las actividades que a diario se realizan en los diferentes sectores y esferas de la sociedad. Para la producción de software la UCI cuenta con una Infraestructura Productiva (IP) la cual en su interior está compuesta por centros que se especializan en ramas específicas de la producción. Todos estos centros utilizan el Redmine, aplicación web desarrollada con el *framework* Ruby on Rails[1](RoR), por ser este un completo sistema de gestión de proyectos, el cual entre sus funcionalidades principales permite mantener de forma organizada el desarrollo de los proyectos y aumentar el control del equipo de trabajo.

[1]**Ruby on Rails**: es un *framework* para el desarrollo de aplicaciones web, *software* libre por naturaleza, está basado en el patrón de diseño Modelo Vista Controlador (MVC). Fue creado por David Heinemeier Hansson.

Actualmente cada centro utiliza el Redmine de manera independiente y no de forma centralizada lo que provoca un gasto en cuanto a recursos dedicados a esta forma de despliegue ya que en la UCI hay alrededor de 15 centros y cada uno de estos posee un Redmine. La explotación de esta potente herramienta ha crecido considerablemente y tenerlo centralizado requiere de un gran número de usuarios de manera concurrente accediendo al mismo, por lo que se necesita de una infraestructura tecnológica altamente sofisticada y muy costosa pero los centros de producción de la UCI no cuentan con dicha tecnología ni el presupuesto para obtenerla. Un aspecto fundamental a destacar es que las aplicaciones que son accedidas por varios usuarios de manera simultánea generan grandes volúmenes de carga en los servidores web sobre los que se ejecutan, esto tiene como consecuencia que el sistema tenga tiempos de respuestas inaceptables lo cual repercute directamente en el rendimiento del mismo.

Por todo lo antes expuesto se plantea como **problema de la investigación**: ¿Cómo obtener mejores tiempos de respuestas de los servidores web donde se despliegan aplicaciones Ruby on Rails con alta concurrencia?

A partir del problema planteado se obtiene como **objeto de estudio** los clúster de aplicaciones web enmarcado en el **campo de acción**: clúster de aplicaciones web para sistemas desarrollados en Ruby on Rails.

Teniendo en cuenta la situación problemática descrita anteriormente la presente investigación tiene como **Objetivo General**: Desarrollar una solución de clúster para los servidores de aplicación Ruby on Rails que permita obtener mejores tiempos de respuestas en sus servicios.

Para dar cumplimento al objetivo trazado se definen como **tareas de la investigación**:

- Identificación de las diferentes formas en las que se puede poner una aplicación Ruby on Rails en producción.
- Selección de las técnicas de clúster de balanceo de carga existentes y el software que se utiliza para implementarlo.
- Diseño del clúster de aplicaciones.
- Implementación del clúster de aplicaciones.
- Inclusión de alta disponibilidad, confiabilidad y facilidades de escalabilidad en la solución propuesta.

- Realización de las pruebas de carga y rendimiento al clúster de aplicaciones para su validación.
- Desarrollo del manual de implantación del clúster de aplicaciones.

Con la presente investigación se esperan los siguientes resultados:

- Obtener el diseño de la propuesta de solución que permita mejorar tiempos de respuestas en los servidores web donde se despliegan las aplicaciones Ruby on Rails.
- Manual de instalación y configuración del clúster de aplicaciones.

La presente investigación científica está estructurada en tres capítulos, a continuación se muestra una breve descripción de cada uno de ellos:

Capítulo 1: Fundamentos teóricos
En este capítulo se expone todos los fundamentos teóricos de la investigación, se hace una breve descripción de los distintos servidores web que soportan aplicaciones desarrolladas en RoR y las formas de configuración para poner una aplicación RoR en producción, así como las diferentes técnicas de balanceo de carga existentes.

Capítulo 2: Diseño de la propuesta de solución
En este capítulo inicialmente se muestra el diseño lógico y el diseño físico de la propuesta de solución así como una descripción de ambos, también se realiza un diseño de interconexión y un diseño de seguridad a la propuesta de solución.

Capítulo 3: Validación de la propuesta de solución
En este capítulo se muestra la validación de la propuesta de solución mediante pruebas de configuración y rendimiento realizadas al clúster, también se describen y se analizan los resultados obtenidos en dichas pruebas.

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS

Introducción

En este capítulo se realiza un estudio de la tecnología clúster, los tipos de balanceo de carga que existen actualmente, el *software* que se necesita para su implementación, los diversos servidores web que soportan aplicaciones RoR, las pruebas que pueden realizarse para la validación de la propuesta de solución así como la descripción de otras herramientas que serán utilizadas en la solución.

1.1. Tecnología Clúster

El objetivo principal de la presente investigación científica es diseñar e implementar una solución de clúster para mejorar los tiempos de respuestas en los servidores web donde se despliegan aplicaciones RoR. Para mejorar estos tiempos de respuestas existen técnicas que permiten dicho objetivo, tal es el caso de la tecnología clúster, iniciada por *Digital Equipment Corporation* (DEC) compañía que se dedicó a la fabricación de microcomputadoras a partir de 1957, para aumentar el rendimiento de aplicaciones o servicios, fue sin dudas DEC quien inicialmente definió como clúster a*: "un grupo de computadoras que están interconectadas y funcionan como una sola unidad de proceso de información"* (1).

Un clúster está formado por: ´´ *Un conjunto de máquinas unidas por una red de comunicación trabajando por un servicio conjunto*´´ (1). La idea es que este tipo de máquinas estén dotadas de lo principal para llevar a cabo un proceso, es decir, de un procesador y una memoria.

1.1.1. Clasificación de clúster

El clúster en dependencia de la función que realiza puede clasificarse de las siguientes formas (1):

Clúster de Alto-Rendimiento (HP, *High Performance*): es aquel que está diseñado para dar altas prestaciones en cuanto a capacidad de cálculo. El objetivo de este tipo de clúster es mejorar el rendimiento en la obtención de la solución de un problema, en términos del tiempo de respuesta y también de su precisión, es utilizado en problemas que requieran grandes tiempos de proceso.

Entre las aplicaciones donde son desplegados este tipo de clúster se encuentran:

- Cálculos matemáticos.
- Renderizaciones de gráficos.
- Compilación de programas.

- Compresión de datos.
- Descifrado de códigos.
- Rendimiento del sistema operativo.

Clúster de Alta-Disponibilidad (HA, *High Availability*) : totalmente diferente a un clúster de Alto-Rendimiento. Estos estan diseñados para garantizar el funcionamiento ininterrumpido de ciertas aplicaciones. Su principio de funcionamiento es proporcionar un servicio ininterrumpido las 24 horas del día, los siete días de la semana.

Su arquitectura está compuesta por un conjunto de dos o más máquinas, que se caracterizan por la redundancia de sus recursos, y porque están en constante monitoreo entre sí, en caso de producirse un fallo de hardware o de las aplicaciones en alguna de las máquinas del clúster, el *software* de Alta-Disponibilidad es capaz de migrar automáticamente los servicios que han fallado hacia las otras máquinas que lo componen.

Para garantizar la alta disponibilidad de sus servicios, la tecnología clúster se implementa en base a tres factores fundamentales (2):

- **Fiabilidad:** Probabilidad de un funcionamiento correcto.
- **Disponibilidad:** La calidad de estar siempre presente, listo para el uso, a mano, accesible.
- **Dotación de servicio:** Debe existir un servicio proporcionado por el clúster.

Este clúster esta diseñado especialmente para resolver múltiples problemas como:

- Sistemas de información redundante.
- Sistemas tolerantes a fallos.

Clúster de Balanceo de Carga (LB, *Load Balancing*) : se encarga de colocar en paralelo varios servidores (servidores reales) capaces de brindar el mismo servicio, y repartir de alguna forma la carga de trabajo entre ellos, tal que los clientes vean servidas sus peticiones en tiempos menores y aceptables. Los clientes deben ver este conjunto de servidores como si fuera uno solo (servidor virtual). Un clúster de balaneo de carga tiene peculiaridades de clúster de alta disponibilidad y alto rendimiento.

Dentro de la tecnología clúster, el balanceo de carga constituye una de las técnicas más utilizadas para mejorar los tiempos de respuestas así como aliviar la carga en el servidor web , pero la implantación de la misma generalmente termina combinando las demás técnicas de la tecnología clúster.

En la presente investigación se utilizará para mejorar los tiempos de respuestas de los servidores web donde se despliegan aplicaciones RoR la técnica balanceo de carga combinado con alta disponibilidad.

1.1.2. Balanceo de carga

El balanceo de carga se define como: ´´ *El balance o balanceo de carga es un concepto usado en informática que se refiere a la técnica usada para compartir el trabajo a realizar entre varios procesos, ordenadores, discos u otros recursos. Está íntimamente ligado a los sistemas de multiprocesamiento, o que hacen uso de más de una unidad de procesamiento para realizar labores útiles*´´ (2).

Su principio de funcionamiento consiste en compartir la carga de trabajo y tráfico de los clientes hacia los servidores que éstos acceden. Al balancear carga se mejora el tiempo de respuesta, acceso y confiabilidad. La caída de un servidor no influye en el funcionamiento de todo el clúster ya que las funciones de este son asumidas por el resto de los servidores. Cuando la carga de trabajo sea mayor se pueden añadir más servidores al clúster y escalar este sistema para garantizar el balanceo de carga.

Para el balanceo de carga existen métodos que definen el tipo de balanceo realizado, que pueden mejorar el rendimiento del servidor virtual. Entre estos métodos se pueden encontrar:

1.1.2.1. Balanceo de Carga por DNS

Este método consiste en crear un dominio DNS[2] al cual se le asignan diferentes direcciones de IP[3] pertenecientes a los servidores que están funcionando, a esta configuración no se le considera un clúster debido a que en la cache del explorador de Internet de los clientes se almacena la dirección de IP del servidor que en ese momento le atendió. Los clientes automáticamente redireccionan las peticiones refiriéndose a la dirección del servidor de la caché cuando un nuevo requerimiento se produce. El

[2]**Sistema de Nombres de dominio o DNS (Domain Name System**): El sistema de nombres de dominio es un estándar de Internet. El propósito de DNS es crear un sistema que permita realizar búsquedas en una base de datos tipo árbol. Estas búsquedas se realizan para la obtención de la dirección de IP o del nombre de host que pertenece a un nodo que se encuentra en el sistema de nombres de dominio.

[3]**IP:** *Internet Protocol*, es un protocolo de comunicaciones de la capa de red de la arquitectura TCP/IP. Es un protocolo no orientado a conexión, no confiable.

balanceo de carga no se produce por completo ya que un servidor puede atender solicitudes de gran procesamiento, mientras otros pueden atender solicitudes más sencillas, esto se debe a que carecen de un dispositivo que controle y redireccione equitativamente el trabajo a los servidores, la falta de este dispositivo hace que la repartición de carga no sea equitativa entre los servidores web (3).

Cuando un servidor queda fuera de servicio, los clientes que eran atendidos por este van a continuar enviando requerimientos sin que sus peticiones sean cumplidas, quedando sin atención hasta el momento en que su caché se actualice y aprenda la dirección de un servidor operativo.

1.1.2.2. Balanceo de Carga mediante Nodo Director

Este mecanismo está basado en la utilización de un nodo director. El nodo director recibe todas las peticiones de los clientes, balancea carga y redirecciona las peticiones a los servidores del clúster. El nodo director vuelve a enrutar las peticiones a los nodos, una vez procesada la petición, los servidores devuelven el resultado al nodo director para que este entregue los resultados al cliente. Existen variaciones donde los nodos o servidores pueden entregar el resultado al cliente directamente (3).

1.1.3. Software dedicado al balanceo de carga

Actualmente existe una considerable cantidad de *software* dedicado al balanceo, los cuales se diferencian en la forma en que balancean la carga. Uno de los más utilizados en plataformas libres es Linux Virtual Server (LVS).

1.1.3.1. Linux Virtual Server (LVS)

Es una solución para gestionar balance de carga en sistemas Linux. Es un proyecto de código abierto iniciado por Wensong Zhang en mayo de 1998. El objetivo es desarrollar un servidor en Linux de alto rendimiento que proporcione buena escalabilidad, confiabilidad y robustez usando tecnología clúster (4).

El LVS es un servidor altamente escalable y altamente disponible construido sobre un clúster de servidores reales, con balanceador de carga corriendo en un Sistema Operativo Linux. La arquitectura del clúster de servidores es completamente transparente a los usuarios finales y los usuarios interactúan como si fuera un solo servidor virtual de Alto-Rendimiento.

1.1.3.2. Crossroads

Este es otro potente software, libre y diseñado con el propósito de balanceo de carga en servicios TCP como: HTTP, HTTPS, SMTP, SSH y DNS. Su principio de funcionamiento es balancear carga de forma normal, pero además este brinda facilidades para la recuperación ante fallos en servicios reales, también puede ser utilizado en sistemas sin discos de almacenamiento (5).

1.1.3.3. Nginx

Este es un servidor web, que constituye un balanceador, ya que mediante el módulo *Upstream* permite realizar balanceo de carga inteligente mediante el algoritmo Run-Rubin, que consiste en repartir una petición a cada servidor de aplicación publicado con Apache, Lighttpd, Thin o incluso el propio Nginx hasta terminar de reenviar todas las peticiones recibidas por el balanceado. Como balanceador de carga, es capaz de detectar la caída de uno de sus servidores reales, dejando de enviarle peticiones a este y repartirlas entre los demás nodos del clúster, utiliza el mecanismo nodo director donde todos las peticiones pasan a través de él antes de ser enviadas a los servidores web, hace uso de *revese proxy* acelerado sin caché, es conocido por su estabilidad, gran conjunto de características, configuración simple, y bajo consumo de recursos (6).

Como balanceador de carga en la presente propuesta solución se hará uso del módulo *upstream* del servidor web Nginx ya que usa un balanceo de carga inteligente entre los servidores reales detectando la caída de los mismos y fue desarrollado para aumentar el rendimiento en aplicaciones RoR.

1.2. Servidor Web a utilizar

Un clúster de balanceo de carga no solo está compuesto por el balanceador, también dentro de su arquitectura están los servidores reales entre los que el balanceador de carga repartirá la carga recibida por el cliente, en estos servidores reales estarán desplegadas las aplicaciones web, en el caso de la presente investigación el Redmine, para esto es necesario de un servidor web que soporte aplicaciones desarrolladas en RoR.

Actualmente hay un gran número de servidores web que permiten publicar aplicaciones web desarrolladas en diversos lenguajes de programación e innumerables plataformas, tal es el caso de Ruby, un lenguaje de programación que permite desarrollar aplicaciones web mediante el *framework* RoR, algunos de estos servidores son Apache, Ligthttpd y Ruby Enterprise Edition (REE).

1.2.1. Apache

Apache es un servidor web de código abierto multiplataforma y modular, se desarrolla dentro del proyecto HTTP Server (httpd) de la empresa Apache Software Fundation. Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, tiene amplia aceptación en la red: desde 1996, es el servidor HTTP más usado alcanzando su máxima cuota de mercado en 2005 siendo el servidor empleado en el 70% de los sitios web en el mundo, sin embargo ha sufrido un descenso en su cuota de mercado en los últimos años. Sus principales ventajas son: es modular, es decir, se le pueden añadir funcionalidades que no tiene por defecto. Algunas de estas funcionalidades son: SSL para comunicaciones seguras vía TLS, *Rewrite* para la reescritura de direcciones (generalmente utilizado para transformar páginas dinámicas como php en páginas estáticas html), Auth_ldap lo que permite autentificar usuarios contra un servidor LDAP, soporte para páginas dinámicas en lenguaje Perl, soporte para páginas dinámicas en lenguaje PHP, soporte para páginas dinámicas en lenguaje .NET de *Microsoft,* filtrado a nivel de aplicación para seguridad, *OpenSource* es decir de código abierto, multiplataforma lo podemos ejecutar en Unix, Linux, Windows, MacOS. Además este potente servidor web cuenta con varias formas de configuración para publicar una aplicación web desarrollada en RoR, tales como (7):

Apache + passenger: es una configuración muy sencilla y utilizada por aquellos que desean publicar una aplicación sin ningún tipo de complicación, passenger es una gema[4] de Ruby que ayuda a que Apache interprete de forma eficiente el lenguaje Ruby, es muy sencillo de instalar y configurar, con passenger los procesos de RoR son descendientes de Apache, es una de las configuraciones más habituales, debido a su sencillez, y su gran escalabilidad, entre los problemas más notables de esta configuración está el gran consumo de memoria y recursos **(8)**.

Apache + mod_fastcgi: la configuración de la aplicación con el servidor y su mantenimiento, con esta variante de Apache es un tanto compleja. FastCGI es una alternativa al CGI[5] estándar, cuya diferencia principal radica en el hecho que el servidor crea un único proceso persistente por cada programa FastCGI en lugar de uno por cada solicitud del cliente y el módulo para Apache, mod_fastcgi es inestable y puede presentar dificultades en cuanto al rendimiento de la aplicación publicada**.**

[4]**Gemas:** Son paquetes que trae el lenguaje Ruby para integrarlo con otros lenguajes e incluso aplicaciones.

[5]**CGI**: Colección de scripts esenciales para la compilación de Ruby on Rails.

Apache + mongrel: esta configuración tiene mayor complejidad, *mongrel* es una biblioteca Ruby que es utilizada para desplegar servidores web, está diseñada para tener un buen rendimiento sirviendo contenido dinámico, pero no estático, es una opción muy robusta, cuenta con un gran manejo de memoria, mejor integración con los *plugins* de RoR, y muy buena rapidez. Mongrel es una de las maneras más simples y escalables para desplegar aplicaciones en RoR, pueden ser distribuidas transparentemente entre servidores independientes y pueden separarse los procesos de los usuarios, en fin, con *mongrel* Apache sólo actúa como pasarela para el contenido dinámico, es recomendable para un servidor con varias aplicaciones RoR publicadas, su mayor dificultad radica que a menudo se debe iniciar y reiniciar los procesos manualmente.

1.2.2. Thin

Es un servidor web realizado en Ruby, catalogado como el más seguro, estable, rápido y ampliable servidor web existente para Ruby, el cual utiliza tres de las mejores librerías de este lenguaje por lo que tiene una gran eficiencia:

- Mongrel parser.
- Event Machine.
- Rack.

Esto le da velocidad, seguridad y una interfaz mínima entre servidores web y el *framework* RoR, además es fácil de instalar y configurar (9).

1.2.3. Lighttpd

Este es un servidor HTTP o HTTPS, seguro, rápido, que respeta estándares y consume muy pocos recursos, todo ello con un código limpio y elegante, ideal para ser usado en entornos donde la carga es máxima, se requieren respuestas rápidas y alta escalabilidad, sus formas de configuración para desplegar aplicaciones RoR son (10):

Lighttpd + fastcgi: *Fastcgi* es una extensión para *CGI*, esto provee un alto rendimiento sin limitaciones específicas para el servidor. Esta opción es una de las más rápidas, altamente configurable y de alta escalabilidad.

Lighttpd + Pound: *Pound* es utilizado por Lighttpd para el balanceo de carga, pero no es confiable ya que

ha tenido dificultades en cuanto al rendimiento. Este servidor tiene muy buena reputación en aplicaciones de mucha concurrencia pero en otro tipo de aplicaciones como es el caso de las desarrolladas con RoR no es muy utilizado por su pobre integración con RoR y su complicada configuración **(10)**.

1.2.4. Ruby Enterprise Edition (REE)

Este es un servidor orientado a la distribución oficial del intérprete Ruby que entre sus características podemos encontrar:

- Una copia por escritura, colector amistoso de basura.
- Tiene un gestor de memoria que mejora el rendimiento.
- La capacidad de ajustar la configuración recolector de elementos para el rendimiento máximo del servidor.
- La capacidad de inspeccionar el estado del colector de basura y el estado del objeto del montón, con fines de depuración.
- Varios ajustes de administración de memoria para que el intérprete de Ruby utiliza menos memoria en promedio (11).

Es conocido sobre todo por su perfecta integración con *passenger* que es utilizado para el despliegue de aplicaciones desarrolladas con RoR.

Como servidor de aplicaciones web para el diseño del clúster de balanceo de carga se usará Thin ya que es un servidor web desarrollado en Ruby, por lo que no necesita de una gema para la integración de las aplicaciones RoR como el caso de la mayoría de los demás servidores estudiados, además de ser rápido, estable, y de fácil configuración, consume poca memoria RAM no así el caso de Apache que consume una gran cantidad de memoria y recursos del ordenador.

1.3. Otros componentes necesarios para el clúster

1.3.1. HEARTBEAT

Es un componente esencial a la hora de diseñar un clúster de balanceo de carga ya que el balanceador constituye un punto de falla único. Este software fue desarrollado para mantener alta disponibilidad en servicios que se ejecutan sobre el Sistema Operativo Linux. Es utilizado para garantizar alta disponibilidad

de los servicios. Es portable y funciona sobre cualquier plataforma Linux y también sobre FreeBSD y Solaris, ofrece detección de nodos caídos, comunicación y gestión de clúster en un solo proceso (12).

Para garantizar la alta disponibilidad Heartbeat utiliza un sistema de latidos para que los nodos chequeen si el resto de ellos están funcionando. Cuando el servidor secundario deja de escuchar los latidos del servidor primario por determinado periodo de tiempo, asume que este dejó de funcionar y toma su lugar, permitiendo que no se interrumpan los servicios que brindaba el servidor caído. Es recomendable utilizar uno o varios recursos dedicados para la comprobación de los otros nodos. Para el chequeo del estado de los servidores Heartbeat mantiene una comunicación directa y actualmente soporta conexiones de puerto serie interfaz de tipo Ethernet (12).

1.3.2. Network file System

Las aplicaciones web no solo guardan datos en el servidor de base de datos, sino que muchas lo hacen también en disco, como es el caso del Redmine, es por esto que existe la necesidad de encontrar una solución que permita que todos los datos en disco estén accesibles por todos los servidores de aplicación.

Network File System (sistema de archivos en red, NFS) se utiliza para la implementación de sistemas de archivos distribuidos en un entorno de red de computadoras de área local, esto posibilita que varios ordenadores conectados a una misma red accedan a ficheros remotos como si se tratara de ficheros locales (2).

NFS está compuesto por dos partes fundamentales: el servidor y los clientes, en el servidor se centraliza toda la información que pueda ser accedida de forma remota por los clientes, evitando la replicación de datos (2).

1.3.3. DRBD

No basta solo que los datos guardados en disco estén accesibles sino también deben estar sincronizados, DRBD permite esto pues es un Dispositivo de Bloques Replicado y Distribuido (*Distributed Replicated Block Device*, DRBD), permite hacer mirror o espejo remoto en tiempo real (equivalente a RAID-1 pero en red), algo muy difícil de conseguir con otros sistemas como rsync ya que éste no puede trabajar en tiempo real por su consumo de memoria y CPU.

DRBD crea un dispositivo de bloques drbd0 accesible desde ambos servidores. El servidor primario es el que tiene acceso RW en el dispositivo drbd0: cada vez que escribe algo en drbd0 lo escribe en la partición física y esos mismos datos se envían por TCP/IP al servidor secundario (que tiene acceso de sólo lectura) consiguiendo que ambas particiones físicas estén sincronizadas, exactamente igual que un RAID-1. A partir de la versión 0.8 DRBD soporta dos servidores primarios, es decir, que ambos servidores leen y escriben en drbd0 indistintamente (13).

1.3.4. Memcached

Otra forma de acelerar los tiempos de respuestas en los servidores web es la utilización de un sistema de cacheo de objetos en memoria que permite en un espacio de memoria asignada, almacenar datos como: secciones de usuarios, consultas a páginas, que una vez repetidas demoran la mitad del tiempo que demoraron en ejecutarse la primera vez que fueron realizadas, entre la lista de software que permiten esto destaca Memcached software definido por *Danga Interactive*, la empresa que lo desarrolló y mantiene el proyecto bajo licencia BSD como un "*sistema distribuido de alto rendimiento para el cacheo de objetos en memoria, genérico por naturaleza, pero pensado para incrementar la velocidad de aplicaciones web dinámicas, aliviando la carga de las bases de datos*" (13).

Es usado por numerosos *webmasters* dada la necesidad de incrementar velocidades de respuesta para las peticiones web en sitios de tráfico masivo, actualmente su uso continúa expandiéndose a medida que el proyecto toma mayor fuerza con la ayuda de varios desarrolladores de la comunidad Open Source que revisan y agregan nuevas capacidades al proyecto (13).

1.3.1. IPtables

En la actualidad uno de los puntos más críticos es la seguridad, muchos sistemas están expuestos a ataques ya que la mayoría no dispone de una infraestructura que los proteja. La configuración correcta de un cortafuego se basa en conocimientos considerables de los protocolos de red y de la seguridad de la computadora. Errores pequeños pueden dejar a un cortafuego como una herramienta sin valor.

IPtables es un sistema de firewall vinculado al kernel de linux que se ha extendido enormemente a partir del kernel 2.24 de este sistema operativo. Con iptables lo que se hace es aplicar reglas, un firewall de iptables es simplemente un script en el que se van ejecutando las reglas de firewall (14).

1.4. Pruebas para validar la solución

A la hora de diseñar el clúster es necesario medir la calidad del diseño que se realizará, para esto se realizan un grupo de pruebas para validar la propuesta de solución.

1.4.1. Pruebas de rendimiento

Prueba que se implementa para la determinación de la rapidez con que se realiza una tarea en un sistema bajo condiciones particulares de trabajo. Además estas pruebas son útiles para validar y verificar otros atributos de la calidad del sistema como el uso de recursos y la escalabilidad.

1.4.1.1. Pruebas de carga

Esta prueba es una subcategoría de las pruebas de rendimiento, su objetivo es comprender el comportamiento de una aplicación ante una carga determinada. Esta carga puede estar determinada por la conexión de un número de usuarios conectados de manera concurrente al sistema en un tiempo determinado. La prueba de carga como resultado dará el tiempo de respuesta de todas las transacciones críticas con el monitoreo del servidor de aplicación.

1.4.2. Pruebas de configuración

El objetivo de esta prueba es garantizar que la aplicación funcione correctamente sobre diferentes configuraciones de hardware y/o software. Mientras dure la realización de la misma tiende a ocurrir un ciclo de perfeccionamiento en el diseño de la solución. Para la realización de estas pruebas es necesaria la utilización de herramientas que permitan generar carga y monitorear el comportamiento de los recursos del sistema durante la misma.

1.5. Herramientas para las pruebas

Existen un gran número de herramientas libres que pueden ser utilizadas en el período de pruebas ya sea por su completo desarrollo en las mismas como en su facilidad de uso, entre las que se pueden encontrar:

1.5.1. JMeter

Aplicación de escritorio desarrollada sobre Java. Originalmente fue diseñada para realizar pruebas de carga y medir el rendimiento de aplicaciones web, pero su uso se ha extendido a otros tipos de pruebas. Actualmente JMeter se puede utilizar para probar el rendimiento en recursos estáticos y dinámicos. Con esta herramienta es posible generar una pesada carga a servidores web, ftp y de bases de datos, para analizar su comportamiento bajo diferentes niveles de carga (8).

1.5.2. Sysstat

El paquete Sysstat contiene utilidades para monitorizar el rendimiento del sistema y la actividad del mismo. Con esta herramienta se pueden monitorear varios recursos del sistema tales como tasa de transferencia a disco, actividad de la red, memoria RAM y utilización de espacio en SWAP, uso de CPU, entre otras (15).

1.6. Métricas para medir el rendimiento

Para la evaluación del rendimiento de una aplicación, es necesario hacer mediciones del comportamiento de algunas de sus características principales. En el momento que se realizan dichas mediciones, se recoge información referente al funcionamiento de todo el sistema y/o de los servidores que lo conforman, donde las más comunes son:

- Número de peticiones respondidas por segundo.
- Número de peticiones completadas.
- % de peticiones fallidas.
- Tiempo de respuesta promedio del sistema.

A la hora de tratar el comportamiento de los servidores que forman parte del sistema las mediciones se enfocan en el consumo de los recursos, donde las realizadas en los servidores por lo general son:

- % de uso de CPU.
- % de consumo de memoria RAM.
- % de uso de la red.

1.7. Conclusiones

Después de realiza un estudio de la tecnología clúster, los tipos de balanceo de carga que existen actualmente, el *software* que se necesita para su implementación, los diversos servidores web que soportan aplicaciones RoR y las pruebas que pueden realizarse para la validación de la propuesta de solución, se concluye:

- La implementación del clúster que constituye la base de la propuesta de solución será de balanceo de carga utilizando el método nodo director.
- El *software* para el balanceo de carga será el servidor web Nginx, la alta disponibilidad en el sistema será manejada con Heartbeat y Memcached para el cacheo de objetos en memoria.

- Como servidor de aplicaciones web se usará Thin por su perfecta integración con Nginx.
- Para la validación de la propuesta de solución es necesario hacer pruebas de configuración y de rendimiento haciendo uso de herramientas especializadas en este tipo de tareas como JMeter y para el monitoreo de los recursos se usará el paquete Sysstat.
- Para el almacenamiento de datos compartidos se utilizará el *software* NFS combinado con DRBD y Heartbeat para brindar alta disponibilidad.
- Las métricas para medir el rendimiento del sistema serán: tiempo de respuesta promedio del sistema, mientras que para medir el rendimiento de los servidores se usarán como métricas: % de uso del CPU, % uso de la memoria RAM y el % del uso de la red.

CAPÍTULO 2: DISEÑO DE LA PROPUESTA DE SOLUCIÓN

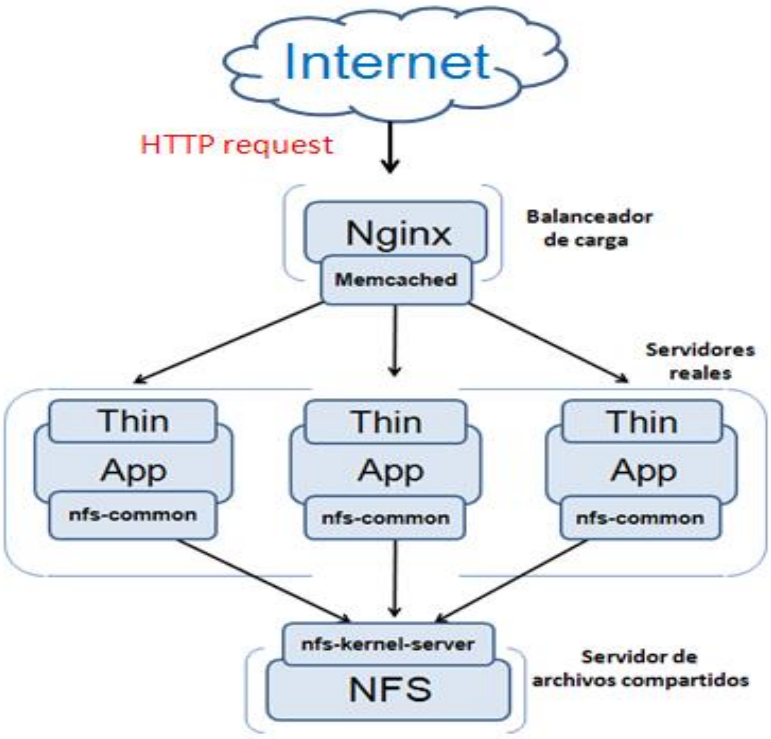
Introducción

El siguiente capítulo está dedicado al diseño de la propuesta de solución de clúster de balanceo de carga, para mejorar los tiempos de respuestas en servidores web donde se despliegan aplicaciones RoR, en este se muestra el diseño lógico, físico y de interconexión de la propuesta de solución, así como un diseño de seguridad.

2.1. Diseño de la propuesta de solución

2.1.1. Diseño lógico del clúster

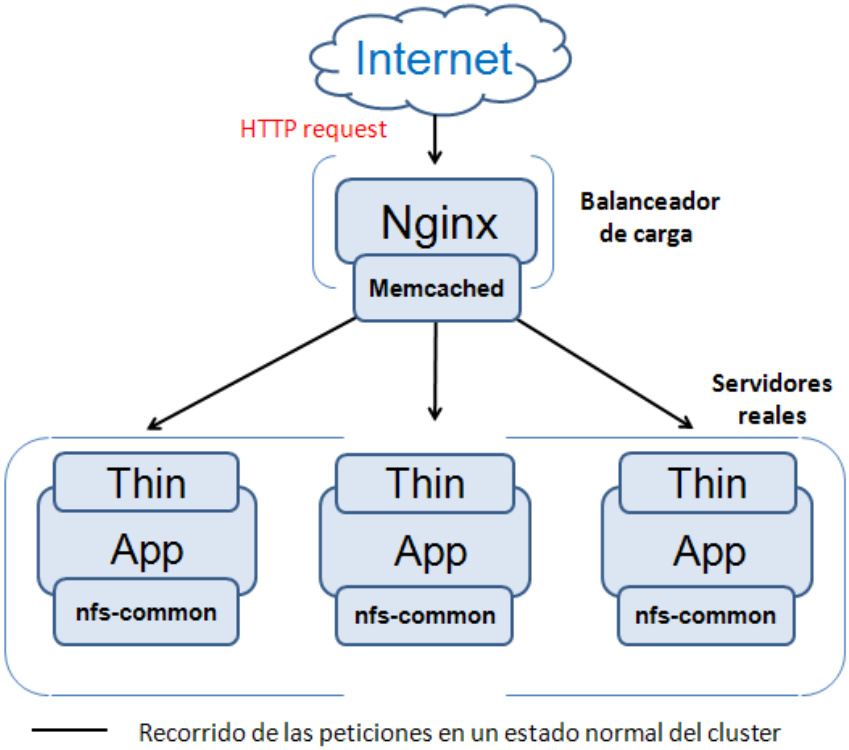
El clúster de balanceo de carga en su arquitectura básica está compuesto por un balanceador de carga, tres servidores de aplicación donde estará publicado el Redmine y un servidor de archivos compartidos (ver Fig. 1). Este debe ser desplegado sobre el Sistema Operativo GNU/Linux en específico la distribución Debian Squeeze, se deberá instalar esta distribución sin entorno gráfico, el idioma con que contará la misma será el Inglés y la zona horaria de Cuba, teniendo correctamente configuradas las fuentes APT (*Advanced Packaging Tool*).



Figuras 1 Arquitectura del clúster.

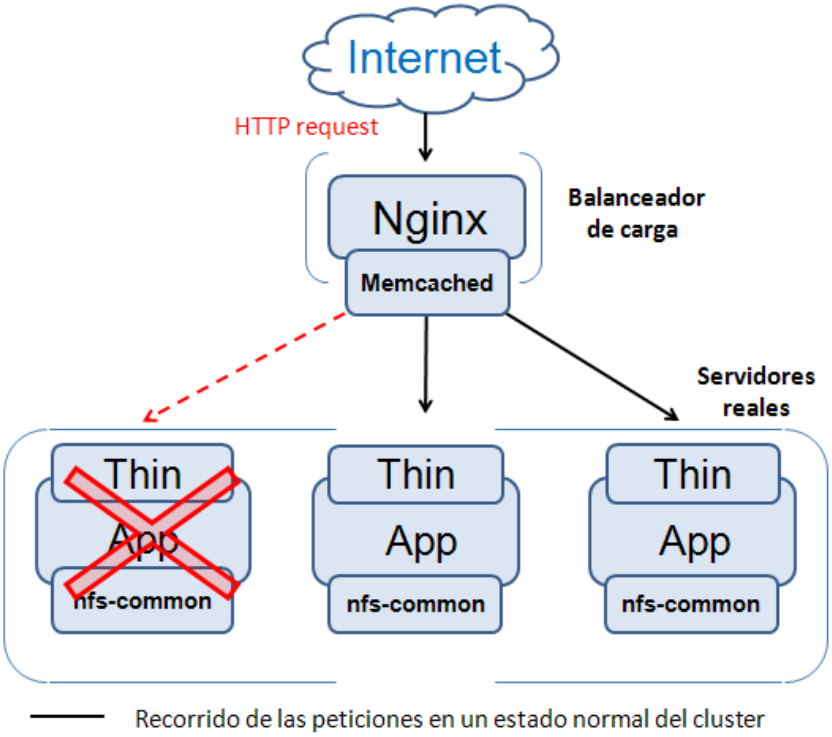
La arquitectura inicial muestra las herramientas que serán utilizadas en cada uno de los servidores: en el balanceador de carga se instalará el *software* Nginx en su versión 0.8.54 este permitirá balancear la carga recibida entre los servidores de aplicación y Memcached en la versión 1.4.5 que será utilizado para dotar a Nginx de una caché que le permita acelerar su funcionamiento, en los servidores de aplicación se instalará el *software* Thin como servidor web en su versión 1.8 para la publicación del Redmine y nfs-common en su versión 1.2.2 para acceder al servidor de archivos compartidos, mientras que el servidor de archivos compartidos la herramienta a utilizar será nfs-kernel-server en la versión 1.2.2 la cuál permitirá exportar los archivos que serán utilizados por los servidores de aplicaciones.

Todas las peticiones enviadas por el cliente al servidor, pasan primero por el nodo donde se encuentra el balanceador de carga el que tiene como objetivo fundamental, repartir las peticiones entrantes entre cada uno de los nodos donde está montada la aplicación Redmine disminuyendo de esta forma la carga existente entre los mismos, para la aceleración de los tiempos en de respuesta en el balanceador de carga se utiliza el *software* Memcached el cuál almacena en caché información que permite a Nginx acelerar su funcionamiento en caso de que esa petición sea realizada nuevamente, ya sea información de los usuarios, todo lo que pasa por Nginx es cacheado (ver Fig. 2).



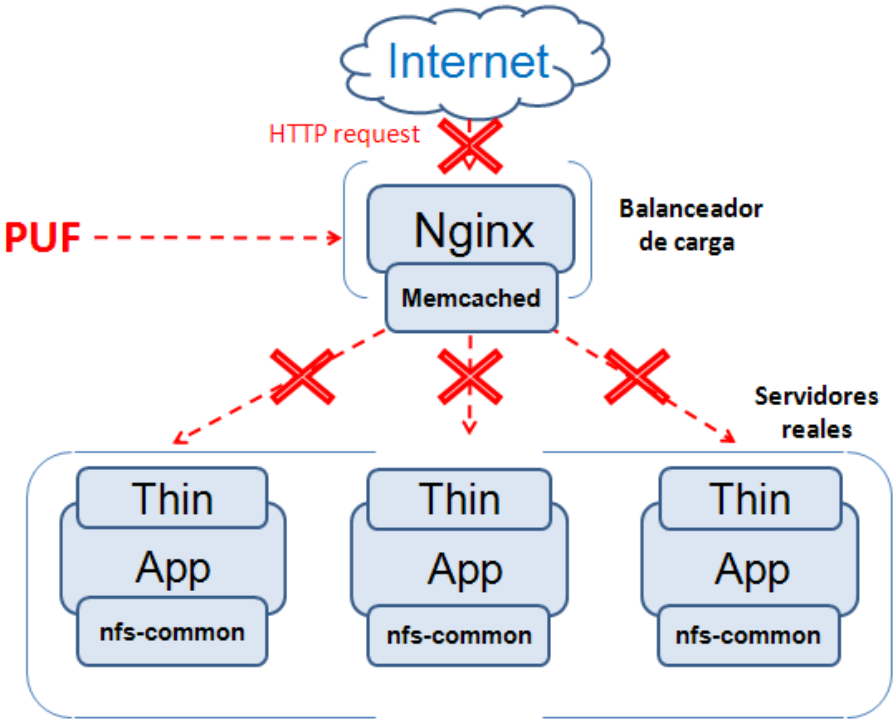
Figuras 2 Recorrido de las peticiones en el balanceador de carga.

Una vez llegadas estas peticiones a los servidores de aplicaciones, son respondidas al cliente a través del balanceador de carga. Cuando el balanceador de carga detecta que un nodo del clúster interrumpe su funcionamiento, deja de enviarle peticiones a este nodo y continúa balanceando carga con los demás nodos que conforman el clúster hasta que el mismo recupere su estado (ver Fig. 3).



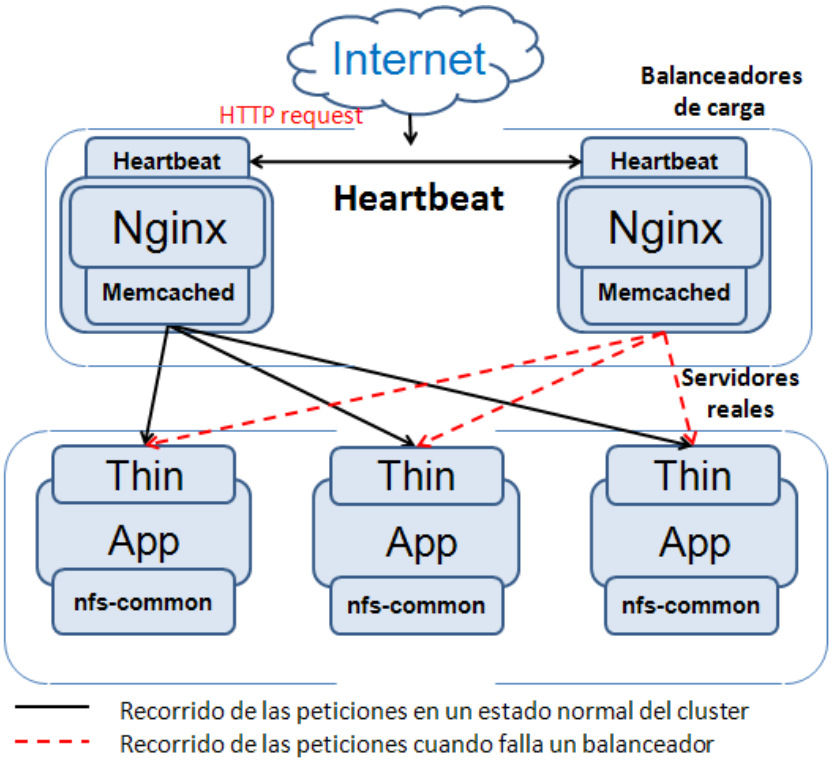
Figuras 3 Representación lógica cuando falla un nodo.

Hasta el momento la arquitectura diseñada tiene un posible punto de falla único (PFU), que constituye la caída de todo el sistema, este se encuentra en el balanceador de carga, ya que una vez que este deje de prestar servicios los clientes dejaran de tener comunicación con el sistema (ver Fig. 4), o sea que un PUF es aquel recurso o nodo por llamarlo de alguna forma, que cuando falla provoca que el sistema deje de prestar sus servicios.



Figuras 4 Fallo del balanceador de carga.

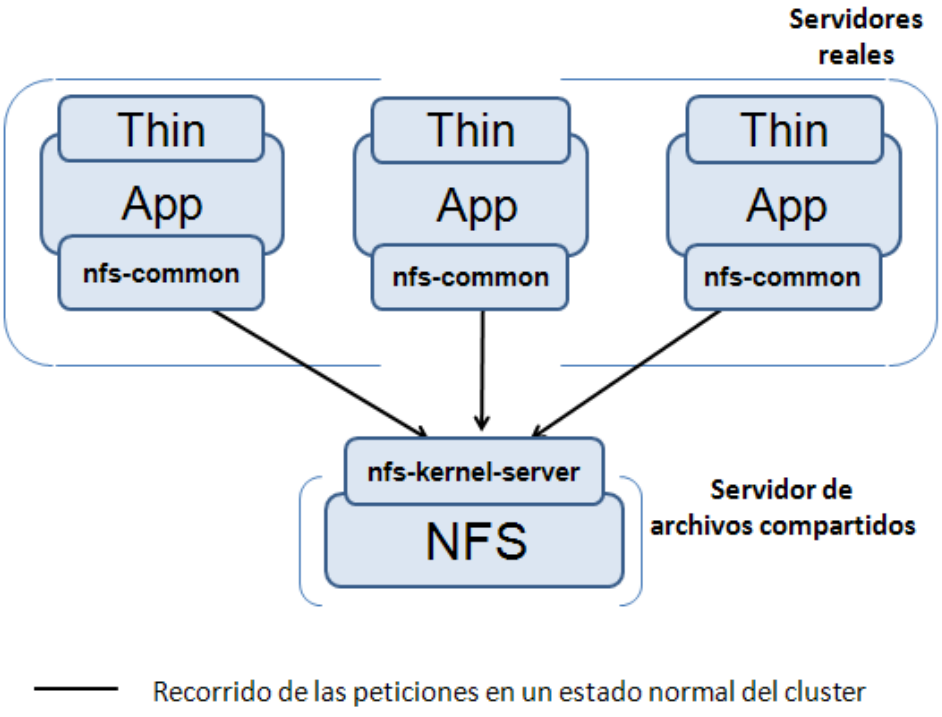
Para dar solución a esta problemática se agregará al actual diseño de la solución un balanceador de carga que actuará como respaldo, el cuál, en caso de falla en el balanceador de carga primario los servicios que antes brindaba dicho servidor continuarán brindándose por el balanceador de carga de respaldo (ver Fig. 5), Esto se puede realizar con ayuda de la herramienta Heartbeat que será utilizada en la solución en su versión 3.0.4 para la recuperación ante fallos y alta disponibilidad en los balanceadores de carga.



Figuras 5 Alta disponibilidad en el balanceador de carga.

Una vez que el balanceador principal se restablece de su falla, el mismo se convierte en balanceador secundario colocándose en espera de ocurrencia de alguna falla, y el balanceador que antes era secundario pasa a ser primario tomando el control del sistema.

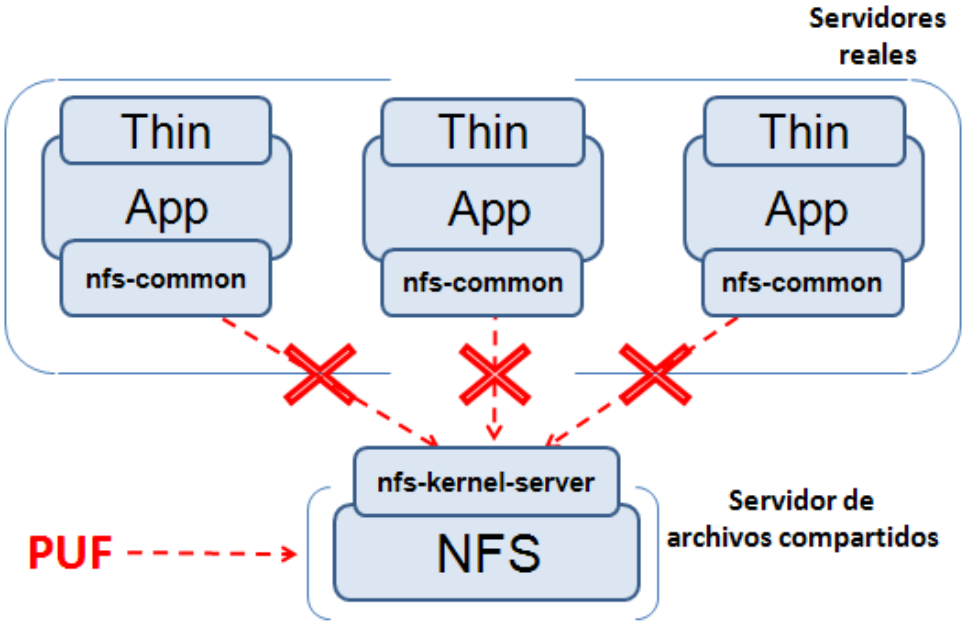
En el caso del servidor de archivos compartidos todos los servidores de aplicación acceden al servidor de archivos compartidos realizando operaciones de escritura y lectura como si lo hicieran en un disco local (ver Fig. 6). Este servidor de archivos compartido solo exportará datos a servidores de aplicación específicos que constituyen a su vez clientes NFS.



Figuras 6 Recorrido de las peticiones en el servidor NFS.

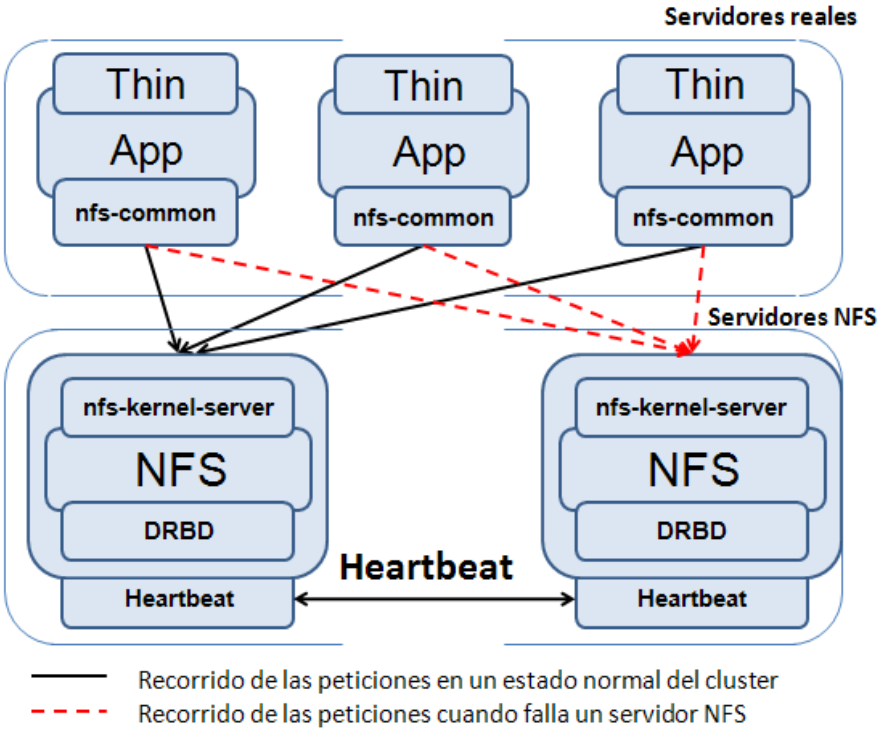
Cuando llegan las peticiones de lectura o escritura al servidor NFS, son respondidas a cada servidor de aplicación. Ante el fallo de un servidor de aplicación cuando este se recupera los archivos no sufren daños ya que se crean, se modifican y se leen en el servidor NFS directamente.

En caso de que falle el servidor NFS este deja inhabilitado a todo el sistema por lo que constituye un PUF ya que no se podrá acceder a los archivos publicados en el mismo (ver Fig. 7).



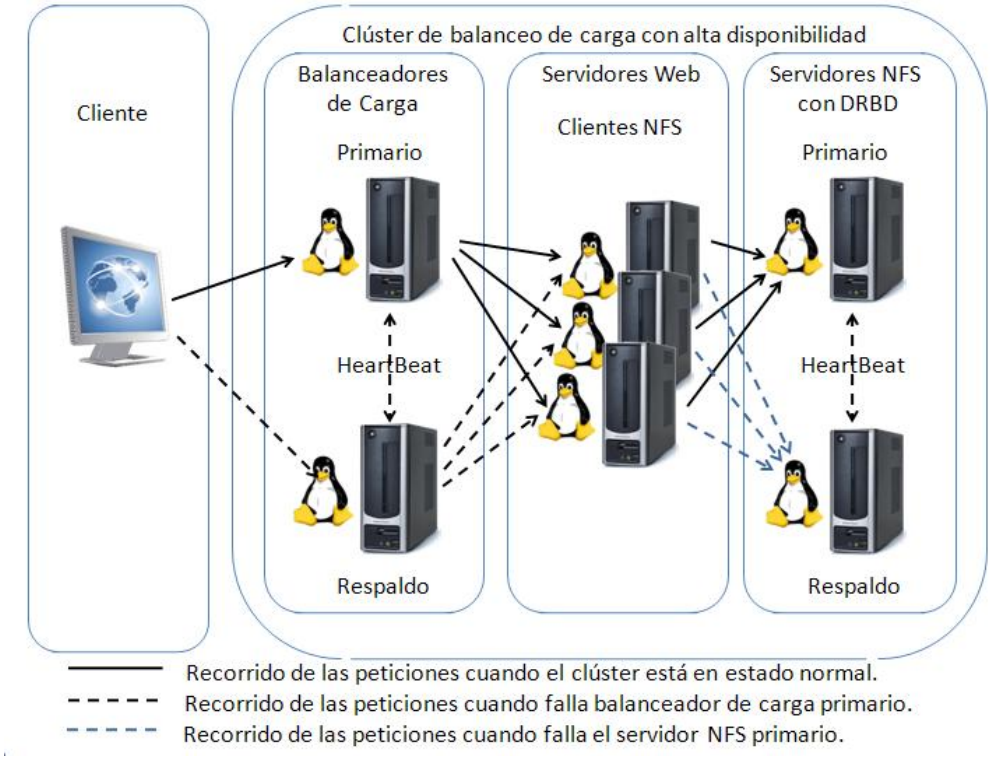
Figuras 7 Falla en el servidor NFS.

Para solucionar dicho problema se agregó al diseño del clúster un servidor NFS de respaldo y con el uso de Heartbeat se garantizó la alta disponibilidad en los servidores de archivos compartidos (ver Fig. 8), con Heartbeat se migraban los servicios del servidor NFS pero con esto no bastaba pues también era necesario migrar los archivos que se habían creado en el servidor, para esto se utilizó la herrameinte DRBD en su versión 8.3 la cuál permitía mantener sincronizados los dispositivos de bloques (los discos duros o particiones) donde se encontraban estos archivos en los dos servidores NFS. Todo esto es manejado por Heartbeat, el montaje de los discos en los servidores NFS, la sincronización de sus archivos mediante DRBD. Cuando el servidor NFS primario tiene una falla todos los recursos de este son migrados al servidor secundario el cuál pasa a ser en ese momento servidor NFS primario (ver Fig. 8).



Figuras 8 Alta disponibilidad en los servidores NFS.

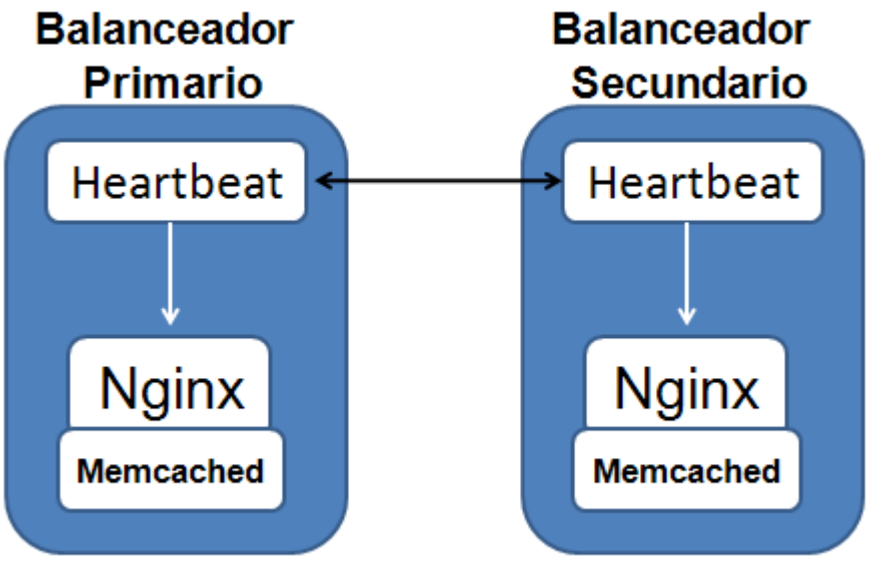
Ya restablecido el servidor NFS que falló pasa a ser respaldo en espera de alguna falla. El diseño lógico del clúster de balanceo de carga quedaría finalmente como se muestra en la figura 9.



Figuras 9 Diseño lógico del clúster.

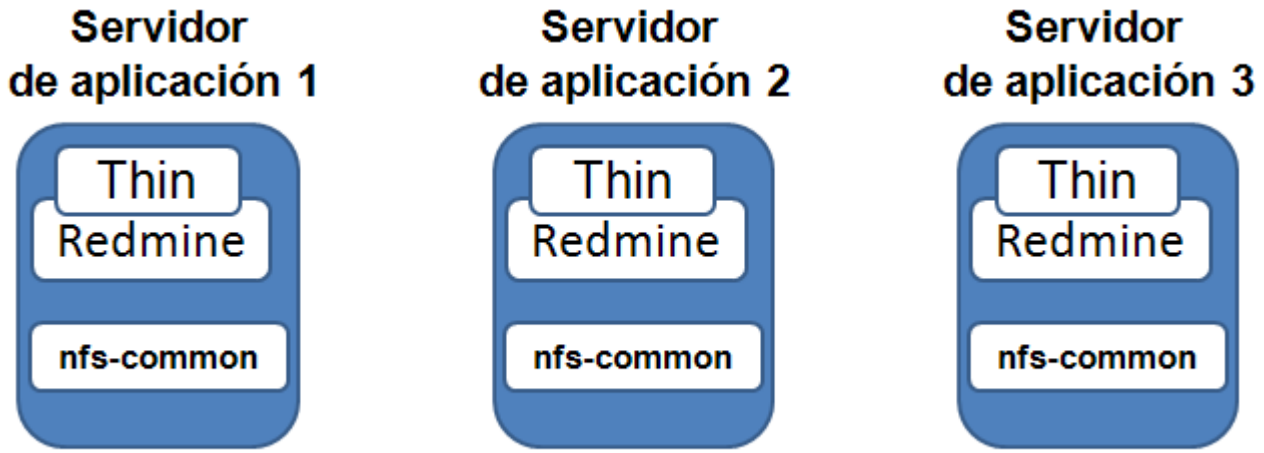
2.1.2. Diseño físico del clúster

El clúster de balanceo de carga físicamente contará con siete servidores de los cuáles dos serán balanceadores de carga, tres estarán destinados a ser servidores de aplicación y dos servidores de archivos compartidos. En los balanceadores de carga estarán instaladas las herramientas Nginx, Memcached, las cuáles serán manejadas por Heartbeat (ver Fig. 10), estos servidores son independientes pero están comunicados por Heartbeat.



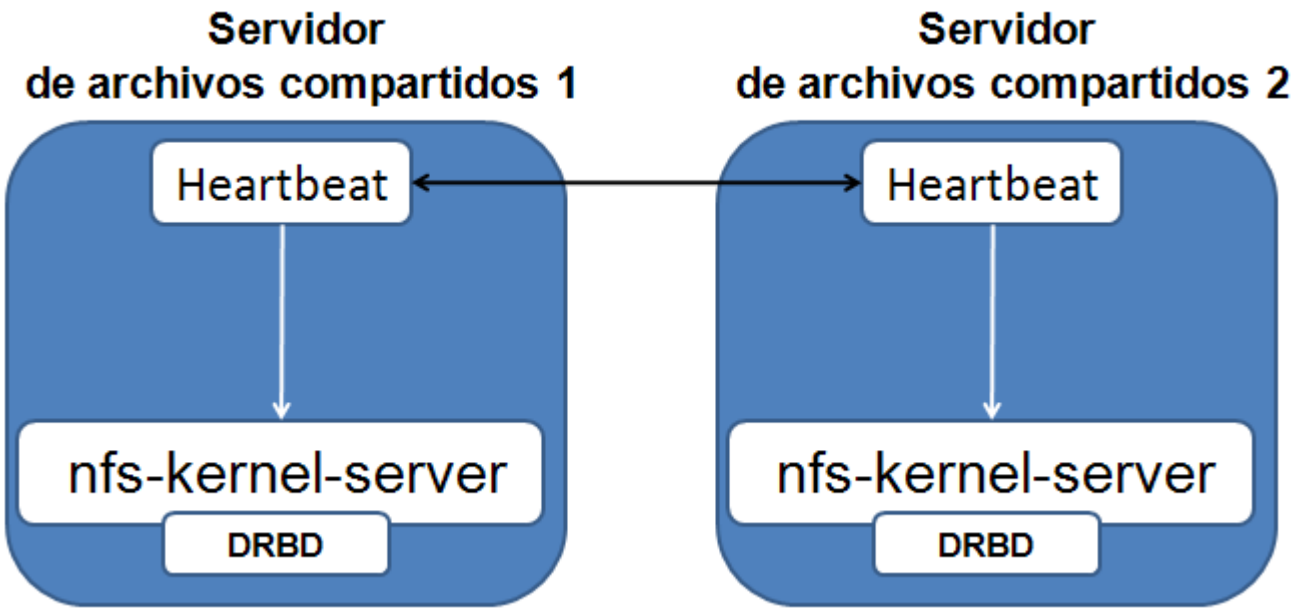
Figuras 10 Diseño físico de los balanceadores de carga.

En el caso de los servidores de aplicación en ellos se encontrarán instaladas las herramientas Thin, nfs-common y el Redmine, estos servidores son independientes, y entre ellos no están comunicados (ver Fig. 11).



Figuras 11 Diseño físico en los servidores de aplicación.

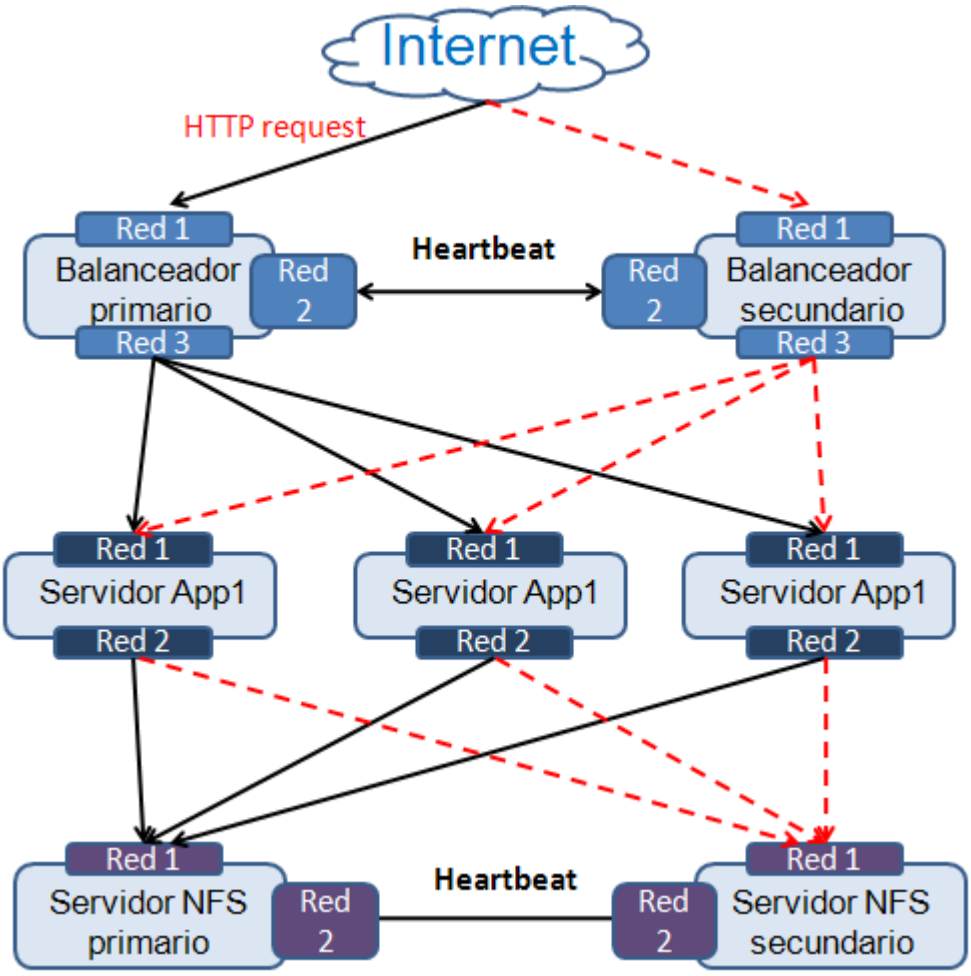
Sin embargo los servidores de archivos compartidos tendrán instalado nfs-kernel-server, DRBD y Heartbeat para manejar la alta disponibilidad entre ambos servidores, estos servidores solo estarán conectados entre ellos por medio de Heartbeat (ver Fig. 12).



Figuras 12 Diseño físico en los servidores de archivos compartidos.

2.1.3. Diseño de interconexión

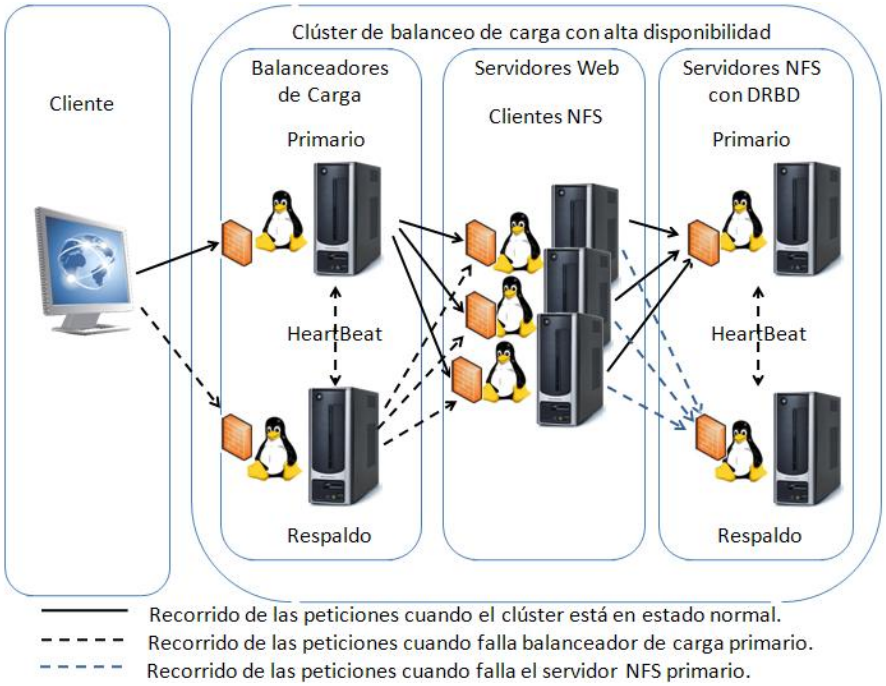
El diseño de interconexión tendrá la siguiente estructura: en los balanceadores de carga será necesaria la utilización de tres tarjetas de red, la primera para atender las peticiones de los clientes, la segunda para la para el monitoreo de Heartbeat y la tercera para la comunicación entre los servidores de aplicación y el balanceador de carga, los servidores de aplicación contarán con dos tarjetas de red de las cuáles una será utilizada para la conexión con los balanceadores de carga y la otra para los servidores NFS, mientras que en los servidores de almacenamiento compartido serán necesarias dos tarjetas de red, una que se usará para las peticiones que realizan los servidores de aplicaciones a los servidores NFS, y la otra para el monitoreo de Heartbeat (ver Fig. 11).



Figuras 13 Diseño de interconexión.

2.1.4. Diseño de seguridad del clúster

Para implentar el diseño de seguridad de la propuesta de solución se utilizará el *software* Iptables el cuál permite implementar cortafuegos y filtrado de paquetes, en el caso de la presente investigación se filtrarán en específico los puertos que usan las aplicaciones que componen el clúster. Se cerrarán todos los puertos en todos los ordenadores del clúster y solo se dejarán abiertos los utilizados para el correcto funcionamiento del clúster (ver Fig. 14).



Figuras 14 Diseño de seguridad.

Se creará un script para que al iniciar el ordenador configure el cortafuego, ya que iptables es un sistema volátil y cada vez que es reiniciado el ordenador este pierde su configuración. Para la configuración de iptables se hará uso de la política denegar todo por defecto y luego se irán abriendo los puertos utilizados por el clúster. La configuración e instalación de este diseño de la propuesta de solución se describe en el documento que se adjunta en formato digital titulado “Guía de instalación de un clúster para el despliegue del Redmine”.

Conclusiones

El estudio de las características del Redmine como aplicación desarrollada en el *framework* RoR al cual se le diseña el clúster permite concluir que:

- El clúster contará con un balanceador de carga de respaldo en caso de fallas en el primario.
- El clúster tendrá tres nodos donde estará publicado el Redmine.
- El clúster dispondrá de un servidor NFS de respaldo en caso de que falle el primario.
- Los balanceadores de carga estarán dotados con tres tarjetas de red, los servidores de aplicación tendrán dos tarjetas al igual que los servidores de archivos compartidos.

CAPÍTULO 3: VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

Introducción

Este capítulo tiene como propósito describir y analizar los resultados de las pruebas de rendimiento y de configuración realizadas a la propuesta de solución. En el mismo se realiza un análisis de los resultados obtenidos en dichas pruebas mediante gráficas y tablas comparativas.

3.1. Descripción del proceso de pruebas

Para garantizar el correcto funcionamiento de cualquier sistema es necesario realizar un proceso de pruebas donde se demuestre que dicho sistema está listo para su despliegue. Este proceso de pruebas, se realizó con el propósito de demostrar que la aplicación funciona correctamente, permitiendo su utilización, y una disminución en los tiempos de respuestas del sistema.

Estas pruebas se realizaron fundamentalmente con los siguientes objetivos:

- Proveer confianza en el sistema.
- Identificar las áreas de debilidad (áreas donde el sistema está más propenso a fallos debido a una serie de condiciones que se pueden presentar una vez que el sistema esté desplegado).
- Establecer un grado de calidad del sistema.
- Proveer un entendimiento general de cómo funciona el mismo.
- Probar además que es usable y operable.

Durante la realización de las pruebas se estudió el comportamiento de las funcionalidades de la solución, las cuáles definen el comportamiento del clúster y puede afectar el correcto funcionamiento de la aplicación. Para medir el rendimiento del clúster y comprobar su correcto funcionamiento se realizaron fundamentalmente pruebas de configuración y de rendimiento.

Las pruebas de rendimeinto realizadas para la validación de la propuesta de solución se realizaron en ordenadores virtualizados con el *software* VMware[6] dotadas de un procesador a 1.171 GHz, una memoria RAM asignada de 512 Mb y 8 GB de disco duro.

[6] **VMware**: *software* que permite la virtualización de ordenadores en un entorno virtual.

Para la medición del rendimiento durante el proceso de pruebas de carga se utilizaron métricas como: tiempo de respuesta promedio de las peticiones, cantidad de peticiones respondidas por segundo, número de peticiones completadas, % de peticiones fallidas, y tiempo de respuesta promedio; mientras que para la medición de los recursos las métricas utilizadas fueron: % de uso del CPU, % de uso de memoria RAM y el % del uso de la red.

Para generar carga en el clúster se utilizó el *software* Jmeter a través del cuál se realizó la emulación de un número concurrentemente de usuarios navegando por el Redmine. Jmeter permite grabar la navegación de un usuario por una aplicación y permite simular varios usuarios realizando la misma navegación. Cuando se configura Jmeter para repetir la navegación grabada es necesario agregarle el elemento HTTP Cookie Manager al plan de pruebas ya que este permite mantener a la herramienta las secciones de usuario mediante el almacenamiento de cookies.

También se le agregó a la configuración del plan de pruebas de Jmeter un elemento de reporte para la recogida de los datos devueltos y las peticiones que conforman cada página se agruparon para conocer el tiempo que tarda el sistema en devolver una página completa.

Para el comienzo de las pruebas se generó carga simulando un determinado número de usuarios concurrentes para que la aplicación a probar se adaptara a esta carga, luego desde un ordenador diferente se realizó la navegación de un usuario un total de cinco veces para obtener el tiempo de respuesta del sistema.

La recogida de datos como % de usos de CPU, % de uso de memoria RAM y % de uso de de la red en los balanceadores de carga, en los servidores de aplicación y en el servidor NFS se realizó mediante la herramienta Sysstat. Al igual que en el caso anterior se esperó a que se adaptaran los servidores a la carga de usuarios para comenzar a recoger los datos. Luego de recogido los datos, los valores obtenidos en cada ordenador se promedian para obtener el consumo de recursos promedio de manera general para cada prueba realizada.

3.2. Pruebas realizadas al diseño de la propuesta de solución

Para validar que el diseño de la propuesta de solución funciona correctamente se le realizaron un grupo de pruebas de configuración a las funcionalidades del clúster. Las pruebas de configuración realizadas y el propósito de las mismas se describen en la tabla 1.

3.2.1. Pruebas de configuración realizadas al diseño de la solución

Funcionalidades del sistema a probar

- Balanceo de carga.
- Almacenamiento de archivos compartidos.
- Alta disponibilidad en el balanceador de carga.
- Recuperación ante fallos en los servidores de aplicación.
- Alta disponibilidad en el servidor de archivos compartidos.

Tabla 1 Pruebas de configuración realizadas al diseño de solución del clúster de balanceo de carga.

Nombre	Tipo	Propósito
DS-CO-01	Configuración	Comprobar que el diseño de solución realiza balanceo de carga.
DS-CO-02	Configuración	Comprobar que el diseño de solución realiza almacenamiento de archivos compartidos.
DS-CO-03	Configuración	Comprobar que el diseño de solución es capaz de recuperarse ante la caída de un servidor NFS.
DS-CO-04	Configuración	Comprobar que el diseño de solución es capaz de recuperarse ante la caída del balanceador de carga primario
DS-CO-05	Configuración	Comprobar que el diseño de solución es capaz de recuperarse ante el fallo de uno de los servidores de aplicación.

Se desplegó el diseño de la solución siguiendo las indicaciones del documento “Guía de instalación de un clúster para el despliegue del Redmine”, el cuál está compuesto por dos balanceadores de carga, tres servidores de aplicación donde estará publicado el Redmine y dos servidores de archivos compartidos NFS.

Prueba de configuración DS-CO-01

En esta prueba se espera como resultado que el balanceador de carga distribuya las peticiones entre los servidores de aplicación que componen el clúster.

Para la comprobación de la realización efectiva del balanceo de carga según el algoritmo Run-Robin, fue necesario la utilización del Jmeter para generar 10 usuarios navegando por el sistema un total de 10 veces, como resultado de esta prueba se realizaron 752 peticiones al balanceador de las cuales 251 peticiones fueron atendidas por el primer servidor de aplicación, 251 fueron atendidas por el segundo servidor de aplicación y 250 peticiones se atendieron por el tercer servidor aplicación, por lo que se concluye que el balanceador funciona correctamente.

Prueba de configuración DS-CO-02

Esta prueba se realiza con el propósito de comprobar que el diseño de solución realiza correctamente almacenamiento de archivos compartidos. Se espera que cuando se suba un archivo al sistema y se apague el servidor de aplicación que subió dicho archivo, el mismo continúe estando disponible para que los demás servidores de aplicación accedan a él.

Para su comprobación se navegó por un servidor de aplicación y se subió un archivo, luego se apagó este servidor de aplicación y se accedió a otro servidor de aplicación para leer dicho archivo, esto se pudo hacer sin problema alguno ya que donde todos los servidores de aplicación copian sus archivos es un directorio compartido por el servidos NFS. Esta prueba fue realizada con éxito y devolvió los resultados esperados.

Prueba de configuración DS-CO-03

Esta prueba se realiza con el propósito de comprobar que el diseño de solución ante la caída de un servidor de archivos compartidos se recupere y no afecte el correcto funcionamiento de la aplicación. Se espera que al fallar un servidor NFS los clientes sigan accediendo a los archivos que estaban publicados en este a través del servidor NFS de respaldo.

Para su comprobación se navegó por un servidor de aplicación y se subió un archivo, luego se apagó el servidor NFS donde el sistema guardó dicho archivo, seguidamente se apagó este servidor y se accedió para comprobar que se podía seguir teniendo acceso a dicho archivo, esto se pudo realizar sin problema alguno ya que el *software* DRBD replicó perfectamente la información del servidor que se apagó hacia el servidor de respaldo. Esta prueba fue realizada con éxito y devolvió los resultados esperados.

Prueba de configuración DS-CO-04

Esta prueba tiene como propósito fundamental comprobar que el sistema siga prestando servicios ante la caída de el balanceador de carga primario.

Para la comprobación de esta funcionalidad del sistema se apagó el balanceador de carga primario y se pudo observar como este migró todos sus servicios al balanceador de carga secundario siendo este proceso transparente para el ususario que estaba navegando en ese momento por la aplicación.

Prueba de configuración DS-CO-05

El propósito de la misma es comprobar que el diseño de la solución es capaz de recuperarse ante la caída de un servidor de aplicación. Se espera como resultado que cuando un usuario este navegando por el sistema sobre un servidor de aplicación y este deje de funcionar, el sistema sea capaz de permitir que el usuario continúe navegando por el sistema sin necesidad de volver a autenticarse.

Para la comprobación de dicha prueba se autenticó un usuario en el sistema , luego se apagó el servidor de aplicación al cual el balanceador envió esta petición y seguidamente se continuó navegando por el sistema sin necesidad de autenticarse nuevamente.

Análisis de los resultados

Durante el proceso de pruebas de configuración realizadas al clúster fueron probadas las funcionalidades: balanceo de carga, almacenamiento de archivos compartidos, alta disponibilidad en el balanceador de carga primario y recuperación ante fallos en los servidores de aplicación. Como se pudo observar durante las pruebas, el comportamiento obtenido por parte del sistema en cada una de las funcionalidades probadas fue satisfactorio, se puede llegar a la conclusión que el diseño del clúster de balaneo de carga está validado funcionalmente.

3.2.2. Pruebas de carga realizadas al diseño de la solución

En la presente investigación se realizan un grupo de pruebas de carga para analizar cómo se mejoran los tiempos de respuestas promedio de la aplicación desplegada sobre el clúster. Las pruebas de carga realizadas y el propósito de las mismas se describen en la tabla 2.

Tabla 2 Pruebas de carga realizadas al diseño de solución del clúster de balanceo de carga.

Nombre	Tipo	Propósito

DS-CA-06	Carga	Demostrar por qué se fue usado el servidor web Thin para los servidores de aplicación y no Apache + passenger.
DS-CA-07	Carga	Demostrar como al aumentar el número de servidores de aplicación en el clúster, manteniendo un nivel de concurrencia constante, disminuye el tiempo de respuesta del sistema y el consumo de recursos en los servidores de aplicación.
DS-CA-08	Carga	Demostrar como al aumentar el nivel de concurrencia en el clúster, manteniendo la cantidad de servidores de aplicación, aumenta el tiempo de respuesta del sistema así como el consumo de recursos en los servidores de aplicación.

Prueba de carga DS-CA-06

Para recoger datos que nos pudiesen dar en medida el rendimiento de cada uno de los servidores (Apache y Thin), fue necesaria la generación de 100 usuarios de forma concurrente con ayuda de Jmeter, navegando por 5 páginas de la aplicación.

En la tabla 3 se muestra como se comportaron los tiempos de respuesta promedio sobre las distintas configuraciones: Apache +passenger y Thin.

Tabla 3 Tiempos de Respuesta promedios del sistema.

	Tiempo de respuesta promedio (s)
Thin	3.0
Apache +passenger	25.0

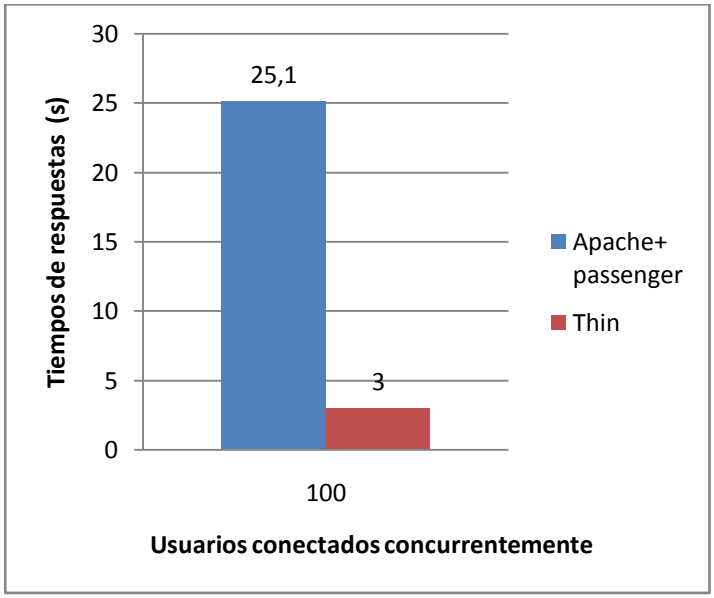
En la tabla 4 se muestra un resumen de los valores promedio en el consumo de recursos en el servidor Thin, y como se comportan dichos recursos en Apache + passenger.

Tabla 4 Consumo de recursos promedio en Thin y Apache + passenger.

Recursos	Servidores	
	Thin	Apache + passenger
% de uso de CPU	75	84
% de uso de memoria RAM	17.6	92
% de uso de la red	6.1	17.0

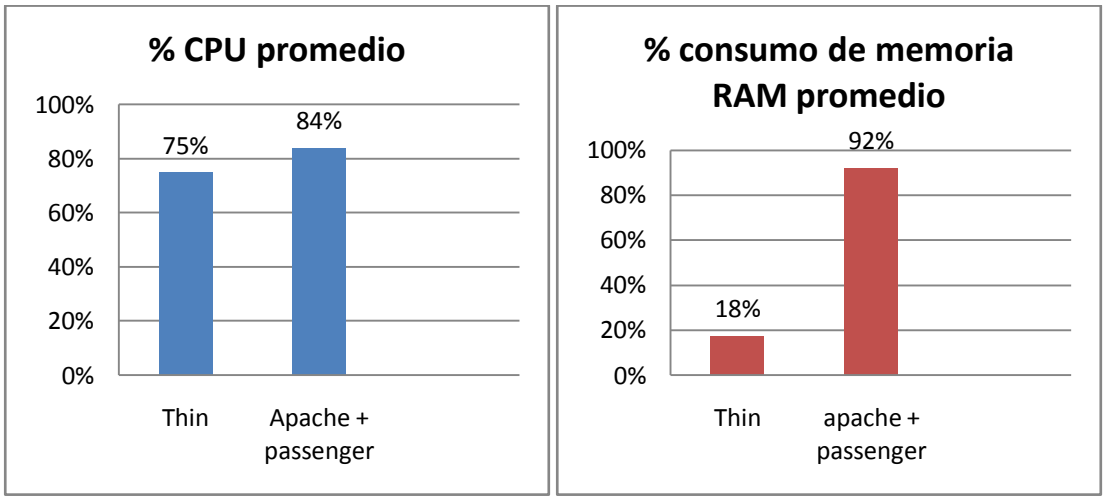
Análisis de los resultados

Como se puede apreciar en el gráfico de la figura 15, con 100 usuarios conectados concurrentemente, navegando por 5 páginas de la aplicación los tiempos de respuesta promedio del servidor Thin son menores que los de Apache + passenger. Esta gráfica muestra la diferencia cuanto a tiempos de respuestas.

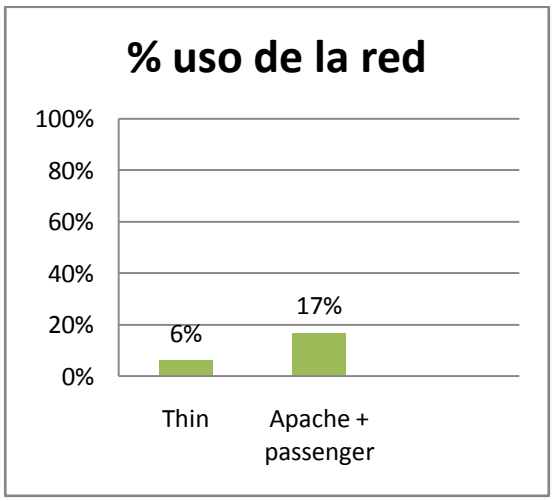


Figuras 15 Tiempos de respuestas promedio.

De manera semejante a lo sucedido con los tiempos de respuestas, el consumo de recursos también disminuye en el servidor Thin, esto se evidencia en la figura 16 y la figura 17.



Figuras 16 Consumo de recursos promedio CPU y RAM en Thin y Apache + passenger.



Figuras 17 Consumo de recursos promedio tráfico de red en los servidores de aplicaciones.

Como se pudo observar en los resultados devueltos en la prueba Thin supera a la variante Apache + passenger para el despliegue del Redmine.

Prueba de carga DS-CA-07

Para la realización de esta prueba la generación de carga sobre el clúster de balanceo de carga desplegado con dos y tres servidores de aplicación se realizó en cinco ordenadores en los que solo se ejecutaba la herramienta Jmeter utilizando un plan de pruebas simulando un usuario navegando por 15 páginas de la aplicación desplegada en el clúster.

En la tabla 5 se muestra como se comportaron los tiempos de respuesta promedio sobre las distintas configuraciones del clúster y aparece en la última fila de la tabla el servidor Thin bajo este mismo plan de pruebas.

Tabla 5 Tiempos de Respuesta promedios del sistema.

Número de servidores de aplicación dentro del clúster (c).	Tiempo de respuesta promedio (s)
2 (c)	2.9
3 (c)	2.2
Thin	3.0

En la tabla 6 se muestra un resumen de los valores promedio en el consumo de recursos en los servidores de aplicaciones desde uno hasta tres nodos, y como se comportan dichos recursos en Thin.

Tabla 6 Consumo de recursos promedio en los servidores de aplicaciones.

Recursos	Número de servidores de aplicación dentro del clúster (c).		
	2 (c)	3 (c)	Thin
% de uso de CPU	72	60	75
% de uso de memoria RAM	16.8	15	17.6
% de uso de la red	3.34	2.70	6.1

En la tabla 7 se recogen los valores promedio del consumo de recursos en el balanceador de carga, de dos y tres nodos así como el comportamiento de los recursos Thin.

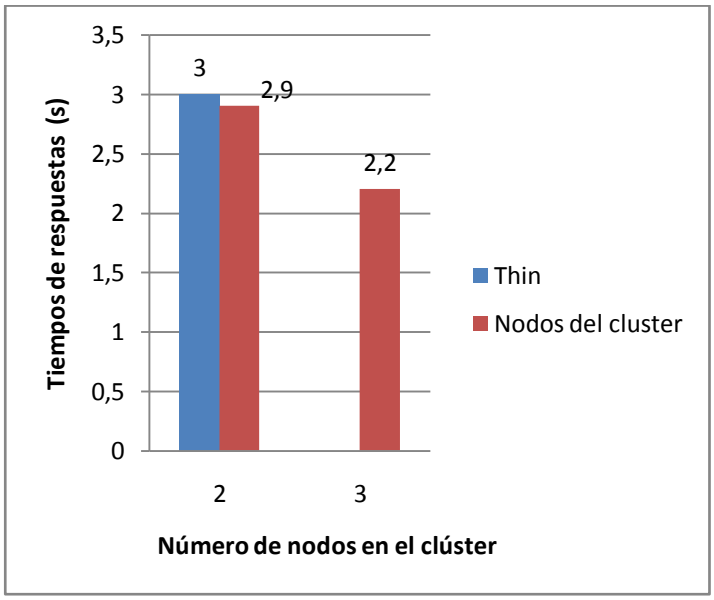
Tabla 7 Consumo de recursos promedio en el balanceador de carga.

Recursos	Número de servidores de aplicación dentro del clúster (c).		
	2 (c)	3 (c)	Thin
% de uso de CPU	52	63	75
% de uso de memoria RAM	9.4	11.2	17.6
% de uso de la red	0.53	0.64	6.1

Análisis de los resultados

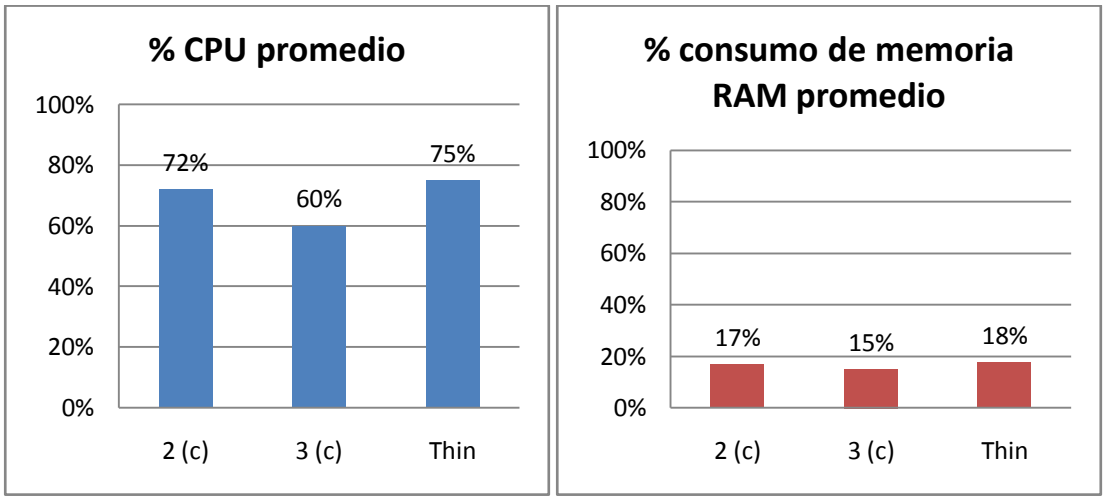
Como se puede apreciar en el gráfico de la figura 18 con el aumento del número de servidores de aplicación en el clúster balanceo de carga los tiempos de respuesta promedio disminuyen.

En dicha gráfica el punto azul representa el servidor real utilizado para medir cuánta carga soportaba la aplicación en su despliegue original y la función representa las distintas configuraciones del clúster. Dicha función permite concluir que para la carga que se estaba generando durante la realización de esta prueba que fue de 100 usuarios concurrentes, con tres nodos en el clúster los tiempos promedio de respuesta son menores que en el servidor Thin.

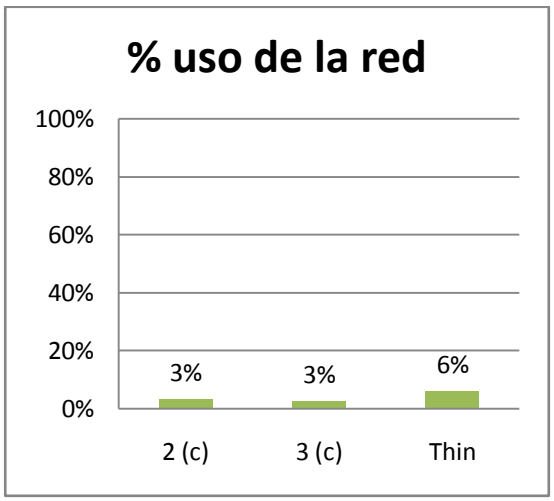


Figuras 18 Tiempos de respuestas promedio contra número de nodos.

De manera semejante a lo sucedido con los tiempos de respuestas, con el aumento del número de servidores de aplicación en el clúster, el consumo de recursos en estos también disminuye debido a que cada servidor tiene que atender menos cantidad de consultas, esto se evidencia en la figura 19 y la figura 20.

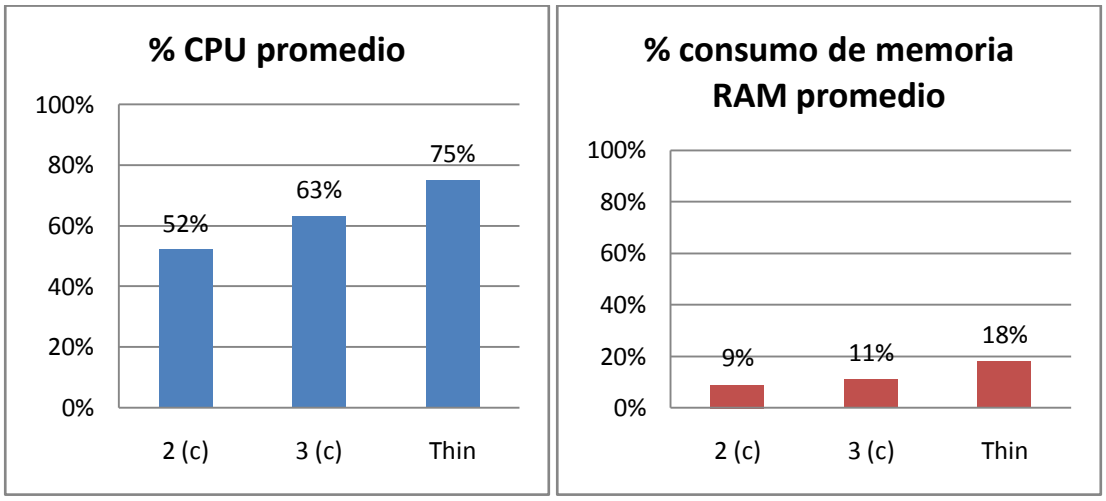


Figuras 19 Consumo de recursos promedio CPU y RAM en los servidores de aplicaciones.

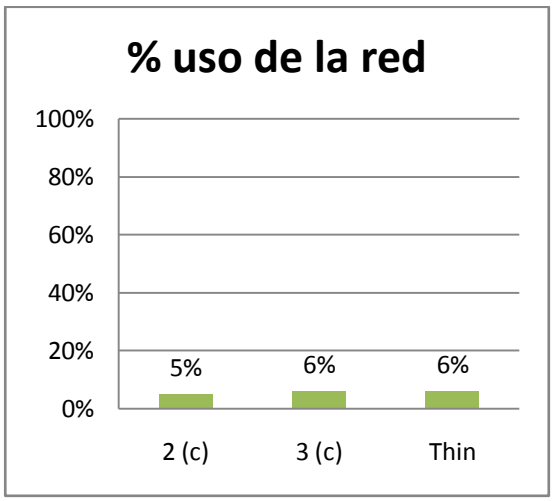


Figuras 20 Consumo de recursos promedio tráfico de red en los servidores de aplicaciones.

Con el aumento del número de servidores de aplicaciones el consumo del CPU y el tráfico de red aumentan en el balanceador de carga, pero para la prueba realizada no alcanza valores preocupantes como se muestra en la figura 21 y la figura 22.



Figuras 21 Consumo de recursos promedio CPU y RAM en el balanceador de carga.



Figuras 22 Consumo de recursos promedio tráfico de red en el balanceador de carga.

Prueba de carga DS-CA-08

La generación de carga sobre el clúster de balanceo de carga desplegado tres servidores de aplicación para realizar esta prueba el mismo plan de pruebas utilizado en la Prueba de carga DS-CA-07.

En la tabla 8 se muestran los tiempos de respuesta promedio por parte del sistema aumentando el número de concurrencia de usuarios desde 100 hasta 2000 y se puede evidenciar como aumentan los tiempos de respuesta por parte del sistema, y como con la variante Thin solo el sistema pudo atender 1000 usuarios concurrentes.

Tabla 8 Tiempos de Respuesta promedios del sistema.

Número de usuarios	Tiempo de respuesta promedio (s)	
	Clúster	Thin
100	2.2	3.0
500	4	12
1000	10.6	22
2000	27.8	-

En la tabla 9 se muestra un resumen de los valores promedio en el consumo de recursos en los servidores de aplicación a medida que aumenta la concurrencia de usuarios navegando en el sistema.

Tabla 9 Consumo de recursos promedio en los servidores de aplicaciones.

Recursos	Número usuarios			
	100	500	1000	2000
% de uso de CPU	75	83	87	90
% de uso de memoria RAM	15	17.4	40.2	57.6
% de uso de la red	3.84	5.65	10.20	15.6

En la tabla 10 se recogen los valores promedio del consumo de recursos en el balanceador de carga a medida que aumenta la concurrencia de usuarios dentro del sistema.

Tabla 10 Consumo de recursos promedio en el balanceador de carga.

Recursos	Número usuarios			
	100	500	1000	2000
% de uso de CPU	63	68	72	76
% de uso de memoria RAM	11.2	20.3	25.4	34.1
% de uso de la red	0.67	1.04	1.50	2.01

En la tabla 11 se muestra un resumen de los valores promedio en el consumo de recursos en el servidor de aplicación Thin a medida que aumenta la concurrencia de usuarios y se puede observar como el sistema resiste navegando concurrentemente alrededor de 1000 usuarios.

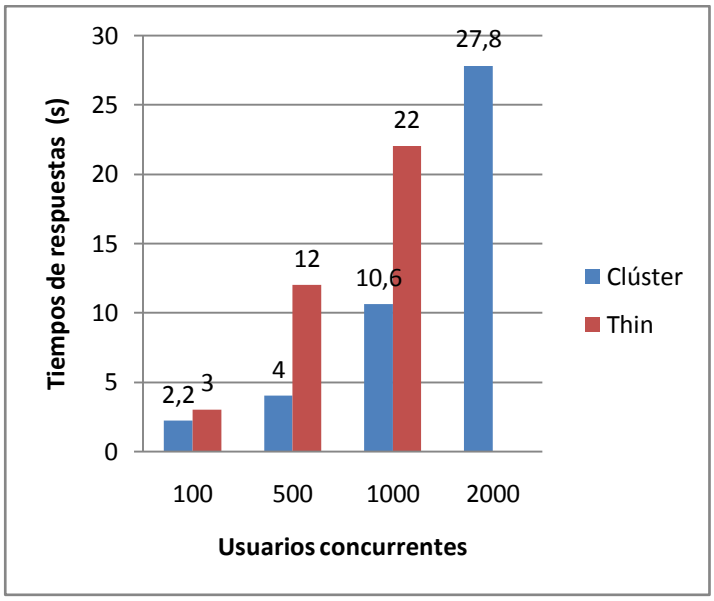
Tabla 11 Consumo de recursos promedio en el servidor de aplicación Thin.

Recursos	Número usuarios			
	100	500	1000	2000
% de uso de CPU	84	100	100	-
% de uso de memoria RAM	75	98	100	-
% de uso de la red	25	30.3	36	

Análisis de los resultados.

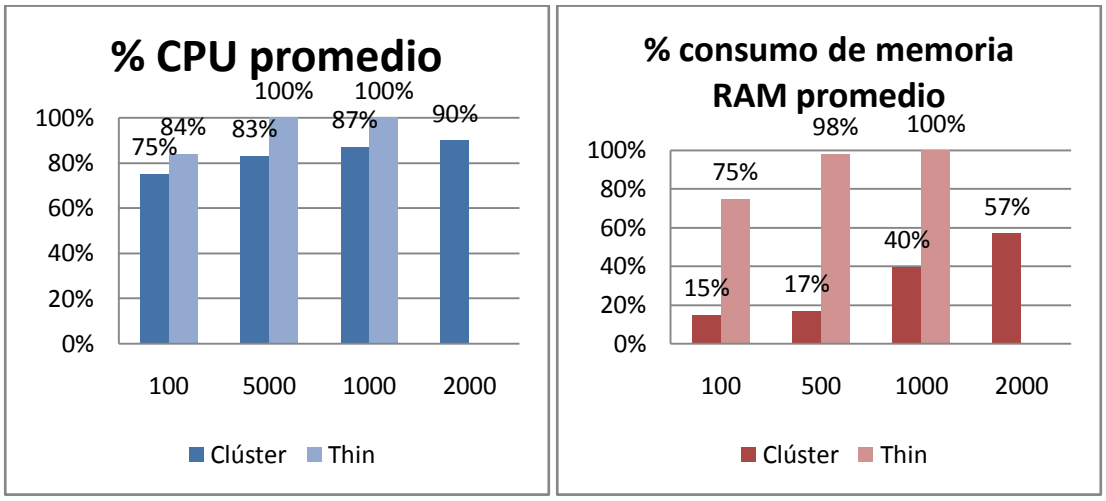
Como se puede apreciar en el gráfico de la figura 23 con el aumento de los niveles de concurrencia en el clúster de alta disponibilidad con balanceo de carga los tiempos de respuesta aumentan.

Esta prueba se realizó con el objetivo de tener una idea o para poder estimar para determinada configuración del clúster cuáles serían sus límites, ya sean por tiempos de respuesta extremadamente altos o por limitaciones en los recursos de los servicios.

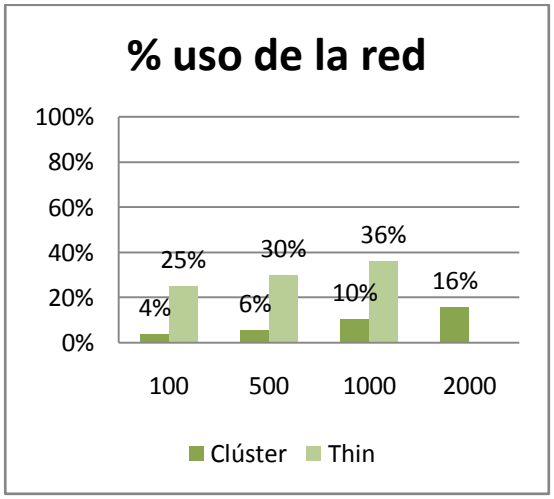


Figuras 23 Tiempos de respuestas promedio contra niveles de concurrencia.

De manera equivalente a lo que sucede con los tiempos de respuestas al aumentar los niveles de concurrencia, el consumo de recursos en los servidores de aplicaciones también aumentan, esto se puede observar en los gráficos de la figura 24 y la figura 25.

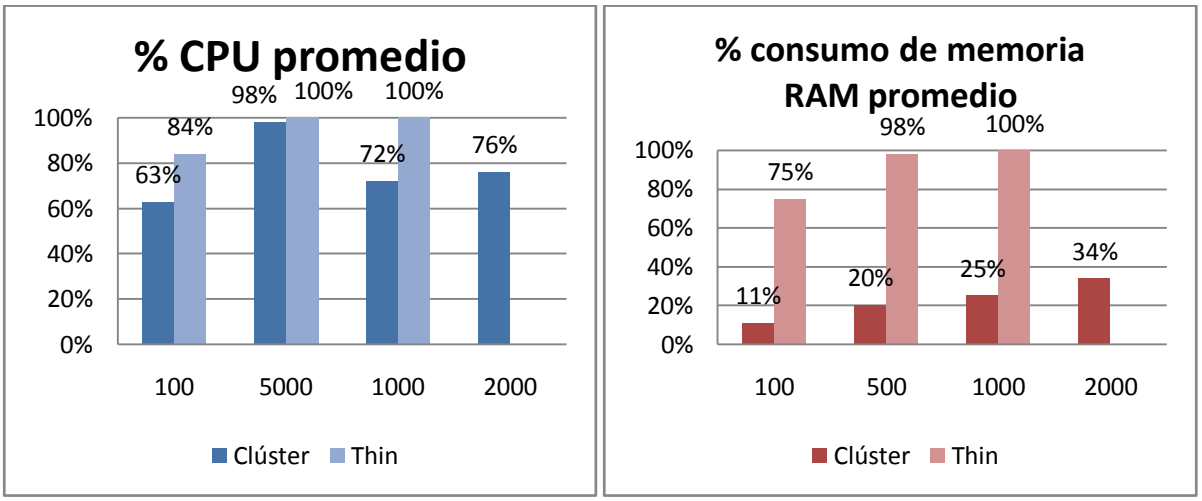


Figuras 24 Consumo de recursos promedio CPU y RAM en los servidores de aplicación.

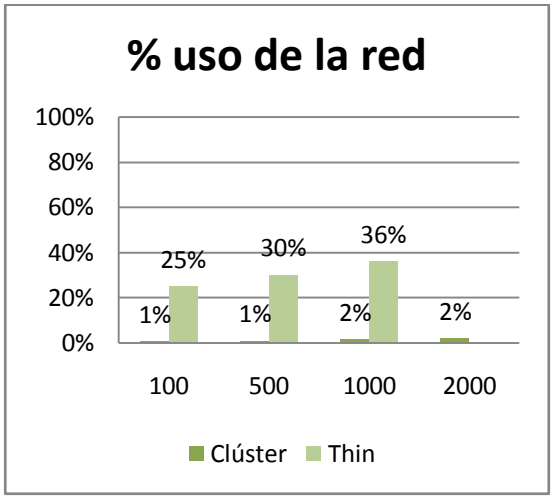


Figuras 25 Consumo de recursos promedio de red en los servidores de aplicación.

Con el aumento de los niveles de concurrencia el consumo de recursos en el balanceador también varía. Para este caso, el consumo de CPU se mantiene casi constante pero el por ciento del uso de la memoria RAM y el porciento de uso de la red van aumentando, esto se puede apreciar en la figura 26 y la figura 27.



Figuras 26 Consumo de recursos promedio CPU y RAM en el balanceador de carga.



Figuras 27 Consumo de recursos promedio de red en el balanceador de carga.

3.3. Conclusiones

Se realizaron varios tipos de pruebas con el objetivo de validar la propuesta de solución de clúster de la presente investigación entre las que se encuentran las pruebas de configuración y pruebas de rendimiento. Las pruebas de configuración se desarrollaron con el objetivo de comprobar que las soluciones no afectaban el correcto funcionamiento de la aplicación desplegada, además se comprobó el correcto funcionamiento del sistema. Las pruebas de rendimiento se utilizaron para medir principalmente los tiempos de respuestas una vez que las soluciones se encontraban desplegadas.

CONCLUSIONES

En el presente trabajo se realizó un estudio de las distintas formas de poner en producción una aplicación RoR para el despliegue del Redmine, herramienta de gestión de proyectos utilizada en la Universidad de las Ciencias Informáticas (UCI), y se diseñó una solución de clúster para ella, dando cumplimiento a los objetivos definidos en la investigación.

Durante las pruebas realizadas para validar la solución se detectó que la propuesta de solución presentada por la presente investigación ofreció los resultados esperados, no así la propuesta implantada actualmente en la UCI, la cual presentó problemas que afectaron el correcto funcionamiento de la aplicación, los cuáles se vieron cuando muchos usuarios accedían de forma concurrente al sistema, estos problemas se evidencian en el crecimiento de los tiempos de respuesta de forma preocupante llegando a colapsar el sistema.

Se obtuvo una guía de instalación de la solución para contar con una documentación a la hora del despliegue de la misma.

RECOMENDACIONES

A pesar de que se alcanzaron los objetivos propuestos en el inicio de la investigación se recomienda:

- Incluir como parte de la Guía de instalación de un clúster para el despliegue del Redmine, el monitoreo de la propuesta de solución.
- Desarrollar un mecanismo para montar automáticamente el sistema de archivos compartido por parte de los servidores de aplicación una vez que estos inicien antes que el servidor de archivos compartidos.

REFERENCIAS BIBLIOGRÁFICAS

1. **Sánchez, Roberto y Luca, Cernuzzi.** www.dei.uc.edu.py. *www.dei.uc.edu.py.* [En línea] 2010. http://www.dei.uc.edu.py/tai2003-2/clustering/html/concepto_de_cluster.html.

2. **Cota, Eduardo.** iie.fing.edu.uy. *iie.fing.edu.uy.* [En línea] [Citado el: 15 de 3 de 2011.] http://iie.fing.edu.uy/ense/asign/admunix/nfs.htm.

3. **www.loadbalancing.org.** www.loadbalancing.org. *www.loadbalancing.org.* [En línea] [Citado el: 25 de 3 de 2011.] http://www.loadbalancing.org/#DNS_Load_Balancing_.

4. www.linuxvirtualserver.org. *www.linuxvirtualserver.org.* [En línea] [Citado el: 25 de 6 de 2011.] http://www.linuxvirtualserver.org/Documents.html.

5. **Kubat, K.** Crossroads Documentation. *Crossroads Documentation.* [En línea] 2008. [Citado el: 13 de 12 de 2010.] http://www.crossroads.e-tunity.com/serverdoc.xr?format=html.

6. **Nginx.** nginx.org. *nginx.org.* [En línea] 2010. [Citado el: 20 de noviembre de 2010.] http://nginx.org/en/docs/introduction.html.

7. **Apache Software Foundation.** apache.org. *papache.org.* [En línea] 2008. [Citado el: 24 de noviembre de 2010.] http://jakarta.apache.org/jmeter/usermanual/index.html.

8. **Apache.** www.apache.org. *www.apache.org.* [En línea] 2010. [Citado el: 16 de noviembre de 2010.] http://www.apache.org/.

9. **Marc-André Cournoyer.** code.macournoyer.com. *code.macournoyer.com.* [En línea] [Citado el: 15 de 4 de 2011.] http://code.macournoyer.com/thin/doc/files/README.html.

10. redmine.lighttpd.net. *redmine.lighttpd.net.* [En línea] [Citado el: 12 de 6 de 2011.] http://redmine.lighttpd.net/wiki/lighttpd.

11. **Enterprise, Ruby.** www.rubyenterpriseedition.com. *www.rubyenterpriseedition.com.* [En línea] 2010. [Citado el: 21 de noviembre de 2010.] http://www.rubyenterpriseedition.com.

12. www.drbd.org. *www.drbd.org.* [En línea] [Citado el: 22 de 1 de 2011.] http://www.drbd.org/users-guide/ch-heartbeat.html.

13. www.drbd.org. *www.drbd.org.* [En línea] [Citado el: 5 de 4 de 2011.] http://www.drbd.org/docs/about/.

14. **Altadill Izura, Pello Xabier.** es.tldp.org. *es.tldp.org.* [En línea] [Citado el: 21 de 3 de 2011.] http://es.tldp.org/Manuales-LuCAS/doc-iptables-firewall/doc-iptables-firewall-html/.

15. **Godar, S.** SysStat Documentation. *SysStat Documentation.* [En línea] 2 de 12 de 2011. [Citado el: 22 de 11 de 2010.] http://pagesperso-orange.fr/sebastien.godard/documentation.html.

BIBLIOGRAFÍA

Altadill Izura, Pello Xabier. es.tldp.org. *es.tldp.org.* [En línea] [Citado el: 21 de 3 de 2011.] http://es.tldp.org/Manuales-LuCAS/doc-iptables-firewall/doc-iptables-firewall-html/.

Apache Software Foundation. 2008. apache.org. *papache.org.* [En línea] 2008. [Citado el: 24 de noviembre de 2010.] http://jakarta.apache.org/jmeter/usermanual/index.html.

Apache. 2010. www.apache.org. *www.apache.org.* [En línea] 2010. [Citado el: 16 de noviembre de 2010.] http://httpd.apache.org/docs/2.2/.

Cota, Eduardo. iie.fing.edu.uy. *iie.fing.edu.uy.* [En línea] [Citado el: 15 de 3 de 2011.] http://iie.fing.edu.uy/ense/asign/admunix/nfs.htm.

Enterprise, Ruby. 2010. www.rubyenterpriseedition.com. *www.rubyenterpriseedition.com.* [En línea] 2010. [Citado el: 21 de noviembre de 2010.] http://www.rubyenterpriseedition.com.

Godar, S. 2011. SysStat Documentation. *SysStat Documentation.* [En línea] 2 de 12 de 2011. [Citado el: 22 de 11 de 2010.] http://pagesperso-orange.fr/sebastien.godard/documentation.html.

Kubat, K. 2008. Crossroads Documentation. *Crossroads Documentation.* [En línea] 2008. [Citado el: 13 de 12 de 2010.] http://www.crossroads.e-tunity.com/serverdoc.xr?format=html.

Marc-André Cournoyer. code.macournoyer.com. *code.macournoyer.com.* [En línea] [Citado el: 15 de 4 de 2011.] http://code.macournoyer.com/thin/doc/files/README.html.

memcache.org. 2009. memcache.org. *memcache.org.* [En línea] 2009. [Citado el: 21 de noviembre de 2010.] http://memcached.org/about.

Nginx. 2010. nginx.org. *nginx.org.* [En línea] 2010. [Citado el: 20 de noviembre de 2010.] http://nginx.org/en/docs/introduction.html.

Olea, Ismael. 2009. es.tldp.org. *es.tldp.org.* [En línea] 2009. [Citado el: 15 de 11 de 2010.] http://es.tldp.org/Manuales-LuCAS/doc-cluster-computadoras/doc-cluster-computadoras-html/node7.html.

redmine.lighttpd.net. *redmine.lighttpd.net.* [En línea] [Citado el: 12 de 6 de 2011.] http://redmine.lighttpd.net/wiki/lighttpd.

Ridruejo, Francisco. 2003. sc.ehu.es. *Estrategias de instalación y gestión de clusters.* [En línea] 2003. [Citado el: 15 de 11 de 2010.] http://www.sc.ehu.es/acwmialj/papers/jornadas03_a.pdf.

S.A., Informáticos. 2009. www.anurix.com. *www.anurix.com.* [En línea] 2009. [Citado el: 22 de noviembre de 2010.] http://www.anurix.com/docs/Introduccion_a_lighttpd.pdf.

Sánchez, Roberto y Luca, Cernuzzi. 2010. www.dei.uc.edu.py. *www.dei.uc.edu.py.* [En línea] 2010. http://www.dei.uc.edu.py/tai2003-2/clustering/html/concepto_de_cluster.html.

Vásquez, Miguel Hernández. 2007. elai.upm.es. *elai.upm.es.* [En línea] 2007. [Citado el: 15 de 11 de 2010.]
http://www.elai.upm.es/spin/Investiga/GCII/personal/mhernandez/pagina_personal_de_Miguel_Hernandez.html.

www.drbd.org. *www.drbd.org.* [En línea] [Citado el: 5 de 4 de 2011.] http://www.drbd.org/docs/about/.

www.drbd.org. *www.drbd.org.* [En línea] [Citado el: 22 de 1 de 2011.] http://www.drbd.org/users-guide/ch-heartbeat.html.

www.linuxvirtualserver.org. *www.linuxvirtualserver.org.* [En línea] [Citado el: 25 de 6 de 2011.] http://www.linuxvirtualserver.org/Documents.html.

www.loadbalancing.org. www.loadbalancing.org. *www.loadbalancing.org.* [En línea] [Citado el: 25 de 3 de 2011.] http://www.loadbalancing.org/#DNS_Load_Balancing_.

www.redmine.org. *www.redmine.org.* [En línea] [Citado el: 23 de 12 de 2010.] http://www.redmine.org/wiki/redmine/Plugin_List.

ANEXOS

Anexo 1. Documento de diseño de la solución propuesta.

En este anexo se muestra el documento de diseño de la solución propuesta. Este se encuentra en formato digital en la carpeta que se adjunta a este informe.

Instalación y configuración de la solución de clúster para el despliegue del Redmine.

Nombre del archivo: Guía de instalación de un clúster de alta disponibilidad con balanceo de carga para el despliegue del Redmine.pdf
Ubicación: Solución\Guía Clúster\