ESCALABILIDAD Y OPTIMIZACIÓN EN APLICACIONES WEB UTILIZANDO TÉCNICAS DE BALANCEO DE CARGA HTTP

TRABAJO DE GRADO

LUIS GIOVANNY CARREÑO ORTIZ

Director:

ISAAC ZUÑIGA SILGADO

Asesor:

JAIRO ENRIQUE SERRANO CASTAÑEDA

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

CARTAGENA DE INDIAS D.T. Y C.

2016

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

Lista de Gráficas

Agradecimientos

Gracias a Dios porque sin el nada sería, gracias a mis padres por apoyarme y ayudarme en esta etapa de mi vida a ellos le debo todo, gracias a mi hermana por ser luz en mi vida, gracias a mi novia por su apoyo incondicional, a todos mis familiares y amigos por su compañía indispensable.

También agradezco a los profesores de la universidad Tecnológica de Bolívar, por su contribución en mi formación, en especial a el profesor Isaac Zuñiga y al profesor Jairo Serrano por ser parte de este trabajo de investigación.

"Totus Tuus Maria"

1 CAPÍTULO I: DESCRIPCIÓN DEL PROYECTO DE INVESTIGACIÓN

1.1 Introducción

La escalabilidad de sistemas web hace referencia a un conjunto de técnicas dentro del campo de la computación, las cuales en conjunto aumentan la capacidad de procesamiento de un sistema, mejorando así el rendimiento del mismo. Por lo tanto, la escalabilidad web es utilizada para implementar soluciones flexibles que soportan: el procesamiento concurrente, la optimización en los tiempos de atención y el bajo consumo de recursos; proporcionando un desempeño superior dentro de una plataforma web.

En el año 1999 Dan Kege visualizo que un servidor web tenía que soportar diez mil clientes simultáneos (Problema C10K), esto fue propuesto debido al gran crecimiento que había experimentado la web; desde entonces muchas investigaciones y diversas tecnologías han sido desarrolladas: sofisticados protocolos, técnicas de balanceo de carga HTTP, servidores especializados, servidores orientados a eventos; han permitido un gran avance dentro del campo de la escalabilidad de sistemas web.

Por esto, es importante para el profesional en computación poder conocer e implementar soluciones de escalabilidad web, las cuales le permitan afrontar diversas problemáticas que se presenten dentro del campo profesional. El desarrollo de aplicaciones y sitios web especializados, actualmente es una de las actividades principales y su utilización dentro del campo empresarial es cada vez más común, mostrando un crecimiento exponencial.

Este trabajo de investigación propone implementar la escalabilidad horizontal en servidores web, basándose en la técnica de balanceo de carga HTTP; utilizando un servidor de balanceo nginx en modo proxy inverso. Este tipo de escalabilidad soporta el manejo de conexiones TCP concurrentes, distribución de peticiones HTTP basadas en el balanceo dinámico, y permite la agregación de nuevas instancias de servidores para soportar el crecimiento continuo.

En este trabajo se presentarán diversos escenarios de prueba, con el fin de realizar análisis comparativos entre cada uno de ellos. También se describirán cada uno de los aspectos técnicos que se deben de realizar para poder implantar una solución integral dentro de una plataforma web, tales como clúster de datos, sincronización de archivos y configuración de servidores.

El aporte de este trabajo se fundamenta en proveer un marco de referencia para los profesionales de computación, en donde puedan obtener las bases fundamentales para aplicar soluciones de escalabilidad dentro de sus proyectos de desarrollo web y también generar nuevas temáticas de investigación dentro de la comunidad académica de la Universidad Tecnológica de Bolívar.

1.2 Planteamiento del problema

1.2.1 Descripción del problema

El aumento de tráfico HTTP simultáneo sobre un servidor web causa una sobrecarga de procesamiento afectando el funcionamiento del sistema. En una aplicación web existen aspectos importantes que definen el rendimiento, uno de ellos son los tiempos de respuesta, estos indican que tan rápido puede responder el servidor a una petición de un cliente.

Cuando se despliega por primera vez una aplicación no siempre se tiene presente diferentes escenarios de funcionamiento. Dentro de la etapa inicial, poco tiempo después de su lanzamiento; el rendimiento y capacidad de trabajo son aceptables y cumplen con los requerimientos de los clientes. Pero a medida que aumentan el número de usuarios al igual aumenta la cantidad de datos almacenados y el acceso a ellos; las peticiones HTTP cada vez son más concurrentes haciendo que aumente el flujo de procesamiento hasta el punto de sobrepasar la capacidad del sistema. En este punto, aspectos como tiempos de respuesta y disponibilidad de servicio se ven afectados negativamente.

Por lo tanto, dentro de este contexto se presenta la necesidad de aplicar soluciones tecnológicas para afrontar los inconvenientes que trae consigo el crecimiento de una aplicación; la escalabilidad horizontal se muestra como un enfoque adecuado, dando la posibilidad de ejercer un mayor control sobre la arquitectura en general.

1.2.2 Formulación del problema

- ¿Cómo se puede mantener tiempos de respuesta aceptables, cuando aumenta el número simultáneo de peticiones HTTP, bajo un sistema CMS Drupal?

- ¿Qué clase de tecnologías son necesarias para soportar las conexiones simultáneas de usuarios sobre servidores web?

- ¿Se puede aumentar la tasa de procesamiento (req/seg) de un sistema CMS Drupal mediante el balanceo de carga HTTP?

1.2.3 Hipótesis

El balanceo de carga HTTP es una técnica ideal para mejorar los tiempos de respuesta cuando aumenta el número de peticiones simultáneas, bajo una aplicación CMS Drupal.

1.3 Justificación

Para poder manejar la escalabilidad de peticiones HTTP y optimizar los tiempos de respuesta entre los clientes y los servidores web, se implementará un sistema de balanceo de carga especializado en aplicaciones y sitios web.

Este sistema cuenta con un nuevo enfoque denominado servidor proxy inverso, el cual mejora la comunicación entre los múltiples servidores web y los clientes, este tipo de servidores tienen una diversa gama de funcionalidades y beneficios tales como: Balanceo de carga (HTTP/TCP), Proxy caching, Soporte multi-protocolo (HTTP/1.1, HTTP/2, HTTPS, SPDY, WebSocket, IMAP, POP3, SMTP, etc.), monitoreo, fácil configuración, seguridad (SSL/TLS), tolerancia a fallos, alta disponibilidad, optimización de hardware entre otros.

Su despliegue disminuye los costos de hardware puesto que es una solución OpenSource, y está diseñada para operar sobre cualquier sistema operativo de la industria, su diseño está enfocado para optimizar el consumo de los recursos de procesamiento tales como: CPU, RAM y almacenamiento; además cuenta con soporte para mejorar la comunicación entre los procesadores de código, utilizando los protocolos (FastCGI, memcached, SCGI, uWSGI), dando así soporte para diversos lenguajes de programación tales como: Python, PHP, Ruby, Node.js, Perl, etc.

1.4 Objetivos

1.4.1 General

Implementar una solución de escalabilidad para aplicaciones web a través del balanceo de carga HTTP, para manejar el alto tráfico de peticiones y la concurrencia de las mismas.

1.4.2 Específicos

- Implementar múltiples servidores web los cuales en conjunto servirán una misma aplicación o sitio web, utilizando la técnica de servidor proxy inverso, donde se aplicarán métodos para el balanceo de carga HTTP.

- Implementar un clúster de base de datos en modo master-master, para la replicación de datos entre los múltiples nodos servidores.

- Elaborar un entorno de pruebas de rendimiento, entre una plataforma web con balanceo de carga HTTP y uno sin él, para analizar los distintos entornos de rendimiento.

1.5 Resumen de capítulos

1.5.1 Capítulo 1 - Descripción del proyecto

En este capítulo inicialmente se presenta la introducción de la investigación, donde se exponen los lineamientos generales del campo investigativo y la naturaleza de la investigación, luego se define el planteamiento del problema donde se describe el porqué de la problemática junto con sus aspectos más relevantes, aquí se definen las preguntas a resolver y la hipótesis investigativa. También se expone la justificación de la investigación y finalmente se estipulan los objetivos generales y específicos.

1.5.2 Capítulo 2 – Contextualización de la investigación

Aquí se desarrollan una serie de conceptos y definiciones, las cuales son base fundamental para obtener el conocimiento mínimo necesario que se requiere para comprender el problema de investigación. Los componentes principales en este capítulo son el marco teórico y el estado del arte, los cuales tiene como objetivo fundamental servir de marco referencial, para comprender, conocer y analizar los detalles y principales aspectos teóricos y técnicos de la escalabilidad y optimización en sistemas web.

1.5.3 Capítulo 3 – Metodología y alcance

La definición de la metodología y los alcances del proyecto se plasman en este capítulo. Aquí se expone el tipo de alcance y el tipo de metodología utilizada, se describe cada una de las etapas metodológicas, las cuales son: planificación, investigación preliminar, diseño y modelado, implementación y pruebas, y documentación. También se define el proceso metodológico de la implementación para cada uno de los objetivos de la investigación.

1.5.4 Capítulo 4 - Implementación del Sistema de Optimización y Escalabilidad web

Este trabajo investigativo propone la implementación de un sistema de balanceo de carga, el cual está conformado por 3 componentes principales, los cuales son: entorno múltiple de servidores, balanceador de carga, y clúster de base de datos. En este capítulo se describen todos los aspectos técnicos de la implementación del sistema, tales como requerimientos de software y hardware. Aquí cada componente del sistema se expone por medio de tres etapas, la etapa de instalación, configuración y despliegue. Finalmente se presenta el funcionamiento total del sistema, utilizando como aplicación de prueba un sitio web sobre un sistema CMS Drupal.

1.5.5 Capítulo 5 - Pruebas y validaciones del Sistema de Optimización y Escalabilidad web

En este capítulo se desarrollan una serie de escenarios de prueba, donde se aplica el balanceo de carga y la optimización de servidores. Cada escenario es sometido a un conjunto de pruebas, por medio de una herramienta de testing llamada TSUNG, con el fin de analizar las variables de rendimiento; tales como tiempos de respuesta, tasa de rendimiento (req/seg), usuarios simultáneos, tasa de transmisión, etc. Los resultados obtenidos en cada escenario, son utilizados para realizar análisis comparativos, y así, poder determinar los mejores entornos de rendimiento.

1.5.6 Capítulo 6 - Conclusiones y trabajos futuros

Se presentan las conclusiones de la investigación, aquí se describe cuáles fueron los puntos importantes durante el desarrollo del trabajo investigativo, se resalta los beneficios hallados, y los factores técnicos de mayor consideración. También se describe el proceso metodológico, y la forma en que se evalúalo la hipótesis de investigación. Se mencionan los resultados obtenidos, los cuales verifican la hipótesis planteada y los objetivos del proyecto de investigación, al final se expone los trabajos de mejoramiento, los cuales son punto de partida para mejorar el trabajo actual.

2 CAPÍTULO II: CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN

2.1 Marco teórico

En esta sección se desarrollarán una serie de conceptos y definiciones, las cuales son fundamentales para obtener el conocimiento mínimo necesario que se requiere para comprender el problema de investigación.

Esta base teórica tiene como objetivo fundamental servir de marco referencial, para comprender y analizar los detalles y principales aspectos de la escalabilidad y optimización en sistemas web.

2.1.1 Escalabilidad web

2.1.1.1 Computación y escalabilidad

La palabra computación hace referencia a la acción y efecto de computar, realizar una cuenta o un cálculo matemático. Un sistema de cómputo (Servidor web) tiene como tarea principal “realizar cálculos” y dicha actividad la realiza de una forma efectiva. Debido a razones de diseño y propiedades físicas todo sistema de cómputo tiene limitaciones en cuanto a la capacidad de procesamiento. Por ejemplo, una persona puede sumar 2 números de una forma efectiva, pero si le pedimos que sume 10 o 100 números simultáneamente, podríamos estar sobrepasando su capacidad de procesamiento.

Un aplicación o sitio web funciona de igual forma, el cálculo o procesamiento requerido para atender peticiones de 10 clientes no es el mismo para atender peticiones de 100 o 1000 clientes, si hablamos de atenciones concurrentes o simultáneas.

Escalabilidad

La escalabilidad es la capacidad de un sistema, que indica su habilidad para adaptarse al crecimiento de trabajo sin perder la calidad. Para una plataforma web existen dos maneras de escalabilidad, enfocada a servicio y enfocada a servidor. La primera llamada escalabilidad vertical, la cual añade más recursos a la máquina. La segunda llamada escalabilidad horizontal, la cual separa las funcionalidades del sistema para crear componentes individuales que conformen un todo.

2.1.1.2 Tipos de escalabilidad web

2.1.1.2.1 Escalabilidad vertical

La escalabilidad vertical se centra en aumentar las especificaciones técnicas de un componente de cómputo. Tomemos como ejemplo un escenario donde una máquina que escribe errores de memoria en el log del sistema advierte que la cantidad de memoria no soporta las condiciones de trabajo.

Por sentido común este problema se podría abordar añadiendo más memoria RAM a la máquina, sin embargo, no sería lo adecuado. La escalabilidad vertical aumenta las características físicas de la máquina, lo cual hace que aumente la capacidad de procesamiento, pero dentro de una arquitectura web los problemas de procesamiento no son de esta índole; controlar conexiones TCP simultáneas, procesar peticiones HTTP de clientes simultáneos y múltiples accesos a un sistema de datos, no son tareas que se puedan soportar añadiendo más CPU o nuevos módulos de memoria RAM, debido a que la arquitectura de un servidor web estipula la lógica de procesamiento para atender las peticiones HTTP y es ella quien determina el rendimiento del servidor independientemente de las características técnicas de la máquina donde esté ejecutándose.

2.1.1.2.2 Escalabilidad horizontal

La escalabilidad horizontal agrega más unidades de procesamiento (Máquinas físicas) para distribuir la carga de procesamiento entre diferentes nodos que componen un todo. Supongamos que el servidor de base de datos está creciendo en gran cantidad de entradas y el servidor web muestra aumentos de peticiones HTTP.

Para ello la separación del servidor web y servidor de base de datos en máquinas diferente sería la clave, más aún podríamos añadir más servidores web que funcionan sirviendo la misma aplicación.

La Figura 1 muestra de una forma gráfica los enfoques que representa cada tipo de escalabilidad, se puede apreciar claramente cuál es el comportamiento de cada tipo.

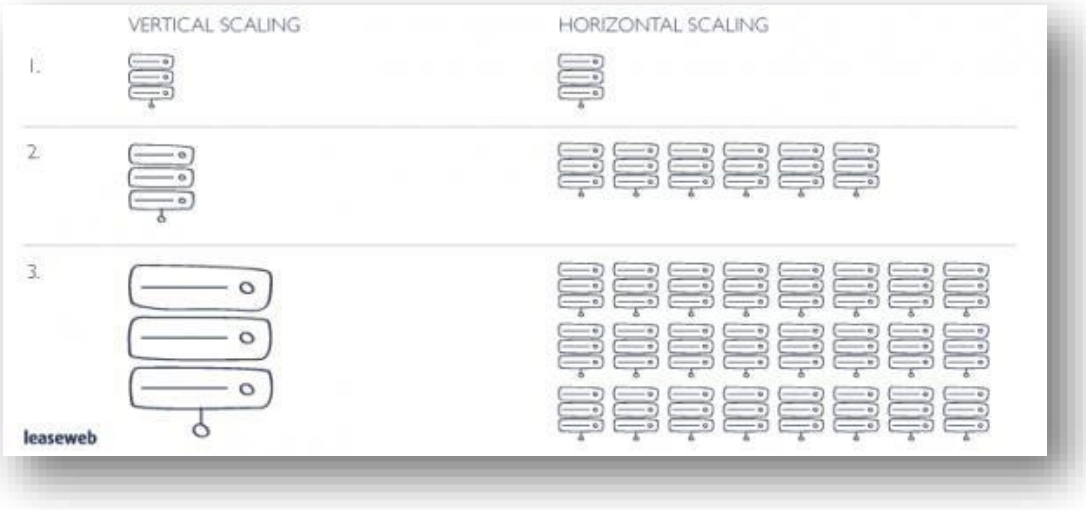


Figura 1: Esquema gráfico escalabilidad vertical y horizontal

Fuente: http://blog.leaseweb.com/uploads/2014/05/horizontal-and-vertical-scaling-e1400767862256.png

2.1.1.3 Rendimiento de sistemas web

El rendimiento de una aplicación o sitio web está directamente relacionado con el tiempo de respuesta, el cual se define como: “la cantidad total de tiempo que se necesita para responder la solicitud de un servicio”. Este se divide en dos facetas, el tiempo de servicio y el tiempo de espera; el tiempo de servicio es el tiempo utilizado para procesar la petición (Procesamiento del servidor) y el tiempo de espera es aquel que transcurre mientras la petición permanece en cola antes de iniciar el tiempo de servicio. Estas dos facetas están representadas por diversos procesos que operan sistemáticamente para responder o generar la respuesta al cliente.

A continuación, se definirán diversos conceptos fundamentales para comprender los procesos que definen el rendimiento de un sistema web.

Protocolo HTTP

Hypertext Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto), es un protocolo de comunicación que permite la transferencia de información dentro de la WWW (World Wide Web). Una forma pedagógica de definir este protocolo podría ser “el idioma de comunicación entre un navegador y un servidor web”, aquí se define la estructura para realizar una petición HTTP y la estructura para generar una respuesta HTTP. Este protocolo es de vital importancia para el rendimiento, puesto que una mejor comprensión entre el cliente y el servidor disminuye el tiempo necesario para transferir información.

Petición HTTP (HTTP request)

Una petición HTTP es el proceso por el cual un navegador o cliente web envía datos al servidor pidiendo algún tipo de información. Este proceso es comúnmente realizado cuando escribimos una dirección URL en el navegador.

- Estructura de una petición GET HTTP

```
GET /docs/index.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

Respuesta HTTP (http response)

Una respuesta HTTP es el mensaje generado por un servidor web luego de haber recibido una petición, este mensaje es enviado hacia el cliente que realizo la petición inicialmente.

- Estructura de una respuesta HTTP

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

Con base a los anteriores conceptos se puede deducir que los sistemas web funcionan bajo un mecanismo de solicitud y entrega, para comprender este comportamiento analizaremos el ciclo de vida de una petición HTTP.

2.1.1.3.1 Ciclo de vida de una petición HTTP

Una petición HTTP culmina en el mismo punto donde inicia. Por ejemplo, la petición inicia en el navegador, luego es trasmitida por una serie de protocolos hacia el servidor, este procesa y genera la respuesta que será enviada de vuelta

hacia el navegador. Por lo tanto, es clave analizar este clico ya que recorre todos los puntos dentro del sistema web.

Estructura del ciclo de vida de una petición HTTP

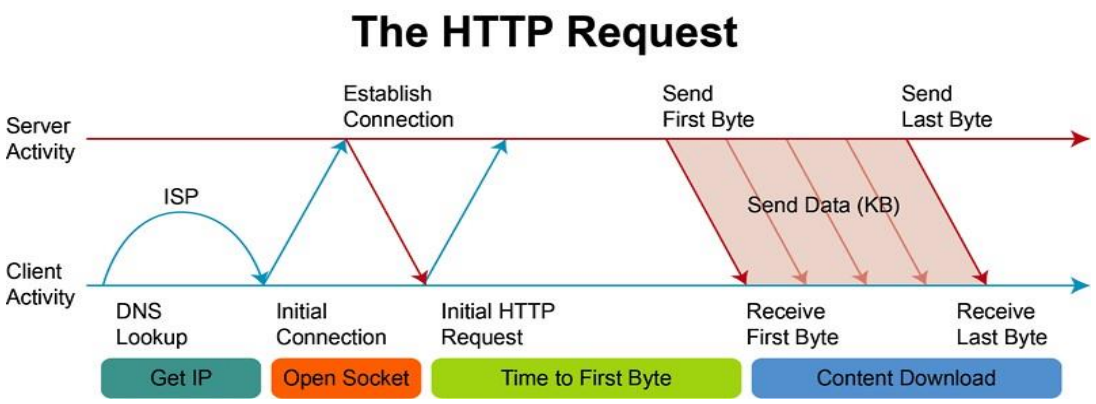


Figura 2: Línea de vida de una petición HTTP

Fuente: http://www.nycwebapps.com/wp-content/uploads/2012/12/httprequest.png

La anterior representación (Figura 2) describe a grandes rasgos los procesos del ciclo de vida de una petición HTTP, cada fase del ciclo involucra una serie de elementos los cuales repercuten en el rendimiento del sistema.

1. Obtención de la dirección IP

Para iniciar cualquier comunicación entre dos computadoras dentro de internet se necesitan dos direcciones IP, una del lado del cliente y otra del lado del servidor. Cuando el cliente va a iniciar la comunicación, primero pregunta cuál es la dirección IP del servidor puesto que no conoce hacia dónde va enviar la petición; el usuario o la persona únicamente ingresa la URL en el navegador y no la dirección IP del servidor. Este proceso de identificación de la dirección IP se realiza mediante el servicio DNS quien realiza la traducción de dominio hacia la dirección. Este proceso de traducción agrega cierta demora o latencia para la comunicación, aumentando el tiempo de respuesta.

2. Establecer la conexión TCP

Para enviar un flujo de datos entre dos computadores inicialmente se debe crear una conexión TCP. Una conexión TCP hace referencia a los mecanismos y procesos que realiza el protocolo de control de transmisión (TCP), para intercambiar datos entre un servidor y un cliente de una forma ordenada y sin pérdida de información.

Conexión TCP

Figura 3: Utilización del puerto 8080 para establecer una conexión TCP

Fuente: Propia

EL manejo de conexiones TCP es vital para el rendimiento de un servidor web, puesto que cada sesión de un usuario web necesariamente debe abrir una conexión TCP. La administración y manejo de estas conexiones son muy costosas, ya que son manejadas por procesos o hilos, los cuales consumen grandes porcentajes de recursos como RAM y CPU.

Por otra parte, el protocolo UDP también posee la capacidad de transmitir información entre dos computadores, este funciona en la capa de transporte del modelo OSI y está basado en el intercambio de datagramas. Aunque no es confiable para transmitir información dentro de un sistema web, puesto que no contiene control de flujo y no hay forma de conocer si los paquetes llegan a su destino; esto se conoce como un protocolo no orientado a conexión.

3. Respuesta HTTP del servidor web

Luego que se establezca una conexión estable TCP, los datos de la petición son enviados hacia el servidor, este los recibe e inicia el procesamiento para generar la respuesta. En este punto el servidor web se encarga de utilizar los procesadores de código para generar páginas dinámicas, y luego enviar diferentes archivos tales como, HTML, CSS, JS, Imágenes, Audio, etc. El rendimiento de esta etapa se centra en los procesadores de código y la arquitectura del servidor; un lenguaje que se ejecute de una forma rápida y un servidor que genere rápidas respuestas (http response) hacen que se minimicen los tiempos de respuesta.

La *Figura 4* describe 3 etapas dentro de un sistema web: Cliente web, Web server y comunicación con la aplicación web. La etapa de “Web server” muestra cómo el servidor utiliza el protocolo CGI para comunicarse con la aplicación web y de esa forma poder generar la respuesta HTTP.

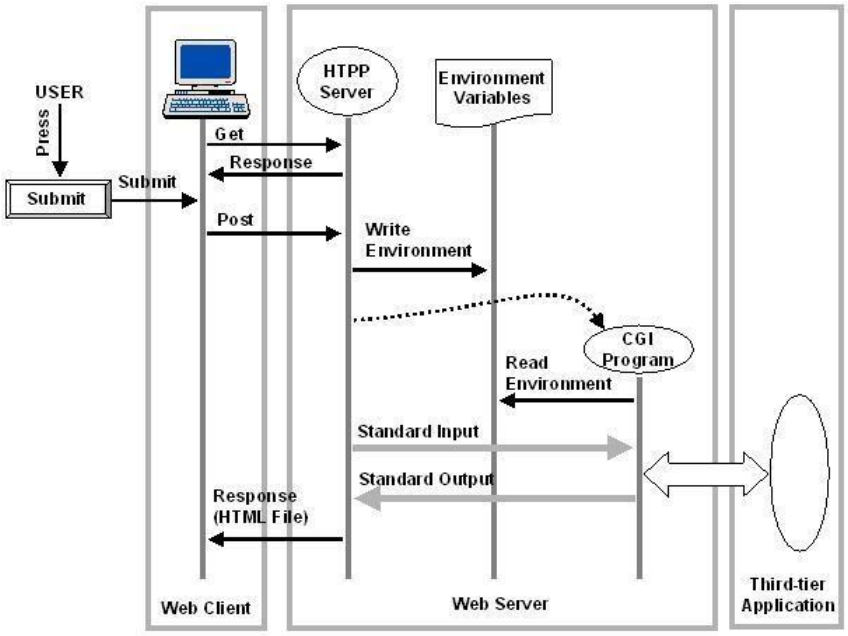


Figura 4: Procesamiento interno de un servidor web utilizando la tecnología CGI (Common Gateway Interface) para generar respuestas HTTP)

Fuente: O'Reilly - Web Performance Tuning [2]

4. Envió y descarga de contenido

Esta etapa inicia cuando el servidor envía el primer byte hacia el cliente, aquí los protocolos de transmisión y las especificaciones de red son los principales agentes. Una arquitectura de red que posea un amplio ancho de banda permite la fácil y rápida transmisión de datos en el menor tiempo posible, esta es una de las especificaciones de red más importantes, aunque su rendimiento depende del ISP y no de la plataforma web como tal.

5. Obtención de la respuesta

El cliente (Navegador web) recibe la respuesta del servidor, la respuesta puede contener diferentes tipos de archivos tales como imágenes, videos, audios, archivos de código HTML, CSS, JavaScript, etc. Estos archivos luego son procesados por el motor del navegador para luego ser presentados en la pantalla del computador.

2.1.1.4 Procesamiento en servidores web

Un servidor web (También llamado Servidor HTTP) es un programa, el cual tiene desarrollado dentro de su núcleo un módulo para utilizar el protocolo HTTP. Este módulo es fundamental para procesar las peticiones y generar la respuesta HTTP, puesto que utiliza los estándares estipulados en el protocolo HTTP para poder transferir contenido entre un cliente y servidor web.

Contenido estático y dinámico

El rendimiento dentro de un servidor web está sometido a una gran variación debido a la magnitud de la carga. Una petición de una página HTML estática, conlleva menos procesamiento que una petición a una página dinámica.

Un contenido estático es un archivo almacenado en el sistema de archivos del servidor, por ejemplo, un archivo que solo tenga código HTML, mientras que un

contenido dinámico es aquel que contiene código PHP, JAVA, C#, etc. La diferencia de rendimiento radica en que un contenido dinámico utiliza los procesadores de código los cuales no son utilizados en un contenido estático. Además, los contenidos dinámicos manejan código que consultan bases de datos y esto aumenta mucho más el procesamiento dentro del servidor web.

Aumento de la demanda

Un servidor web es un programa que consume recursos del sistema como cualquier otro, sin embargo, cada petición HTTP entrante hace que este aumente el consumo. Un sistema operativo UNIX contiene diversos recursos que manejar y procesar, por ejemplo, los recursos de red, disco, memoria, CPU, etc. Si dentro de este se instala un servidor web (Programa), este mediante el kernel de UNIX consume dichos recursos, por lo tanto, el aumento de peticiones demanda mucho procesamiento dentro del sistema.

La figura 5 muestra cada uno de los componentes tanto de hardware y software que intervienen en el procesamiento de una petición HTTP.

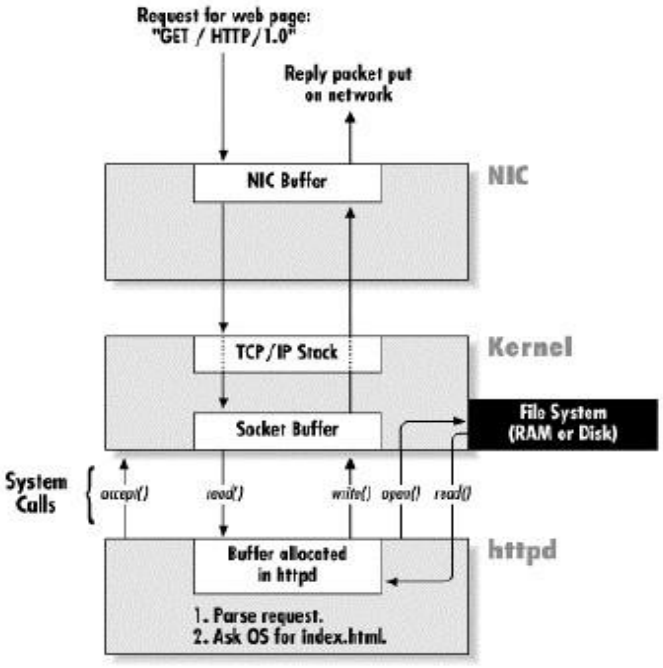


Figura 5: Interacción entre un S.O. y un servidor web para el procesamiento de peticiones HTTP

Fuente: O'Reilly - Web Performance Tuning [2]

La anterior figura (Figura 5) muestra la interacción del recurso de red (NIC), el cual es administrado por el sistema (kernel), luego presenta la interacción entre el servidor web (http demonio ejecutado en el núcleo del servidor web) y el sistema operativo. Esta interacción fluctúa dependiendo el diseño y arquitectura del servidor web, el mejoramiento en cuanto al manejo de procesos e hilos por parte del demonio del servidor web es fundamental para el óptimo procesamiento de peticiones HTTP concurrentes.

2.1.1.4.1 Capacidad del servidor

A diferencia de otros sistemas el parámetro relevante de un servidor web son las operaciones HTTP por segundo, también referidas como atenciones por segundo. Los servidores web no mantienen una dedicada conexión con el navegador porque HTTP fue implementado fundamentalmente como un protocolo muy simple, para poder conservar el ancho de banda, y permitir a las páginas web alojar contenidos de múltiples servidores. Aunque los usuarios perciben la impresión de estar conectados cuando están leyendo una página web, desde el punto de vista del servidor, los usuarios desaparecen después de cada petición y vuelven aparecen solo cuando realicen una nueva petición. Actualmente el protocolo HTTP tiene muchas mejoras y es capaz de percibir sesiones de usuarios mediante el uso de “cookies.

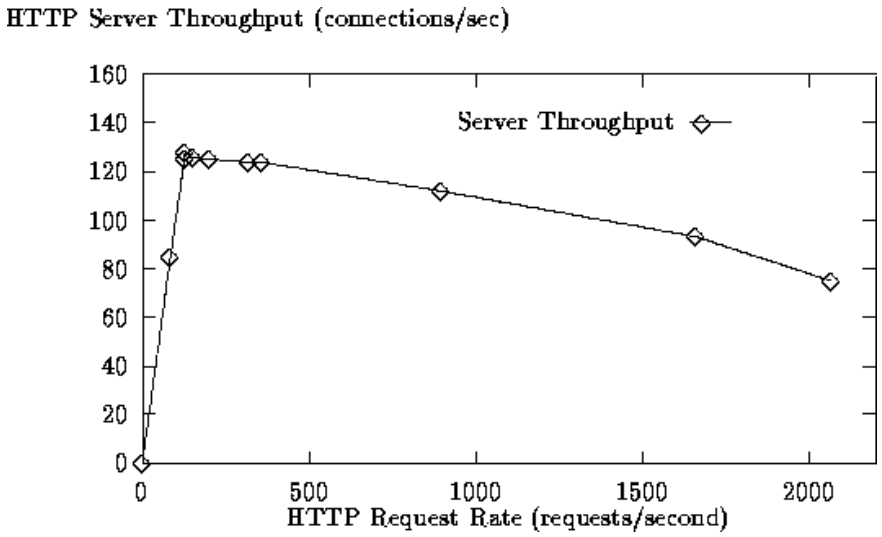
Las cookies son archivos que utiliza el servidor web para identificar un cliente, estos archivos son generados inicialmente dentro del servidor y luego enviados al cliente. Tanto el cliente como el servidor deben de almacenar la misma cookie para poder mantener la sesión. También existen otro mecanismo o técnica el cual consiste en enviar el ID de usuario como parámetro dentro de la URL al servidor, para que este pueda identificar la sesión correcta.

Una sesión HTTP inicia cuando el cliente realiza por primera vez una petición HTTP al servidor (Petición HTTP: Es el proceso por el cual un cliente web solicita a un servidor web la realización de una operación específica) y termina poco

tiempo después de la última, dependiendo de las características de servidor web. Una sesión HTTP se crea para poder intercambiar datos de una forma interactiva entre un cliente y un servidor, durante el intercambio de datos el servidor almacena información relevante, tal como: tipo request, ID del cliente, etc. Luego que se detiene el intercambio de información el servidor elimina dentro de su sistema todos los datos relacionado con ese cliente y así se termina la sesión HTTP.

Cuando múltiples usuarios inician una sesión dentro del servidor en un mismo lapso de tiempo, se presentan las denominadas peticiones u operaciones HTTP por segundo. Esto demuestra que la carga real de un servidor web obedece a una función estadística con respecto al tiempo, la cual tiene diferentes variaciones.

Si se realiza una medición durante un minuto teniendo 10 sesiones (usuarios) simultáneas los cuales realizan múltiples peticiones cada lapso de tiempo determinado, podemos observar que durante cada segundo el servidor atenderá múltiples peticiones de distintos usuarios, teniendo en cuenta que debido a su comportamiento aleatorio en todos los segundos no se atenderán el mismo número de peticiones simultáneas. Por ejemplo, en los 20 primero segundo se atenderán muchas más peticiones que en los 40 segundo restantes.



Gráfica 1: Tasa de rendimiento de un servidor HTTP

Fuente: aosabook.org

Un sitio web puede recibir 1 millón de peticiones diarias distribuidas uniformemente, pero particularmente no es una alta carga por segundo, cuando se realiza el cálculo de la tasa de rendimiento del día, 100000 / (60 x 60 x 24) = 11.6 req/segundo, teniendo en promedio 10k de transferencia por segundo; esta carga está dentro de las capacidades del servidor teniendo una plataforma de hardware modesta.

Variables de capacidad

La capacidad del servidor está sujeta a la variación de carga, la cual depende del tipo de petición y su concurrencia.

Parámetros clave en cuanto al tipo de petición HTTP:

- Tamaño del archivo pedido.
- Contenido adicional, imágenes, videos, etc.
- Ejecución de código dinámico (PHP, Java#, etc.)
- Número de consultas a la base de datos

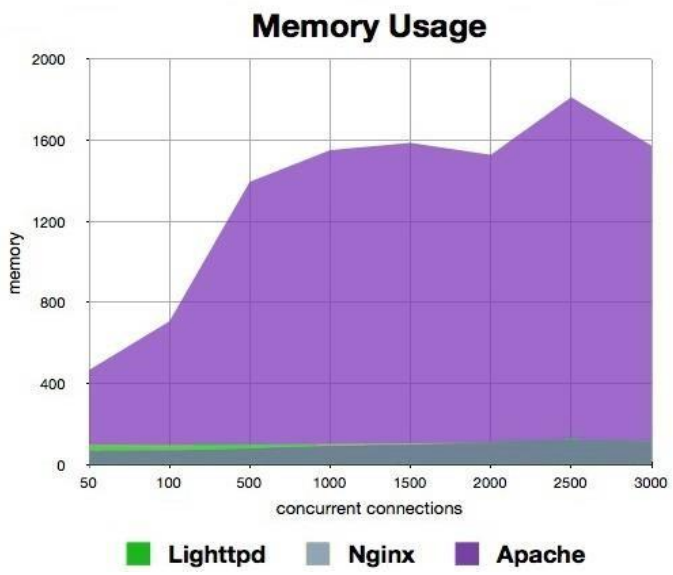
En un supuesto panorama un servidor web o sistema web puede soportar cientos de peticiones por segundo mientras transmite una simple página web (Archivo de código HTML puro); si este panorama cambia y las peticiones conllevan accesos a la base de datos, carga de imágenes y videos, este servidor no podría soportar la misma tasa de rendimiento y en muchos casos afecta su funcionamiento.

2.1.1.5 Consumo de recursos en servidores web

2.1.1.5.1 Uso de memoria RAM

La creación de procesos e hilos son muy costoso en cuanto al consumo de memoria, una petición HTTP es manejada por un proceso o un hilo dependiendo del diseño del servidor o su configuración. Cada servidor web (Apache, Nginx, IIS,

etc.) consume cierta cantidad de memoria por cada proceso o hilo creado dentro del sistema.



Gráfica 2: Consumo de memoria RAM en servidores web

Fuente: http://wiki.dreamhost.com/File:Webserver_memory_graph.jpg

Existen varias actividades por la cual los servidores web utilizan o consumen memoria RAM:

- Por creación de procesos o hilos
- Por creación de conexiones
- Por creación de host virtuales

En el manejo de procesos e hilos interviene la ejecución del código (PHP, C#, Python, etc.), se carga librerías y demás componentes del SO para poder procesar la petición; en la creación de conexión la utilización de puertos TCP y la admiración de socket de red son tarea fundamental y conllevan un gran consumo de memoria; en la parte de Host virtuales se presenta un alto consumo puesto que se sirven múltiples sitios o aplicación web dentro del mismo servidor físico, aumentando la demanda de procesamiento.

2.1.1.5.2 Uso de CPU

La relación de consumo entre un servidor web y los recursos de CPU, están ligados a la creación de procesos dentro del sistema operativo. La siguiente imagen muestra la descripción de un proceso creado dentro de una máquina Linux.

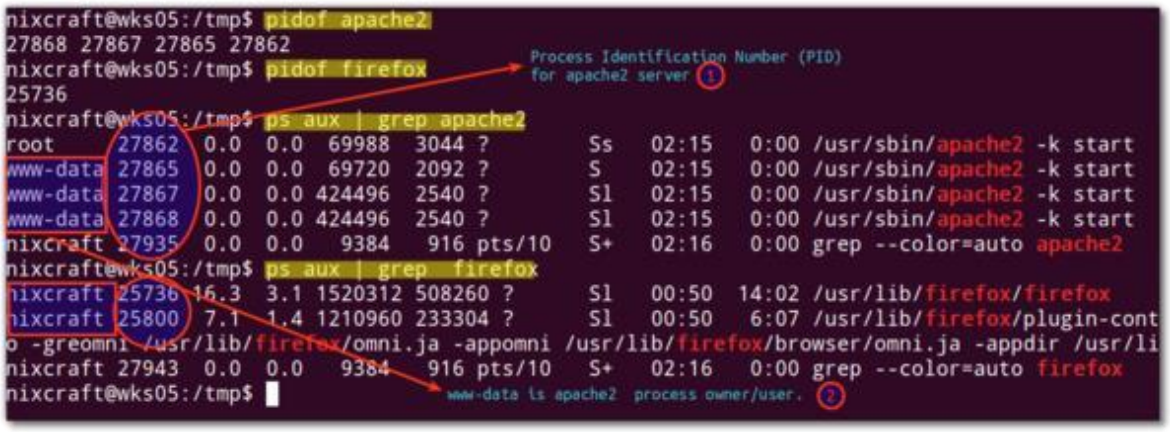


Figura 6: Procesos de Apache Web server creados en Linux

Fuente: nixcraft.com

El proceso dentro de Linux es identificado por un ID único de proceso (PID) y es asociado al usuario que lo ejecuta, en el caso anterior es www-data usuario creado para Apache web server. Los procesos usan un porcentaje específico de la CPU, este porcentaje varía dependiendo la magnitud de la petición y lo que requiera el núcleo de apache para procesar la petición. Existen algunas herramientas para plataformas Linux que identifican el consumo de los recursos de CPU, uno de ellas se llama Glances, esta permite conocer con gran exactitud el porcentaje de un programa en ejecución, La siguiente figura muestra un ejemplo del uso de Glances.

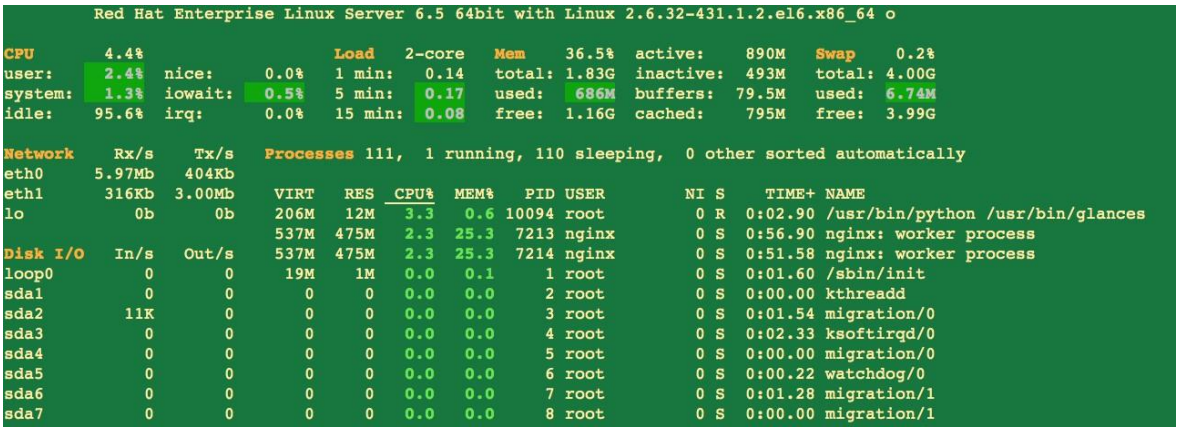


Figura 7: Sistema de monitoreo de consumo de recurso, Glances 1.7

Fuente: nixcraft.com

La *figura 7* muestra un reporte realizado por Glances ejecutándose en un servidor Red Hat Server 6.5, se muestra en la sección de procesos que el porcentaje de CPU utilizado por el servidor web Nginx es 2.3 %, cabe mencionar que este porcentaje no obedece a la carga de una única petición HTTP, si no a todas las peticiones que en ese determinado instante este procesando Nginx; por lo tanto, este porcentaje varía dependiendo la carga actual dentro del sistema.

2.1.2 Balanceo de carga (Load balancing)

En computación, el balanceo de carga es la técnica por el cual se distribuyen las cargas de trabajo/procesamiento a través de múltiples recursos computacionales tales como: computadores, clúster de computadores, enlaces de red, unidades centrales de procesamiento (CPU) o unidades de disco.

El balanceo de carga tiene como objetivo:

- Optimizar el uso de recursos
- Maximizar el rendimiento
- Minimizar el tiempo de respuesta
- Evitar la sobrecarga de cualquier recurso

Utilizar múltiples componentes para el balanceo de carga, en lugar de un solo componente, puede aumentar la fiabilidad y la disponibilidad a través de la redundancia. El balanceo de carga difiere de la técnica unión de canales ("channel bonding"), debido a que este divide el tráfico entre interfaces de red en un mismo socket de red (Modelos OSI, capa de red), mientras que la unión de canales implica una división de tráfico entre interfaces físicas en un nivel inferior al Modelos OSI (Capa de enlace de datos), utilizando el protocolo del camino más corto para determinar él envió de paquetes.

2.1.2.1 Tipos de balanceo de carga

Según [1] el balanceo de carga tiene diferentes enfoques dentro de los sistemas distribuidos, por lo cual existen dos tipos de balanceo; balanceo estático y balanceo dinámico.

A continuación, se muestra una descripción gráfica acerca de la perfección del balanceo de carga en un entorno distribuido. En la imagen se representan una serie de procesos individuales, procesando una misma tarea.

BC perfecto vs. BC imperfecto

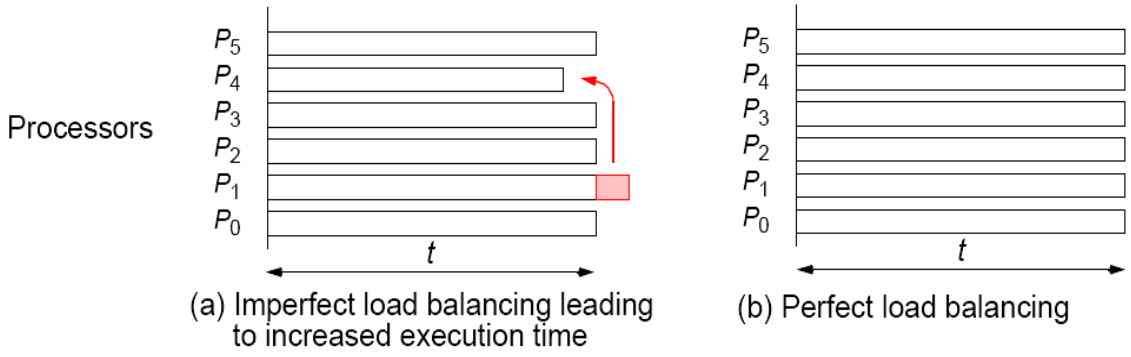


Figura 8: Representación gráfica del proceso de balanceo de carga perfecto e imperfecto

Fuente: Dynamic Load balancing University of Berkeley [17]

[1] Distributed Computing: Principles, Algorithms, and Systems. Autores : Ajay D. Kshemkalyani , Mukesh Singhal

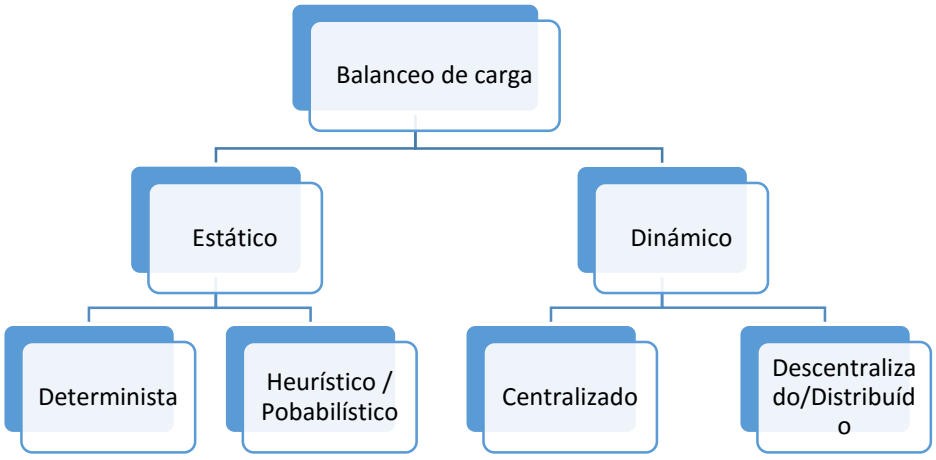
El arte de repartir

Una de las principales actividades a la hora de distribuir cargas entre diferentes procesadores o fuentes computacionales, es definir el tamaño de la proporción de trabajo que se le asignará a cada uno. Si a todos se les asigna una carga de igual tamaño, esto quiere decir que se está manteniendo una carga computacional uniforme dentro del sistema. Cuando no existe una carga uniforme, es decir, hay un desbalance en la carga, entonces algunos procesadores estarán inactivos mientras que otros todavía estarían calculando.

La forma de distribución de la carga es el principal concepto para determinar el tipo de balanceo; tanto el balanceo estático como el dinámico, manejan de forma diferente la distribución de carga. Cabe mencionar que existen diversos algoritmos enfocados a cada tipo de balanceo, pero la utilización de los mismos está directamente ligada con la problemática que se desea resolver.

Los tipos de balanceo de carga se rigen por políticas estipuladas, las políticas estáticas basan su decisión en la información estática sobre el sistema, ellas no toman en consideración el estado actual del sistema. Las políticas dinámicas basan su decisión en el estado actual del sistema, por lo cual son más complejas que las políticas estáticas.

TAXONOMÍA DE BALANCEO DE CARGA



2.1.2.1.1 Balance de carga estático

En este tipo de balanceo las tareas se distribuyen al inicio de la computación. Dentro de un contexto distribuido, nodo maestro y nodos esclavos; el nodo principal inicialmente participa en el proceso computacional asignando una fracción de la carga a cada nodo esclavo, para que luego estos inicien su tarea de procesamiento. Aquí la asignación puede ser realizar una sola vez o puede ser de manera cíclica.

En un algoritmo de balanceo estático los procesos son asignados en el tiempo de compilación acorde al desempeño de los nodos, luego de la asignación no es posible volver asignar ningún proceso a los nodos en el tiempo de ejecución. Por lo cual los algoritmos no recogen ninguna información sobre los nodos.

Algunas características del balanceo de carga estático que lo ponen en desventaja, contra el balanceo dinámico:

- Dificultad para estimar tiempo de ejecución de las partes en que se divide la carga, sin antes ejecutarlas.

- Algunos sistemas presentan retardos en la comunicación, los cuales varían dependiendo de las circunstancias. El balanceo de carga estático es incapaz de determinar los tiempos de retardo, dificultando la sincronización del sistema.

- A veces los problemas necesitan un número indeterminado de pasos para alcanzar la solución, Por ejemplo, los algoritmos de búsqueda normalmente atraviesan un grafo buscando la solución y a priori no saben cuántos caminos hay que recorrer. El balanceo estático debido a que no recoge información de los nodos, se le dificulta conocer si ya se ha alcanzado la solución.

APLICACIONES DE BALANCEO DE CARGA ESTÁTICO

La técnica de balanceo de carga estática es muy útil cuando se utilizan miles de procesadores, en donde intervienen técnicas heurísticas.

Fragmentación molecular orbital (FMO)

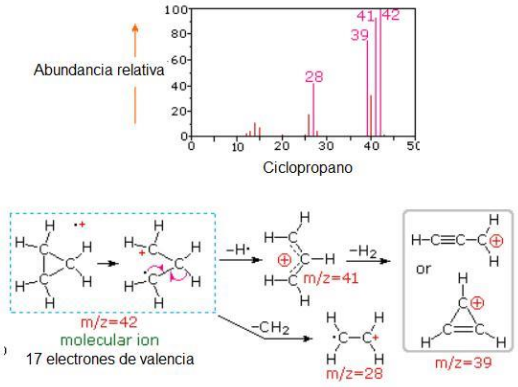


Figura 9: Balanceo de carga aplicado a la fragmentación molecular orbital

Fuente: FRAGMENT AND LOCALIZED ORBITAL METHODS IN ELECTRONIC STRUCTURE THEORY. DMITRI G. FEDOROV (11)

La fragmentación molecular orbital es un método computacional que puede computar sistemas moleculares de gran envergadura con miles de átomos; este método es utilizado aplicando en el balanceo de carga estático, un ejemplo de ello es el trabajo realizado por Dmitri G. Fedorov en el Instituto de avances de industria, ciencia y tecnología del Japón.

Networking

El enrutamiento es uno de los procesos principales en los dispositivos de red e interconexión, estos utilizan diferentes algoritmos basados en el balanceo de carga estático para distribuir la carga computacional dentro de la red.

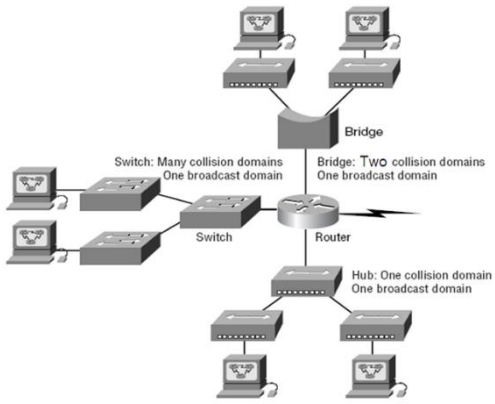


Figura 10: Topología de Red

Fuente: netacad.com

Sistemas de mallas no estructuradas

Los sistemas de mallas no estructuradas, son utilizados para la distribución de memoria en paralelo. Aquí el balanceo de carga estático tiene su aplicación, puesto que el problema de mallas, tiene una mejor solución cuando se divide en subdominios más pequeños.

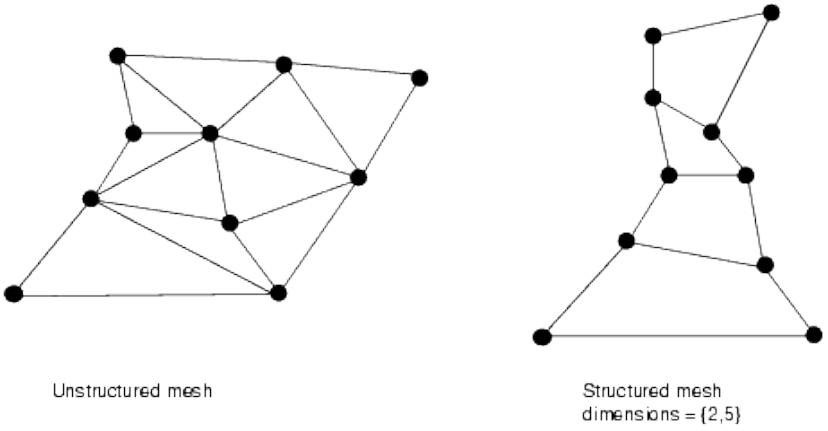


Figura 11:Balanceo de carga aplicado a la distribución de memoria en paralelo

Fuente: mathmoo.unam.mx

2.1.2.1.2 Balanceo de carga dinámico

Este tipo de balanceo se trata durante la ejecución de los procesos. La división de la carga se hace dependiendo de las partes que inicialmente se han ejecutado.

La distribución de carga dinámica inicia con un primer paso, en donde se reparten las cargas de una forma equitativa, luego las divisiones siguientes se hacen dependiendo de las partes que ya se están ejecutando inicialmente.

Este tipo de balanceo tiene como enfoque principal mantener un equilibrio de carga uniforme dentro del sistema, de modo que se minimicen dentro de los nodos el tiempo medio de respuesta de trabajo. Con el balanceo de carga dinámico, se tiene en cuenta las debilidades del balanceo de carga estático, puesto que se depende netamente de la información tomada en tiempo de ejecución y no de cálculos de estimación basados en técnicas probabilísticas. Cabe mencionar que este tipo de balanceo adiciona una sobrecarga de ejecución, debido a que inspecciona los nodos cíclicamente durante su procesamiento; sin embargo, resulta una técnica mucho más eficiente que el balanceo de carga estático.

Evaluación de rendimiento de un balanceo estático y dinámico, para computación paralela.

L#	CPUs	Speeds (GHz)	Load Distribution (%)	Load Distribution (KCells)	Aver Time/Iter (sec)	Speed-up (a)	Ideal Speed-up (b)	Efficiency (%) (a)/(b)
1	1	1.7	100	353	22	1.0	1	100
2*	**2**	**1.7-1.7**	**50-50**	**176-177**	**11**	**2.0**	**2**	**100**
3	2	1.7-1.7	70-30	247-106	16	1.4	2	70
4	2	1.7-1.7	97-03	342-11	21	1.0	2	50
5*	**3**	**1.7-1.7-1.7**	**34-33-33**	**121-116-116**	**8**	**2.8**	**3**	**93**
6*	**4**	**1.7–1.7–1.7–1.7**	**25-25-25-25**	**89-88-88-88**	**6**	**3.7**	**4**	**93**
7	4	1.7–1.7–1.7–1.7	33-33-33-01	116-116-116-5	8	2.8	4	70
8	4	1.7–1.7–1.7–1.7	70-10-10-10	248-35-35-35	14	1.6	4	40
9	4	1.7-1.7-1.7-1.7	97-01-01-01	341-4-4-4	21	1.0	4	25

* highlighted lines represent initial distribution of workload that is proportional to the speeds of the processors

Tabla 1: Registro del rendimiento utilizando Balanceo de carga estático

Registro del rendimiento utilizando Balanceo de carga estático

L#	CPUs	Speeds (GHz) (a)	SMP or Cluster (b)	Without Load balancing			With load balancing			
				Initial workload Load Distribution (%) (c)	Total Elapsed Time (sec) (d)	Avg Time/ Iteration (sec) (e)	Final workload load Distribution (%) (f)	Total Elaps Time (sec) (g)	Avg Time/ Iter (sec) (h)	Improvement or Slowdown in Elapsed Time (d)/(g)
1*	**4**	**1.7–1.7–1.7–1.7**	**SMP**	**25-25-25-25**	**467**	**6.0**	**24-25-25-26**	**515**	**7**	**0.9**
2	4	1.7–1.7–1.7–1.7	SMP	97-01-01-01	1446	21	25-26-25-24	630	6	2.3
3	4	1.7–1.7–1.7–1.7	SMP	70-10-10-10	969	14	24-27-25-24	535	6	1.8
4*	**4**	**1.7–1.7–1.7–1.7**	**Cluster**	**25-25-25-25**	**719**	**9**	**25-28-23-24**	**916**	**10**	**0.8**
5	4	1.7–1.7–1.7–1.7	Cluster	97-1-1-1	1571	21	26-27-23-24	890	10	2.1
6	4	1.7–1.7–1.7–1.7	Cluster	70-10-10-10	1128	15	24-28-23-25	535	6	2.1
7	4	1.7–1.1–1.7–1.7	Cluster	28-16-28-28	1041	10	28-15-28-29	1058	11	1.0
8*	**4**	**1.7–1.1–1.7–1.7**	**Cluster**	**25-25-25-25**	**857**	**11**	**28-15-28-29**	**948**	**10**	**0.9**
9	4	1.7–1.1–1.7–1.7	Cluster	97-1-1-1	1557	21	29-15-27-29	1230	12	1.3
10	4	1.7–1.1–1.7–1.7	Cluster	70-10-10-10	1127	14	29-15-27-29	1117	11	1.0

* highlighted lines represent balanced initial distribution of workload that is proportional to the speeds of the CPUs

Tabla 2: Registro del rendimiento utilizando Balanceo de carga dinámico

APLICACIONES DE BALANCEO DE CARGA DINÁMICO

Las diferentes aplicaciones del balanceo de carga dinámico, están dirigidas a soluciones de problemas no uniformes, con impredecible carga.

Computación Paralela

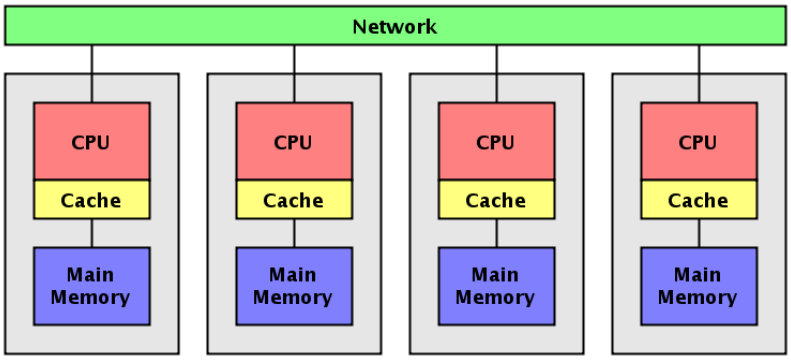


Figura 12:Balanceo de carga dinámico aplicado a la computación paralela en red (Clúster)

Fuente: OpenHMPP, A New Standard for Manycore, openhmpp.org

Las técnicas de balanceo de carga dinámica en sistemas paralelos son utilizadas para minimizar el tiempo de ejecución de una aplicación.

Escalabilidad de sistemas distribuidos

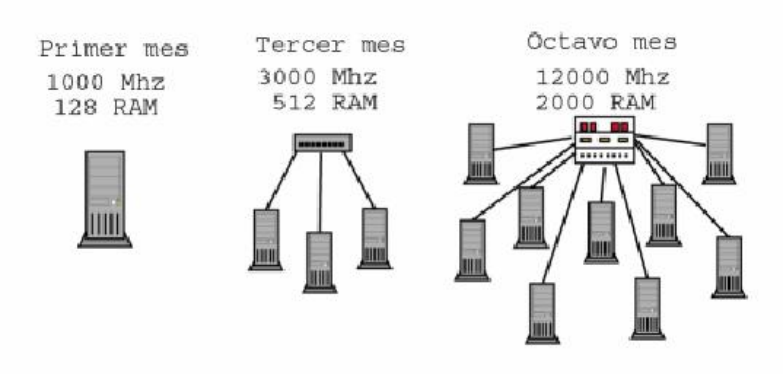


Figura 13:Balanceo de carga dinámico aplicado a sistemas distribuidos

Fuente: highscalability.com

El balanceo de carga dinámico, es utilizado en sistemas de escalabilidad puesto que la arquitectura maestro-esclavo es indispensable para el funcionamiento en sistemas distribuidos. En esta arquitectura las cargas pueden ser distribuidas entre los nodos esclavos teniendo claro conocimiento del porcentaje de ocupación de cada uno de estos.

2.1.2.1.2.1Tipos de balanceo de carga dinámico

El balanceo de carga dinámico dependiendo de donde y como se almacenan y repartas las tareas se puede clasificar como:

- Centralizado
- Descentralizado

Balanceo de carga dinámico centralizado: Se corresponde con la estructura típica de Maestro/Esclavo.

Balanceo de carga dinámico distribuido o descentralizado: Se utilizan varios maestros y cada uno controla a un grupo de esclavos.

2.1.2.1.2.1.1 Balanceo de carga dinámico centralizado

En el balanceo de carga dinámico centralizado las tareas son manejadas desde una ubicación centralizada, manejando una estructura Maestro-Esclavo. El nodo maestro almacena en su registro la colección completa de tareas a realizar, este envía las tareas a cada nodo esclavo; cuando estos terminen su procesamiento solicita al maestro una nueva. Esta técnica se denomina programación por demanda, y puede ser aplicable a tareas que tengan igual o diferente tamaño.

En algunos casos donde existan tareas con distintos tamaños, la mejor forma de distribución, se fundamenta en repartir primero las tareas que tengan mayor carga computacional. En dado caso, si la tarea con mayor carga se dejaría para el final, las tareas más pequeñas serían procesadas rápidamente y luego algunos nodos estarían esperando sin hacer nada, hasta que se termine de procesar la tarea de mayor complejidad.

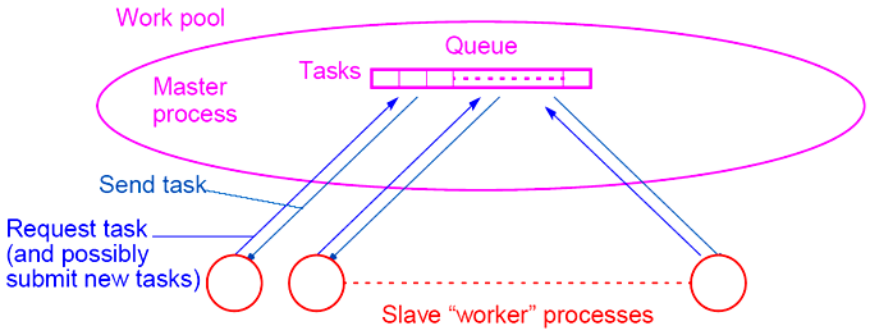


Figura 14: Sistema de balanceo de carga dinámico centralizado

Fuente: Dynamic Load balancing University of Berkeley.(17)

2.1.2.1.2.1.2 Balanceo de carga dinámico descentralizado

El balanceo de carga dinámico descentralizado, existen varios nodos maestros; los cuales interactúan con un determinado grupo de nodos esclavos.

Los nodos maestros manejan comunicación entre ellos, y los nodos esclavos pueden tener interacción dentro de un mismo grupo maestro. Esta característica aumenta la confiabilidad para tener una mejor distribución de carga uniforme, puesto que existe una mayor exactitud en los cálculos de tiempo medio de trabajo, en cada nodo esclavo.

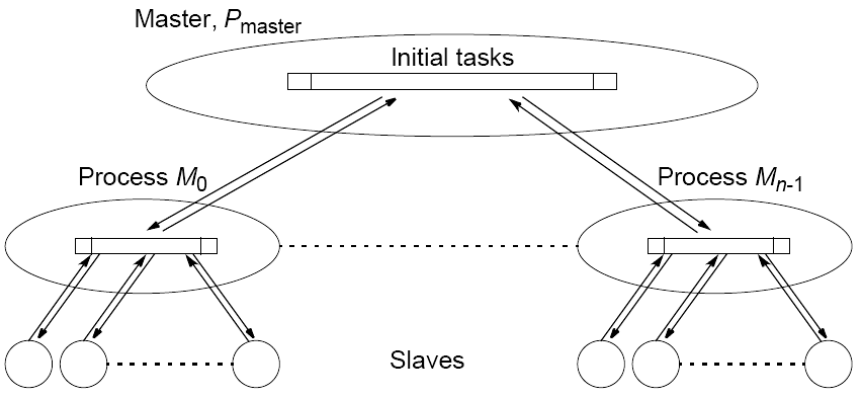


Figura 15: Sistema de balanceo de carga dinámico descentralizado

Fuente: Dynamic Load balancing University of Berkeley (17)

2.1.2.2 Métodos para el balanceo de carga

Dentro del campo computacional el balanceo de carga ha tenido un sin número de aplicación, dichas aplicaciones son utilizadas en diferentes áreas dentro del campo de las ciencias computacionales; tales como las redes informáticas, el procesamiento en paralelo, los sistemas distribuidos, etc.

Para cada uno de los campos mencionados anteriormente, existen diversos métodos de balanceo de carga, los cuales son aplicados de distintas maneras y utilizados dependiendo a su contexto. Dichos métodos se ven materializados en diferentes formas, para luego poder ser utilizados de una forma aplicativa.

Con esto se quiere dejar claro, que pueden existir un sin número de aplicaciones, materializadas de distintas formas, pero cada una de estas debe estar fundamentada en un tipo determinado de balanceo de carga.

A continuación, se describirán algunos de los métodos de balanceo más comunes:

Método	Descripción	Aplicación
Round Robin	Este es el método de balanceo de carga por defecto. El modo round Robin pasa cada nueva conexión al próximo server en línea, eventualmente distribuye las conexiones uniformemente a través de una lista (Array) de máquinas inicialmente creadas.	El modo Round Robin trabaja bien en muchas configuraciones, especialmente si existen en los nodos igual velocidad de procesamiento y memoria.
Ratio (miembro) Ratio (nodo)	Distribuye conexiones entre miembros de la reserva de nodos en una rotación estática de acuerdo con la relación de los pesos que se definan. Los pesos son definidos por el administrador.	Este es un método de balance de carga estático, básicamente distribuye las cargas dependiendo a los pesos los cuales son proporcionales a la capacidad de los servidores.
Ratio Dinámico (miembro) Ratio Dinámico (nodo)	El método ratio dinámico selecciona un servidor basado en varios aspectos, que analizan el rendimiento de los nodos en tiempo de ejecución. (Real-time). Este método es similar a los demás métodos de ratio excepto que la asignación de los pesos es una tarea del sistema y no del administrador.	Este método es usado específicamente para balancear tráfico dentro de: Equipos de Networking, Plataformas de servidores como Windows Management. Trabaja en el protocolo SNMP y servidores Windows 2000 server.
Fastest (node) Fastest (Aplicación)	El método Fastest selecciona un servidor basado en el menor número de conexiones activas.	El método Fastest es usado en entornos donde los nodos están separados lógicamente a través de redes.
Least Connections	Este método es relativamente simple, pasa a las nuevas conexiones a los nodos que tengan menor número de conexiones activas.	Este método funciona muy bien en entornos donde los servidores tienen similares capacidades.
Weighted	Es similar al método Least	Weighted Least Connections

Least Connections	Connections, basando la selección del servidor dependiendo a las capacidades del mismo. Los pesos son especificados por el sistema y tiene la capacidad de escoger diferentes algoritmos dependiendo a las especificaciones de los nodos.	trabaja muy bien en entornos donde los servidores tienen diferentes capacidades.
Observerd	En los métodos Observed, los nodos son clasificados basado en el número de conexiones. Este método realiza un seguimiento a las conexiones de capa 4, y crea una relación de equilibrio de carga.	Este método no se recomienda para grandes grupos de nodos.

2.1.2.3 Balanceo de carga en el modelo OSI

Cada protocolo de trasmisión/comunicación como por ejemplo TCP, UDP, HTTP, etc. Está clasificado en su respectiva capa; basándose en esa clasificación el balanceo de carga también se puede realizar por capas.

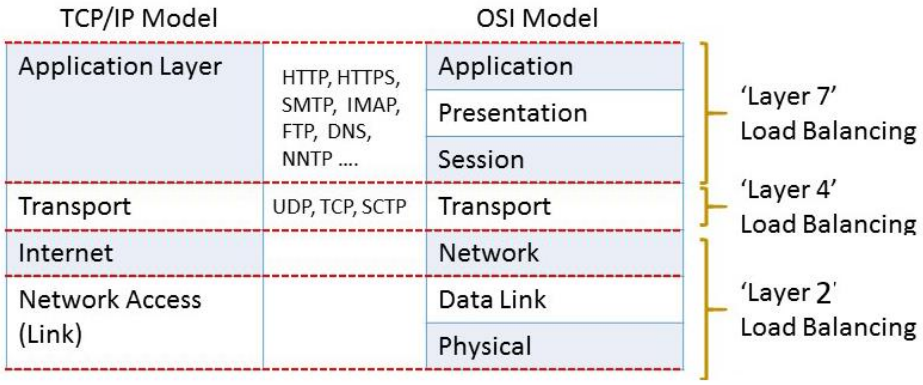


Figura 16: Balanceo de carga por capas del modelo OSI

Fuente: Dynamic Load Balancing on Web-Server Systems [8]

2.1.2.3.1 Balanceo de carga – capa 2

El balanceo de carga Capa 2, opera en las primeras 3 capas del Modelo OSI (física, enlace de datos y red). Aquí se determina como deben ser procesados los paquetes, y a donde debería ser enviado; basados en la información de la cabecera (Tramas) de los paquetes que se transmiten a nivel de estas capas. Las variables principales son las direcciones MAC.

La técnica llama “Agregación de enlaces” (Link aggregation) por sus siglas en inglés. Ofrece balanceo de carga a nivel físico, uniendo dos canales entre dos puntos; esto permite distribuir fácilmente las cargas del tráfico de red. Esta técnica está enmarcada en el balanceo de carga capa 2.

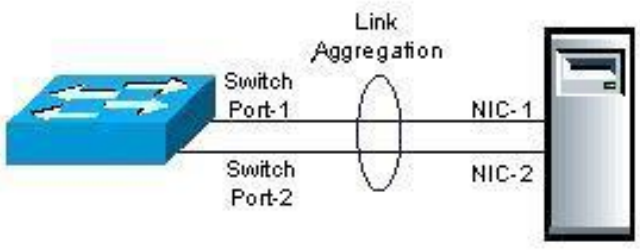


Figura 17: Agregación de enlaces entre un switch y un servidor

Fuente: Web performance tuning [2]

2.1.2.3.2 Balanceo de carga – capa 4

El balanceo de carga en la red (Networking), es uno de los más sofisticados a nivel de capa 4 del modelo OSI; puesto que está fundamentado en la interconexión lógica y no física de la red. Sus principales variables para realizar el balanceo de carga son las direcciones IP, puertos TCP, protocolos de enrutamiento, tablas de enrutamiento y segmentos de red. Estas variables son utilizadas para poder definir los mecanismos de envió y la mejor ruta para cada paquete. Los dispositivos de red como router, tiene a su disposición múltiples

rutas; la configuración manual o automática le permite tomar decisiones entre estas, de tal forma que pueden ejercer el balanceo de carga dentro de la red.

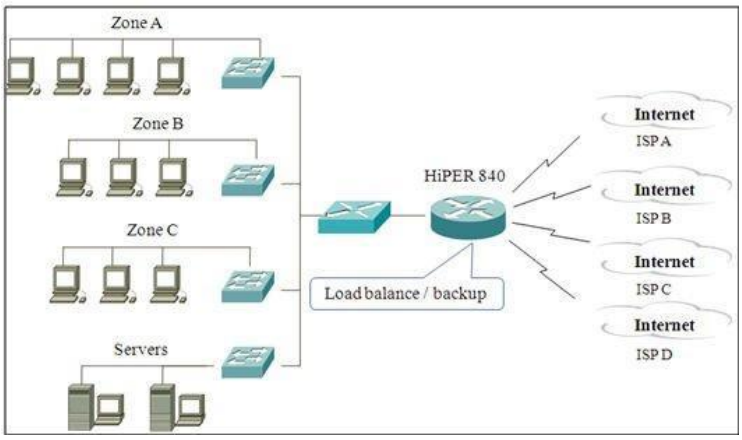


Figura 18: Balanceo de carga dentro de una red WAN

Fuente: Web performance tuning [2]

2.1.2.3.3 Balanceo de carga – capa 7

Los balanceadores de carga capa 7 manejan particularmente el tipo de tráfico basado en TCP, como lo es el contenido HTTP. Los balanceadores en esta capa antes de reenviar las peticiones web, primero revisan el contenido, así obtienen mayor información para luego enviar al servidor.

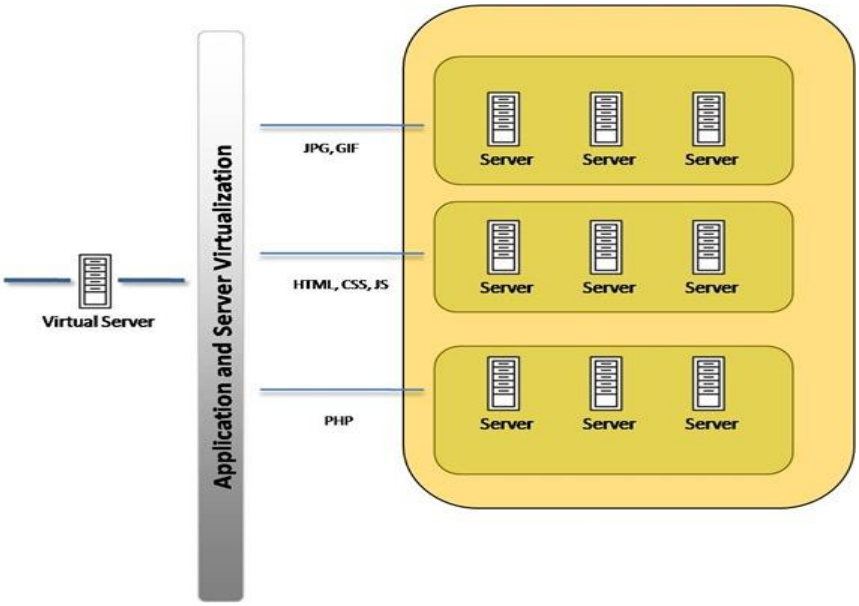


Figura 19: Balanceo de carga HTTP - Capa 7

Fuente: devcentral.f5.com

La decisión de reenvío es tomada basándose en información relevante, como por ejemplo (URL, Cookies, etc.), luego de la revisión se inicia una nueva conexión TCP con el servidor seleccionado, o se reutiliza una ya existente usando el método “HTTP Keepalives” y luego se escribe una petición al servidor.

El balanceo en esta capa está orientado al contenido web, por lo tanto, los balanceadores de carga capa 7 tiene total conocimiento del porcentaje de procesamiento de los nodos donde se alojan los contenidos web (web server).

2.2 Estado del arte

El incremento de la carga de peticiones HTTP en muchas ocasiones sobrepasa la capacidad de procesamiento de los servidores web; para soportar esta sobrecarga la implementación de sistemas web distribuidos, donde se pueda planificar la distribución de peticiones entre diferentes nodos ha sido una tarea que se ha realizado desde varios años atrás, para proveer escalabilidad y disponibilidad.

A continuación, se describirán diferentes técnicas y métodos de escalabilidad que se han adoptado a lo largo del tiempo, donde se resaltan las características fundamentales de las mismas las cuales servirán como marco referencial para esta investigación.

2.2.1 Técnicas y enfoques de escalabilidad web basadas en el balanceo de carga

2.2.1.1 Replicación de información (servidores espejos)

La replicación de contenido consiste en diferentes servidores web espejos, los cuales proveen una lista de URL independientes, las cuales tienen que ser

manualmente seleccionadas por los usuarios. Este enfoque tiene una serie de desventajas, dentro de las cuales están la no transparencia al usuario para seleccionar un servidor determinado y la ausencia del control para la distribución de peticiones.

La Figura 20 muestra un esquema gráfico del proceso de selección de servidor, el cual no es transparente al usuario.

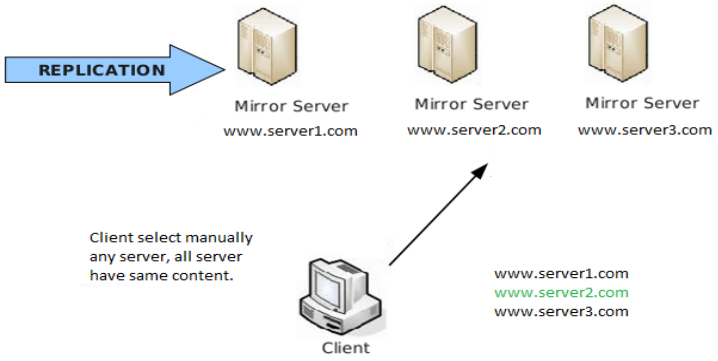


Figura 20: Sistema de servidores web espejos

Fuente: Distributed information system

El usuario manualmente tenía que seleccionar algún servidor, este proceso causaba ciertos inconvenientes puesto que el usuario debía de estar actualizando la lista de servidores. Por otra parte, cada nombre de dominio estaba ligado a una dirección IP diferente, lo cual hace que todas las peticiones estén dispersas y no centralizadas; de esta forma la distribución de las mismas era una tarea que presentaba mucha dificultad.

2.2.1.2 Enrutamiento de peticiones (routing request)

La estructura típica de una arquitectura web consta de varios clientes y un servidor, en este nuevo enfoque se añade un nuevo componente posicionado entre los clientes y el servidor.

Este nuevo componente denominado **enrutador de peticiones**, es el encargado de distribuir las peticiones HTTP entre los servidores web (nodos).

Con este nuevo funcionamiento nacen las arquitecturas web distribuidas donde se utilizan múltiples servidores web para servir una única aplicación o sitio web.

Según la arquitectura de sistemas web distribuidos se puede clasificar dependiendo de la entidad encarga de distribuir las peticiones hacia los múltiples servidores. De esta manera son identificadas 4 clases de métodos, los cuales son:

- Enfoque basado en el cliente
- Enfoque basado en DNS
- Enfoque basado en el despachador (A nivel de red)
- Enfoque basado en el servidor

2.2.1.2.1 Enfoque basado en el cliente

Técnicamente este enfoque se basa en los navegadores web, los cuales almacenan las direcciones URL de cada nodo (servidor web) y luego utilizan una serie de protocolos para seleccionar el nodo más apropiado.

2.2.1.2.2 Enfoque basado en DNS

Este enfoque es implementado del lado de los servidores, aquí los usuarios o clientes no intervienen para seleccionar el servidor de destino. Esta arquitectura de transparencia al usuario se obtiene a través de una simple interfaz virtual la cual se muestra al mundo exterior a nivel de la URL. La responsabilidad de separar las peticiones entre los distintos servidores es tomada por un clúster DNS (Ver figura 21). Al momento de realizar el proceso de traducción del nombre simbólico a la dirección IP, el clúster DNS puede seleccionar cualquier nodo servidor dependiendo a ciertas políticas establecidas dentro del sistema.

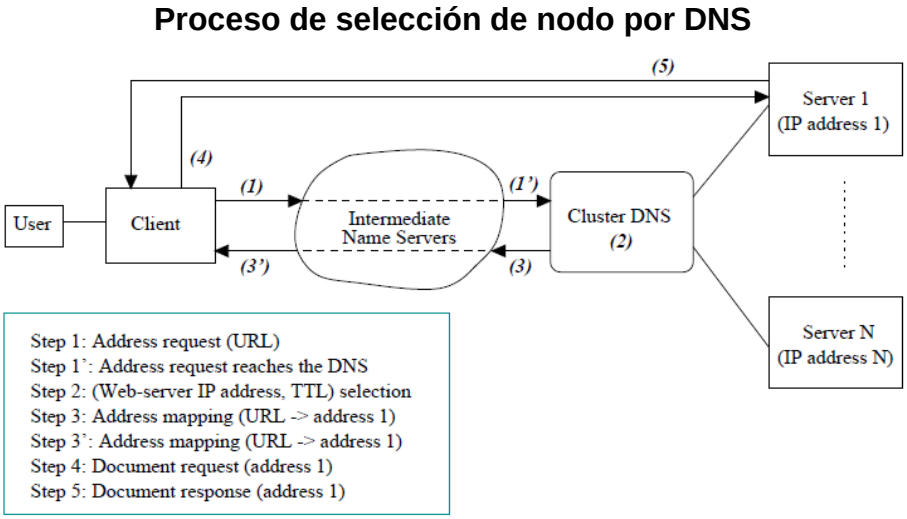


Figura 21:Enfoque basado en DNS

Fuente: Load balancing in distributed systems [9]

2.2.1.2.3 Enfoque basado en el despachador (a nivel de red)

El anterior enfoque está fundamentado en la URL, donde existen diferentes IP que apuntan a una misma URL. Este nuevo enfoque basado en despachador a nivel de red no se centra en la URL si no en la IP, donde se da paso a las direcciones IP virtuales. Aquí se utiliza una dirección IP virtual, la cual es traducida a otras direccione IP físicas, las cuales corresponde cada una a un nodo servidor.

2.2.1.2.4 Enfoque basado en el servidor

Esta técnica implementa el re-direccionamiento dentro del mismo servidor web. Cuando una petición llega a uno de los servidores nodo, este puede enviarla hacia otro servidor nodo que esté con menor carga de procesamiento. Este enfoque es

utilizado luego que los servidores DNS realicen su proceso, lo cual indica que esta técnica es una extensión del enfoque basado en DNS.

2.2.2 Actualidad de la industria

Se presentarán las corporaciones más representativas dentro de la industria, junto con sus principales productos para el balanceo de carga dentro de arquitecturas web distribuidas.

2.2.2.1 F5 networks

Compañía dedicada a la optimización y rendimiento de servidores, la cual provee una serie de dispositivos llamados BIG-IP (Appliance). Los cuales funcionan como **switch web,** estos trabajan en la capa 7 del modelo OSI como balanceador de carga para servidores web.

2.2.2.2 Fundación de software apache

Esta cooperación la cual ha desarrollo el servidor web de mayor uso en la industria, ha elaborado una serie de módulos (Apache módulo: mod_proxy) para el balanceo de carga de peticiones HTTP los cuales funcionan junto con su servidor web nativo (Apache web server).

2.2.2.3 Cisco system

Compañía dedicada a la fabricación de equipos activos de red, ha desarrollado un nuevo producto denominado Cisco ACE, el cual funciona como motor de control de aplicaciones a nivel web, este trabaja sobre la capa 7 del modelo OSI.

2.2.2.4 Microsoft

Compañía de tecnología, fabricante de sistemas operativos y demás. Ha desarrollado una funcionalidad para el procesamiento de servidores web en granja. Esta solución hace que una serie de servidores IIS sirvan una única aplicación.

2.2.2.5 Kemp technologies

Esta compañía produce únicamente dispositivos para el balanceo de carga, en los últimos años ha sido una referente dentro del mercado. Sus productos principales son los dispositivos llamados LM-Series, estos están basados en su propia plataforma de software llamados OVM que se ejecuta como un LoadMasters Virtual (VLM), y también pueden ser implementados en un artefacto de hardware (Switch).

2.2.2.6 Oracle corporación

Esta compañía dedicada a los motores de base de datos, ha diseñado bajo la firma de SUN microsystem, un Plug-in para su servidor de aplicaciones web llamado GlassFish. Este plug-in es llamado **Glassfish Loadbalancer**, el cual funciona junto con el servidor Glassfish.

2.2.3 Balanceadores de carga – OpenSource

2.2.3.1 Pound

Última versión: V 2.7, 2010/Jun/01. Estable
Es un proxy inverso balanceador de carga HTTP y front-end para servidores web. Es un pequeño software el cual no soporta altos porcentajes de peticiones HTTP.

2.2.3.2 Haproxy

Última versión: V 1.6, 2015/Oct/20. Estable
Es un servidor de balanceo de carga TCP/HTTP diseñado especialmente para alta disponibilidad, es una solución para manejar el alto tráfico HTTP. Está escrito en C y lanzado en el año 2000. Su mayor funcionalidad es el balanceo de carga TCP y HTTP.

2.2.3.3 Linux Network Load Balancing

Última versión: V 0.1.3, 2008/Dic/08. Estable
Es un proyecto enfocado a realizar balanceo de carga a nivel de red descentralizado usando en clúster entre host Linux. No es usado para balanceo de carga a nivel de aplicaciones web. En su sitio web describen su funcionamiento de la siguiente forma: LNLB funciona usando una misma IP compartida entre todos los nodos (en una interface virtual)

2.2.3.4 Linux Virtual Server (LVS)

Última versión: V 1.2.1, 2004/Dic/24. Estable
LINUX VIRTUAL SERVER es un servidor construido para la alta disponibilidad y escalabilidad, el cual trabaja incorporado en un clúster de servidor, corriendo un balanceador de carga dentro del sistema operativo Linux. Esta solución es utilizada para mantener la disponibilidad de los servicios de red, tales como: MAIL, FTP, MEDIA, VOIP, ETC.

2.2.3.5 Nginx

Última versión: V 1.8, 2015/Abril/21. Estable

Es un servidor proxy inverso, ligero y de alto rendimiento. Utilizado como servidor de balanceo de carga HTTP/TCP. Su diseño modular también hace que funcione como servidor proxy caching; también tiene soporte multi-protocolo (HTTP, TCP, SMTP, IMAP etc). Tiene soporte para comunicación con aplicaciones web bajo los protocolos: FastCGI, SCGI, uWSGI, etc.

Es programado en c y fue lanzado en el año 2004 y actualmente está habilitado para soportar más de 10.000 conexiones simultáneas.

2.2.4 Soluciones más utilizadas y relevantes dentro de la industria

La utilización de una solución de escalabilidad para una aplicación web está condicionada en muchos escenarios por la tecnología con la cual esta fue desarrollada.

Para balancear la carga de una aplicación web la cual fue desarrollada utilizando tecnologías Microsoft; se necesita por lo tanto utilizar herramientas y técnicas de balanceo de carga desarrolladas por la firma de Microsoft. Este comportamiento prevalece dentro de la industria en casi todas las compañías que son propietarias de herramientas y tecnologías de desarrollo.

Por otra parte, existe un amplio abanico de herramientas y tecnologías para el desarrollo, las cuales son de índole OpenSource, estas herramientas y tecnologías tiene una característica muy favorable, puesto que utilizan los lenguajes de programación de mayor uso en la web, tales como: PHP, Python, JavaScript, Ruby, etc.

Uno de los alcances de este proyecto es la utilización de soluciones OpenSource, por lo tanto, a continuación, se describirán las dos soluciones OpenSource de mayor uso y rendimiento dentro de la industria.

2.2.4.1 Haproxy

Su nombre se refiere a las palabras en inglés High Availability Proxy, es un software OpenSource usado para mejorar el entorno de rendimiento de un servidor.

HAProxy es usado como un software Balanceador de Carga para los protocolos TCP y HTTP, el cual puede ejecutarse en Linux, Solaris, y FreeBSD. Aplica su capacidad de proxy inverso para distribuir el flujo de carga entre múltiples servidores de backend.

Su creador **Willy Tarreau** lo describe como: HAPorxy es una herramienta de Balanceo de carga TCP/HTTP y proxy inverso particularmente adecuada para construir arquitecturas de alta disponibilidad

Desde su creación en el año 2000 ha tenido una evolución muy favorable y se ha implementado en grandes compañías, tales como: GitHub, Imgur, Instagram, y Twitter.

Principales características

Actualmente la versión estable de HaProxy es la versión 1.6, la cual soporta las siguientes funcionalidades:

- Algoritmos de balanceo de carga: Soporta un conjunto amplio de diversos algoritmos de balanceo de carga.
- ACL (Acces Control Lists): Contiene listas de control de acceso, para controlar el flujo de contenido, por medio de reglas establecidas.
- Soporte SSL nativo: Para manejo seguro de contenido.
- Protección DDoS: Brinda protección contra ataques de denegación de servicio, a nivel de servidor proxy.

- Soporte IPv6: Soporta el nuevo estándar de direccionamiento IP
- Compresión HTTP/1.1: Para minimizar el consumo de ancho de banda.

HAProxy dentro de una plataforma web

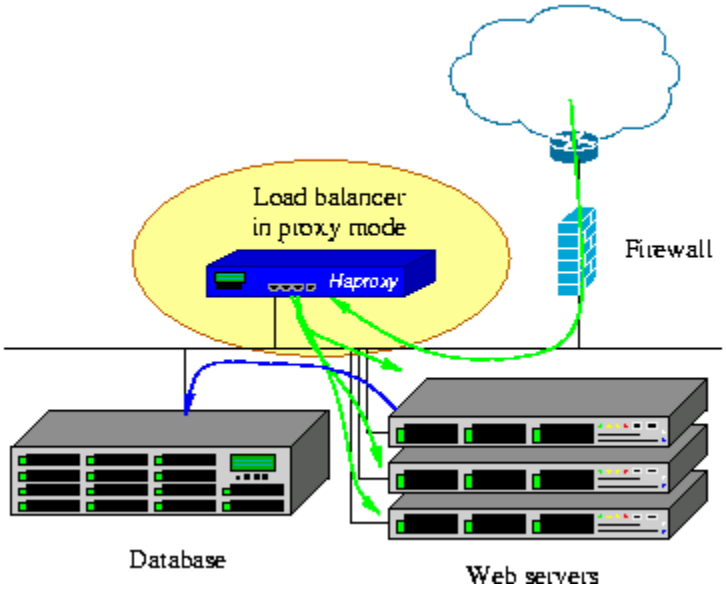


Figura 22: HAProxy utilizado como Balanceador de carga en modo proxy

Fuente: Haproxy.org

HAProxy se presenta como una de las soluciones más utilizadas ya que se puede integrar dentro una plataforma web OpenSource, sin necesidad de aplicar grandes cambios dentro de la arquitectura física y lógica, este puede adaptarse a proyectos de gran envergadura, sin afectar el funcionamiento del mismo.

2.2.4.2 Nginx

NGINX pronunciado “Ngine X” es un software OpenSource diseñado específicamente para la alta disponibilidad y alto rendimiento de una plataforma web. Este fue desarrollado por el ingeniero de software ruso **Igor Sysoev** y fue lanzado en el año 2004.

NGINX está diseñado bajo un enfoque asíncrono orientado a eventos, el cual optimiza el manejo de peticiones HTTP, debido a esto NGINX es capaz de manejar grandes flujos de carga y es una solución ideal para aplicaciones y sitios web con alto tráfico de contenido.

La plataforma de series y películas **Netflix** ha hecho de NGINX su pieza fundamental para manejar su alto tráfico de contenido, el ingeniero de desarrollo **Gleb Smirnoff** en el blog de Nginx.com muestra toda la información referente a la implementación de NGINX sobre la plataforma Netflix.

NGINX en los últimos años se ha consolidado como una de las herramientas de mayor uso para la escalabilidad y optimización de plataformas web. Según el reconocido sitio de estadísticas y encuestas tecnológicas w3techs.com, NGINX ha aumentado su porcentaje de uso en los últimos años dentro del mercado.

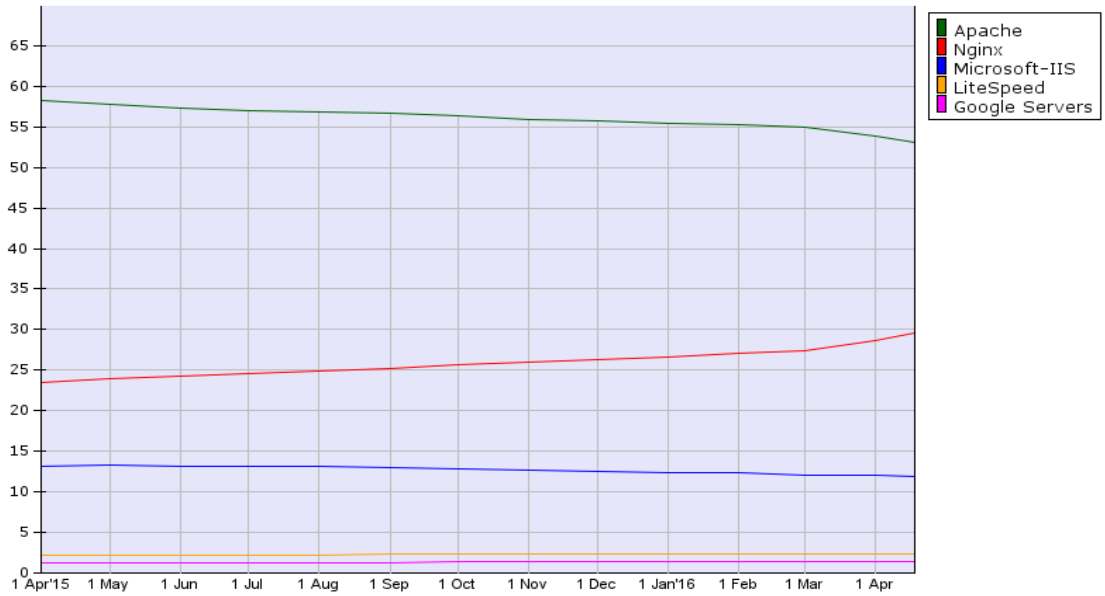


Figura 23: Uso de servidores web, 18 abril 2016.

Fuente: W3Techs.com

Principales Características

La última versión estable de NGINX es la versión 1.9

- Balanceador de carga TCP/UDP: Utiliza un amplio conjunto de algoritmos de balanceo para distribuir flujo de carga entre múltiples servidores.
- Terminación SSL para TCP: Optimiza los procesos de encriptación entre el cliente y los servidores.
- ACL: Filtra el acceso basado en la dirección IP del cliente.
- Web server: Nginx tiene la capacidad y la flexibilidad para ser un servidor web de alto rendimiento.
- Mail Proxy server: Brinda soporte para protocolos POP3, IMAP, SMTP, para distribuir flujo de carga entre servidores de correo.

3 CAPITULO III - ALCANCE Y METODOLOGÍA

3.1 Alcance y delimitación de la investigación

El presente estudio pretende determinar de una forma cuantitativa que técnicas de escalabilidad web permiten optimizar el rendimiento de una plataforma web de una forma efectiva y a bajo costo. La efectividad hace referencia a la tasa de atenciones por segundo que puede soportar una plataforma web. El bajo costo implica aspectos económicos, computacionales, de mantenimiento e implementación.

I. La investigación abarca las tecnologías OpenSource más relevantes dentro de la industria, principalmente dentro de los campos de servidores web, balanceadores de carga HTTP y bases de datos.
II. La aplicación o plataforma web que se utilizará en esta investigación como elemento de prototipo será un CMS Drupal 7; esta plataforma es utilizada para gestionar sitios web con grandes volúmenes de contenido.
III. El entorno de pruebas de rendimiento se realizará únicamente sobre máquinas virtuales, las cuales están alojadas en dos máquinas físicas de alto rendimiento; ubicadas en el laboratorio de software de la Universidad Tecnológica de Bolívar.

3.1.1 Tipo de alcance

El tipo de alcance de esta investigación se identifica como correlacional, en donde se pretende analizar el comportamiento de diferentes entornos de rendimiento. Se evaluarán múltiples variables dentro de cada entorno y su relación entre ellas.

Las principales variables a analizar en los diferentes entornos de rendimiento son:

Variables para analizar		
Rendimiento	Hardware	Software
Tiempos de respuesta (seg.)	Cantidad CPU	Servidores web
Tasa de request (req/seg)	Cantidad RAM	Base de datos
Conexiones/Usuarios simultáneos		Balanceadores HTTP
Transferencia de datos (KB/seg)		

Tabla 3: Principales variables de análisis

Los resultados serán comparados y analizados teniendo en cuenta cada entorno de configuración. Los datos de prueba serán generados a través de una herramienta de Testing para carga HTTP, llamado Tsung.

Las variables de rendimientos son las variables dependientes las cuales varían teniendo en cuenta la infraestructura de hardware y configuración de software.

3.2 Metodología

3.2.1 Descripción metodológica

La metodología para desarrollar este proyecto de grado consta de las siguientes etapas:

1. Planificación
2. Investigación preliminar
3. Diseño y modelado
4. Implementación y pruebas
5. Documentación

Dicha metodología está basada en el modelo metodológico **Ciclo de Vida de Desarrollo de Sistemas** (Systems Development Life Cycle). El cual es ampliamente utilizado en la ingeniería de software.

PLANIFICACIÓN

Inicialmente se realizará un análisis acerca del problema de investigación, para identificar una serie de soluciones posibles, las cuales sean viables y puedan ser ejecutadas en la etapa de implementación.

Luego se determinarán una serie objetivos dentro de la investigación, los cuales den soporte a la problemática o mejoramiento del sistema. Como medida principal se evaluarán aspectos económicos, operativos y de implementación, en cuanto a las alternativas de solución que se visualicen en esta etapa.

INVESTIGACIÓN PRELIMINAR

Se realizará una búsqueda conceptual, donde se tomará información de diferentes fuentes acerca de la escalabilidad y alto rendimiento en servidores web, dichas fuentes serán: Libros, investigaciones anteriores, artículos científicos y

principalmente profesionales y empresas con experiencia en el campo en cuestión. Se tomará específicamente trabajos de grado alojados en la biblioteca de la Universidad Tecnológica de Bolívar, aquellos que sean afines con la línea investigativa. También se analizarán datos estadísticos acerca del uso de la escalabilidad web y su evolución; de igual forma se determinarán las soluciones actuales para el análisis de las mismas.

Se analizará información acerca de empresas que manejan alto tráfico web las cuales necesariamente debe aplicar técnicas de escalabilidad y alto rendimiento, esto con el fin de tener un amplio conocimiento del problema en estudio.

DISEÑO Y MODELADO

El presente proyecto de investigación se centra en un sistema de cómputo distribuido, en donde intervienen múltiples componentes, tales como:

- Sistemas operativos
- Servidores web
- Balanceadores de carga
- Máquinas virtuales
- etc.

En esta etapa de diseño y modelado se determinará los diferentes entornos de funcionamiento, en donde cada componente tendrá un rol específico y una configuración determinada en cuanto a sus aspectos técnicos. Para cada componente se determinará la ubicación física y lógica dentro del sistema, para establecer el flujo del procesamiento dentro del mismo.

IMPLEMENTACIÓN Y PRUEBAS

La implementación de este trabajo de investigación se llevará a cabo dentro de los laboratorios de la Universidad Tecnológica de Bolívar. Para esto se realizará un análisis de los componentes computacional que se requieran dentro del laboratorio

de la universidad. La implementación consiste en aplicar técnicas de escalabilidad y optimización a una aplicación o sitio web.

Para las pruebas del sistema se desarrollarán diversos entornos de prueba, con el fin de realizar análisis comparativos entre los mismos, y así determinar el entorno de mejor rendimiento.

DOCUMENTACIÓN

Se anexarán archivos con las configuraciones de cada uno de los componentes técnicos junto con su rol dentro del sistema, tanto de software como de hardware. También se expondrán y describirán cada uno de los esquemas gráficos de la solución en general, con el fin de apoyar futuros cambios y etapas de mantenimiento o rediseño.

3.2.2 Definición metodológica

A continuación, se describirá como se llevará a cabo cada uno de los objetivos de este trabajo de investigación, con el fin de exponer claramente la metodología de trabajo.

Objetivo 1

Implementar múltiples servidores web los cuales en conjunto servirán una misma aplicación o sitio web, utilizando la técnica de servidor proxy inverso, donde se aplicarán métodos para el balanceo de carga HTTP.

Para manejar múltiples servidores web que sirvan una misma aplicación o sitio web, se utilizará en este caso específico 3 máquinas virtuales las cuales compartirán los mismos archivos de código fuente, a través de un sistema de sincronización de archivos. Cada máquina virtual debe de contener igual sistema operativo, igual versión de servidor web y debe tener configurado el sistema SSH para permitir la transferencia segura de archivos entre cada máquina virtual.

Para que los múltiples servidores web funcionen como una misma plataforma web, se utilizará un servidor de balanceo HTTP, el cual se encargará de distribuir las peticiones de los clientes entre las 3 máquinas virtuales; mostrando al cliente una única interfaz de comunicación.

Objetivo 2

Implementar un clúster de base de datos en modo master-master, para la replicación de datos entre los múltiples nodos servidores.

Para implementar un clúster de base de datos en este caso específico se utilizarán 3 máquinas virtuales, las cuales tendrán el mismo sistema operativo y el mismo motor de base de datos, junto con un aplicativo adicional; el cual estará encargado de gestionar la replicación de cada una de las bases de datos creadas dentro de cualquiera máquina virtual. La replicación dentro de este clúster de base de datos operará en modo master-master, lo cual indica que cualquier nodo (máquina) puede leer y escribir datos dentro del clúster.

Objetivo 3

Elaborar un entorno de pruebas de rendimiento, entre una plataforma web con balanceo de carga HTTP y uno sin él, para analizar los distintos entornos de rendimiento.

Para la implementación de un entorno de pruebas de rendimiento, se utilizará una herramienta de Testing para inducir carga HTTP, la cual crear múltiples clientes HTTP los cuales generan una carga de peticiones simultáneas. Esta herramienta permite configurar el número de clientes simultáneos y el número de peticiones por cliente.

El entorno de pruebas será elaborado para poder determinar qué escenario de configuración, muestra el mayor rendimiento en cuanto a tasa de atenciones y tiempos de respuesta del sistema.

4 CAPÍTULO IV: IMPLEMENTACIÓN - SISTEMA DE OPTIMIZACIÓN Y ESCALABILIDAD WEB

La metodología utilizada en este proyecto, se fundamenta en la elaboración de un sistema de validación de hipótesis, como parte del proceso de investigación. Para realizar la implementación del sistema se debe tomar como referencia la hipótesis de investigación planteada inicialmente, para luego realizar conjeturas con respecto a cada uno de los aspectos principales del proyecto.

Durante el proceso de investigación se realizaron diversas pruebas con varios prototipos a escalas menores, en donde se validaron conceptos fundamentales anteriormente expuestos en el marco teórico. La utilización de estos prototipos, permitió identificar y aclarar todos los aspectos teóricos y técnicos de la investigación; esto fortaleció los alcances y objetivos del proyecto, dando lineamientos específicos para la implementación final del sistema.

La implementación del sistema de optimización y escalabilidad para aplicaciones web consta de tres componentes fundamentales: **balanceador de carga HTTP**, **clúster de bases de datos** y **entorno múltiple de servidores**. El desarrollo de la implementación tendrá como lineamiento estos tres componentes, cada uno de estos contiene aspectos técnicos internos, los cuales son independientes de los demás componentes; aunque también existen aspectos técnicos transversales los cuales se interrelacionan entre dos o más componentes.

Las herramientas y tecnologías utilizadas para la implementación son de índole OpenSource, en el estado del arte se expusieron las principales funcionalidades de cada uno de ellas. Como sistema operativo principal se optó por la distribución Linux Debian 8 con arquitectura x64, la cual ofrece un excelente estabilidad y soporte total para la instalación de diversas herramientas tales como: servidores web, balanceadores de carga HTTP, generadores de carga HTTP, motor de base de datos, clúster de base de datos, soporte SSH, sincronizador de sistema de archivos, etc.

4.1 Generalidades de la implementación

La solución a implementar busca utilizar un conjunto de herramientas y técnicas de escalabilidad y optimización, con el fin de mejorar el rendimiento y soportar la alta concurrencia de peticiones dentro de una aplicación o sitio web.

4.1.1 Topología de la implementación

La *figura 24* muestra un esquema gráfico de los componentes físicos y lógicos de la solución a implementar.

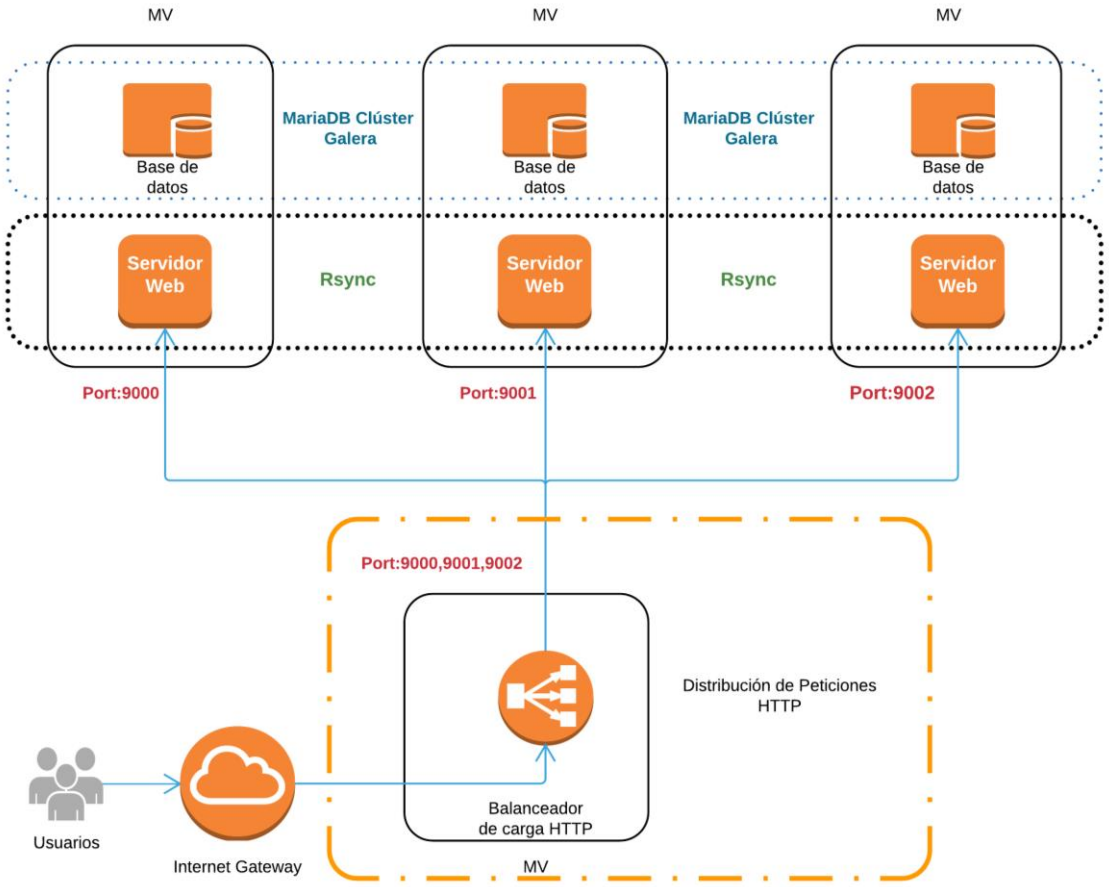


Figura 24:Topología sistema de optimización y escalabilidad para aplicaciones web
Fuente: Propia

La anterior figura (Figura 24) muestra una descripción gráfica de la solución en general, está busca ilustrar a grandes rasgos los principales componentes que

intervienen en la implementación. En la parte superior de la figura se expone el componente de clúster de bases de datos y el componente de entorno múltiple de servidores, y en la parte inferior se expone el componente balanceador de carga HTTP. De esta forma se puede tener una mejor noción de la implementación y clarifica la estructura física y lógica de la solución a desarrollar.

4.1.2 Arquitectura física de la implementación

La arquitectura computacional para el despliegue del sistema se llevará a cabo bajo una máquina física de alto rendimiento perteneciente al clúster HPCLAB de la Universidad Tecnológica de Bolívar. Este clúster permite la creación de máquinas virtuales mediante la herramienta de virtualización **Virtual Machine Manager**. La Tabla 4 describe en detalle cada una de las máquinas creadas para la implementación.

Lista de Máquinas virtuales

#	Nombre del HOST	CPU	RAM	SO	Dirección IP
1	Balanceador de Carga HTTP	2	2 GB	Debian 8 - x64	172.16.9.49
2	Servidor Web - 01 (FPM-PHP)	4	4 GB	Debian 8 - x64	172.16.9.46
3	Servidor Web - 02 (FPM-PHP)	4	4 GB	Debian 8 - x64	172.16.9.47
4	Servidor Web - 03 (FPM-PHP)	4	4 GB	Debian 8 - x64	172.16.9.48
5	Servidor Web - 01 (HHVM)	4	4 GB	Debian 8 - x64	172.16.9.50
6	Servidor Web - 02 (HHVM	4	4 GB	Debian 8 - x64	172.16.9.51
7	Servidor Web - 03 (HHVM	4	4 GB	Debian 8 - x64	172.16.9.52
8	Generador de carga HTTP	2	2 GB	Debian 8 - x64	172.1.9.44

Tabla 4: Máquinas virtuales para la implementación

Debido a la virtualización de todos los componentes físicos no se utilizó equipos de red, tales como routers, switches, tarjetas de red, etc. Cabe mencionar que una implementación para una aplicación web en producción debe contener una fuerte infraestructura de red donde se contemplen aspectos técnicos como: ancho de banda, canal de internet, seguridad y disponibilidad de los servicios de red.

4.1.2.1 Descripción de máquinas virtuales

Máquina virtual # 1

Esta máquina se utiliza únicamente para la implementación del balanceador de carga HTTP, dentro de ella se realizaron configuraciones pertinentes a la distribución de peticiones y utilización de métodos de balanceo de carga.

Máquinas virtuales # 2,3, y 4

Cada una de estas máquinas contiene dos elementos importantes, los cuales son: un servidor web y un nodo del clúster de bases de datos. Una característica importante de estas máquinas es el servidor web, puesto que este utiliza como procesador de código PHP la implementación FastCGI denominada PHP-FPM (PHP- FastCGI Process Manager, por sus siglas en inglés).

Máquinas virtuales # 5,6, y 7

Al igual que las máquinas virtuales 2,3 y 4; estas máquinas cada una contiene un servidor web y un nodo del clúster de bases de datos. La diferencia radica en el cambio del procesador de código PHP, en este caso se utilizará el procesador de código HHVM (Máquina virtual utilizada para ejecutar código HACK y PHP, enfocada para mejorar el rendimiento).

Máquina virtual # 8

Esta máquina virtual se utilizará para generar tráfico HTTP de prueba, mediante la instalación y configuración de un software de testing llamado Tsung. Aquí se

especifica el número de clientes simultáneos y el número de peticiones por cliente, para generar la carga de prueba.

De acuerdo a la estructura planteada, a continuación, se describirán en detalle cada uno de los componentes de la implementación; iniciando por el balanceador de carga HTTP, luego el componente de múltiple de servidores y finalmente el componente de clúster de bases de datos. Por último, se expondrá el funcionamiento en general de todos los componentes y la relación entre cada uno de ellos.

4.2 Implementación del balanceadora de carga HTTP

El componente balanceador de carga se fundamenta en distribuir un gran conjunto de peticiones HTTP entrantes, entre diferentes servidores de back-end. Para el despliegue de este componente se utilizó el servidor proxy inverso especializado en balanceo de carga llamado Nginx.

4.2.1 Instalación del balanceador de carga Nginx

Para instalar el balanceador de carga nginx en el sistema operativo Debian 8 x64, inicialmente se tuvieron en cuenta dos aspectos fundamentales, el sistema de repositorios (Manejo de fuentes de instalación, soruce.list) y el sistema gestor de privilegios del sistema (Privilegios *sudo*).

Debian 8 a diferencia de otras distribuciones Linux, no viene pre configurado con la herramienta <<*sudo*>>, la cual permite gestionar los privilegios del sistema. Para instalar sudo, se ejecutó el siguiente comando:

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

El sistema de repositorios se configuró mediante el archivo sources.list, a diferencia de otros sistemas, Debian 8 contiene por defecto los paquetes de

instalación de Nginx por lo cual no fue necesario agregar nuevas fuentes de instalación.

Para instalar nginx se ejecutó el siguiente comando:

```
$ sudo apt-get install nginx
```

Con este comando el sistema descarga los paquetes y luego instala el servidor nginx, la versión instalada fue 1.6.0.

4.2.2 Configuración del balanceador de carga nginx

Los archivos de configuración de nginx, luego de la instalación son alojados en el directorio /etc/nginx/.

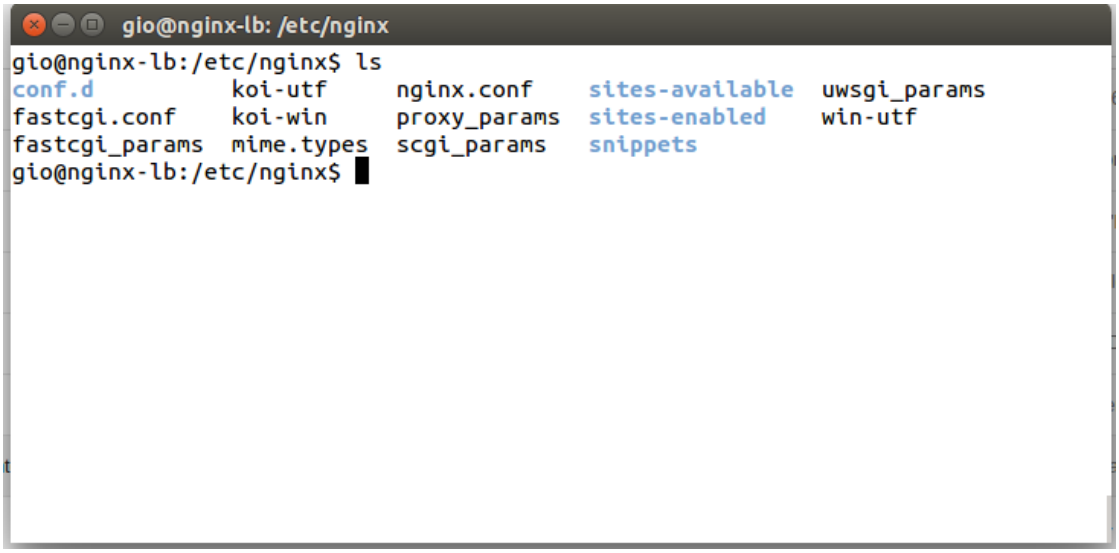


Figura 25: Archivos de configuración servidor nginx

Para la configuración de nginx en modo balanceador de carga, se debe crear un ***host virtual***, para esto se creó un archivo de configuración dentro del directorio */sites-available*. En esta implementación se aplicará balanceo de carga a una aplicación CMS Drupal, por lo tanto, el archivo de configuración tendrá como nombre “drupal” /sites-available/drupal.

4.2.2.1 - Código del archivo de configuración nginx

Ruta: /etc/nginx/sites-available/drupal

****

```
http {

    upstream appDrupal {
        server 172.16.9.50:9000 max_fails=3 fail_timeout=30s;
        server 172.16.9.51:9001 max_fails=3 fail_timeout=30s;
        server 172.16.9.52:9002 max_fails=3 fail_timeout=30s;
    }

    server {

        listen 80;
        access_log /var/log/nginx/drupal-access.log;
        error_log  /var/log/nginx/drupal-error.log error;


      location / {

            include proxy_params;
            proxy_set_header X-Forwarded-Port $server_port;


            proxy_pass http://appDrupal;
            proxy_redirect off;


            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
       }

    }

}
```

****

Para la configuración de este archivo, se tomó en cuenta el flujo de trabajo y el ciclo de procesamiento de las peticiones entrantes al sistema. El balanceador de carga es quien recibe inicialmente las peticiones provenientes de los clientes que acceden desde internet y luego las envía hacia los servidores de back-end, los cuales procesan las solicitudes y las regresan al balanceador, el cual envía la respuesta finalmente a los clientes.

El archivo de configuración /etc/nginx/sites-available/drupal maneja una estructura por bloques y directivas; las directivas son parámetros de configuración los cuales se especifican con una línea de código y los bloques son aquellas porciones de código marcadas con llaves.

La estructura general del archivo de configuración “drupal” se denota de la siguiente manera:

```
http {

       upstream {
      }
      server {
      }
}
```

Los bloques principales en esta estructura son **http, upstream,** y **server**. El bloque http es el bloque general de la configuración, y es colocado por defecto dentro del archivo de configuración.

El bloque *upstream* especifica el conjunto de servidores de back-end que procesaran las peticiones, y el bloque *server* configura los parámetros necesarios para poder recibir las peticiones HTTP entrantes provenientes de internet.

4.2.2.1 Bloque de configuración upstream

```
upstream appDrupal {
        server 172.16.9.50:9000 max_fails=3 fail_timeout=30s;
        server 172.16.9.51:9001 max_fails=3 fail_timeout=30s;
        server 172.16.9.52:9002 max_fails=3 fail_timeout=30s;
}
```

En el bloque upstream se configuró un nombre de identificación, el cual fue “appDrupal”; este nombre es utilizado para que el proxy inverso de nginx pueda conocer hacia donde enviará las peticiones. Dentro del bloque upstream se definieron los servidores back-end, los cuales se comunican con el balanceador de carga a través de una conexión TCP, en este caso se utilizaron los puertos 9000,9001, y 9002. Las direcciones IP de los servidores back-end y el balanceador de carga se establecieron dependiendo a la topología de red diseñada, en este caso el balanceador de carga utiliza la dirección IP 172.16.9.49 y los servidores back-end, las direcciones 172.16.9.50, 172.16.9.51, y 172.16.9.51.

Para conocer el estado de los servidores de back-end, se utilizaron las directivas max_fails y fail_timeout, la primera indica el número máximo de veces que un servidor de back-end puede fallar antes que el balanceador de carga pare de enviarle peticiones; la segunda establece la cantidad de tiempo que el balanceador de carga espera para volver a enviar tráfico hacia un servidor que ha sido marcado como fallido.

4.2.2.2 Bloque de configuración server

```
server {

        listen 80;
        access_log /var/log/nginx/drupal-access.log;
        error_log  /var/log/nginx/drupal-error.log error;

           location / {
```

```
            include proxy_params;
            proxy_set_header X-Forwarded-Port $server_port;

            proxy_pass http://appDrupal;
            proxy_redirect off;

            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
          }
}
```

En el bloque server se especificó el puerto TCP 80 para recibir las peticiones HTTP proveniente de internet y la ruta de almacenamiento para registrar el log del sistema mediante las directivas access_log y error_log. También se configuró el ***bloque location*** el cual contiene las directivas necesarias para procesar las cabeceras HTTP y realizar el proceso de "proxying" mediante la directiva "proxy_pass"; la definición de esta directiva activa el re-direccionamiento de peticiones hacia el bloque upstream. La siguiente figura x ilustra de una forma gráfica la configuración de los bloques server y upstream.

4.2.2.3 Esquema gráfico - Bloques de configuración server y upstream

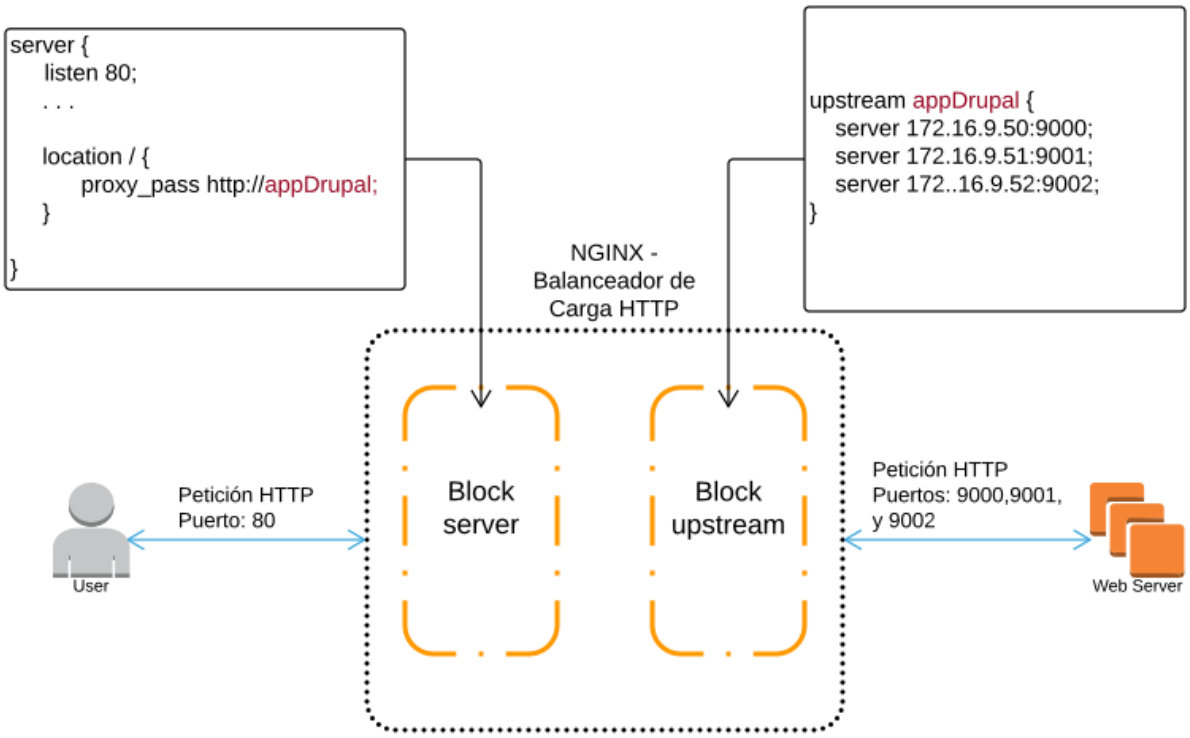


Figura 26: Configuración bloques server y upstream

Fuente: Propia

4.2.3 Despliegue balanceador de carga HTTP

Luego de la etapa de instalación y de configuración del balanceador de carga, se inició la etapa de despliegue. Para esto se realizaron pequeñas pruebas de validación de la configuración para determinar el correcto funcionamiento del balanceador de carga.

4.2.3.1 Modelo de despliegue

Se utilizaron tres servidores web de prueba junto con el balanceador de carga, para verificar el balanceo de peticiones entre los distintos servidores web.

La *figura 27* ilustra el modelo de prueba que se utilizó para realizar el despliegue.

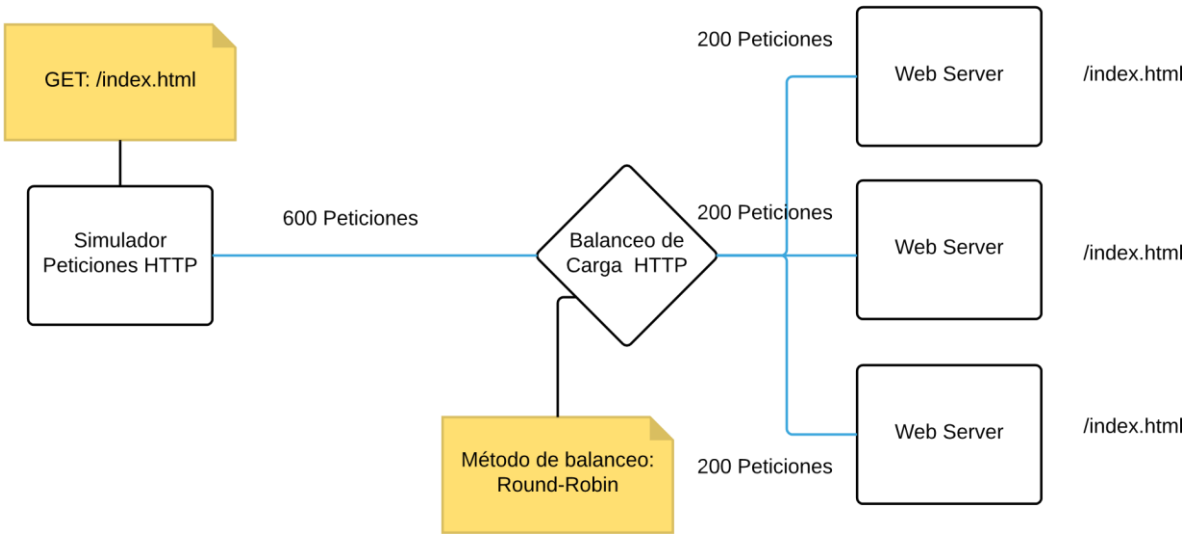


Figura 27: Esquema de despliegue para el balanceador de carga

Fuente: propia

Este modelo consiste en generar un número fijo de peticiones HTTP y verificar cuantas peticiones fueron atendidas por los servidores de back-end, con esto se puede determinar el funcionamiento correcto del balanceador de carga. El método de balanceo de carga utilizado es definido dentro del bloque server de la configuración, para este caso se utilizó el método round robin.

4.2.3.2 Implementación del despliegue

Se utilizó la máquina virtual #8 para generar 600 peticiones, las cuales realizan una petición GET al archivo index.html alojado en los servidores web de prueba, los servidores de pruebas fueron implementados con las máquinas #2,3, y 4 y estos servidores ejecutan Nginx web server.

Para verificar la distribución de las peticiones se utilizó el log del sistema de acceso, tanto en el balanceador de carga como en los servidores de prueba.

A continuación, se presenta evidencia de los archivos log de los servidores web de prueba y el balanceador de carga.

Archivo log balanceador de carga

Ruta archivo Log: /var/log/nginx/drupal.access.log

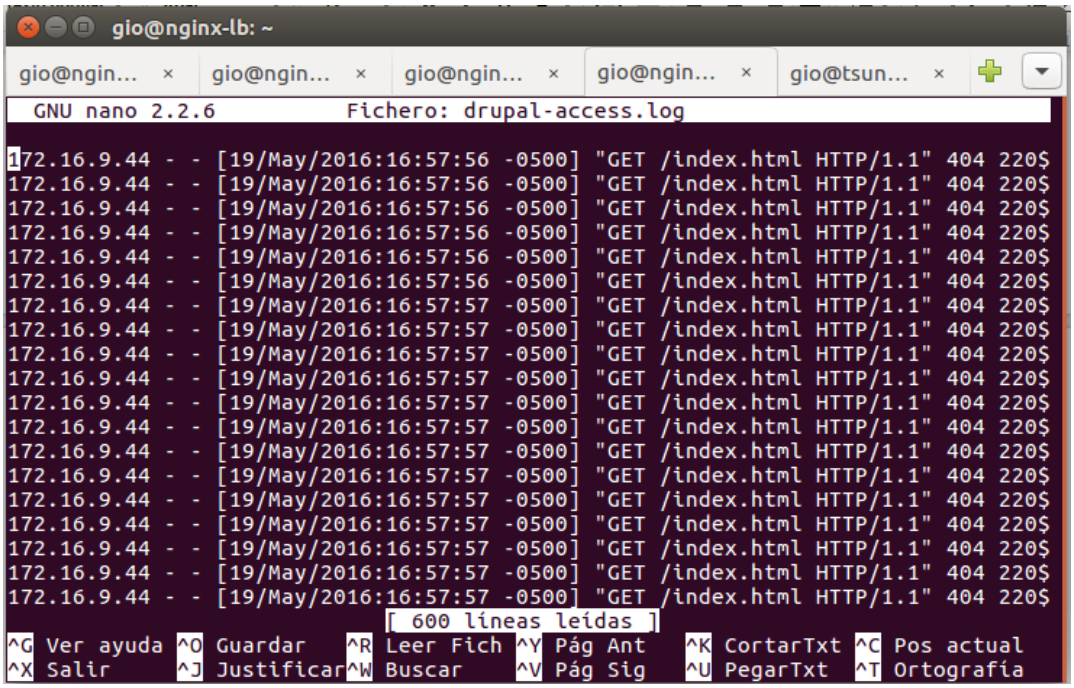


Figura 28:Archivo access.log del balanceador de carga

Como se puede apreciar en la *figura 28* el archivo contiene 600 líneas, esto indica que el balanceador atendió 600 peticiones provenientes del generador de carga HTTP. A continuación, se presenta una línea del archivo **drupal.access.log** del balanceador de carga.

172.16.9.44 -- [19/May/2016:16:57:56 -0500] "GET /index.html HTTP/1.1" 404 22062 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8) Gecko/20050513050513 Galeon/1.3.21"

Una línea del archivo hace referencia a un registro de acceso, cada línea contiene información pertinente al *Host* del cliente que está accediendo al balanceador.

El esquema de los registros de acceso es el siguiente:

IP Host - Fecha de acceso - Tipo petición y URI - Código HTTP - Navegador Host Cliente

Para determinar si las peticiones HTTP proviene del generar de carga, podemos ver que la dirección del host cliente de cada línea de acceso es 192.16.9.44, la cual es la dirección asignada la MV que contiene el generador de carga.

Archivo log Web server 01

Ruta archivo Log: /var/log/nginx/access.log.1

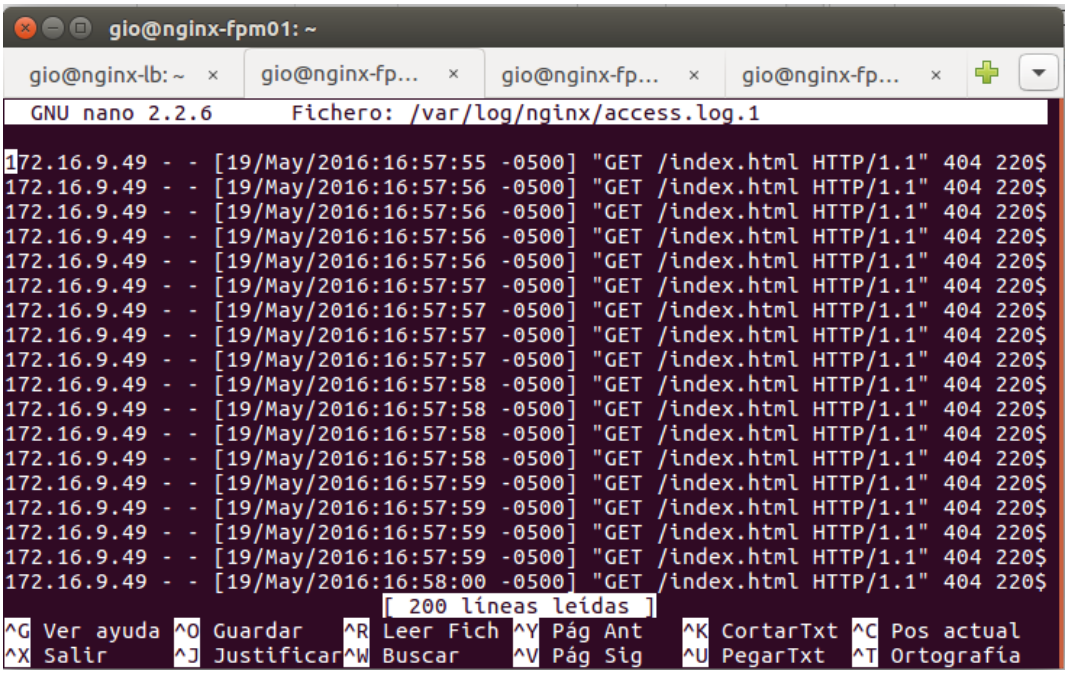


Figura 29: Archivo access.log.1 web server 01

El archivo del servidor web 01 ilustrado en la *figura 29* muestra que posee 200 líneas, lo que indica que atendió 200 peticiones. Cada línea tiene como IP de host cliente la dirección 172.16.9.49, lo que indica que recibió peticiones del balanceador de carga, el cual está instalado en la MV con la IP 17.16.9.49. Estos resultados son correctos, puesto que los servidores de prueba o de **back-end** deben recibir peticiones provenientes del balanceador de carga y no del generador de carga.

Los archivos del servidor 02 y servidor 03, evidencia el mismo resultado, a continuación, se presenta cada uno de ellos.

Archivo log Web server 02

Ruta archivo Log: /var/log/nginx/access.log.1

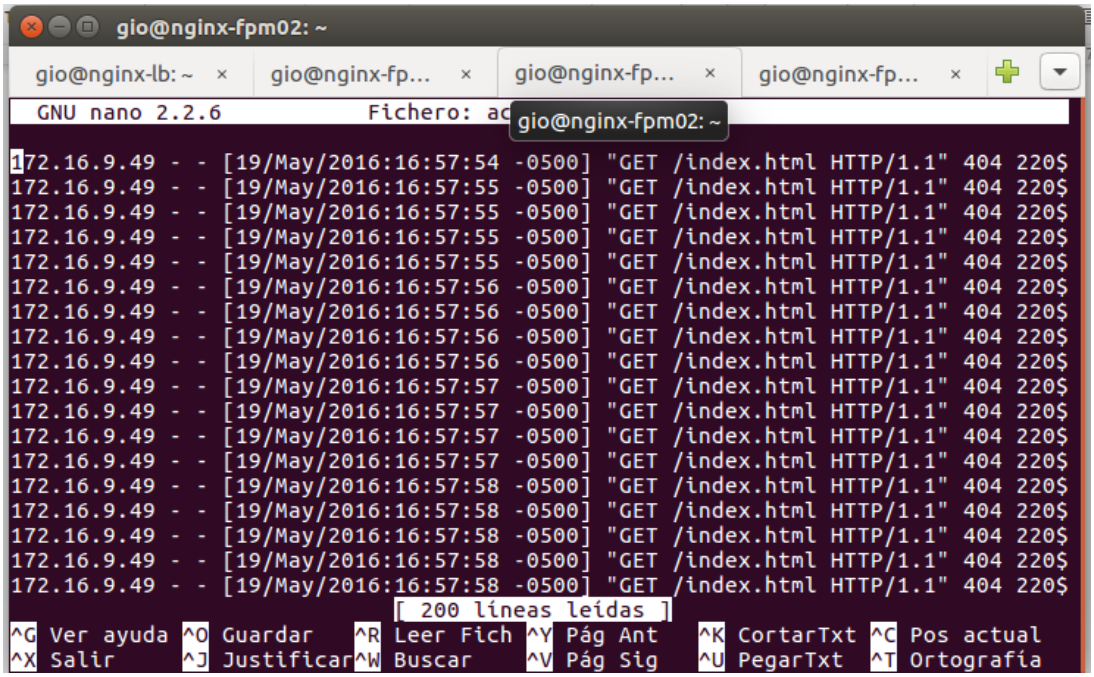


Figura 30: Archivo access.log.1 web server 02

En la *figura 30* se puede apreciar que la dirección IP host para cada registro de acceso es 172.16.9.49 la cual pertenece al balanceador de carga.

Archivo log Web server 03

Ruta archivo Log: /var/log/nginx/access.log.1

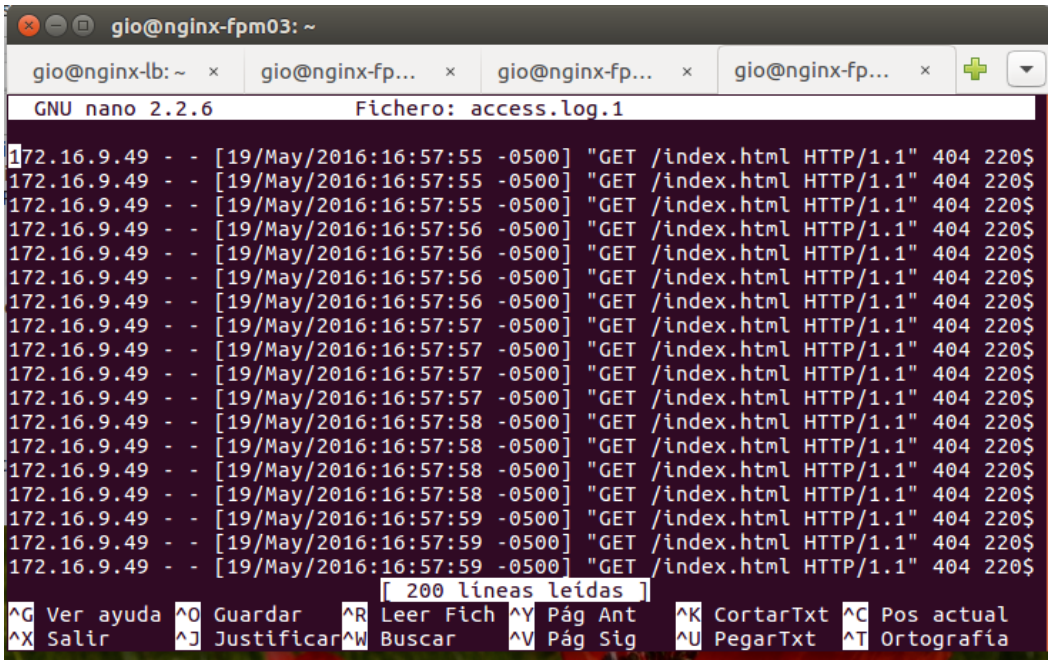


Figura 31: Archivo access.log.1 web server 03

En la *Figura 31* se puede apreciar que la dirección IP host para cada registro de acceso es 172.16.9.49 la cual pertenece al balanceador de carga.

En resumen, la implementación del balanceador de carga, consistió en tres fases principales, las cuales fueron: instalación, configuración y despliegue. La configuración tuvo como actor principal la preparación del sistema operativo, en este caso fue Debian 8 Jesse; el cual tuvo que ser configurado para poder descargar los paquetes binarios de Nginx correspondientes a la arquitectura Debian x64. Luego la fase de configuración se realizó mediante la implementación de un Host Virtual, en donde se configuraron parámetro para recibir peticiones de clientes y luego enviarlas a los servidores de back-end. Por último, se llevó a cabo la fase de despliegue o demostración, en donde se realizó un montaje de prueba para determinar el correcto funcionamiento del balanceador de carga.

4.3 Implementación entorno múltiple de servidores

El componente de múltiples servidores se fundamenta en utilizar diferentes Hosts (Máquinas virtuales o Físicas) para servir una misma aplicación. Para esta implementación la aplicación a servir es un CMS Drupal 7 la cual contiene un sitio web. Una característica importante de ese componente es la sincronización de código, cada servidor de back-end debe contener una réplica exacta del código de la aplicación Drupal y debe ser lo suficientemente flexible durante su etapa de producción.

4.3.1 Instalación de entorno múltiples de servidores

Como se especificó anteriormente todas las máquinas virtuales ejecutan el sistema operativo Debian 8 Jessie x64, las máquinas virtuales que se utilizaron en este componente fueron las máquinas número 2, 3, 4, 5, 6, y 7. Las máquinas número 2, 3, y 4 conforman un clúster donde el procesador de código es PHP-FPM; las máquinas 5, 6, y 7 conforman otro clúster donde el procesador de código es HHVM. La implementación de estos dos clústeres tiene como motivo el análisis de rendimiento entre cada uno de ellos, expuesto en el siguiente capítulo de pruebas y validación; en este capítulo se expondrá únicamente todo lo relaciona con su implementación.

4.3.1.1 Instalación clúster de servidores web PHP-FPM

Las máquinas virtuales utilizadas para este clúster fueron: MV número 2, 3, y 4. Cada máquina ejecuta un servidor web nginx junto con el procesador de código PHP-FPM, tanto el servidor web como el procesador de código son aplicativos distintos, a continuación, se describe el proceso de instalación de cada uno de estos.

4.3.1.1.1 Instalación servidores web NGINX

Los siguientes pasos de instalación se repiten para cada uno de los servidores dentro del clúster.

Paso 1 - Configuración de fuentes de instalación Debian 8

En el archivo /etc/apt/sources.list se agregan los repositorios del servidor de archivo debian, ubicado en la dirección URL: http://ftp.br.debian.org/debian, la siguiente imagen muestra el archivo original de configuración.

Archivo sources.list

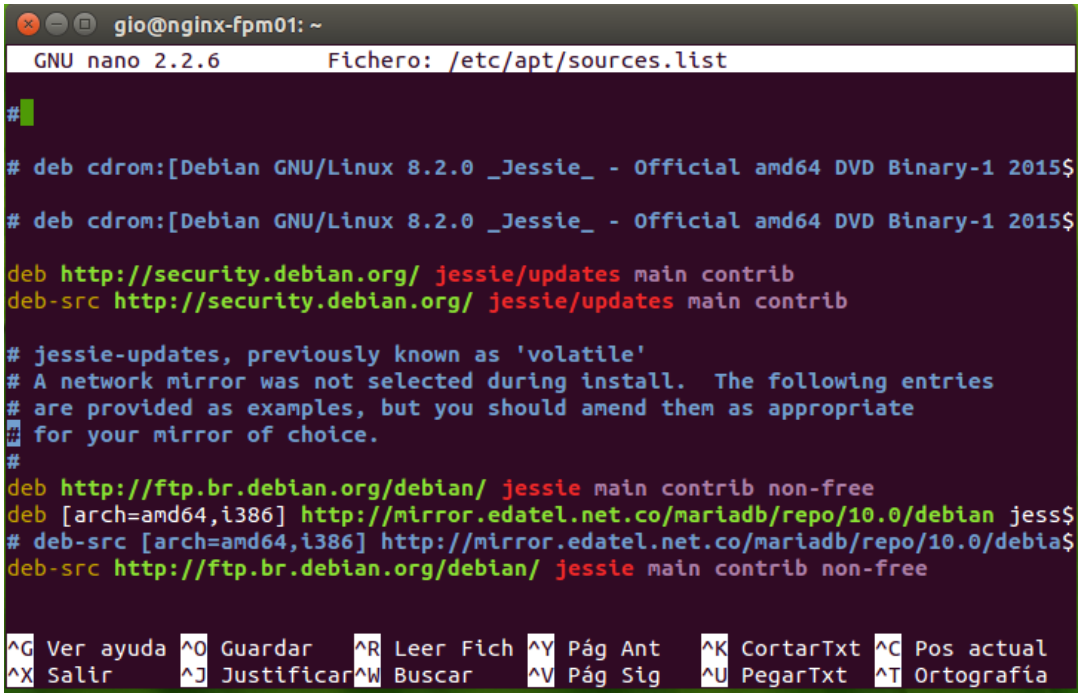


Figura 32: Archivo source.list Debian 8

Paso 2 - Actualización de fuentes de instalación

Al ejecutar el siguiente comando se actualizan todas las fuentes de instalación de los paquetes debian.

```
$ sudo apt-get update
```

Paso 3 - Instalación de paquetes binarios NGINX

Al ejecutar el siguiente comando se instalan todos los paquetes necesarios para el servidor web nginx.

```
$ sudo apt-get install nginx
```

La siguiente imagen muestra el directorio raíz de los archivos Nginx instalados dentro del sistema Debian.

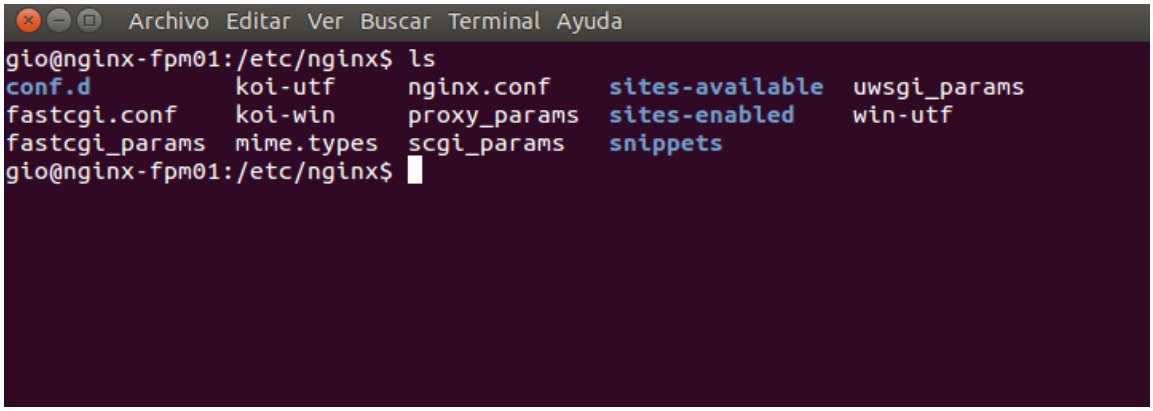


Figura 33: Listado de directorios y archivos de NGINX

4.3.1.1.2 Instalación del procesador de código PHP-FPM

Para procesar código PHP junto con el servidor web Nginx se utilizó la implementación FastCGI PHP-FPM. Cabe mencionar que los servidores web Nginx por defecto no contiene ningún procesador de código. Para instalar PHP-FPM se ejecuta el siguiente comando:

```
$sudo apt-get install php5-fpm
```

La implementación de PHP-FPM utiliza la versión de PHP 5.6.7 y es ejecutado bajo un proceso demonio ubicado en la ruta /var/run/php5-fpm.sock, esta ruta es

de vital importancia puesto que se debe tener en cuenta en la configuración del servidor web. La actividad fundamental de PHP-FPM es manejar el contenido dinámico (páginas con código PHP), estableciendo una comunicación fastCGI entre el servidor web (nginx) y su demonio php5-fpm.sock por medio de un socket unix.

La *Figura 34* muestra el directorio de instalación de PHP-FPM.

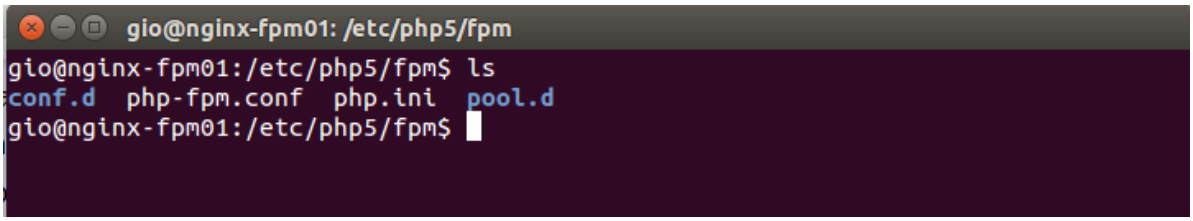


Figura 34: Directorio y archivos del procesador de código PHP-FPM

La *Figura 35* ilustra la ubicación del archivo que ejecuta el demonio php5-fpm.sock bajo el sistema operativo Debian 8.

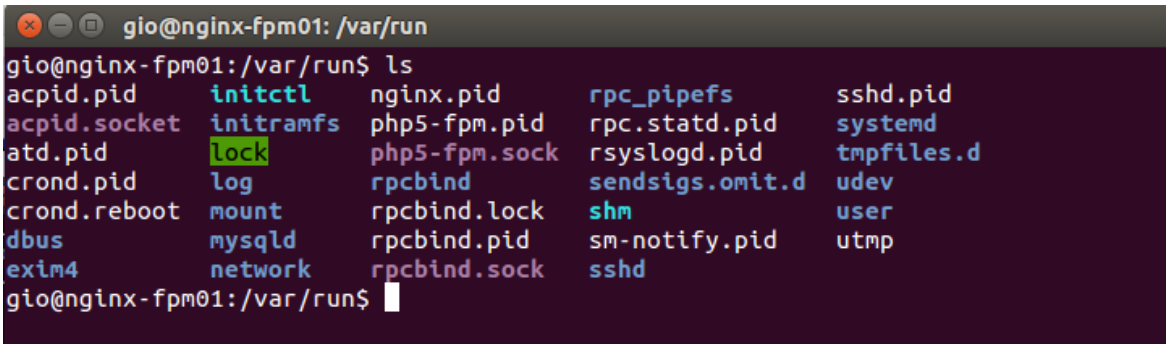


Figura 35: Ubicación del archivo php5-fpm.sock

4.3.1.2 Instalación clúster de servidores web HHVM

Las máquinas virtuales en este clúster fueron: MV número 5, 6, y 7. En cada una de ellas se instaló el servidor web Nginx y un procesador de código PHP. En el clúster anterior el procesador de código fue PHP-FPM, para este clúster se instaló el procesador de código HHVM.

La instalación del servidor nginx en las máquinas número 5, 6 y 7 se realizó de la misma manera que en las máquinas número 2,3 y 4; por esta razón se omite en el proceso de instalación en esta sección y únicamente se describe el proceso de instalación de HHVM.

4.3.1.2.1 Instalación del procesador de código HHVM

Como se mencionó anteriormente los servidores web nginx por defecto no procesan código de ningún lenguaje de programación, por lo tanto, para procesar código PHP en este clúster se utilizó el aplicativo HHVM. El proceso de instalación de HHVM se describe a continuación.

Paso 1 - Configuración de fuentes de instalación HHVM

Para instalar HHVM inicialmente se utilizó el repositorio oficial de Ubuntu 14.04, el cual provee soporte para el sistema operativo Debian 8.

Para utilizar este repositorio instaló una llave pública, para esto se ejecutó el siguiente comando.

```
$sudo apt-key adv --recv-keys --keyserver
hkp://keyserver.ubuntu.com:80 0x5a16e7281be7a449
```

Luego de agregar la llave pública se agregó la ruta del repositorio de instalación de HHVM, ejecutando el siguiente comando:

```
$ sudo add-apt-repository "deb http://dl.hhvm.com/ubuntu
$(lsb_release -sc) main"
```

Finalmente se actualizó el gestor de paquetes de debian utilizando el siguiente comando.

```
$ sudo apt-get update
```

Paso 2 - Instalación de paquetes binarios HHVM

Luego de tener el sistema operativo Debian configurado para poder descargar los archivos binarios de HHVM, se ejecutó la siguiente línea de comando para instalar los paquetes.

```
$sudo apt-get install hhvm
```

La siguiente Figura (*Figura 36*) muestra que HHVM se instaló correctamente dentro del sistema Debian 8, ejecutando la versión 3.12.0

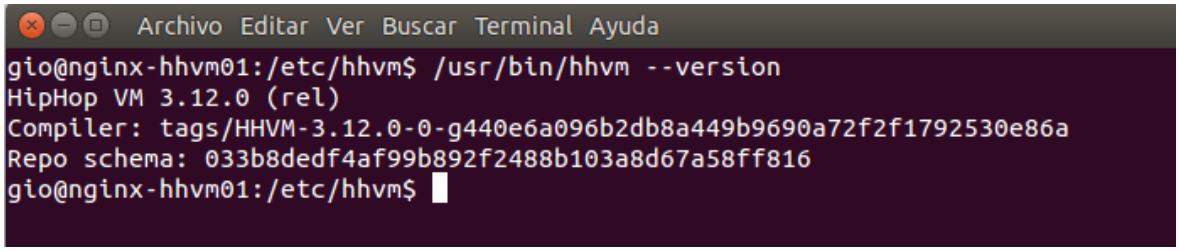


Figura 36:HHVM instalado en un sistema Debian 8 Jessie

HHVM es una solución desarrollada por Facebook para el procesamiento de código PHP por servidores web, en esta implementación HHVM se comunica con el servidor web nginx por medio de un socket UNIX, para poder procesar el contenido PHP que llegue al servidor.

La siguiente figura (figura 37) ilustra la ubicación del archivo hhvm.sock con el cual realiza la comunicación entre el servidor Nginx y HHVM.

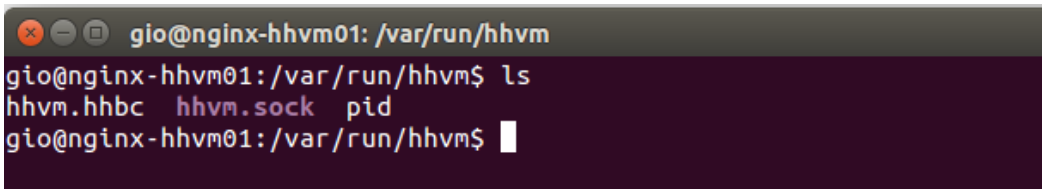


Figura 37: Ubicación archivo hhvm.sock

4.3.2 Configuración de entorno múltiples de servidores

En la anterior sección se instalaron todos los paquetes necesarios para poder ejecutar los servidores web nginx y los procesadores de código PHP-FPM y HHVM en sus respectivas máquinas virtuales. A continuación, se describe la configuración de los servidores web nginx junto con su respectivo procesador de código.

4.3.2.1 Configuración servidores web nginx PHP-FPM

Para configurar un servidor web Nginx junto con el procesador de código PHP-FPM, se especificó un ***host virtual*** mediante la creación de un archivo de configuración nginx dentro del directorio /etc/nginx/sites-available/ llamado "drupal".

Dentro del archivo "drupal" se realizaron las configuraciones pertinentes para el funcionamiento entre el servidor web Nginx y el procesador de código PHP-FPM, a continuación, se presenta el código de configuración del archivo "drupal".

4.3.2.1.1 Código archivo de configuración servidor web nginx PHP-FPM

La estructura principal del archivo de configuración consta de un bloque **server** principal y 5 bloques ***location*** internos, especificados dentro del bloque server, como se indica a continuación.

****

```
server {
```

```
    ...
    location / {
     ...
    }
    location ~ '\.php$|^/update.php' {
    ...
    }
    location ~ ^/sites/.*/files/styles/ {
    }
    location @rewrite {
    …
    }
    location ~ ^/sites/.*/files/styles/ {
    ...
    }
}
```

****

A continuación, se explica la funcionalidad de cada bloque de configuración

Bloque server

Este bloque habilita un host virtual el cual se encarga de manejar las peticiones entrantes al servidor.

```
server {
    listen 9000;

    server_name www.drupal.com.co;
    root /usr/share/nginx/drupal;
```

```
    index index.php index.html index.htm;

    ….

}
```

Las directivas listen habilita el puerto TCP 9000 para establecer conexión con el balanceador de carga, la directiva ***server_name*** establece el nombre del host www.drupal.com.co, la directiva ***root*** especifica la ruta raíz del host en donde se aloja el código de la aplicación, la directiva ***index*** define el nombre del archivo que será establecido como index de la aplicación.

Bloque location 1

```
 location / {
  try_files $uri $uri/ /index.php?$query_string; # For Drupal >= 7
 }
```

Los bloques **location** determinan como el servidor nginx procesa las peticiones, esto ayuda al servidor a manejar los archivos. La definición del modificador “/” indica que el servidor buscará dentro del directorio raíz y todos los subdirectorios de la raíz. La directiva **try_files** verifica si existe un archivo o directorio dependiendo de la URI especificada en la petición. Este bloque location es utilizado para procesar peticiones que requieran contenido estático como archivos o directorios.

Bloque location 2

```
location ~ '\.php$|^/update.php' {
        fastcgi_split_path_info ^(.+?\.php)(|/.*)$;
        fastcgi_intercept_errors on;
        fastcgi_pass unix:/var/run/php5-fpm.sock;
```

```
        fastcgi_index index.php;

        include fastcgi.conf;
        fastcgi_param PATH_INFO $fastcgi_path_info;
        fastcgi_param ENV development;
  }
```

La definición de este bloque se utilizó para comunicar el servidor web NGINX con la implementación FastCGI PHP-FPM, este proceso se implementó mediante la directiva **fastcgi_pass**, la cual establece la comunicación UNIX con el proceso demonio de PHP-FPM. Con esta configuración este bloque atenderá todas las peticiones que requieran un archivo con extensión (.php), el modificador "~" indica que se está definiendo una expresión regular.

La directiva **fastcgi_split_path_info** es utilizada para capturar los valores de la variable $fastcgi_path_info, esta variable contiene la ruta de la petición. La directiva **fastcgi_intercept_errors** determina si los códigos de respuesta del servidor FastCGI (PHP-FPM) son errores para procesarlos con la página de error de NGINX. La directiva **include** añade un archivo de configuración que utiliza el procesador de código PHP-FPM. La directiva **fastcgi_param** establece parámetro los cuales son pasados al servidor FastCGI.

Bloque 3

```
 location ~ ^/sites/.*/files/styles/ { # For Drpal >= 7
        try_files $uri @rewrite;
    }
```

Bloque 4

```
  location @rewrite {
    rewrite ^/(.*)$ /index.php?q=$1;
   }
```

Bloque 5

```
location ~ ^/sites/.*/files/styles/ {
    try_files $uri @rewrite;
  }
```

Los bloques 3, 4, y 5 se configuraron por requerimientos propios de la aplicación Drupal, puesto que las imágenes cargadas al sitio se alojaban en la ruta /files/styles y el servidor Nginx denegada su acceso por defecto.

Código de configuración completo (PHP-FPM)

****

```
server {
     listen 9000;

     server_name www.drupal.com.co;
     root /usr/share/nginx/drupal;
     index index.php index.html index.htm;

     location / {
     try_files $uri $uri/ /index.php?$query_string; #For Drupal >=
7
     }
     location ~ '\.php$|^/update.php' {
        fastcgi_split_path_info ^(.+?\.php)(|/.*)$;
        fastcgi_intercept_errors on;
        fastcgi_pass unix:/var/run/php5-fpm.sock;
        fastcgi_index index.php;
```

```
        include fastcgi.conf;
        fastcgi_param PATH_INFO $fastcgi_path_info;
        fastcgi_param ENV development;

    }

   location ~ ^/sites/.*/files/styles/ { # For Drpal >= 7
        try_files $uri @rewrite;
    }
    location @rewrite {
    rewrite ^/(.*)$ /index.php?q=$1;
    }
    location ~ ^/sites/.*/files/styles/ {
    try_files $uri @rewrite;
    }
}
```

****

Cabe mencionar que esta configuración se realiza de igual forma en cada una de las máquinas virtuales que conforman el clúster de servidores.

4.3.2.1.2 Esquema de configuración servidores web nginx PHP-FPM

La *figura 38* presenta un esquema gráfico de configuración de las máquinas virtuales número 2,3, y 4 las cuales conforman el clúster de servidores web nginx PHP-FPM.

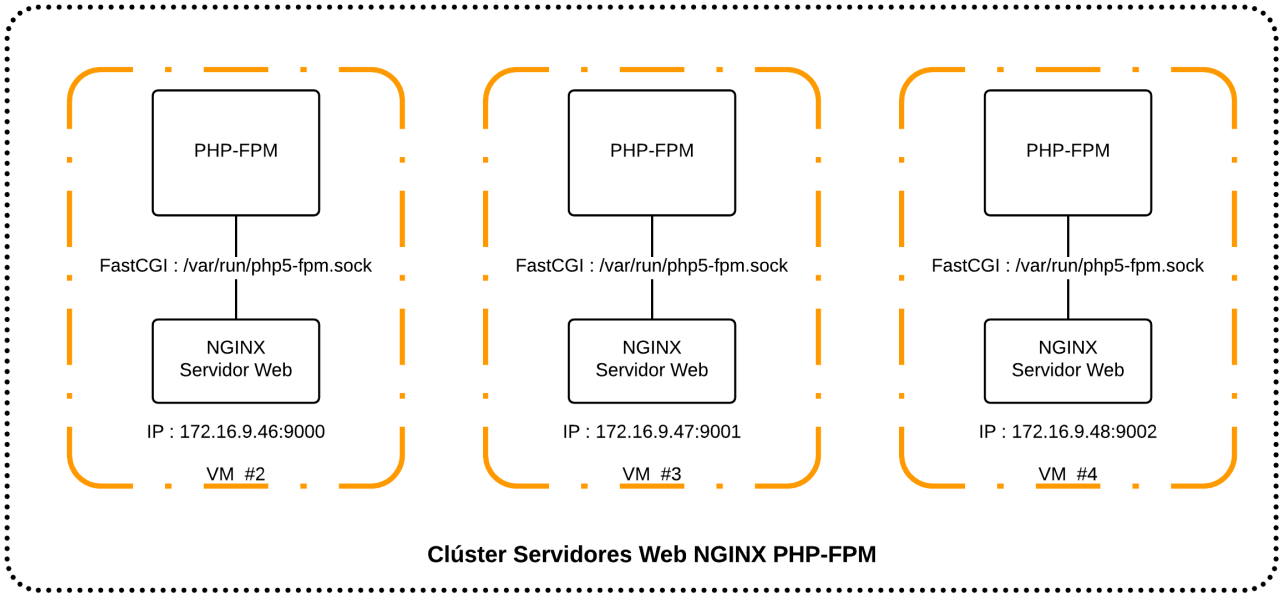


Figura 38: Esquema de configuración clúster de servidores web NGINX PHP-FPM

En la figura anterior se puede apreciar la comunicación FastCGI entre el servidor web y el procesador de código PHP-FPM, cada máquina contiene una dirección IP y un puerto TCP para recibir las peticiones provenientes del balanceador de carga.

4.3.2.2 Configuración servidores web nginx HHVM

Las máquinas virtuales que se configuraron para este clúster de servidores web NGINX HHVM fueron la número 5, 6, y 7. Para configurar un servidor web NGINX junto con el procesador de código HHVM se implementó un host virtual mediante la creación de un archivo de configuración NGINX dentro de la carpeta

/etc/nginx/sites-available/, puesto que la aplicación a servir por el servidor es un CMS Drupal al archivo de configuración se le estableció el nombre de “drupal” /etc/nginx/sites-available/drupal.

4.3.2.2.1 Código archivo de configuración servidor web nginx HHVM

La estructura del archivo de configuración /etc/nginx/sites-available/drupal para procesadores de código HHVM maneja la misma lógica de configuración que el archivo de configuración para procesadores PHP-FPM, puesto que se implementan bajo la misma plataforma de servidores web NGINX. A continuación, se presenta el código completo del archivo y se resalta la diferencia existente.

Código de configuración completo (HHVM)

```
**********************************************************************************************
server {
     listen 9000;

     server_name www.drupal.com.co;
     root /usr/share/nginx/drupal;
     index index.php index.html index.htm;

    #Bloque 1
    location / {
     try_files $uri $uri/ /index.php?$query_string; # For Drupal
>= 7
     }
```

```
#Bloque 2
     location ~ '\.(hh|php)$|^/update.php'  {
        fastcgi_keep_conn on;
        fastcgi_pass   unix:/var/run/hhvm/hhvm.sock;
        fastcgi_index  index.php;
 fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
      }
  #Bloque 3
    location ~ ^/sites/.*/files/styles/ { # For Drpal >= 7
    try_files $uri @rewrite;
    }
  #Bloque 4
    location @rewrite {
    rewrite ^/(.*)$ /index.php?q=$1;
    }
 #Bloque 5
    location ~ ^/sites/.*/files/styles/ {
    try_files $uri @rewrite;
    }
}
**************************************************************
```

La estructura de bloques **server** y **location** se configuró de la misma forma que el archivo de configuración para PHP-FPM, la diferencia radica en el bloque location número 2, en donde se cambió la definición de la directiva **fastcgi_pass** por la ruta del proceso demonio de HHVM, para este caso es

unix:/var/run/hhvm/hhvm.sock. Cabe mencionar que esta configuración se realiza para todas las máquinas del clúster de servidores Nginx HHVM.

4.3.2.2.2 Esquema de configuración servidores web nginx HHVM

La figura 39 presenta un esquema gráfico de configuración de las máquinas virtuales número 5, 6, y 7 las cuales conforman el clúster de servidores web Nginx HHVM.

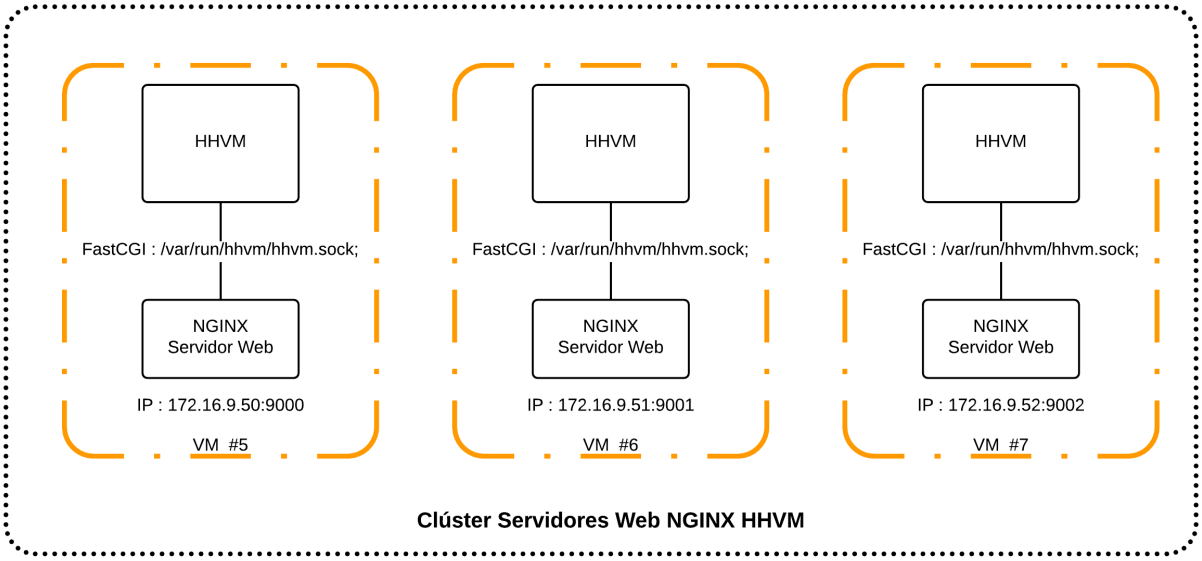


Figura 39: Esquema de configuración clúster de servidores web NGINX HHVM

En la figura anterior (*figura 39*) se ilustra la comunicación de los servidores web con el procesador de código HHVM, mediante un socket UNIX. Cada máquina virtual es configurada con una dirección IP y un puerto TCP para la comunicación con el balanceador de carga.

4.3.3 Despliegue de entorno múltiples de servidores

Para el despliegue del entorno múltiple de servidores web se realizaron pequeñas pruebas de laboratorio para verificar la ejecución de código PHP utilizando los procesadores de código PHP-FPM y HHVM.

Para la prueba se creó un archivo con extensión PHP llamado ***info.php***, este archivo contiene el siguiente código:

```
<?php

    phpinfo();

?>
```

El archivo se alojó en la ruta raíz de cada servidor web.

Ruta: /usr/share/nginx/drupal/info.php

4.3.3.1 Prueba del despliegue

Para verificar la prueba se utilizó la información expuesta por la función phpinfo() la cual nos indica que tipo de procesador de código PHP está ejecutando el servidor web. La tabla 5 describe cada una de las pruebas realizada en las respectivas máquinas virtuales.

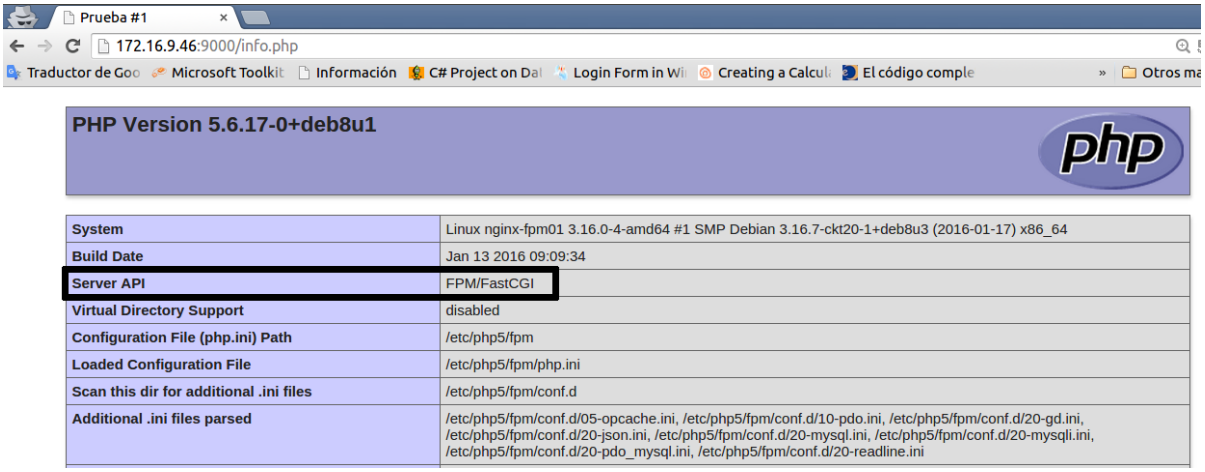
N° de prueba	**Máquina Virtual**	**Servidor WEB**	**IP**	**Puerto TCP**	**Tipo de Petición**	**Ruta de Archivo**
1	MV #2	Servidor NGINX PHP-FPM 01	172.16.9.46	9000	GET	/usr/share/nginx/drupal/info.php
2	MV #3	Servidor NGINX PHP-FPM 02	172.16.9.47	9001	GET	/usr/share/nginx/drupal/info.php

3	MV #4	Servidor NGINX PHP-FPM 03	172.16.9.48	9002	GET	/usr/share/nginx/drupal/info.php
4	MV #5	Servidor NGINX HHVM 01	172.16.9.50	9000	GET	/usr/share/nginx/drupal/info.php
5	MV #6	Servidor NGINX HHVM 02	172.16.9.51	9001	GET	/usr/share/nginx/drupal/info.php
6	MV #7	Servidor NGINX HHVM 03	172.16.9.52	9002	GET	/usr/share/nginx/drupal/info.php

Tabla 5: Lista de prueba de despliegue en entorno múltiple de servidores

A continuación, se evidencia cada prueba realiza, mediante las siguientes imágenes. En cada imagen se identifica la IP del servidor el puerto TCP y el procesador de código que está utilizando el servidor web.

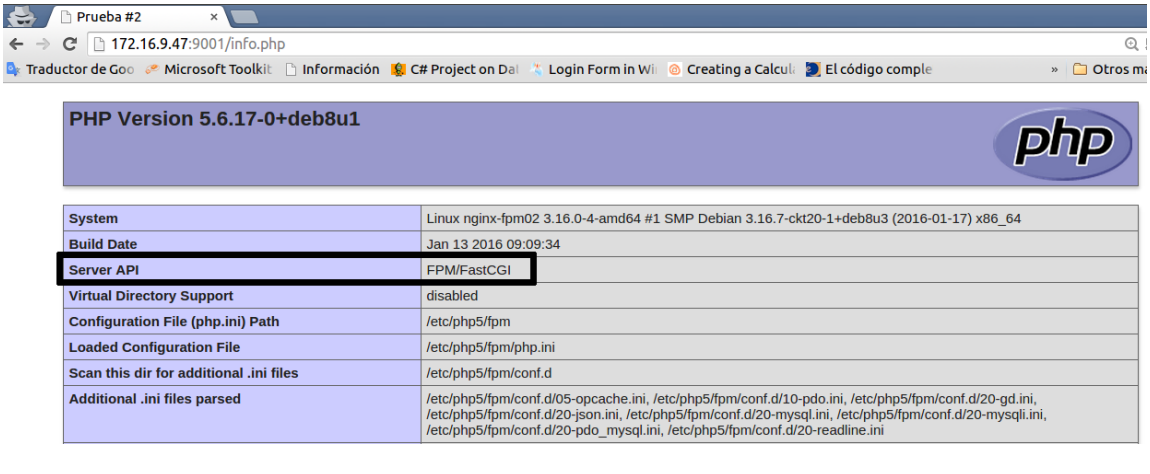
Prueba #1



System	Linux nginx-fpm01 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt20-1+deb8u3 (2016-01-17) x86_64
Build Date	Jan 13 2016 09:09:34
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/fpm
Loaded Configuration File	/etc/php5/fpm/php.ini
Scan this dir for additional .ini files	/etc/php5/fpm/conf.d
Additional .ini files parsed	/etc/php5/fpm/conf.d/05-opcache.ini, /etc/php5/fpm/conf.d/10-pdo.ini, /etc/php5/fpm/conf.d/20-gd.ini, /etc/php5/fpm/conf.d/20-json.ini, /etc/php5/fpm/conf.d/20-mysql.ini, /etc/php5/fpm/conf.d/20-mysqli.ini, /etc/php5/fpm/conf.d/20-pdo_mysql.ini, /etc/php5/fpm/conf.d/20-readline.ini

Figura 40: Prueba servidor NGINX PHP-FPM 01

Prueba #2



System	Linux nginx-fpm02 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt20-1+deb8u3 (2016-01-17) x86_64
Build Date	Jan 13 2016 09:09:34
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/fpm
Loaded Configuration File	/etc/php5/fpm/php.ini
Scan this dir for additional .ini files	/etc/php5/fpm/conf.d
Additional .ini files parsed	/etc/php5/fpm/conf.d/05-opcache.ini, /etc/php5/fpm/conf.d/10-pdo.ini, /etc/php5/fpm/conf.d/20-gd.ini, /etc/php5/fpm/conf.d/20-json.ini, /etc/php5/fpm/conf.d/20-mysql.ini, /etc/php5/fpm/conf.d/20-mysqli.ini, /etc/php5/fpm/conf.d/20-pdo_mysql.ini, /etc/php5/fpm/conf.d/20-readline.ini

Figura 41: Prueba servidor NGINX PHP-FPM 02

Prueba #3

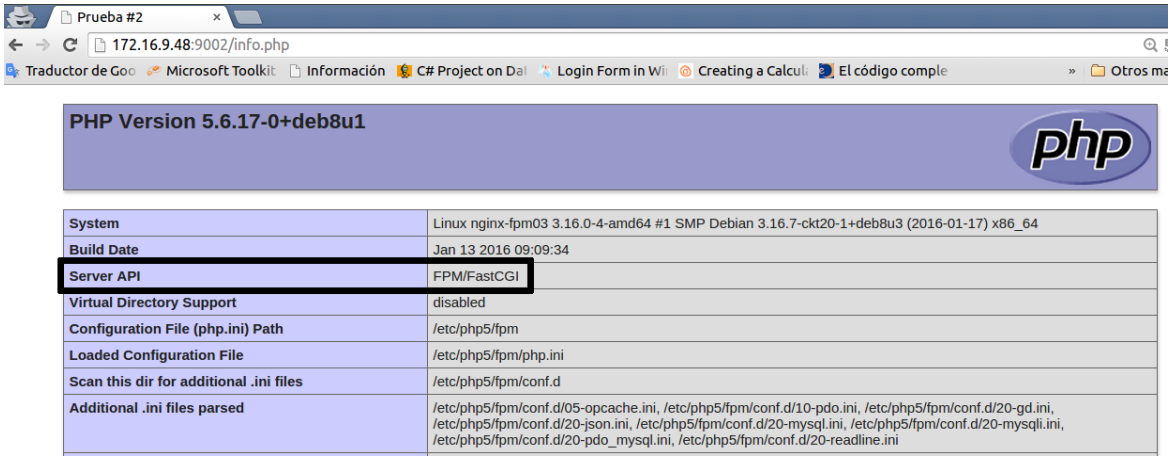


Figura 42: Prueba servidor NGINX PHP-FPM 03

Prueba #4

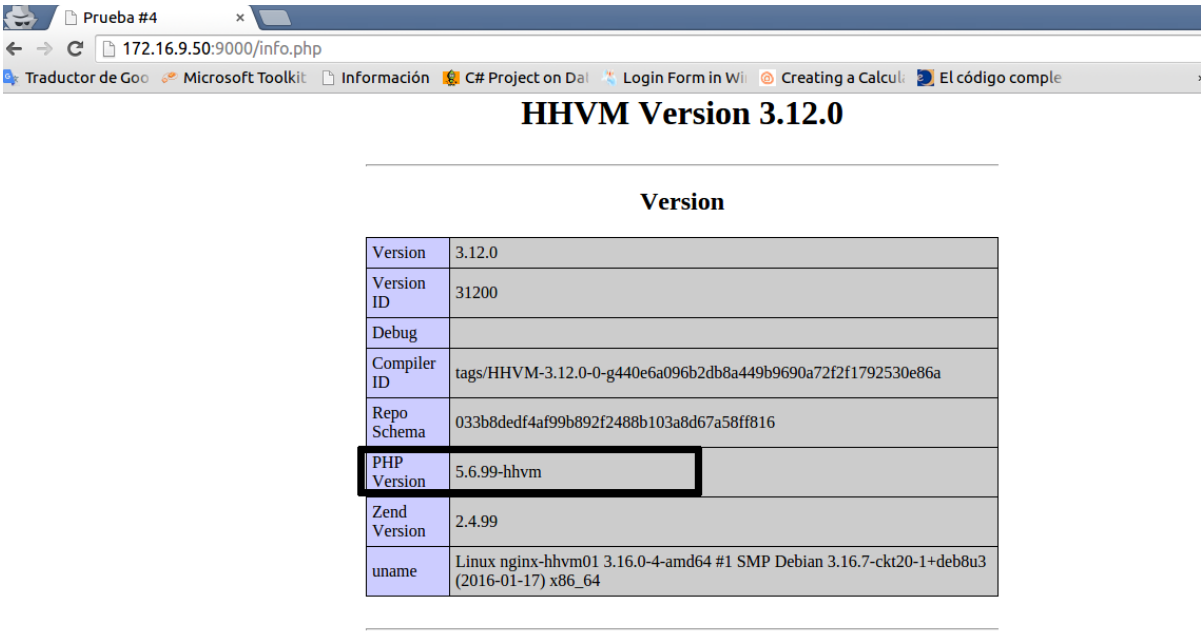


Figura 43: Prueba servidor NGINX HHVM 01

Prueba #5

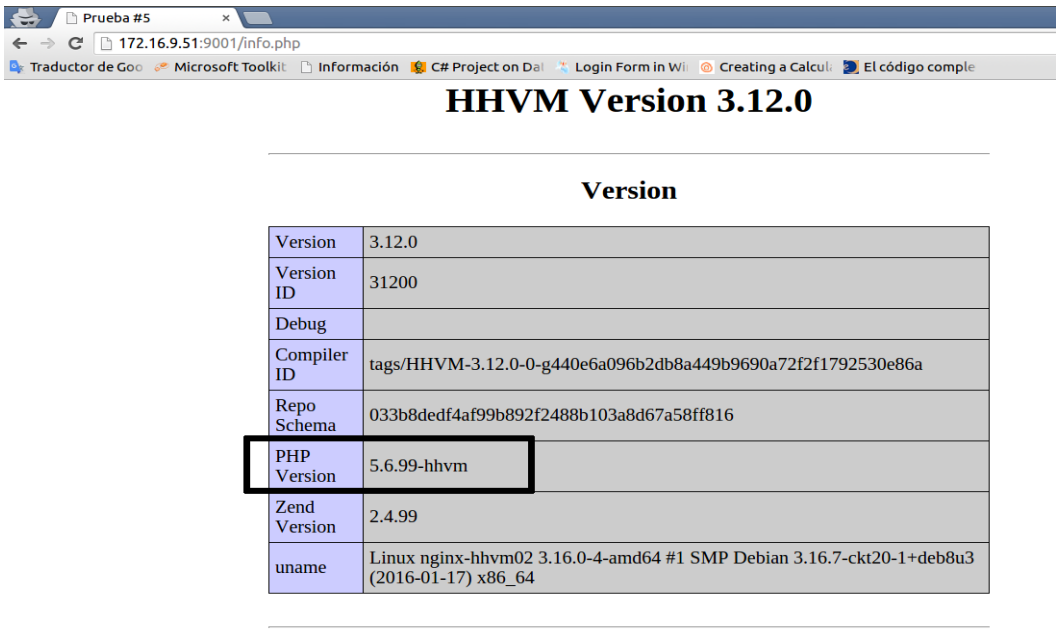


Figura 44: Prueba servidor NGINX HHVM 02

Prueba #6

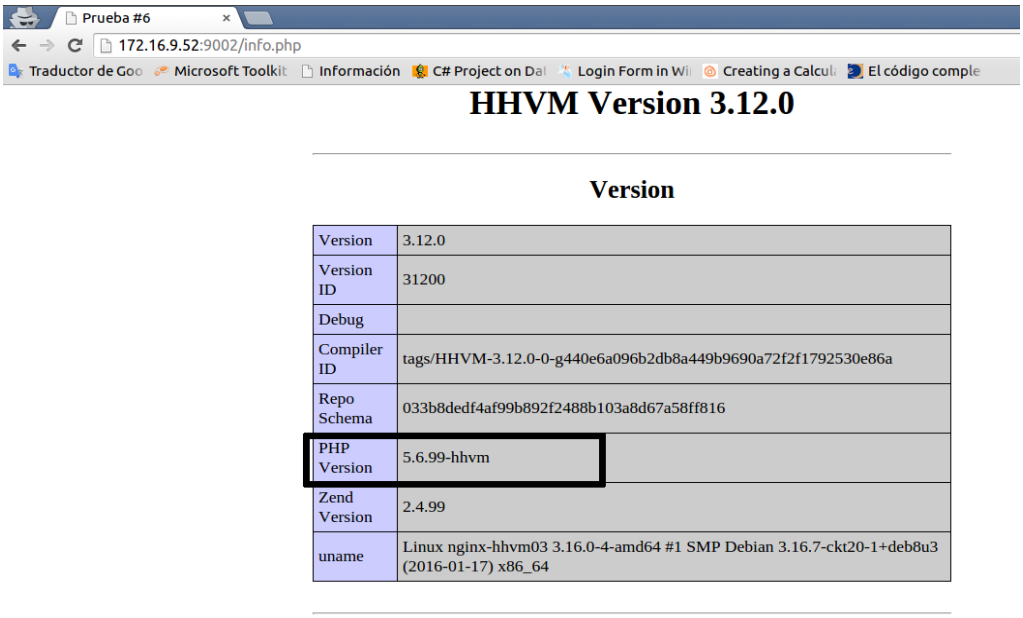


Figura 45: Prueba servidor NGINX HHVM 03

En resumen, la implementación del entorno múltiple de servidores, se llevó a cabo mediante la instalación y configuración de 6 máquinas virtuales, 3 de ellas conforman el clúster de servidores web que procesan el código PHP con PHP-FPM y las otras tres conforman un segundo clúster las cuales procesan el código con HHVM.

El aspecto fundamental de esta implementación fue prepara todos los componentes técnicos para poder servir aplicaciones desarrollada con el lenguaje PHP, agregando optimización en la ejecución del código y permitiendo la correcta comunicación con el balanceador de carga HTTP.

4.4 Implementación Clúster de Datos

El componente de clúster de datos se implementó mediante 3 máquinas virtuales, cada máquina virtual se utilizó como un nodo del clúster en modo master-master. Cabe mencionar que las máquinas virtuales número 2, 3, y 4 se utilizaron tanto para la instalación de los servidores web como para la instalación del clúster de datos.

4.4.1 Instalación clúster de datos

Las máquinas virtuales número 2, 3 y 4 tiene como sistema operativo Linux Debia 8, este por defecto no contiene los paquetes de instalación galera-wsrep y mariadb-galera-server, los cuales son el motor del clúster y el motor de base de datos respectivamente. Para descargar los paquetes de instalación se debe agregar los repositorios de instalación.

4.4.1.1 Agregar repositorios clúster galera MariaDB

Para agregar los repositorios del clúster Galera MariaDB se realizó el siguiente proceso en cada una de las MV's:

1 Paso - Instalar el paquete de administración para software propietario:

```
$ sudo apt-get install python-software-properties
```

2 paso - Agregar archivo de llaves para el repositorio

```
$ sudo apt-key adv –recv-keys –keyserver
hkp://keyserver.ubuntu.com:80 0xcbcb082a1bb943db
```

3 paso – Agregar el repositorio

```
$ sudo add-apt-repository 'deb [arch=amd64,i386]
http://mirror.edatel.net.co/mariadb/repo/10.0/debian jessie main'
```

4.4.1.2 Instalación de paquetes binarios clúster galera MariaDB

El motor del clúster y el motor de la base de datos se instaló en cada MV's, para realizar esto se utilizó el siguiente comando en cada uno de MV's.

```
$ sudo apt-get install mariadb-galera-server galera
```

4.4.2 Configuración clúster de datos MariaDB galera

Luego de realizar la instalación en cada una de las tres MV's, se realizó el proceso de configuración del clúster. Este proceso tiene como principal actor el archivo de configuración **mariadb.cnf** ubicado en el directorio /etc/mysql/conf.d, la *figura 46* ilustra los archivos de configuración del clúster.

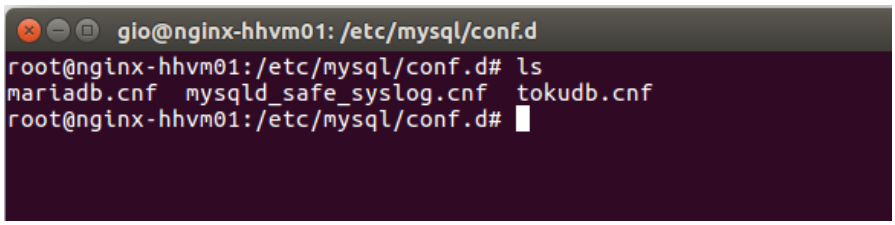


Figura 46: Archivos de configuración clúster Galera MariaDB

El clúster Galera MariaDB necesariamente tiene que compartir la configuración en cada uno de los nodos, por lo tanto, cada nodo posee una copia del archivo de configuración mariadb.cnf. A continuación, se presenta el código del archivo mariadb.cnf.

4.4.2.1 Código de configuración archivo mariadb.cnf

****

```
[client]
#Default is Latin1, if you need UTF-8 set this (also in server
section)
#default-character-set = utf8

[mysqld]

query_cache_size=0

binlog_format=ROW

default-storage-engine=innodb

innodb_autoinc_lock_mode=2

query_cache_type=0

bind-address=0.0.0.0

# Galera Provider Configuration

wsrep_provider=/usr/lib/galera/libgalera_smm.so

#wsrep_provider_options="gcache.size=32G"
```

```
# Galera Cluster Configuration

wsrep_cluster_name=mycluster

wsrep_cluster_address="gcomm://172.16.9.46,172.16.9.47,172.16.9.48
"


# Galera Synchronization Congifuration

wsrep_sst_method=rsync

#wsrep_sst_auth=user:pass


# Galera Node Configuration

wsrep_node_address="172.16.9.46"

wsrep_node_name=node1
```

****

Para la configuración del clúster únicamente se utilizaron los parámetros resaltados en color azul. El parámetro **wsrep_cluster_name** se utiliza para identificar el clúster con un nombre en este caso fue “mycluster”, **wsrep_cluster_address** define las direcciones IP de cada uno de los nodos, **wsrep_sst_method** define el método de sincronización, **wsrep_node_address** define la dirección IP del nodo actual, **wsrep_node_name** define el nombre del nodo actual.

Los parámetros **wsrep_node_address** y **wsrep_node_name** cambian su valor dependiendo del nodo donde se esté configurando. A continuación, se ilustra la configuración de cada nodo del clúster.

Nodo 1

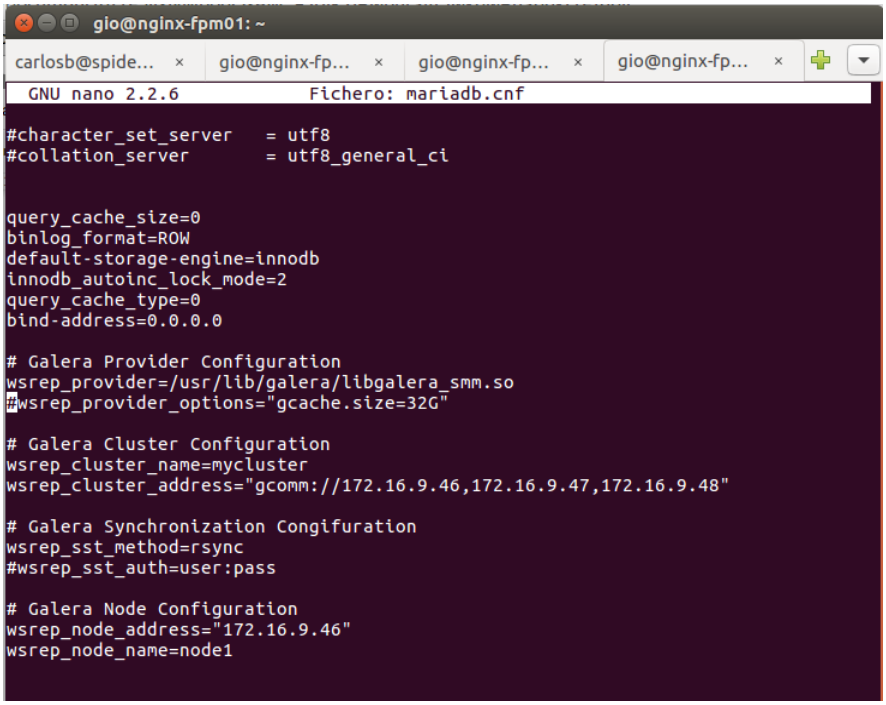


Figura 47: Configuración nodo 1, clúster MariaDB Galera

Nodo 2

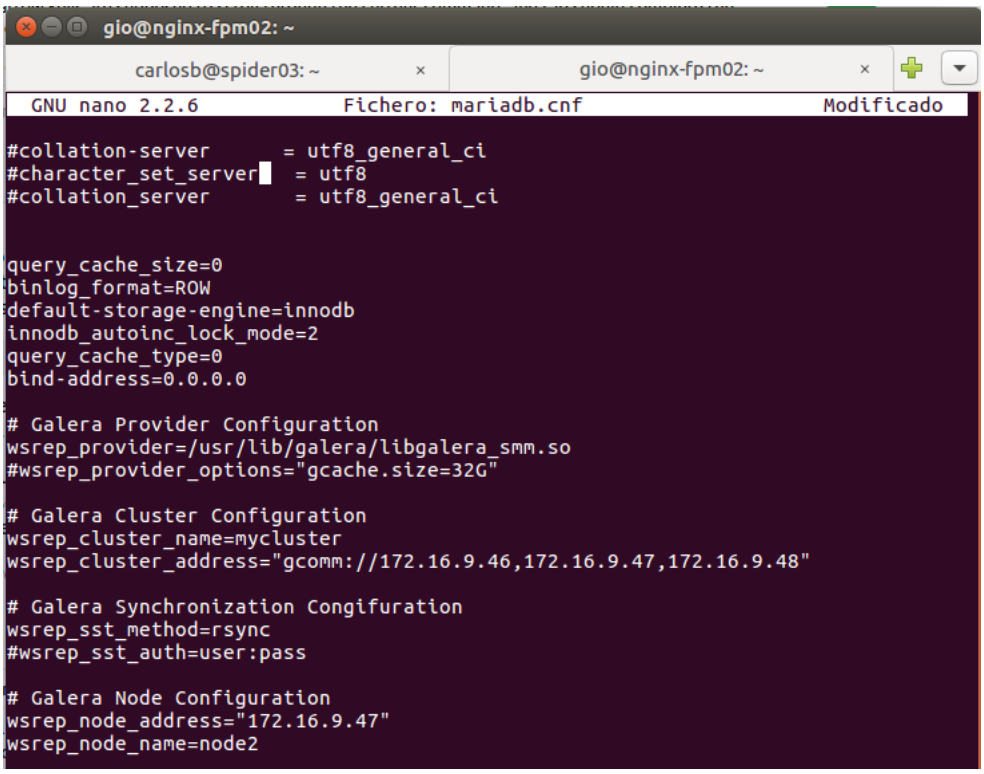


Figura 48: Configuración nodo 2, clúster MariaDB Galera

Nodo 3

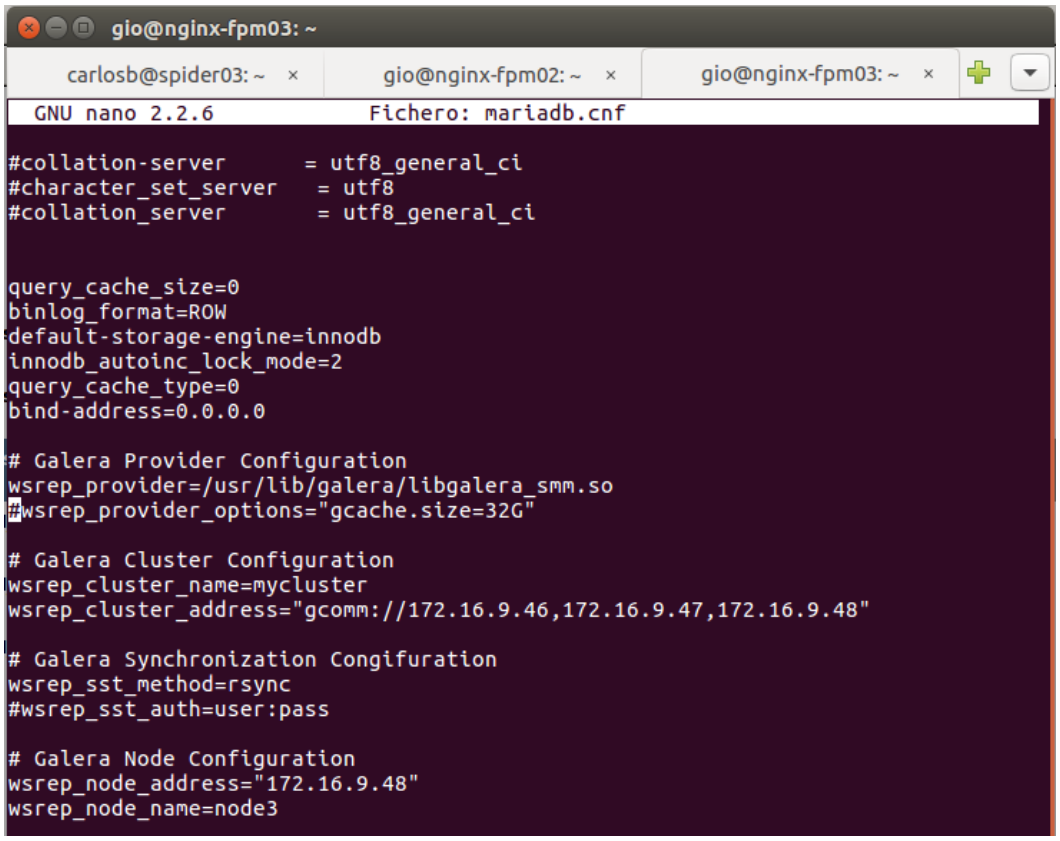


Figura 49: Configuración nodo 3, clúster MariaDB Galera

4.4.2.2 Esquema de configuración clúster galera MariaDB

La *figura 50* ilustra de una forma gráfica la configuración del clúster, donde se resalta los parámetros utilizados para su configuración.

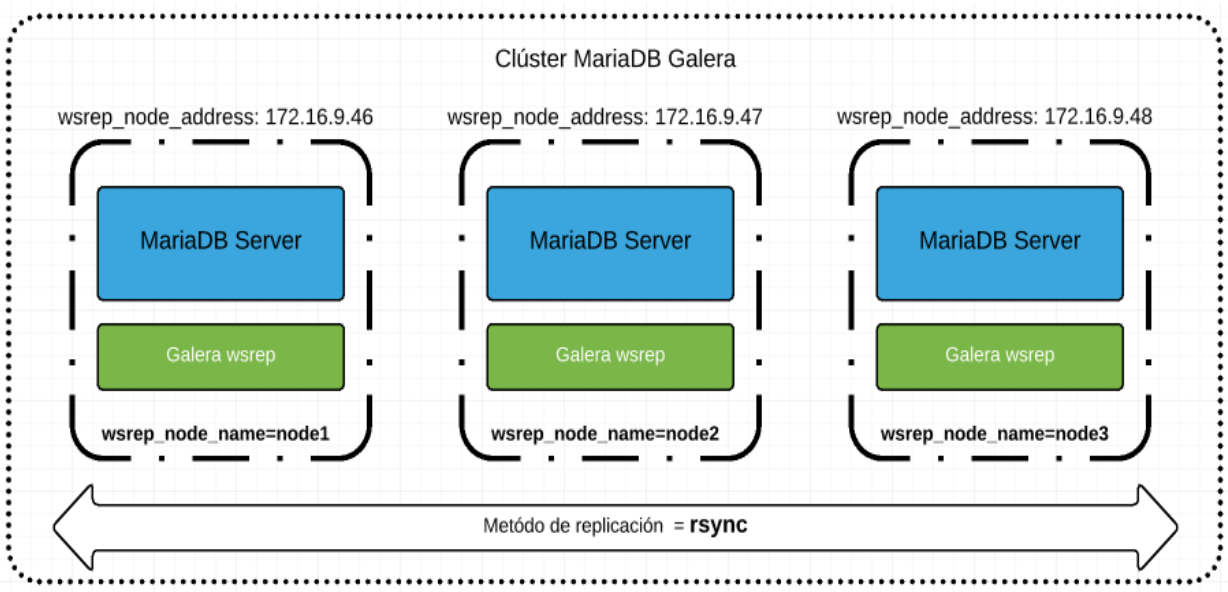


Figura 50: Esquema de configuración Clúster MariaDB Galera

4.4.3 Despliegue clúster de datos

Luego de terminar la instalación y configuración del clúster de datos, se realizó una prueba de laboratorio para determinar su correcto funcionamiento. La prueba consistió en crear una base de datos llamada “prueba” en el nodo 1 y verificar su replicación desde los demás nodos.

4.4.3.1 Inicio del servicio de clúster de datos

La instalación y configuración del clúster no activa automáticamente el servicio del mismo, para poder manejar el clúster se necesita iniciar el servicio del clúster en cada uno de los nodos, para esto se realizó el siguiente proceso.

Paso 1 - Iniciar el clúster nodo 1

Dentro del nodo 1 se inició el cluster por medio del siguiente comando:

```
$ sudo service mysql start --wsrep-new-cluster
```

Para verificar el inicio del clúster, se abre una terminal mysql y se ejecuta la siguiente sentencia:

```
> SHOW STATUS LIKE 'wsrep_%';
```

Esta sentencia muestra las variables de configuración del clúster, la imagen x muestra el resultado de la sentencia. Para verificar si el nodo 1 está conectado con el cluster, la variable **wsrep_cluster_size** debe ser 1, puesto que el clúster solo tiene conectado el nodo 1.

Sentencia SQL dentro del clúster MariaDB Galera

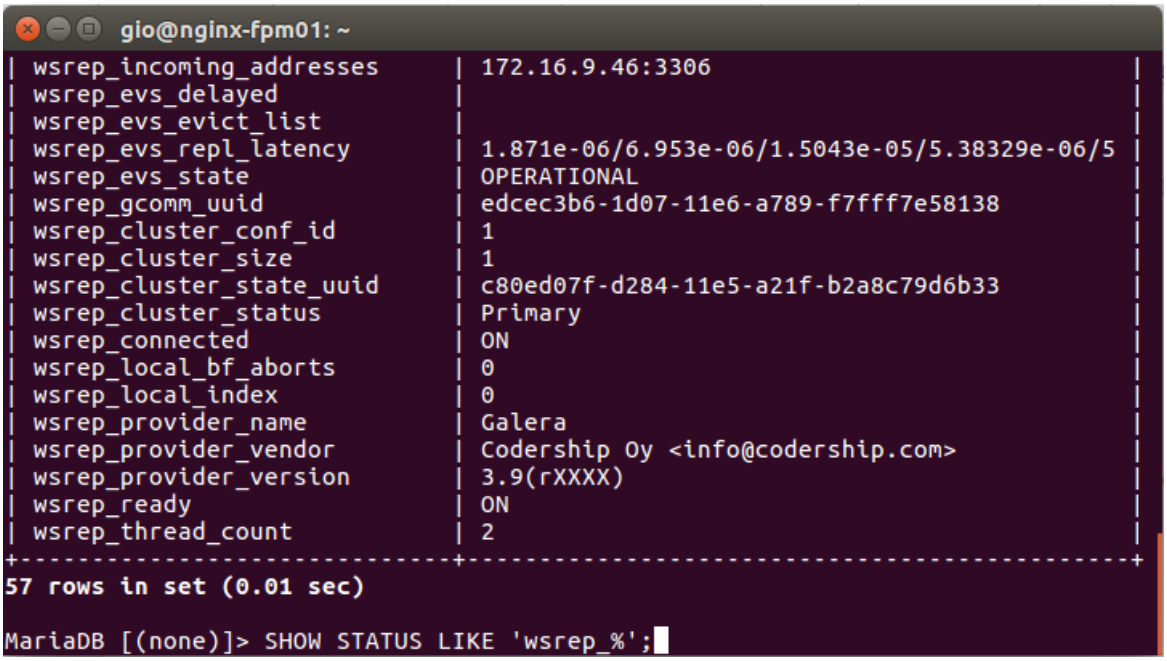


Figura 51: Tabla de resultado sentencia SQL (SHOW STATUS LIKE 'wsrep_%';) nodo 1

Paso 2 - Conexión nodo 2

Para la conexión del nodo 2 se utilizó el siguiente comando, este comando se ejecutó desde el nodo 2.

```
$ sudo mysqld --wsrep_cluster_address=gcomm://172.16.9.46
```

La dirección IP utilizada al final de la sentencia, es la dirección del nodo que inició el clúster, para este caso es la dirección del nodo 1. Para verificar que el nodo 2 está conectado al clúster se realizó el mismo proceso que en el paso 1. La *figura 52* ilustra el resultado, donde se determina que el parámetro wsrep_**cluster_size** cambió su valor a 2.

Sentencia SQL dentro del clúster MariaDB Galera

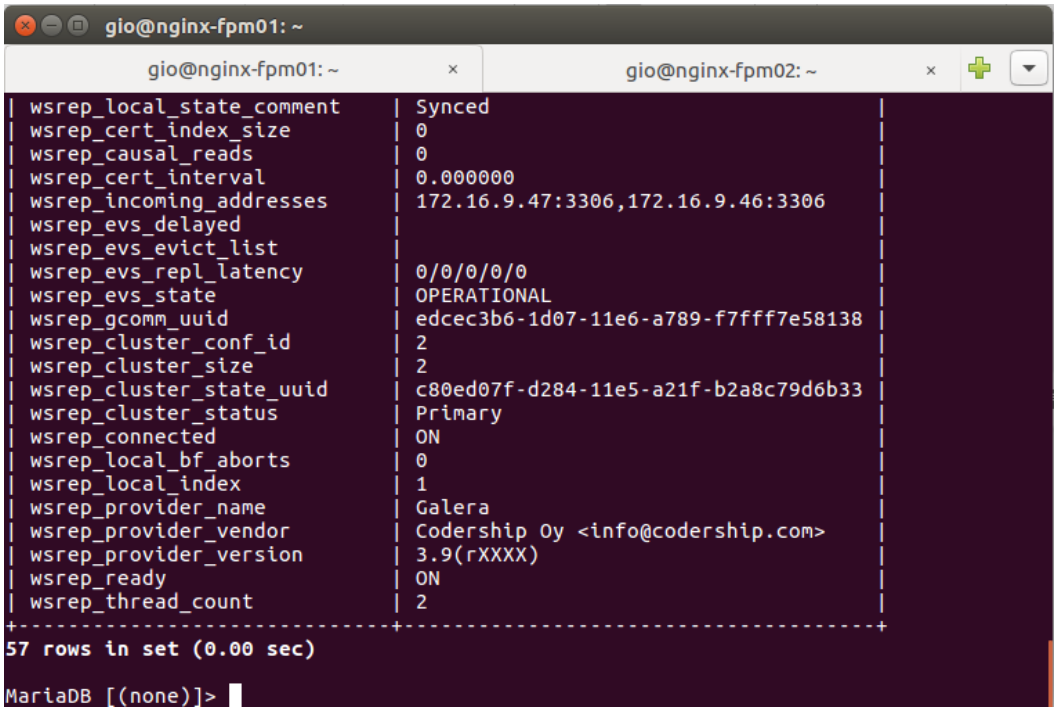


Figura 52: Tabla de resultado sentencia SQL (SHOW STATUS LIKE 'wsrep_%';) nodo 2

Paso 3 - Conexión nodo 3

Para la conexión del nodo 3, se realizó el mismo proceso descrito en el paso 2, teniendo en cuenta que la configuración se debía realizar sobre el nodo 3. La *figura 53* ilustra el aumento del valor de parámetro **wsrep_cluster_size,** en este caso el valor fue 3.

Sentencia SQL dentro del clúster MariaDB Galera

Figura 53: Tabla de resultado sentencia SQL (SHOW STATUS LIKE 'wsrep_%';) nodo 3

4.4.4 Test de replicación clúster de datos

Luego de haber iniciado el clúster y verificar su funcionamiento dentro de los tres nodos, se realizó el test de replicación.

Dentro del nodo 1 se inició una sesión mysql y se declaró la siguiente sentencia:

```
> MariaDB [(none)]> create database prueba;
```

Esta sentencia crea la base de datos dentro del clúster, la *figura 54* muestra la existencia de la base de datos dentro del clúster. La base de datos “prueba” fue creada desde el nodo 1.

Sentencia SQL nodo 1

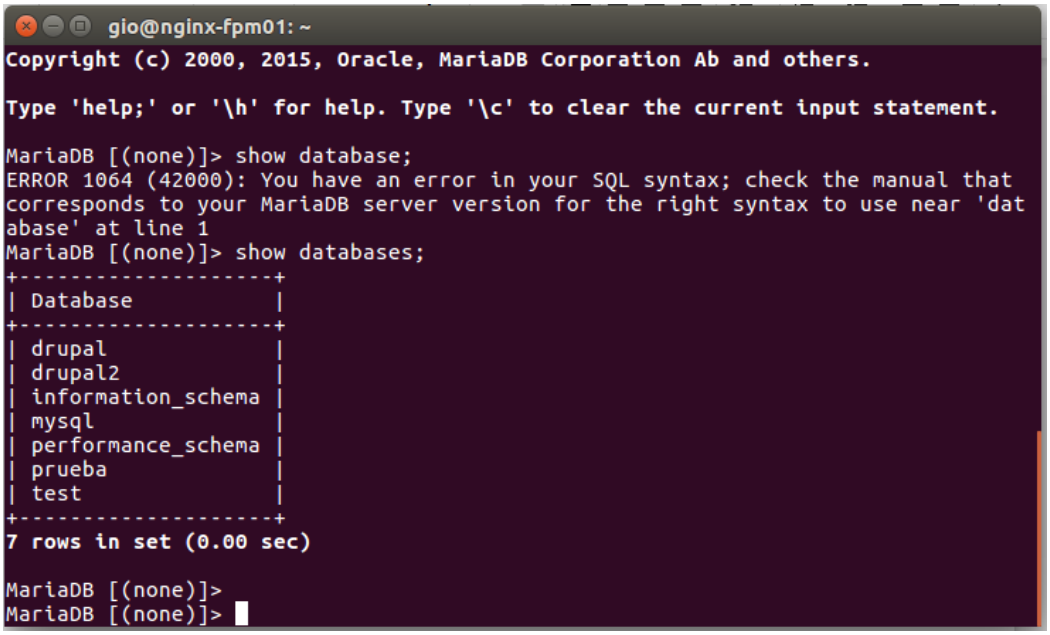


Figura 54: Consulta SQL nodo 1 (show databases;)

Luego desde el nodo 2 y 3 se verificó la existencia de la base de datos. La figura 55 y la imagen 56, evidencian este proceso.

Sentencia SQL nodo 2

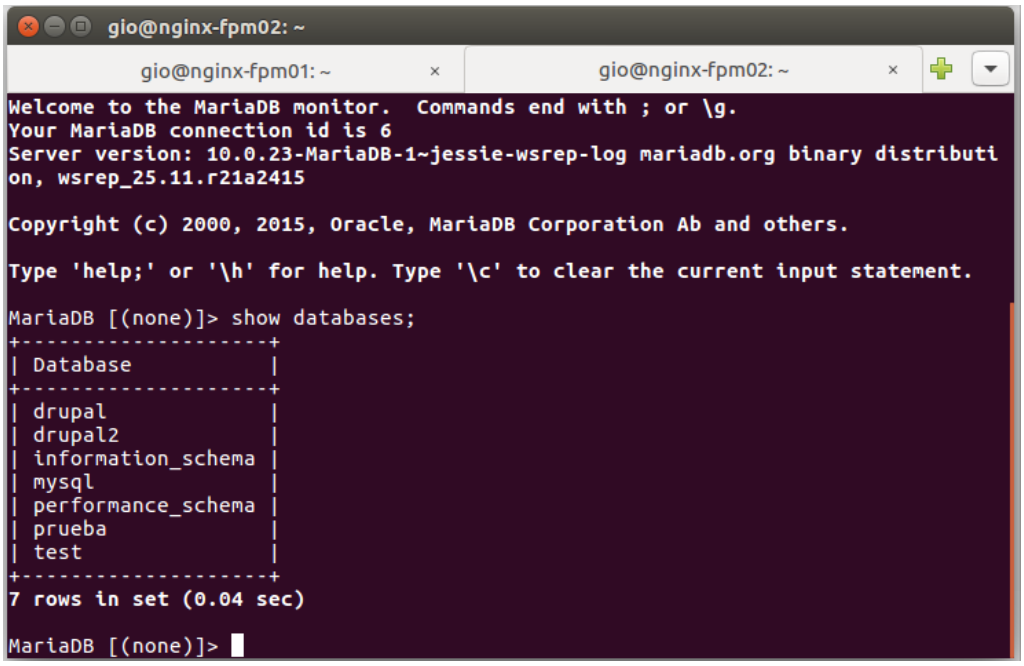


Figura 55: Consulta SQL nodo 2 (show databases;)

Sentencia SQL nodo 3

Figura 56: Consulta SQL nodo 3 (show databases;)

En resumen, el componente de clúster de datos básicamente se utiliza para replicar la base de datos de la aplicación CMS Drupal, de esta forma los servidores web pueden acceder a los datos de la aplicación de una forma más eficiente.

Las MV's número 2,3 y 4 contienen un clúster de datos junto con los servidores web NGINX-PHP-FPM y las MV's 5,6 y 7 también contienen un clúster de datos junto con los servidores web NGINX-HHVM.

4.5 Despliegue del sistema de optimización y escalabilidad web para una aplicación Drupal 7.43

En esta sección se expondrá el despliegue del sistema de optimización y escalabilidad web para una aplicación Drupal, este despliegue será el punto de convergencia de los tres componentes anteriormente desarrollados, los cuales fueron: Balanceador de carga HTTP, entorno múltiple de servidores, y clúster de datos.

Cada componente fue validado a través de una etapa de verificación, en donde se probó su correcto funcionamiento, esto soporta el uso de cada uno de ellos en esta etapa sin ningún inconveniente.

4.5.1 Generalidades del despliegue final

Inicialmente se realizará la instalación y configuración del aplicativo CMS Drupal en cada uno de los servidores web junto con los archivos y la base de datos del sitio web a servir, para esto es necesarios la utilización del entorno múltiple de servidores y el clúster de datos. Luego se utilizará el componente balanceador de carga para aplicar balanceo al sitio web alojado en el aplicativo Drupal.

La aplicación CMS Drupal utiliza una base de datos para almacenar información referente a la administración del núcleo de su sistema; dentro de esa base de datos también se almacenan información referente al desarrollo del sitio web por lo

tanto es de vital importancia mantener un ***clúster de datos*** que replique la información entre cada uno de los nodos.

Los servidores web encargados de servir el contenido manejan dentro de su sistema de archivo todo el código de la aplicación CMS Drupal. Cada vez que se realicen cambios en el código estos deben de ser reflejados en cada uno de los servidores web, por lo tanto, se debe tener un aplicativo de sincronización de archivos, para este despliegue se utilizó **rsync**.

4.5.2 Esquema del despliegue final

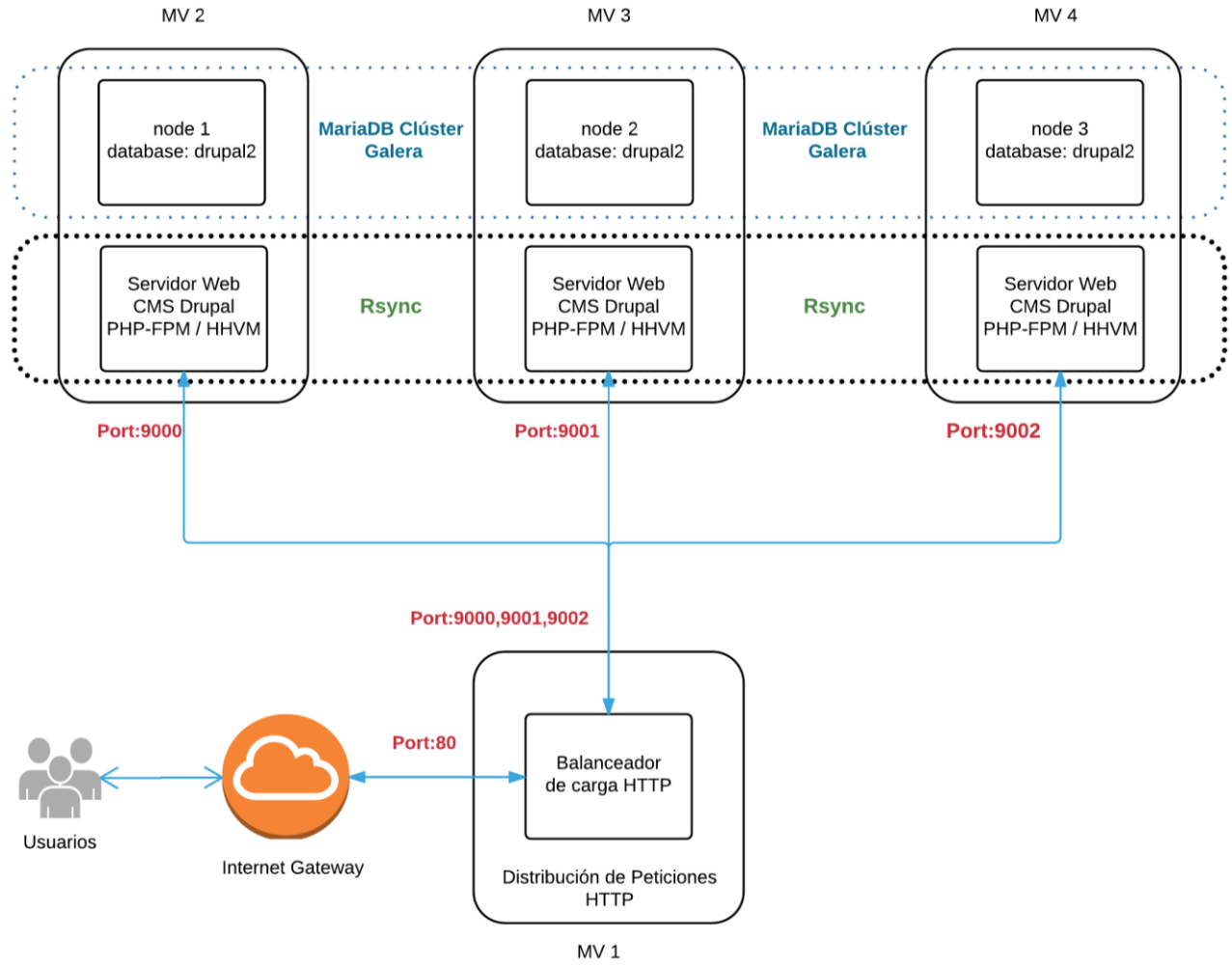


Figura 57: Esquema de despliegue final

4.5.3 Instalación y configuración CMS Drupal

4.5.3.1 Instalación MV número 2

La instalación y configuración del CMS Drupal se realizó mediante el siguiente procedimiento:

Inicialmente este procedimiento se realizó sobre la MV #2, luego se replicó a la MV 3 y 4.

1 Paso - Creación de la base de datos para el CMS Drupal

Para crear la base de datos se ejecutó la siguiente sentencia

```
MariaDB [(none)]> create database drupal2;
```

La base de datos se creó desde el nodo 1 (MV #2) con el nombre de drupal2, y fue replicada automáticamente en los nodos 2 y 3, por medio del clúster MariaDB Galera.

Luego de crear la base de datos, se importó el backup del sitio web; este proceso se realizó por medio del siguiente comando:

```
> mysql -u root -p drupal2 < web.sql
```

2 Paso - Descarga de CMS Drupla 7

El sistema Drupal 7 se descargó desde el siguiente enlace:

https://ftp.drupal.org/files/projects/drupal-7.43.tar.gz

3 Paso - Instalación CMS Drupal 7

Para la instalación de Drupal inicialmente se configuró el servidor web NGINX, puesto que Drupal realiza su instalación por medio del navegador web. Para esto se realizó lo siguiente:

1. Creación de un host virtual NGINX
2. Creación del directorio raíz para los archivos Drupal

Para implementar el ***host virtual*** se creó un archivo llamado “drupal” dentro del directorio /etc/nginx/sites-available/. Este archivo contiene la configuración necesaria para que el servidor web NGINX sirva la aplicación drupal en el puerto TCP 9000. A continuación, se muestra el contenido del archivo “drupal”.

****

```
server {
     listen 9000;

     server_name www.drupal.com.co;

     root /usr/share/nginx/drupal;

     index index.php index.html index.htm;

     location / {
     try_files $uri $uri/ /index.php?$query_string; # For Drupal
>= 7
     }
     location ~ '\.php$|^/update.php' {
        fastcgi_split_path_info ^(.+?\.php)(|/.*)$;

        fastcgi_intercept_errors on;
```

```
        fastcgi_pass unix:/var/run/php5-fpm.sock;
        fastcgi_index index.php;

        include fastcgi.conf;
        fastcgi_param PATH_INFO $fastcgi_path_info;
        fastcgi_param ENV development;

    }

   location ~ ^/sites/.*/files/styles/ { # For Drpal >= 7
        try_files $uri @rewrite;
    }
    location @rewrite {
    rewrite ^/(.*)$ /index.php?q=$1;
    }
    location ~ ^/sites/.*/files/styles/ {
    try_files $uri @rewrite;
    }
}
```

****

En la sección 4.3 Implementación Entorno múltiples de servidores se presenta la explicación del anterior código.

Luego se creó el directorio raíz donde se almacenaron los archivos descargados, el directorio fue creado con el nombre “drupal”.

Ruta: /usr/share/nginx/drupal/

La *figura 58* muestra el directorio raíz junto con los archivos de instalación Drupal.

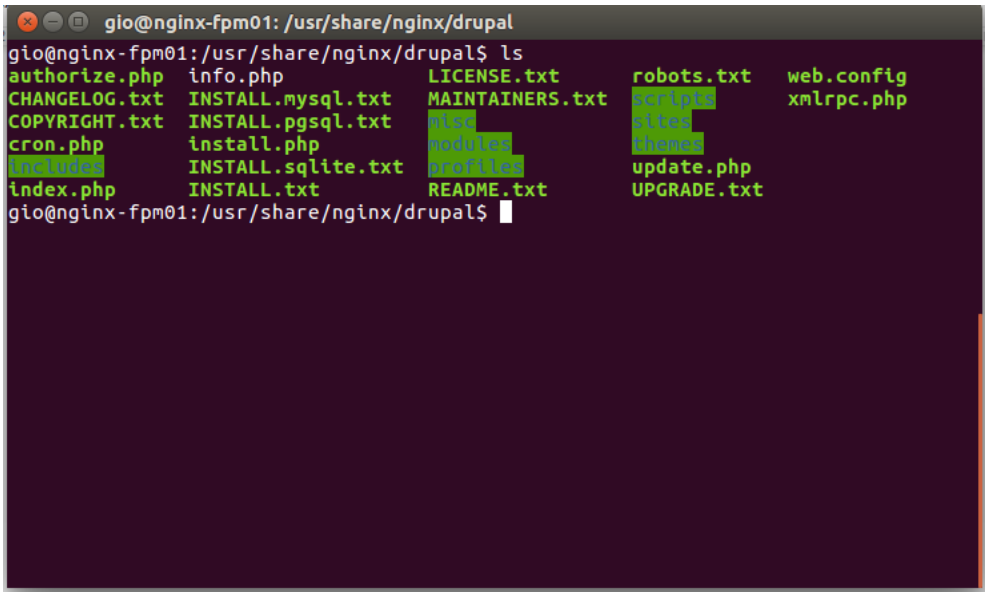


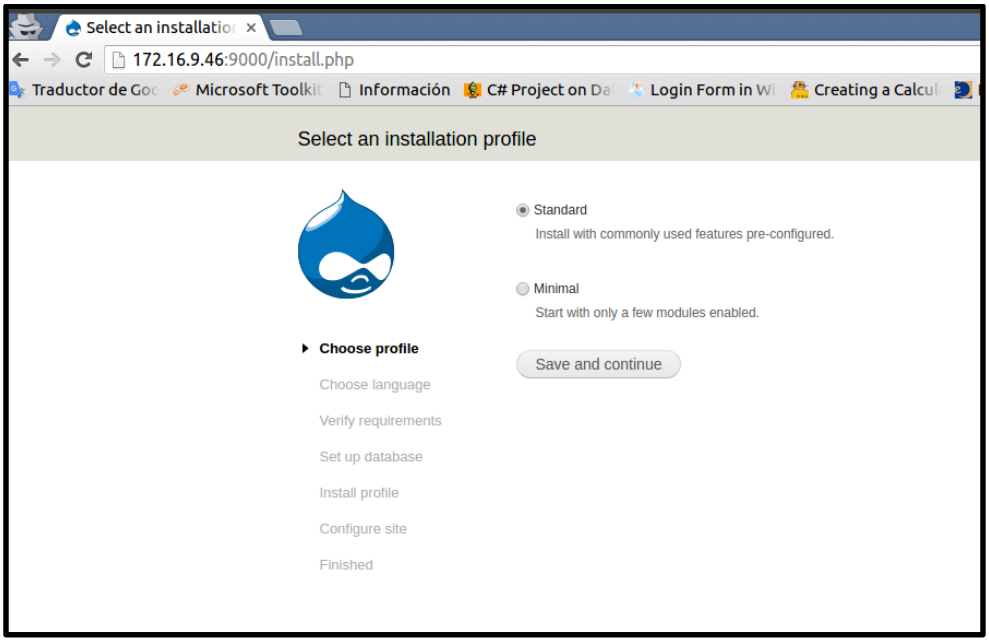
Figura 58: Archivos sistema Drupal 7.43

Teniendo listo el host virtual y el directorio raíz, se ingresa a través del navegador para iniciar el proceso de instalación.

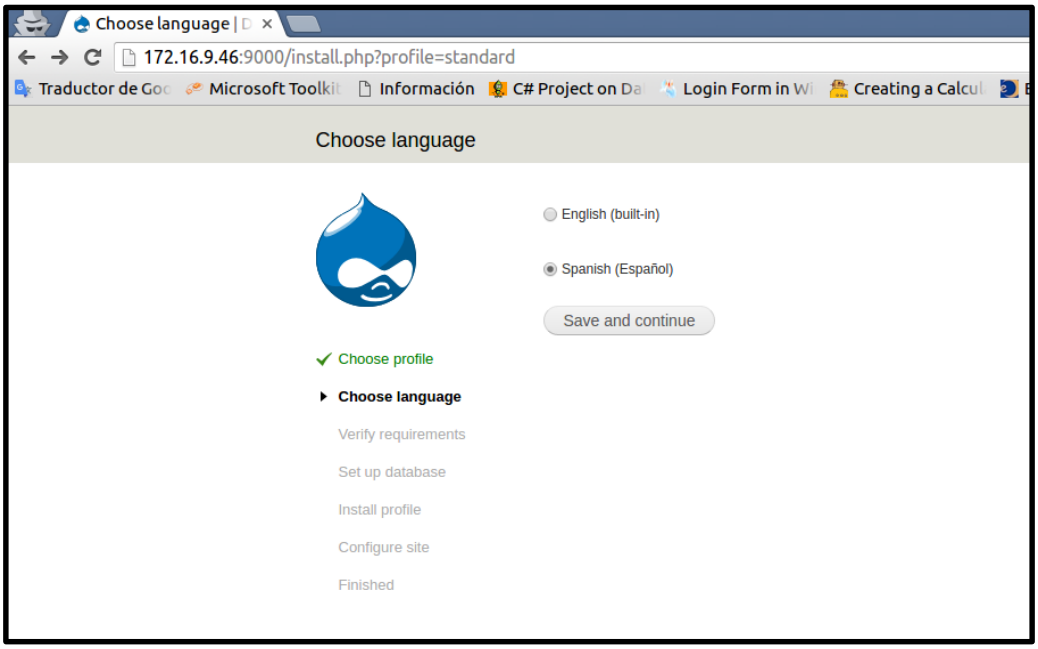
La dirección IP de la MV #2 es 172.16.9.46, el servidor web está configurado bajo el puerto 9000. Por lo tanto, ingresamos a la URL: http://172.16.9.16:9000 para iniciar el proceso de instalación.

El proceso de instalación de la aplicación CMS Drupal se realizó mediante los siguientes pasos:

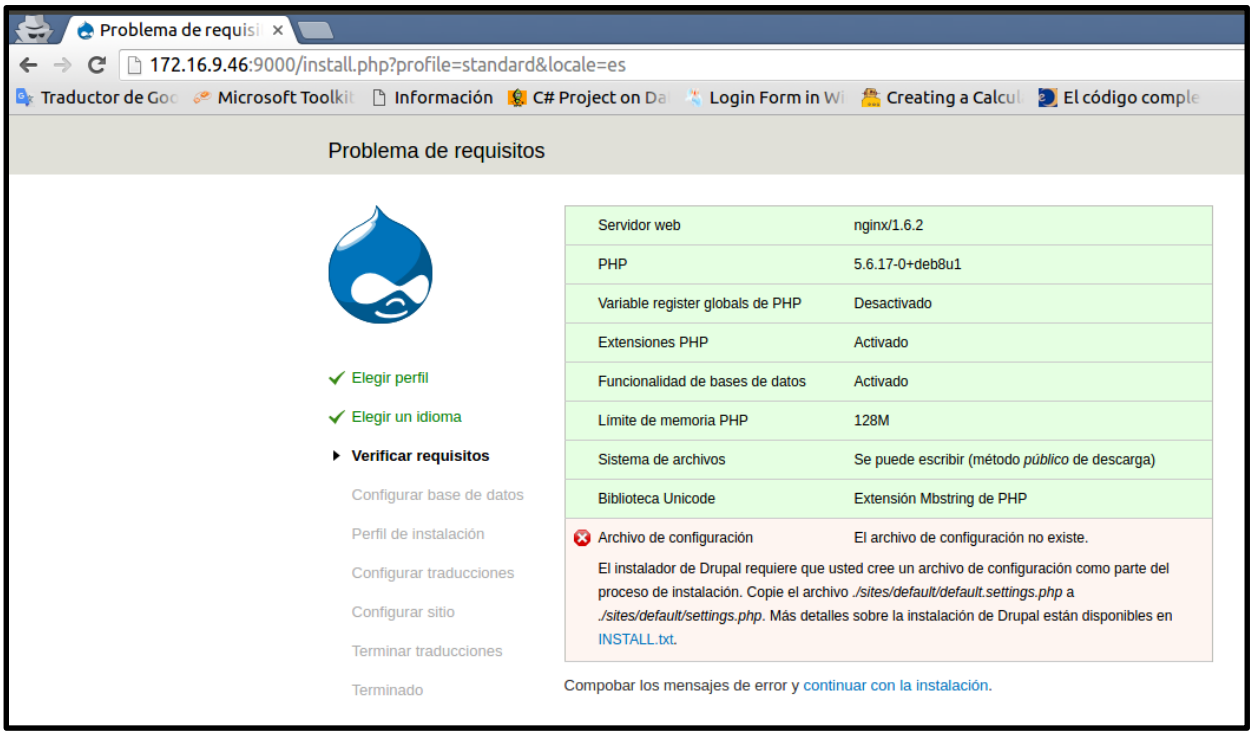
1 Paso - Selección del perfil de instalación



2 Paso - Selección del lenguaje



Paso 3 - Verificación de los requisitos

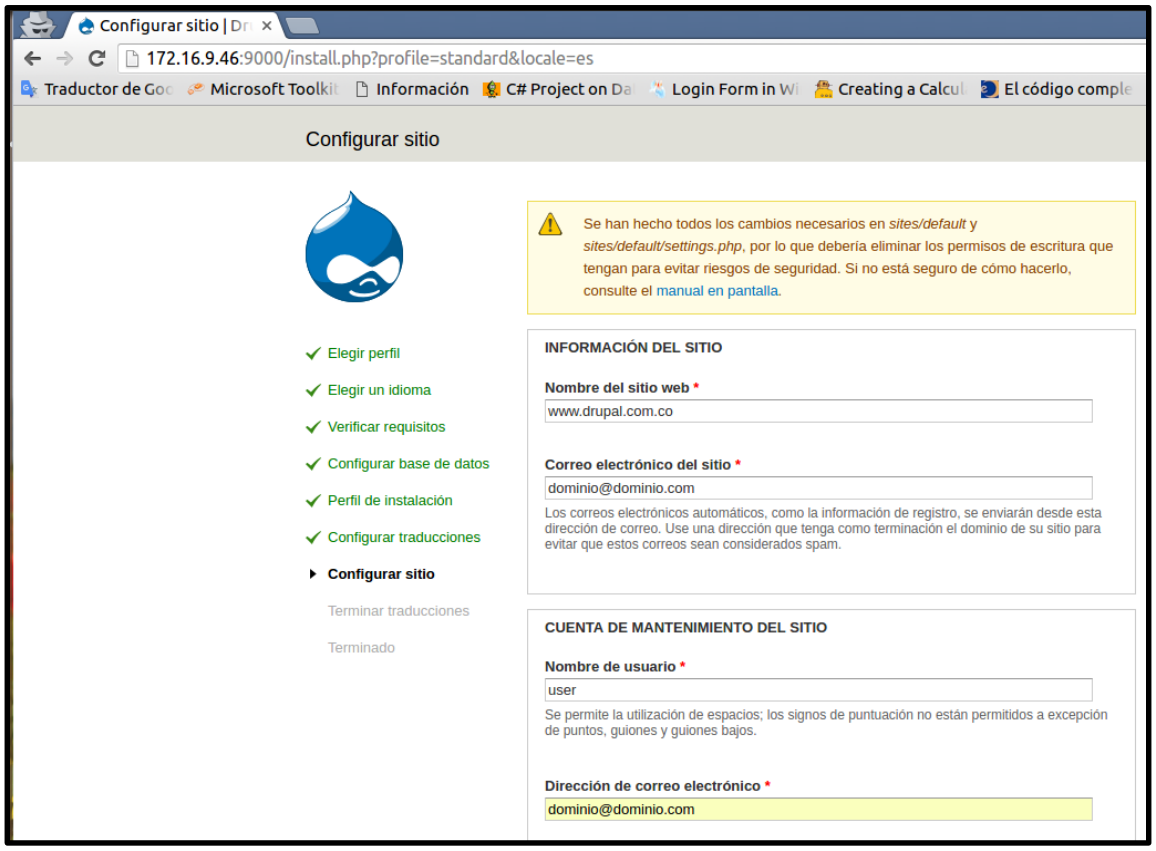


En este paso el archivo de configuración *settings.php* debe ser creado y tener permisos de acceso.

Paso 4 - Configuración de la base de datos

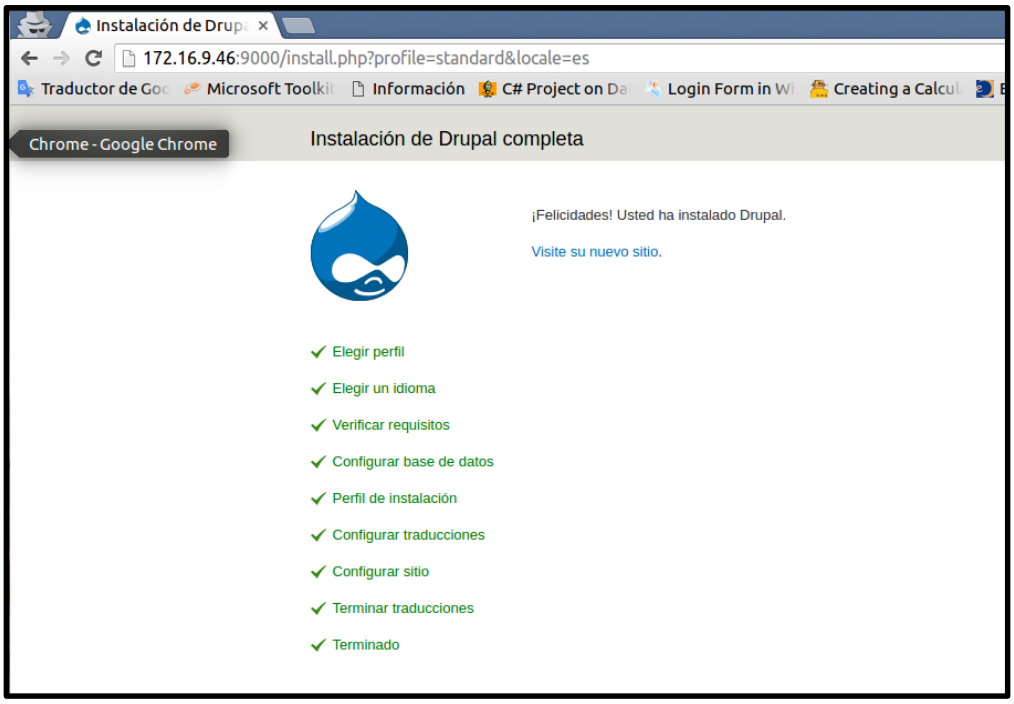
Aquí se establece el nombre de la base de datos, en este caso es “drupal2”, también se establece el nombre de usuario y la contraseña del motor de la base de datos.

Paso 4 - Configuración del sitio



En este paso se establece el usuario y contraseña del sistema Drupal, junto con los datos de la información del sitio web.

Paso 4 - Finalización de la instalación



Luego de instalar el sistema base Drupal se realizó la importación de los archivos del sitio web, para esto, se copió dentro del directorio raíz de drupal en la carpeta /site, todos los archivos relacionados con el sitio web, tales como módulos, temas, librerías, etc.

Finalmente ingresando a la URL http://172.16.9.46:9000, verificamos el funcionamiento del sitio web. La imagen x evidencia lo mencionado anteriormente.

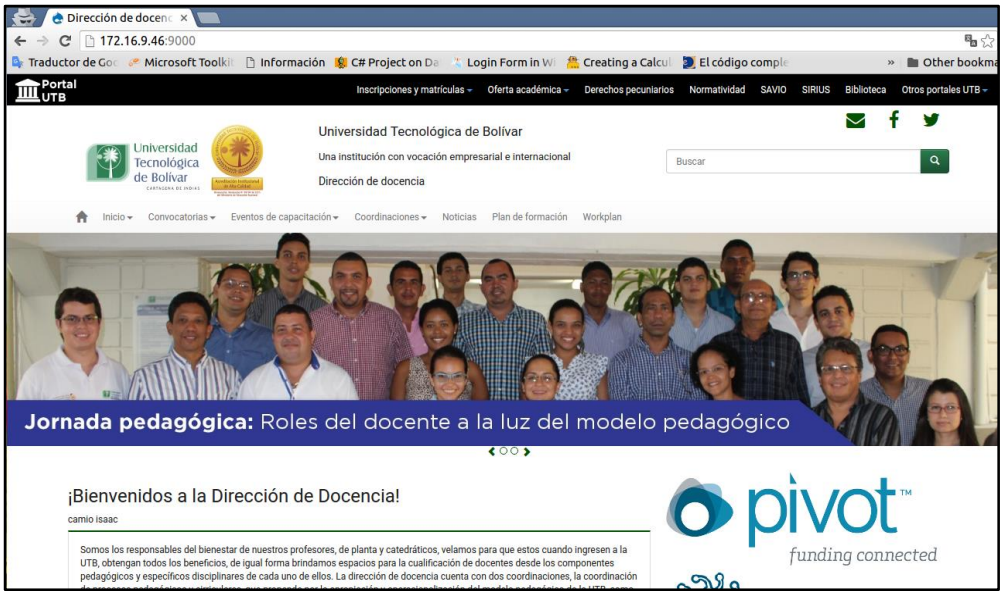


Figura 59: Verificación del funcionamiento del sitio web sobre la MV #2

4.5.3.2 Replicación de instalación MV número 3

Luego de instalar el sistema Drupal junto con el sitio web de prueba sobre la MV número 2, se replicó la configuración de esta dentro de las MV número 3.

Para la replicación de la configuración se realizó el siguiente proceso; cabe mencionar que la base de datos ya fue replicada sobre la MV#3 mediante el clúster de datos.

1 Paso - Creación del host virtual

Dentro del directorio /etc/nginx/sites-available/ se creó de igual forma un archivo llamado “drupal”. Este archivo contiene el mismo código que el archivo de la MV #2, únicamente cambia el valor de la directiva *listen* de 9000 a 9001.

```
**********************************************************************************************

server {
     listen 9001;

     server_name www.drupal.com.co;
     root /usr/share/nginx/drupal;
     index index.php index.html index.htm;

     location / {
     try_files $uri $uri/ /index.php?$query_string; # For Drupal
>= 7
     }
     location ~ '\.php$|^/update.php' {
        fastcgi_split_path_info ^(.+?\.php)(|/.*)$;
        fastcgi_intercept_errors on;
        fastcgi_pass unix:/var/run/php5-fpm.sock;
        fastcgi_index index.php;

        include fastcgi.conf;
        fastcgi_param PATH_INFO $fastcgi_path_info;
        fastcgi_param ENV development;

    }

   location ~ ^/sites/.*/files/styles/ { # For Drpal >= 7
```

```
        try_files $uri @rewrite;
    }
    location @rewrite {
    rewrite ^/(.*)$ /index.php?q=$1;
    }
    location ~ ^/sites/.*/files/styles/ {
    try_files $uri @rewrite;
    }
}
```

****

2 Paso - Replicación del directorio raíz Drupal

Para la replicación del directorio raíz se utilizó la herramienta rsync, esta herramienta está incluida por defecto dentro del sistema operativo Debian 8.

Para transferir los archivos entre las dos máquinas inicialmente se configuró el acceso SSH.

Para realizar la replicación del directorio se utilizó el siguiente comando:

```
sudo rsync -a /usr/share/nginx/drupal/
gio@172.16.9.47:/usr/share/nginx/drupal/
```

El comando se ejecutó desde la MV#2 hacia la MV#3 la cual tiene la dirección IP 172.16.9.47. Como se puede apreciar el directorio replicado fue el directorio raíz /usr/share/nginx/drupal/ donde se alojan todos los archivos de instalación y configuración del sitio web y el sistema Drupal 7.43.

La *figura 60* ilustra los archivos replicados dentro de la MV #3.

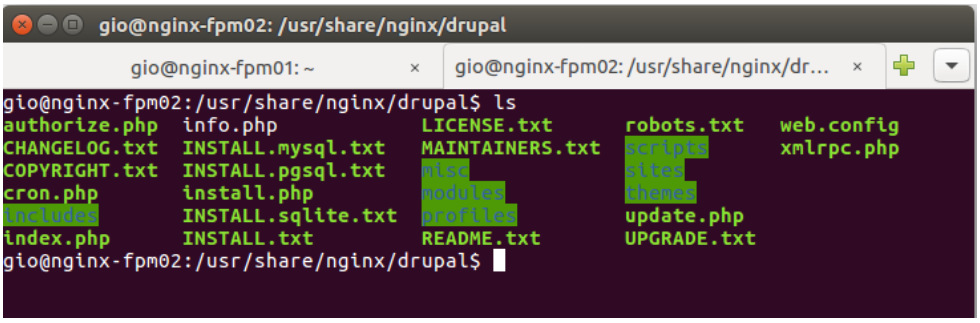


Figura 60: Directorio raíz Drupal MV #3

Luego de tener el host virtual y el directorio raíz replicado se puede verificar el funcionamiento del sitio web, para esto se ingresa a la URL http://172.16.9.47:9001.

La *figura 61* ilustra el correcto funcionamiento del sitio web sobre la MV número 3.

Figura 61: Verificación del funcionamiento del sitio web sobre la MV #3

4.5.3.3 Replicación de instalación mv número 4

La replicación de la instalación sobre la MV #4 es similar a la replicación sobre la MV #3, este proceso también cuenta con la creación de un host virtual y la replicación del directorio raíz del sistema Drupal.

En la creación del host virtual únicamente se cambió el valor de la directiva listen, y se estableció en el puerto 9001.

La replicación del directorio raíz se realizó mediante el siguiente comando:

```
$ sudo rsync -a /usr/share/nginx/drupal/
gio@172.16.9.48:/usr/share/nginx/drupal/
```

En este caso la ruta de destino es la MV #4 la cual tiene la dirección IP 172.16.9.48.

La *figura 62* ilustra los archivos replicados dentro de la MV #4.

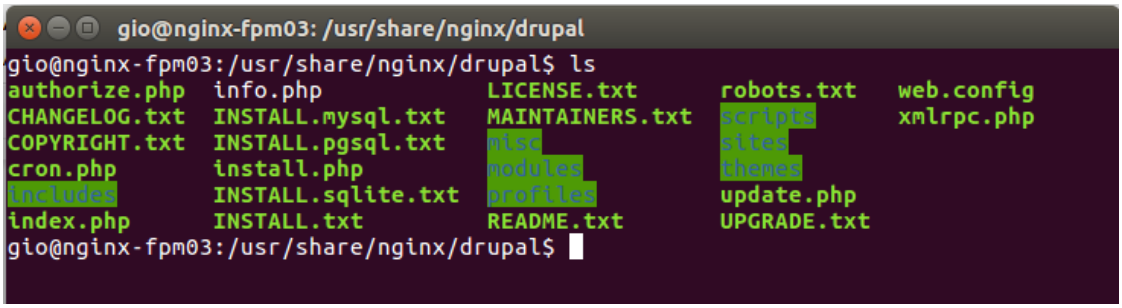


Figura 62: Archivos replicados MV #4

Para la verificación del sitio web sobre la MV #4 ingresamos a la URL http://172.16.9.48:9002.

La figura 63 ilustra el correcto funcionamiento del sitio web sobre la MV #4.

Figura 63: Verificación del funcionamiento del sitio web sobre la MV #4

4.5.4 Balanceador de carga http para CMS Drupal

Luego de la instalación y configuración de Drupal sobre las MV's número 2,3 y 4, se integró el balanceado de carga HTTP el cual se ejecuta sobre la MV número 1.

Para integrar el balanceador de carga se realizó el siguiente proceso:

1 Paso - Requerimientos servidores de back-end

Para realizar la configuración en el balanceador de carga se necesitan las direcciones IP y los puertos TCP configurados en las MV's número 2,3 y 4. La tabla 6 contiene los datos de cada una de la MV's.

Id	Dirección IP	Puerto TCP
MV #2	172.16.9.46	9000
MV #3	172.16.9.47	9001
MV #4	172.16.9.48	9002

Tabla 6: Datos de MV's para la configuración

2 Paso - Configuración del balanceador de carga

Dentro del directorio /etc/nginx/sites-available/ ubicado dentro de la MV #1, se creó un archivo llamado “drupal”.

Este archivo contiene el siguiente código:

```
##############################################################################################
http {
    upstream appDrupal {
        server 172.16.9.46:9000 max_fails=3 fail_timeout=30s;
        server 172.16.9.47:9001 max_fails=3 fail_timeout=30s;
        server 172.16.9.48:9002 max_fails=3 fail_timeout=30s;
    }
    server {
        listen 80;
        access_log /var/log/nginx/drupal-access.log;
        error_log  /var/log/nginx/drupal-error.log error;
```

```
    location / {
          include proxy_params;
          proxy_set_header X-Forwarded-Port $server_port;

          proxy_pass http://appDrupal;
          proxy_redirect off;

          proxy_http_version 1.1;
          proxy_set_header Upgrade $http_upgrade;
          proxy_set_header Connection "upgrade";
     }
  }
```

****

Para la configuración de este archivo se definieron las direcciones IP y los puertos TCP de las MV's número 2,3 y 4 dentro del bloque ***upstream*** (Texto color azul). La explicación de la configuración del anterior archivo fue expuesta en la sección 4.2, Implementación balanceador de carga HTTP.

2 Paso - Despliegue del balanceador de carga para CMS Drupal

Como se indica en el esquema de despliegue final, el balanceador de carga recibe peticiones HTTP por el puerto 80, y luego las re-direcciona hacia los servidores back-end por medio de los puertos 9000,9001,9002.

Los servidores back-end luego de recibir la petición proveniente del balanceador generan la respuesta HTTP y la devuelven al mismo, este finalmente entrega la respuesta al usuario que la solicitó; si este proceso es correcto el usuario recibe el un código HTTP 200.

Para verificar el acceso por medio del balanceador al sitio web alojado en los servidores back-end, se realizó una petición GET utilizando la herramienta **curl**, la cual permite verificar el código de respuesta HTTP; adicional a esto el sistema Drupal incluye un meta tag indicando que el contenido generado es de su propiedad.

Meta tag drupal

```
<meta name="Generator" content="Drupal 7 (http://drupal.org)" />
```

Prueba CURL

$ curl -I 172.16.9.49/eventos

La dirección 172.16.9.49 es la IP del balanceador de carga, a continuación, la *figura 64* ilustra la respuesta recibida.

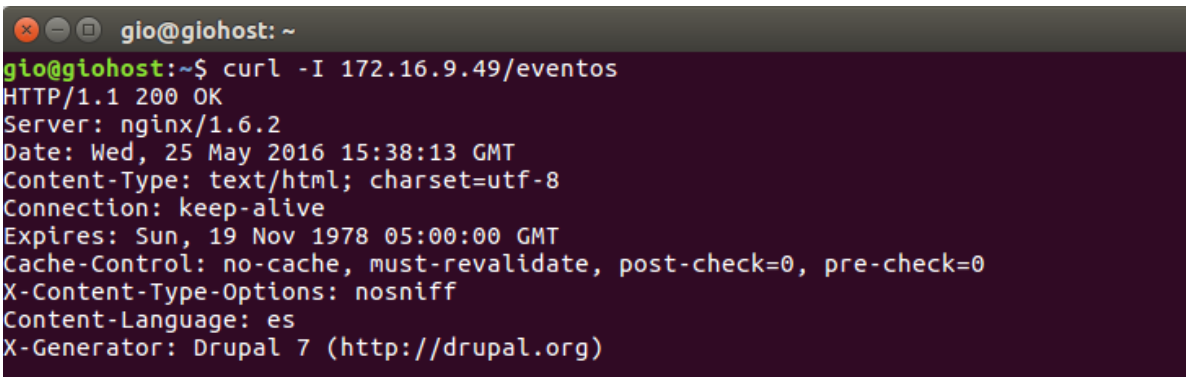


Figura 64: Resultado de la prueba realizada con la herramienta curl para el balanceador de carga.

El primer parámetro de la respuesta contiene el código HTTP 200, esto indica que la petición fue correcta. Por lo tanto, la comunicación entre el balanceador y los servidores back-end funciona correctamente.

El último parámetro de la respuesta curl:

X-Generator: Drupal 7 (http://drupal.org)

Indica que el archivo HTML recibido en la respuesta fue generado por un sistema Drupal, lo cual verifica que la respuesta fue generada por los servidores back-end y no directamente dentro del balanceador de carga.

Prueba del navegador chrome

Se realizó una petición GET al balanceador de carga mediante un navegador web utilizando la URL http://172.16.9.49, la *figura 65* ilustra el resultado.

Figura 65: Resultado de la prueba por medio del navegador web google chrome para el balanceador de carga.

Con las anteriores pruebas se verificó el acceso al sitio web de prueba alojado en los servidores back-end, a través del balanceador de carga. En este despliegue no se realizaron pruebas para verificar el balanceo de carga puesto que fueron realizadas en la sección 4.2 - Implementación balanceador de carga HTTP.

5 CAPÍTULO V: PRUEBA Y VALIDACIÓN DEL SISTEMA DE OPTIMIZACIÓN Y ESCALABILIDAD WEB

Para realizar las pruebas y validaciones del sistema se definió un conjunto de escenarios, donde se analizaron los diferentes factores que intervienen en el rendimiento de una plataforma web. Con ayuda de la herramienta de testing TSUNG se definió un conjunto de variables, las cuales determinan de una forma cuantitativa el rendimiento de un sistema web, además esta herramienta es la encargada de indicar la carga de prueba.

A continuación, la tabla 7 define el conjunto de variables analizadas en cada uno de los escenarios.

Tabla de variables de rendimiento

Tiempos de respuesta	Petición	Tiempo de duración de una petición
	Conexión	Tiempo de duración de una única conexión
Rendimiento	Petición	Número de peticiones atendidas por segundo
	Tráfico de red	Kbits enviados/recibidos por segundo
Usuarios del sistema	Usuarios y conexiones simultáneas	Número simultáneo de usuarios y conexiones TCP

Tabla 7: Variables de rendimiento analizadas en cada escenario

5.1 Escenarios de prueba

Para cada escenario se diseñó un modelo de prueba en donde se especificó los componentes de hardware y de software utilizados. En cada escenario se describe los aspectos técnicos de cada componente, tales como: tipo de servidor web, tipo de motor de base de datos, cantidad de recursos de hardware, procesador de código, etc.

Los escenarios de pruebas fueron configurados teniendo en cuenta la arquitectura de un sistema web real, para este caso se tuvo como referencia la plataforma web de la Universidad Tecnológica de Bolívar.

5.1.1 Escenario de prueba real - Plataforma web Universidad Tecnológica de Bolívar

Este escenario de prueba fue utilizado como marco referencial para los demás escenarios, puesto que con los resultados obtenidos bajo esta plataforma real se pudo recrear diversos escenarios teóricos, con el fin de asemejarse a las condiciones reales de una plataforma web en producción. La infraestructura física de los servidores de la universidad se ubica dentro de la plataforma cloud de la empresa Rackspace Inc.

Modelo del escenario

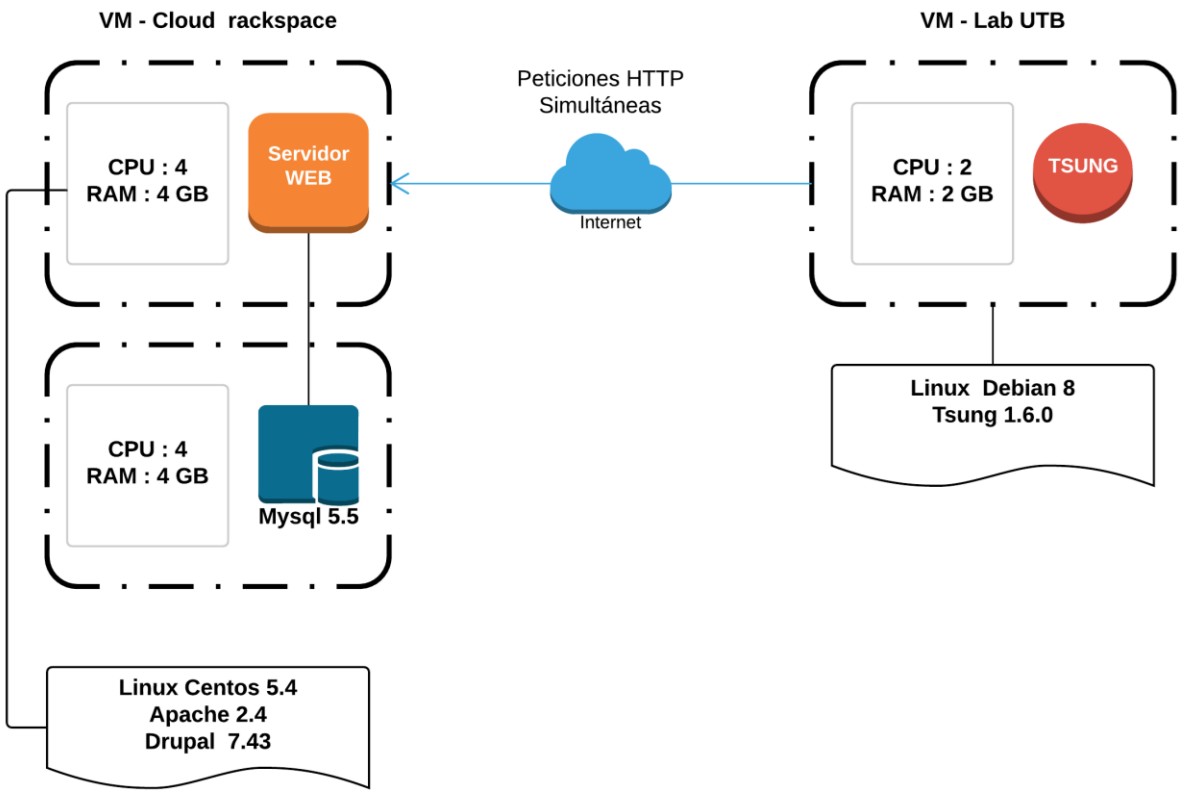


Figura 66: Modelo del escenario de prueba real

Este primer escenario de prueba es utilizado para medir el rendimiento actual de la plataforma web de la universidad, esta maneja un sistema Drupal 7 para la administración y desarrollo del sitio web.

Magnitud de la carga HTTP

Para definir la carga HTTP con la cual se realizaron las pruebas de rendimiento, primero se realizaron pequeñas pruebas aleatorias con el fin de determinar un conjunto de carga ideal para poner a prueba el rendimiento de los servidores.

Finalmente se definieron 3 tipos de prueba, la tabla 8 describe cada una de las pruebas.

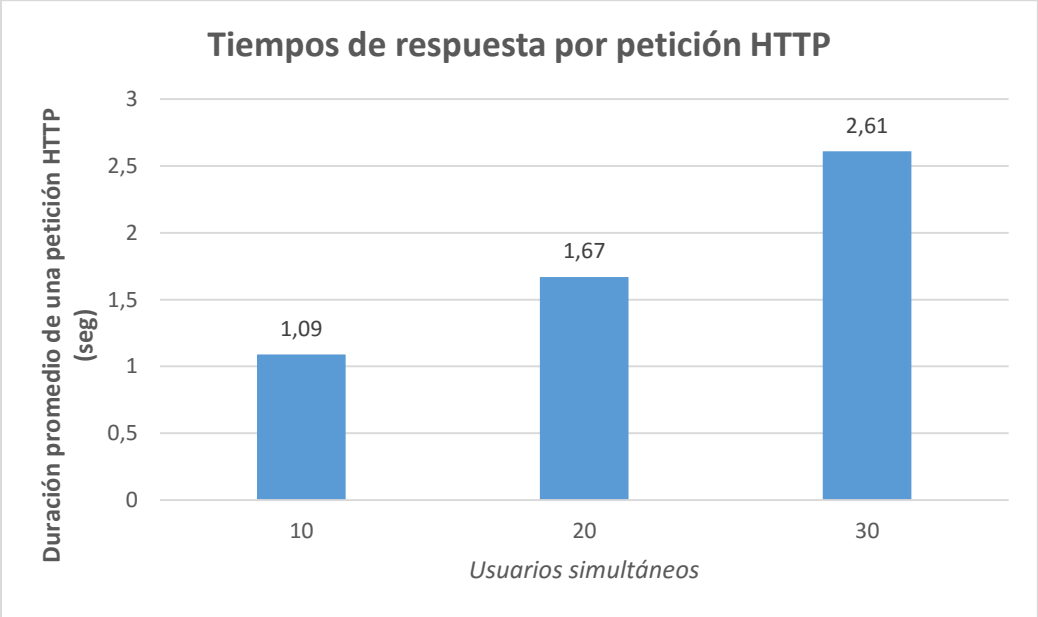
Tipo de prueba	Usuarios simultáneos	Peticiones por usuario
Tipo 1	10	200
Tipo 2	20	200
Tipo 3	30	200

Tabla 8:Tipos de prueba de rendimiento

Resultados de las pruebas

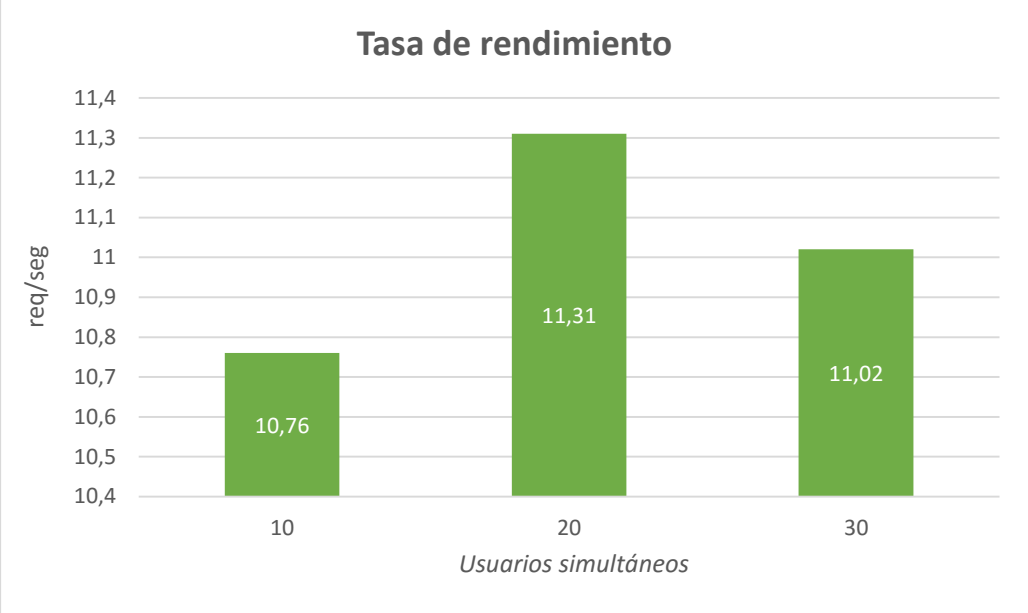
Las pruebas se realizaron en horas de bajo tráfico, en donde el promedio de sesiones iniciadas no era mayor a 4.

Resultado 1



Gráfica 3: Tiempos de respuesta del escenario de prueba real

Resultado 2



Gráfica 4: Tasa de rendimiento del escenario de prueba real

La prueba tipo 3 la cual maneja 30 usuarios simultáneos, será la prueba realizada dentro de los siguientes escenarios de prueba, esto con el fin de analizar si existe o no una mejora en cuanto a las variables de rendimiento. Esta prueba tuvo como resultado tiempos de respuesta de 2.61 segundos por petición y una tasa de rendimiento de 11.02 *req/seg.*

5.1.2 Escenario de prueba 1 – Replica local de la Plataforma web Universidad Tecnológica de Bolívar

En este escenario no se realizaron las pruebas sobre los servidores reales de la plataforma cloud de rackspace Inc; por el contrario, se utilizaron los servidores del laboratorio de la Universidad Tecnológica de Bolívar, teniendo una configuración similar a los servidores reales. Esto con el fin de estimar parámetros de rendimiento de una plataforma real, los cuales sirvan como base de comparación para los siguientes escenarios.

Como se mencionó en el escenario anterior, la prueba utilizada en este escenario es la prueba tipo 3, la figura x ilustra el modelo de la prueba.

Modelo de la prueba

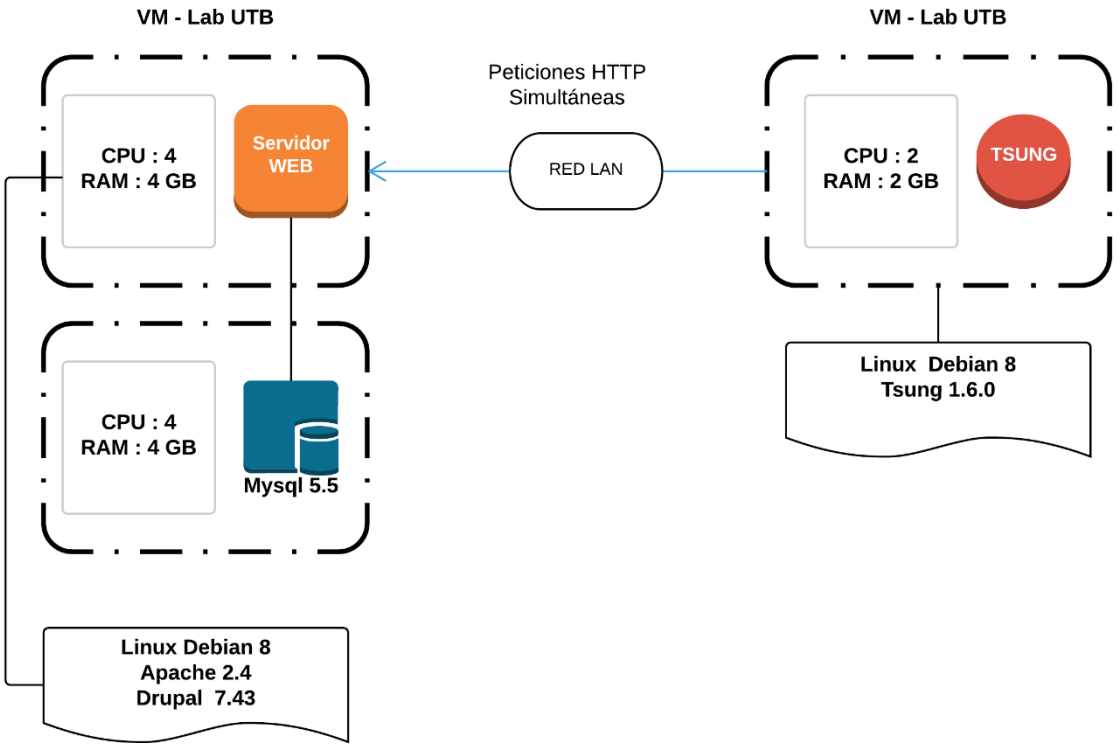
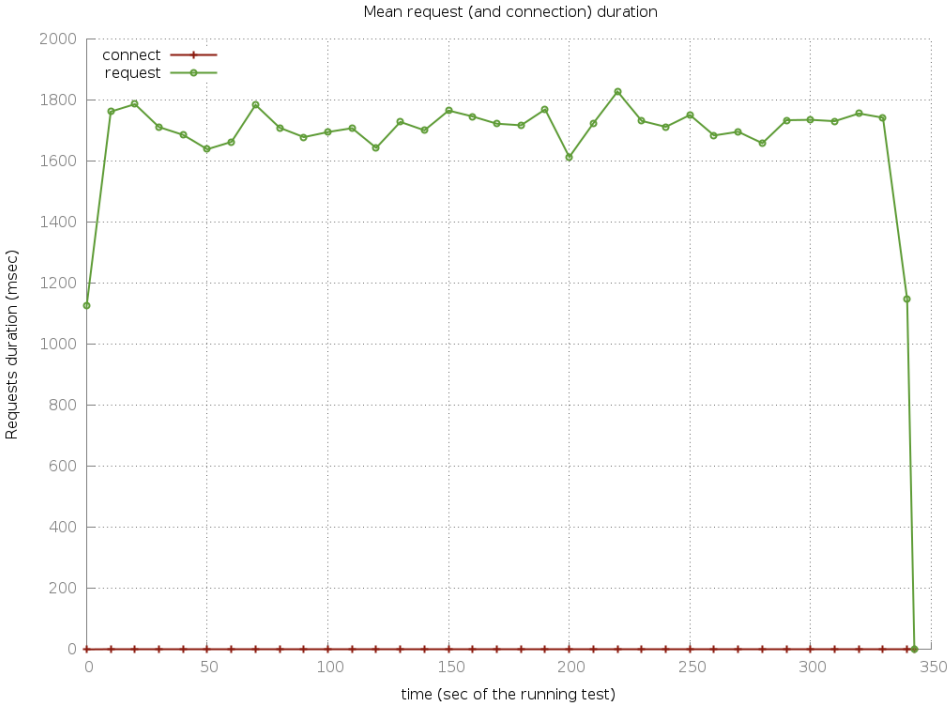


Figura 67: Modelo del escenario de prueba 1

Resultados de las pruebas

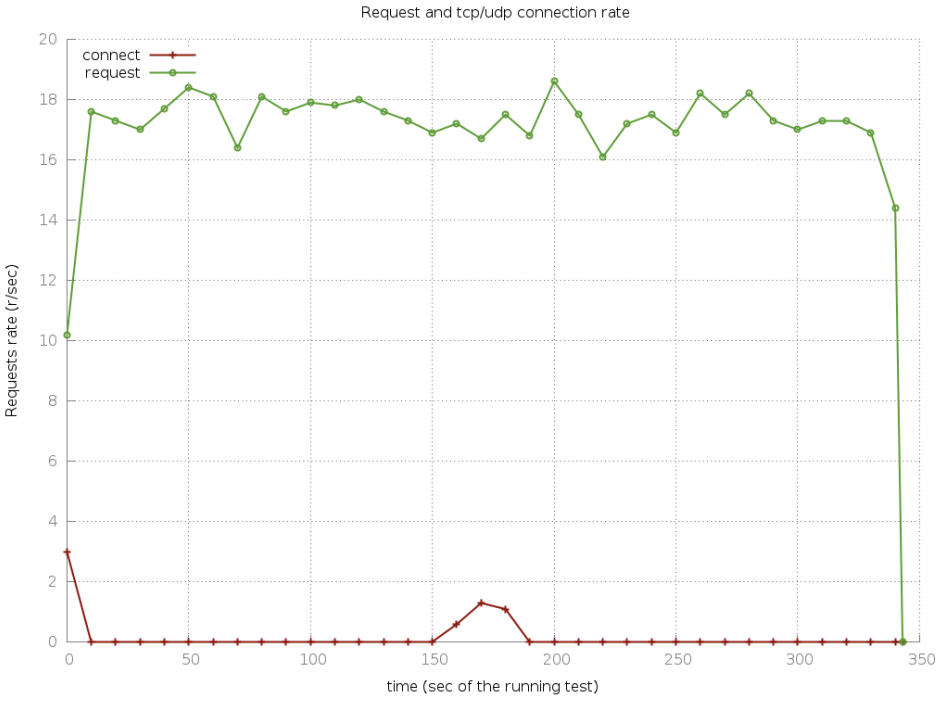
Las siguientes gráficas describen en detalle los resultados de la prueba, cada grafica expone todas las peticiones realizadas durante la prueba.

Tiempos de respuesta



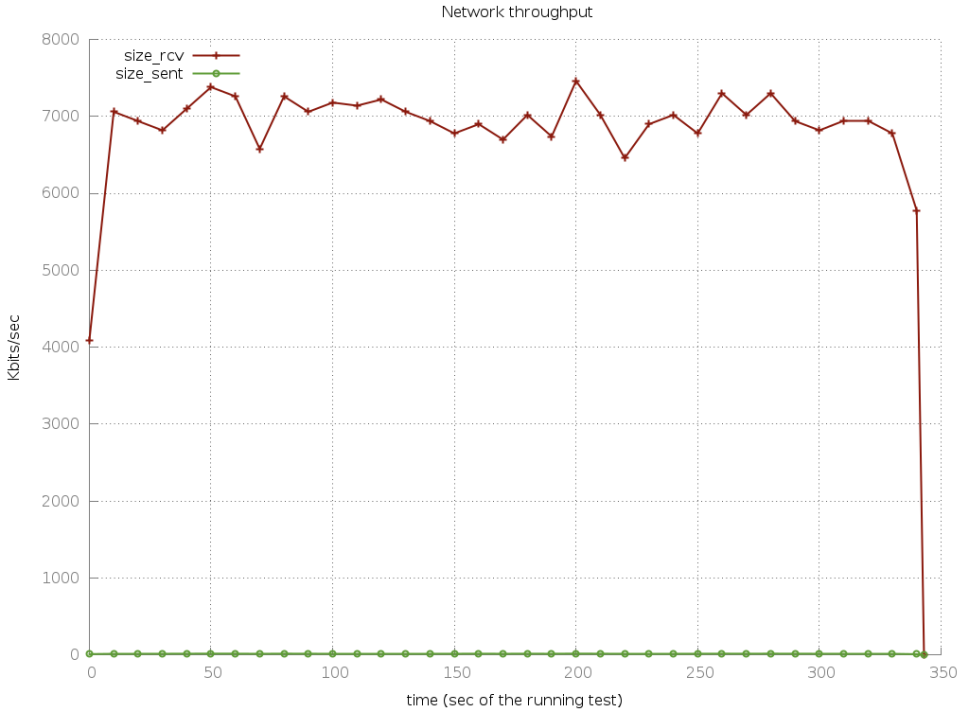
Gráfica 5: Tiempos de respuesta escenario de prueba 1

Tasa de rendimiento



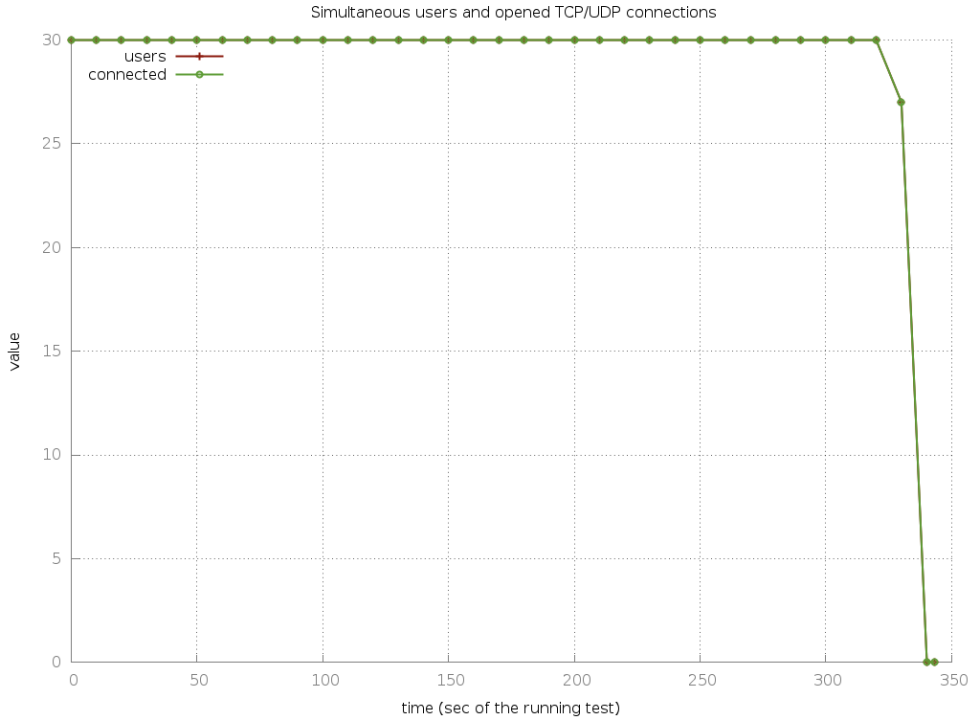
Gráfica 6: Tasa de rendimiento escenario de prueba 1

Tráfico de red



Gráfica 7: Tráfico de red escenario de prueba 1

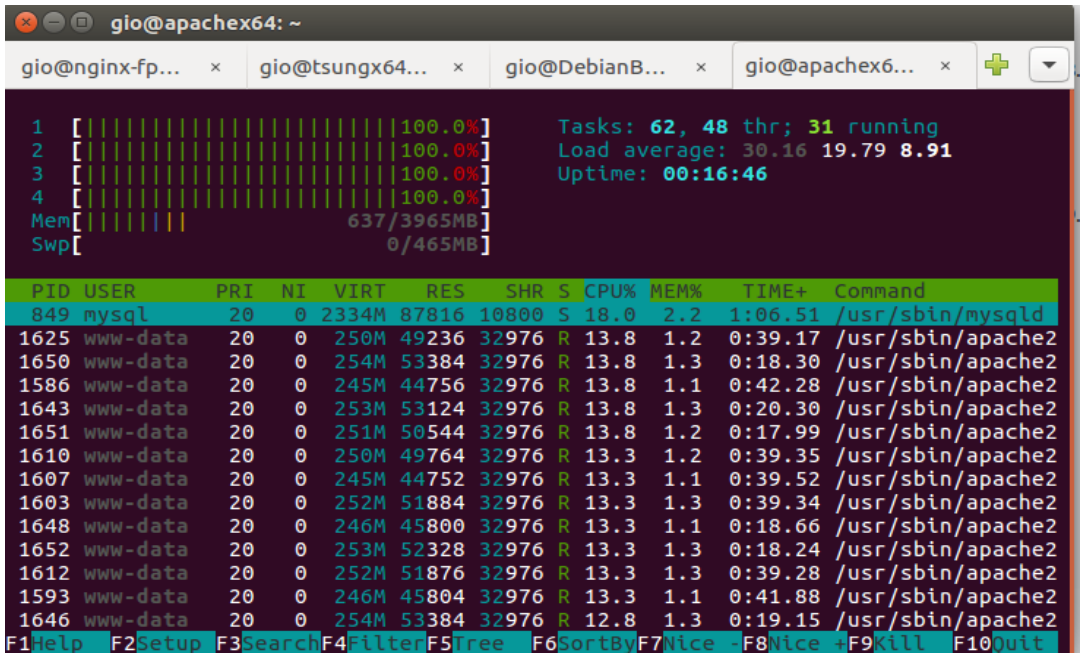
Usuarios simultáneos



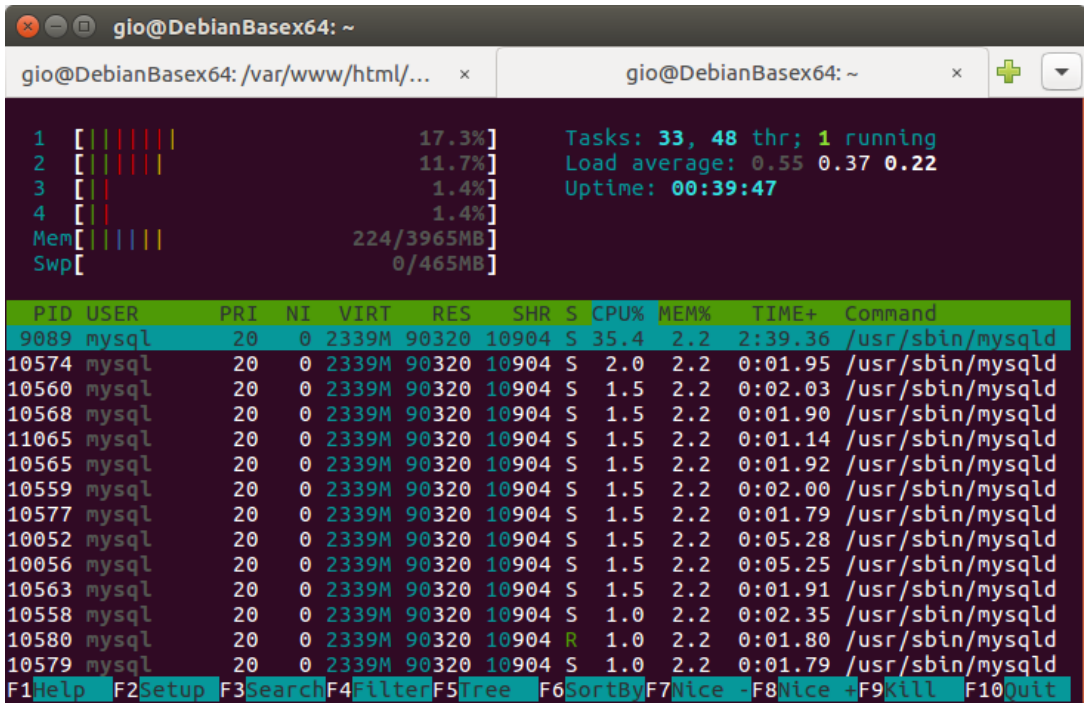
Gráfica 8: Usuarios simultáneos escenario de prueba 1

Utilización de CPU y RAM

Captura 1 – Servidor web



Captura 2 – Servidor de base de datos



Estadísticas generales

Nombre	Tasa más alta	Tasa promedio	Media
Conexión	1.3 /sec	0.09 / sec	0.660 msec
Request	18.6 /sec	16.85 / sec	1.69 sec
Sesión	2.7 /sec	1.00 / sec	5mn 39sec

Rendimiento de red

Nombre	Tasa	Total
Tamaño recibido	7.28 Mbits/sec	293.73 MB
Tamaño enviado	19.63 Kbits/sec	791.41 KB

Código de retorno HTTP

Código	Tasa más alta	Tasa promedio	Total
200	18.3 / sec	16.84 / sec	6000

5.1.3 Escenario de prueba 2 – Implementación de nginx web server y PHP-FPM

Para esta prueba se realizaron cambios de software, se cambió el servidor web apache por el servidor nginx y se utilizó como aplicativo adicional el procesador de código PHP-FPM, también se cambió el motor de base de datos MySQL por el motor MariaDB. El servidor web y el motor de base de datos se instalaron la misma máquina virtual.

Modelo de la prueba

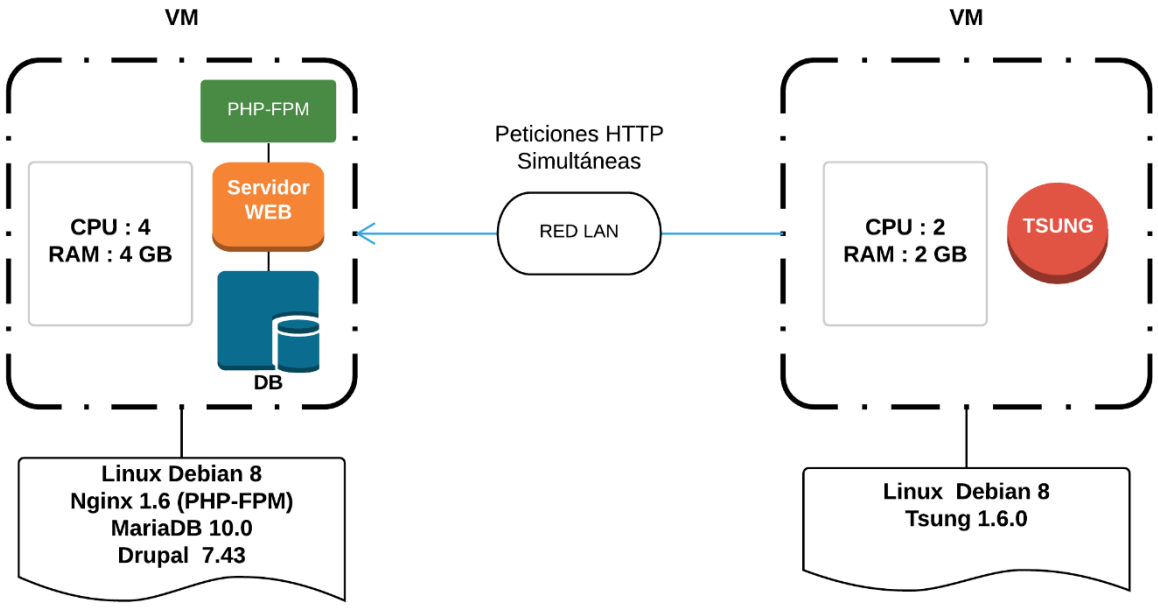
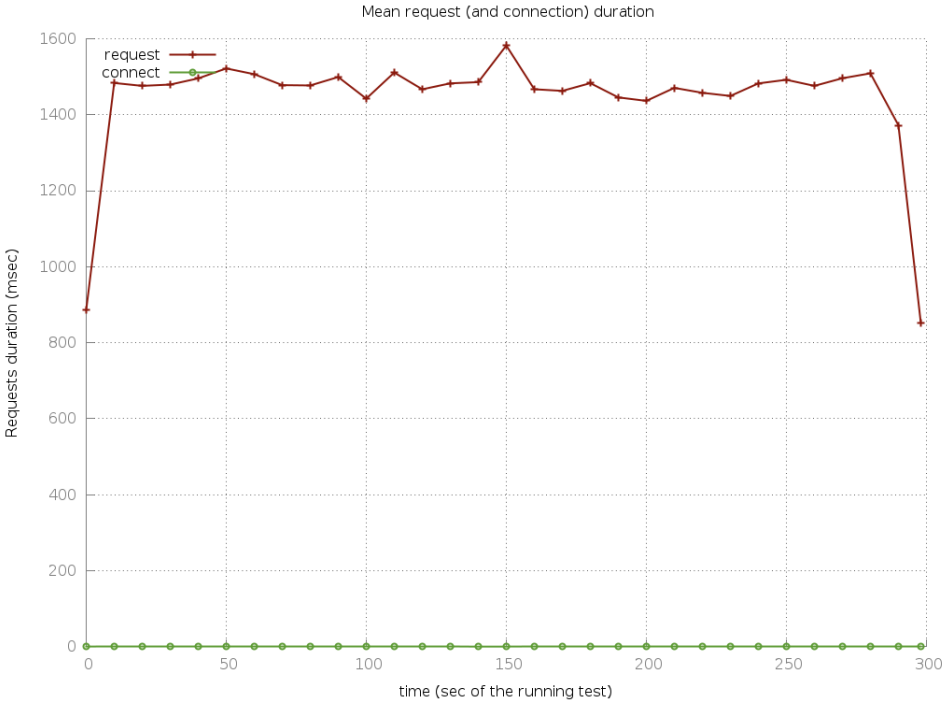


Figura 68: Modelo del escenario de prueba 2

Resultados de las pruebas

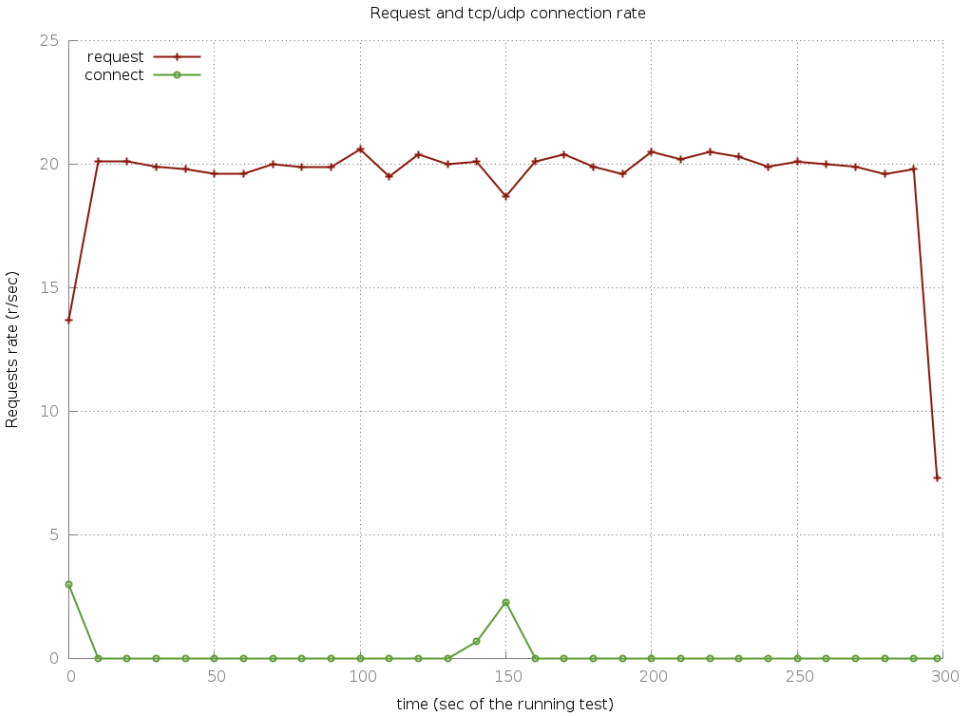
Las siguientes gráficas ilustran los resultados obtenidos.

Tiempos de respuesta



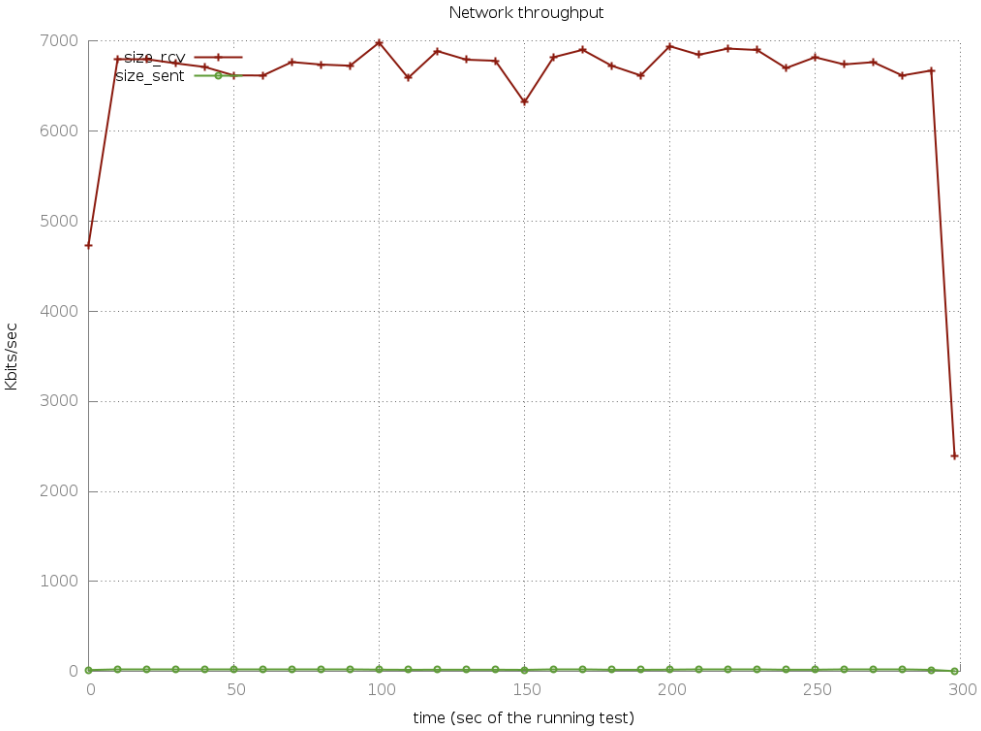
Gráfica 9: Tiempos de respuesta escenario de prueba 2

Tasa de rendimiento



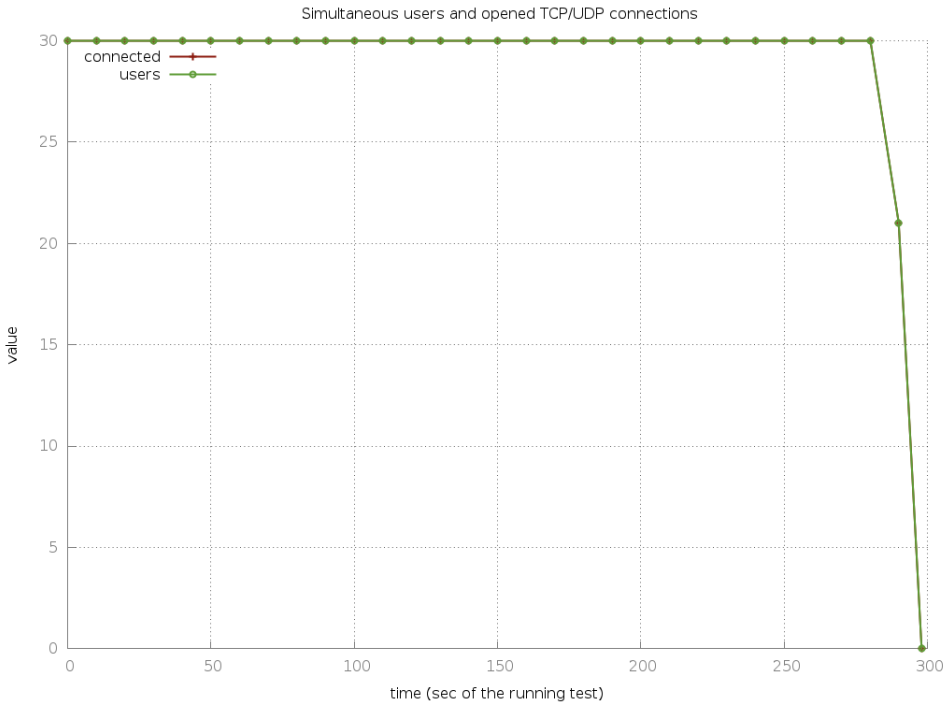
Gráfica 10: Tasa de rendimiento escenario de prueba 2

Tráfico de red



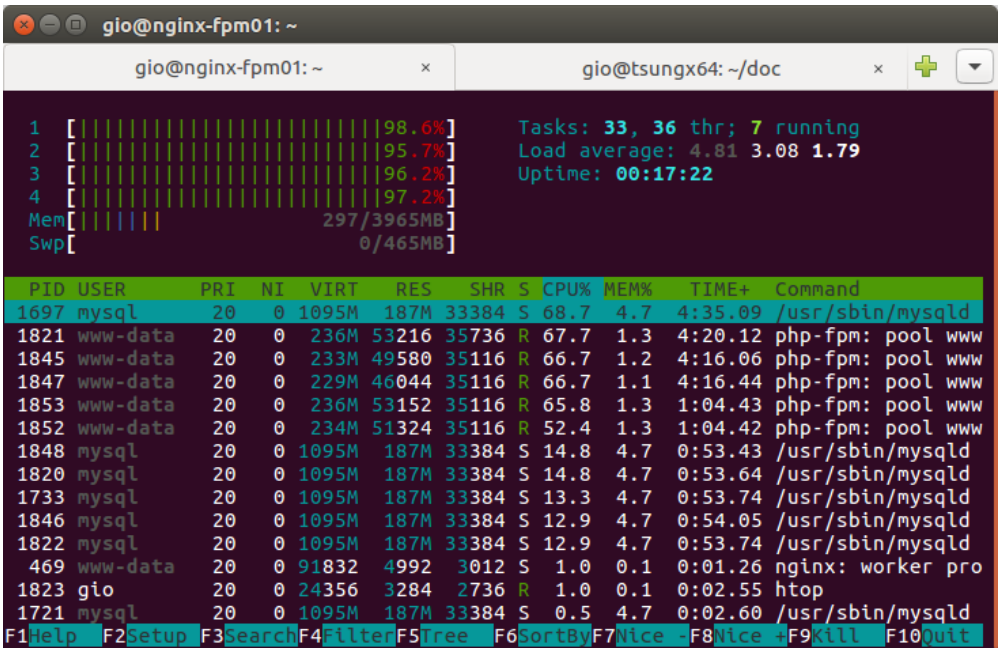
Gráfica 11: Tráfico de red escenario de prueba 2

Usuarios simultáneos



Gráfica 12: Usuarios simultáneos escenario de prueba 2

Utilización de CPU y RAM



Estadísticas generales

Nombre	Tasa más alta	Tasa promedio	Media
Conexión	2.3 /sec	0.10 / sec	0.642 msec
Request	20.6 /sec	19.60 / sec	1.46 sec
Sesión	2.625 /sec	1.00 / sec	4mn 53sec

Rendimiento de red

Nombre	Tasa	Total

Tamaño recibido	6.82 Mbits/sec	247.66 MB
Tamaño enviado	22.38 Kbits/sec	814.84 KB

Código de retorno HTTP

Código	Tasa más alta	Tasa promedio	Total
200	20.6 / sec	19.60 / sec	6000

5.1.4 Escenario de prueba 3 – Implementación de nginx web server y HHVM

Pare este escenario únicamente se cambió el procesador de código PHP-FPM por el procesador de código HHVM.

Modelo de la prueba

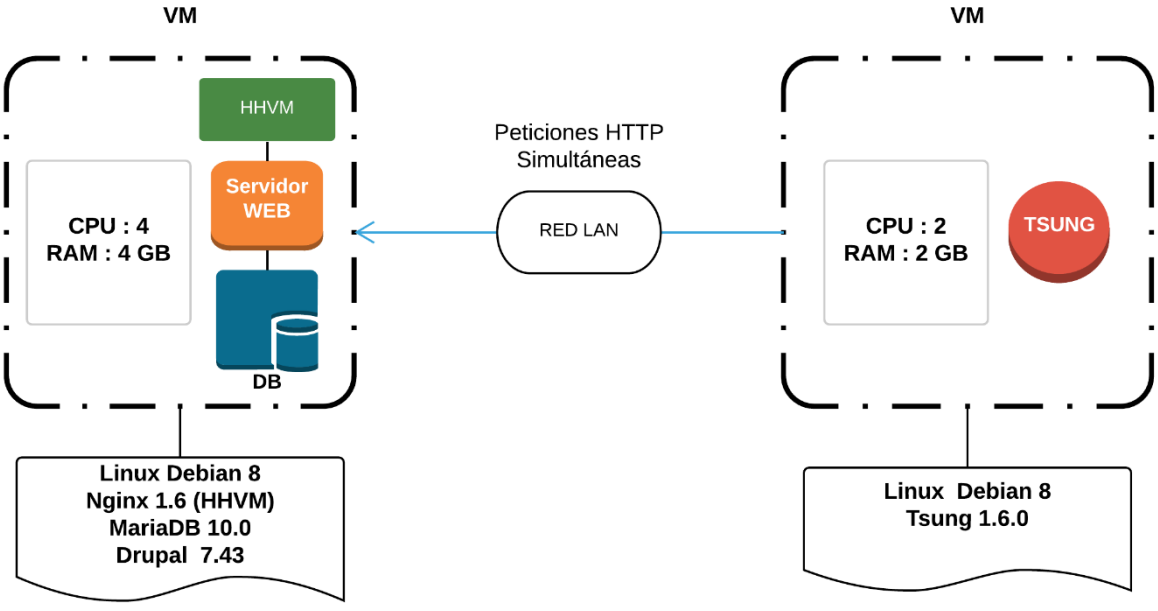
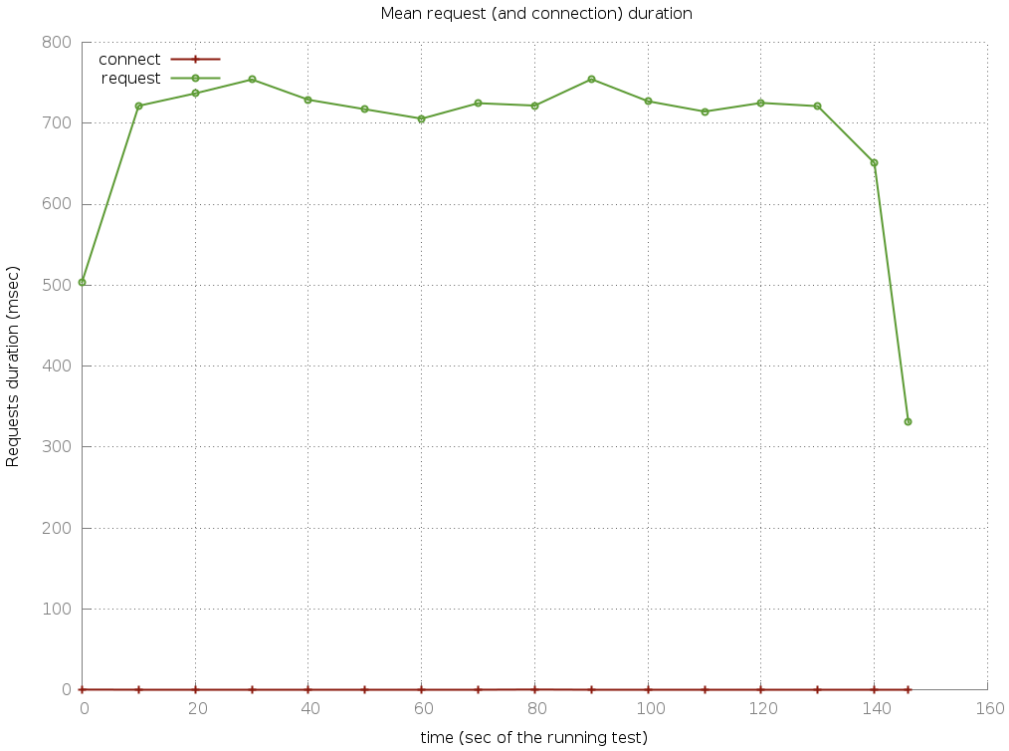


Figura 69:Modelo del escenario de prueba 3

Resultados de las pruebas

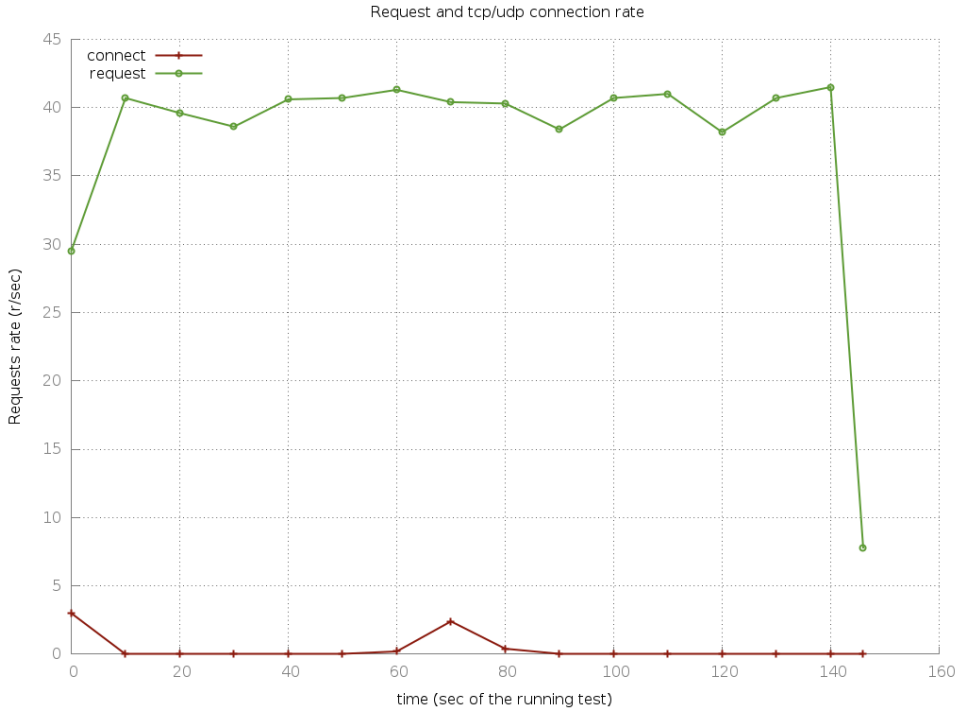
Las siguientes gráficas ilustran los resultados obtenidos.

Tiempos de respuesta



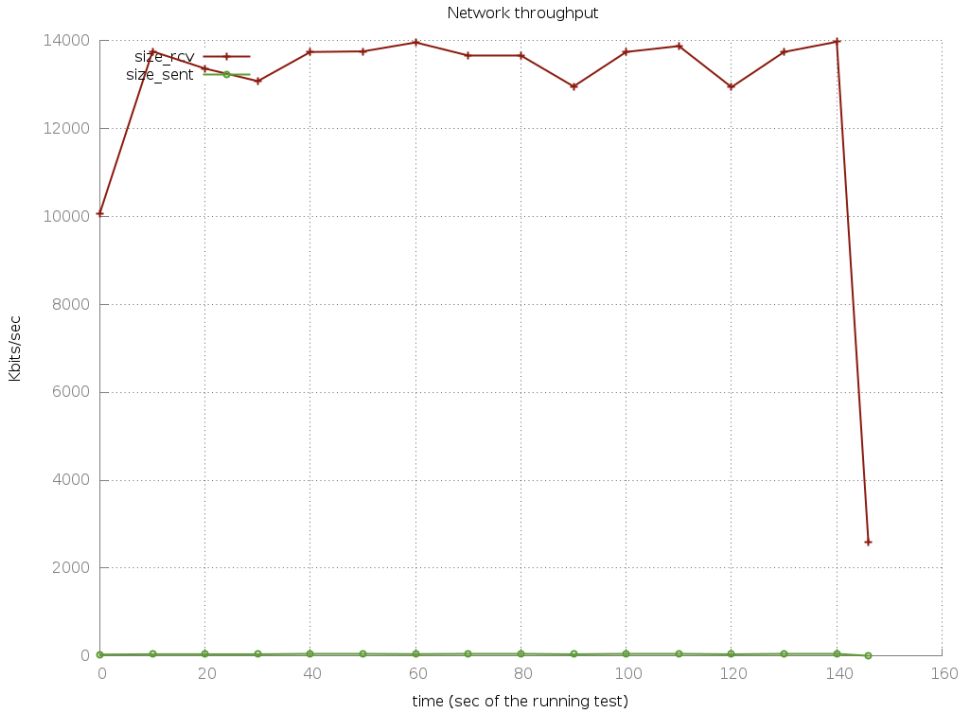
Gráfica 13: Tiempos de respuesta escenario de prueba 3

Tasa de rendimiento



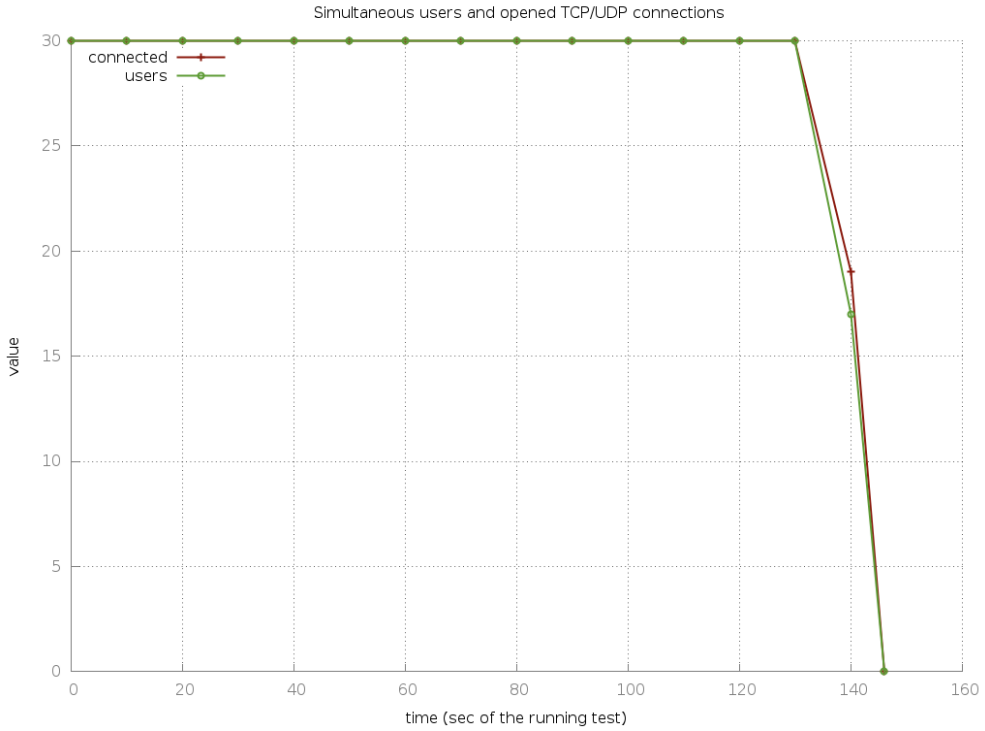
Gráfica 14: Tasa de rendimiento escenario de prueba 3

Tráfico de red



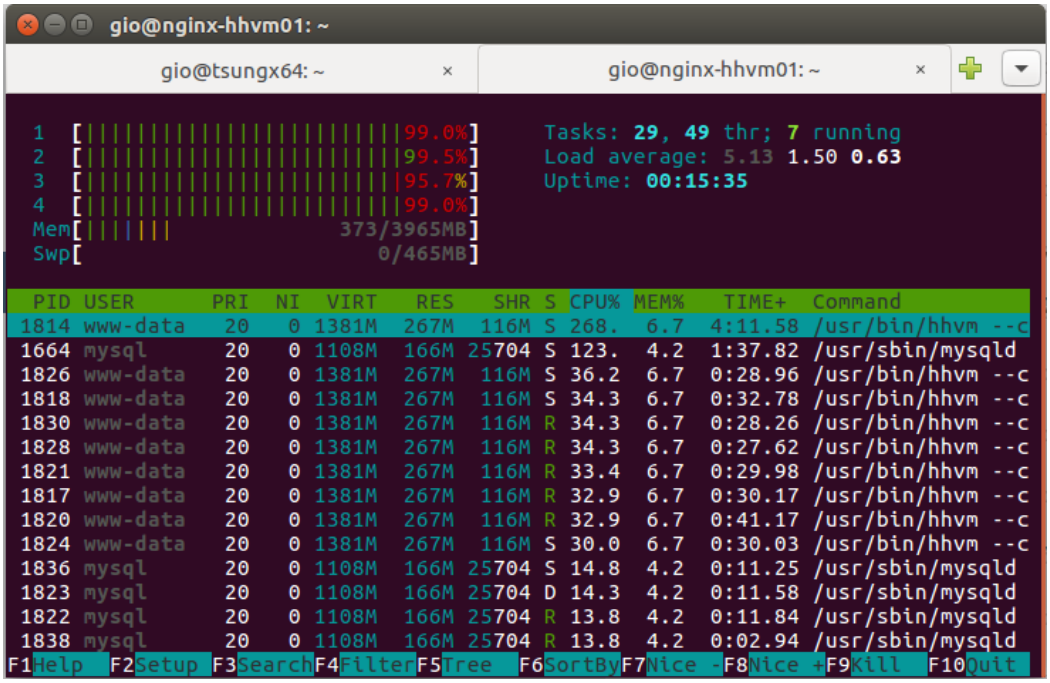
Gráfica 15: Tráfico de red escenario de prueba 3

Usuarios simultáneos



Gráfica 16: Usuarios simultáneos escenario de prueba 3

Utilización de CPU y RAM



Estadísticas generales

Nombre	Tasa más alta	Tasa promedio	Media
Conexión	2.4 /sec	0.20 / sec	0.613 msec
Request	41.5 /sec	38.38 / sec	0.71 sec
Sesión	2.83 /sec	2.07 / sec	2mn 24sec

Rendimiento de red

Nombre	Tasa	Total
Tamaño recibido	13.64 Mbits/sec	247.54 MB
Tamaño enviado	44.98 Kbits/sec	814.84 KB

Código de retorno HTTP

Código	Tasa más alta	Tasa promedio	Total
200	41.5 / sec	38.38 / sec	6000

5.1.5 Escenario de prueba 4 – Implementación de balanceo de carga con 2 nodos

En este escenario se adiciona el componente de balanceo de carga, por lo tanto, se utilizan 3 MV's; una para el balanceador de carga y dos para los servidores de bakc-end, esta dos últimas contiene un servidor web NGINX con procesador de código HHVM y un nodo del clúster MariaDB galera.

Modelo de la prueba

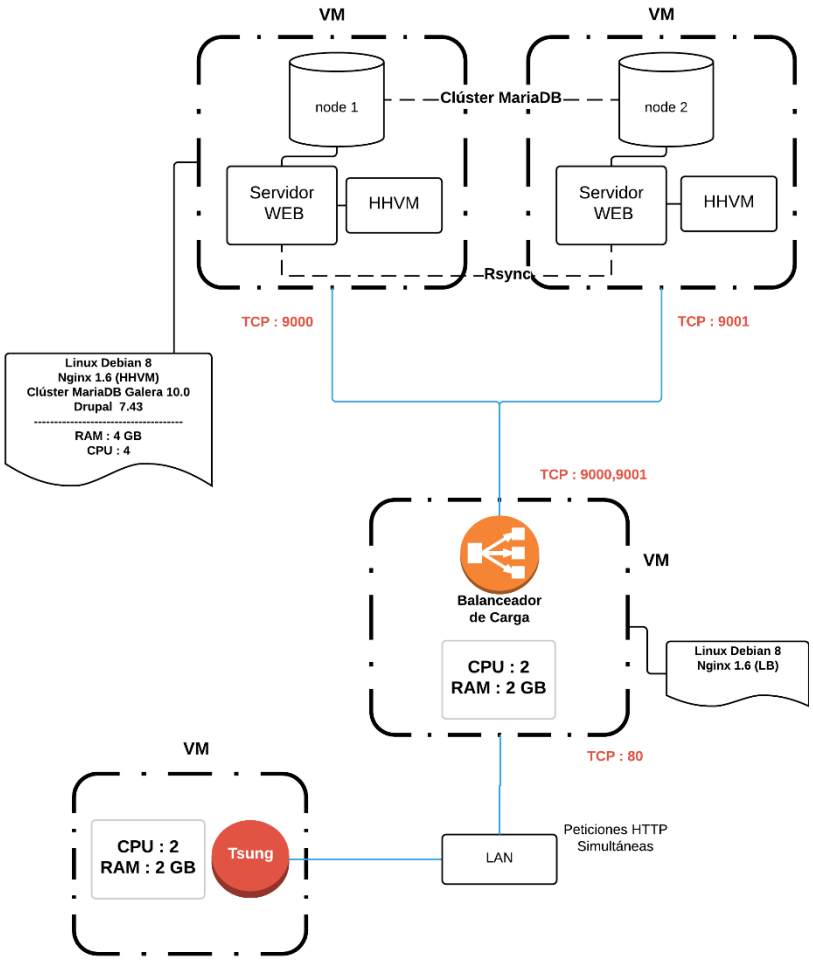
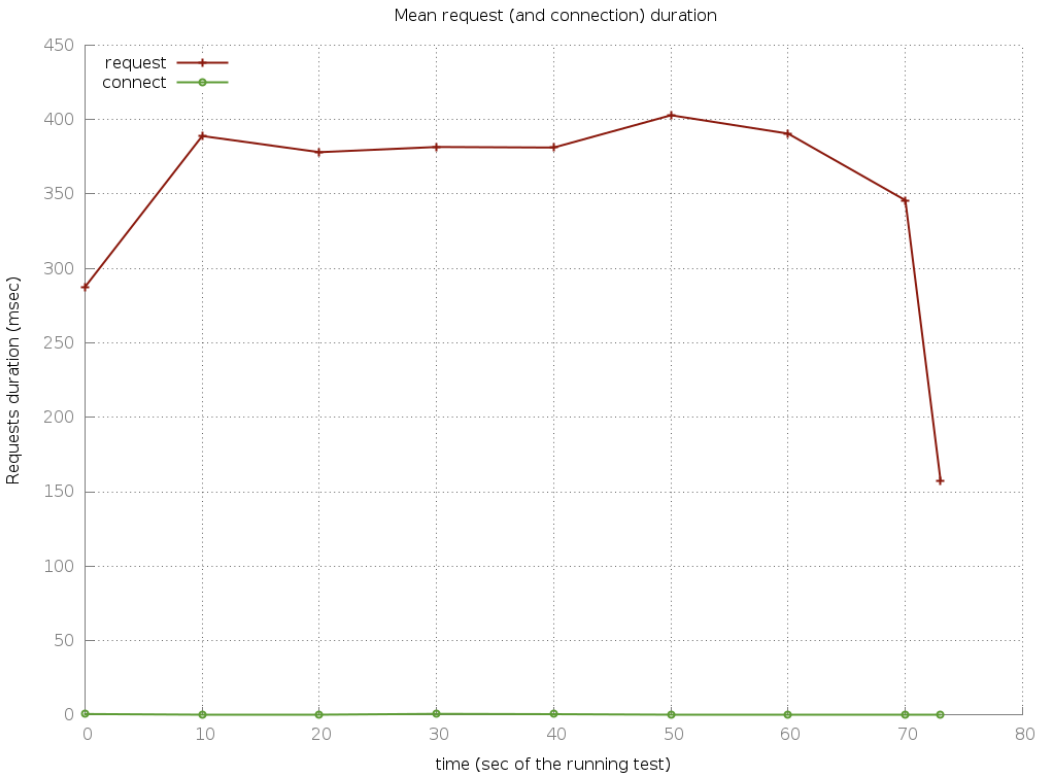


Figura 70: Modelo del escenario de prueba 4

Resultados de la prueba

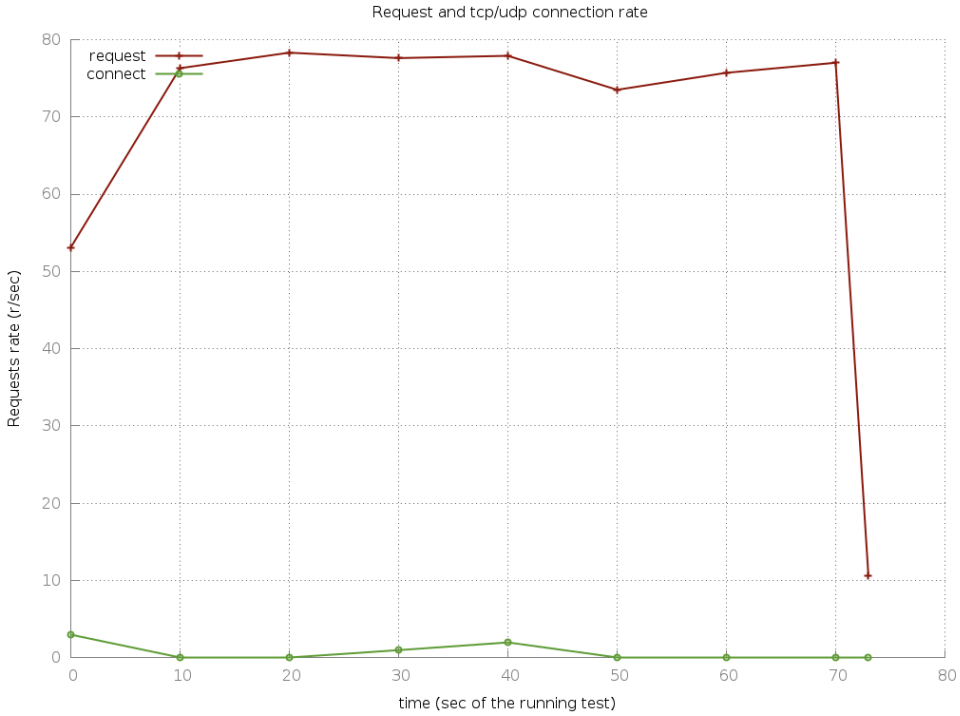
Las siguientes gráficas ilustran los resultados obtenidos.

Tiempos de respuesta



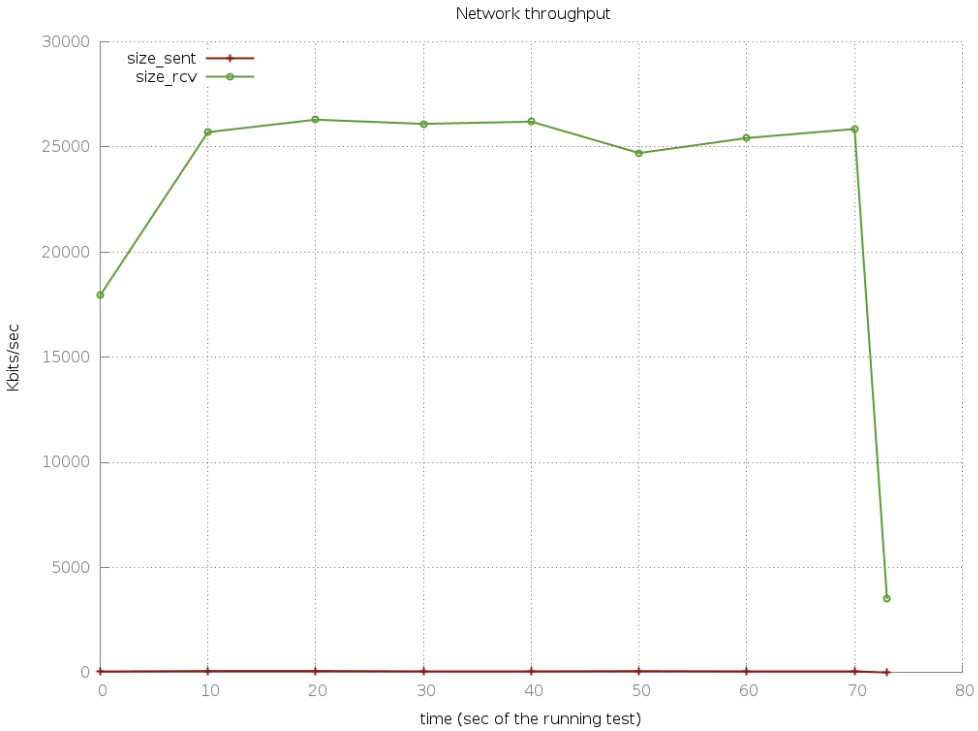
Gráfica 17: Tiempos de respuesta escenario de prueba 4

Tasa de rendimiento



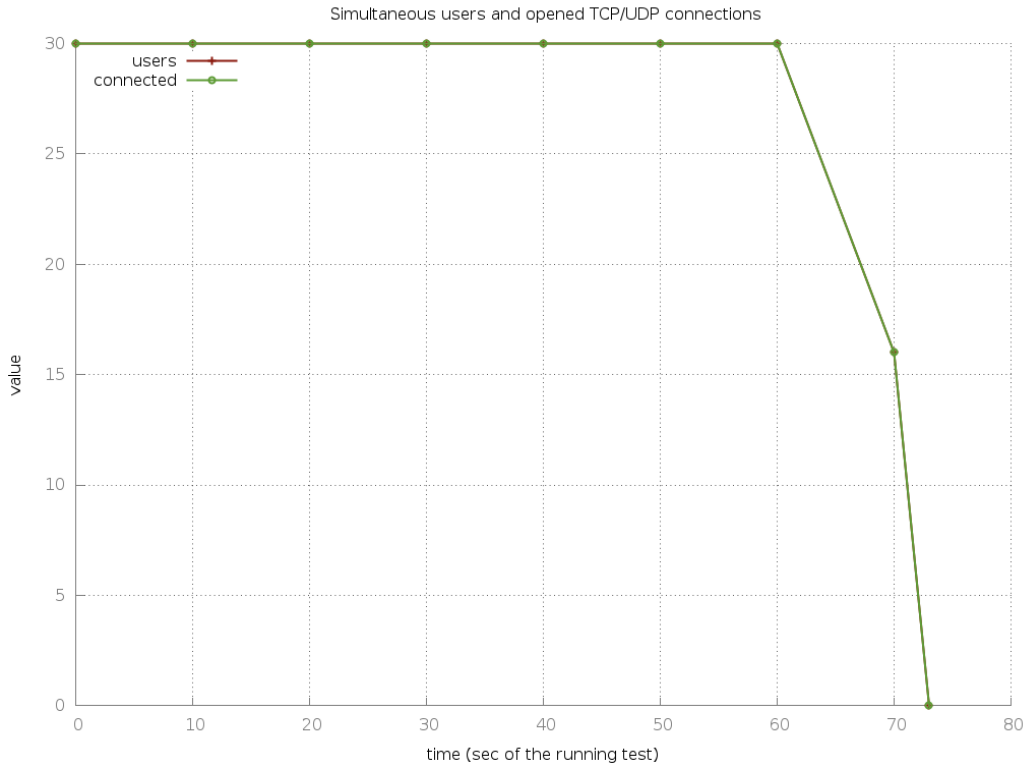
Gráfica 18: Tasa de rendimiento escenario de prueba 4

Tráfico de red



Gráfica 19: Tráfico de red escenario de prueba 4

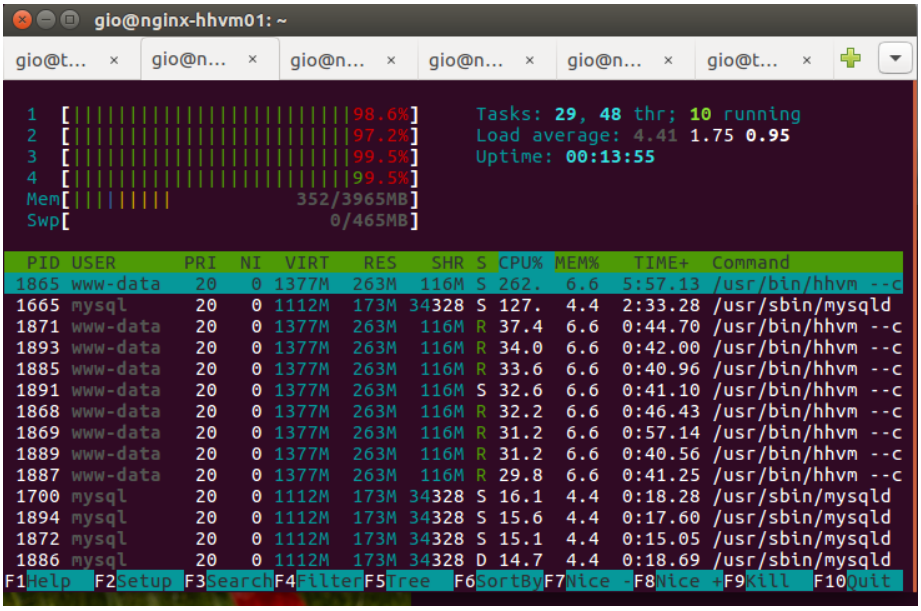
Usuarios simultáneos



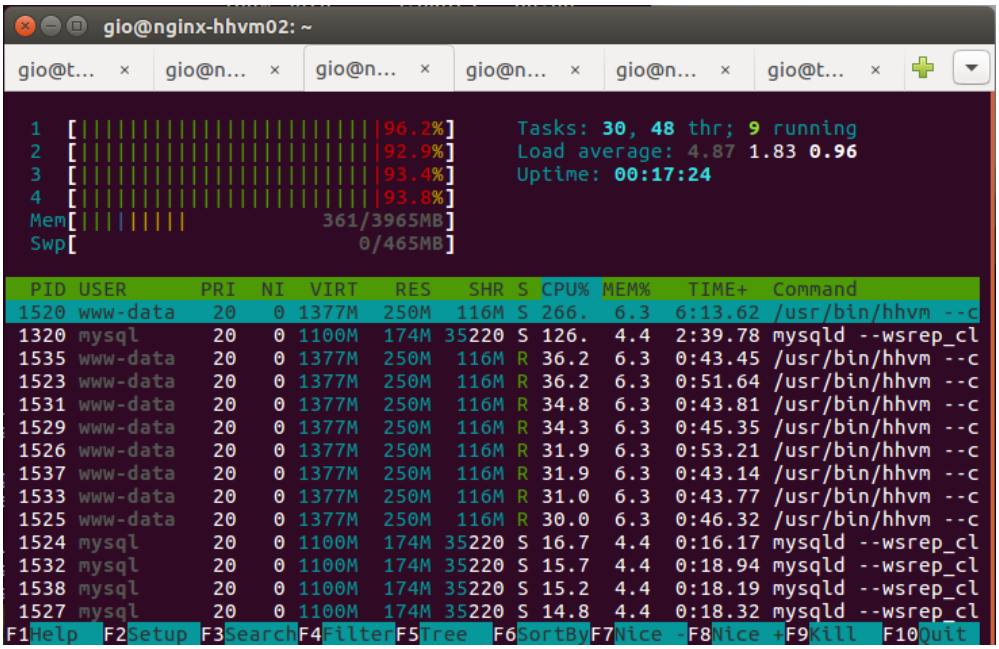
Gráfica 20: Usuarios simultáneos escenario de prueba 4

Utilización de CPU y RAM

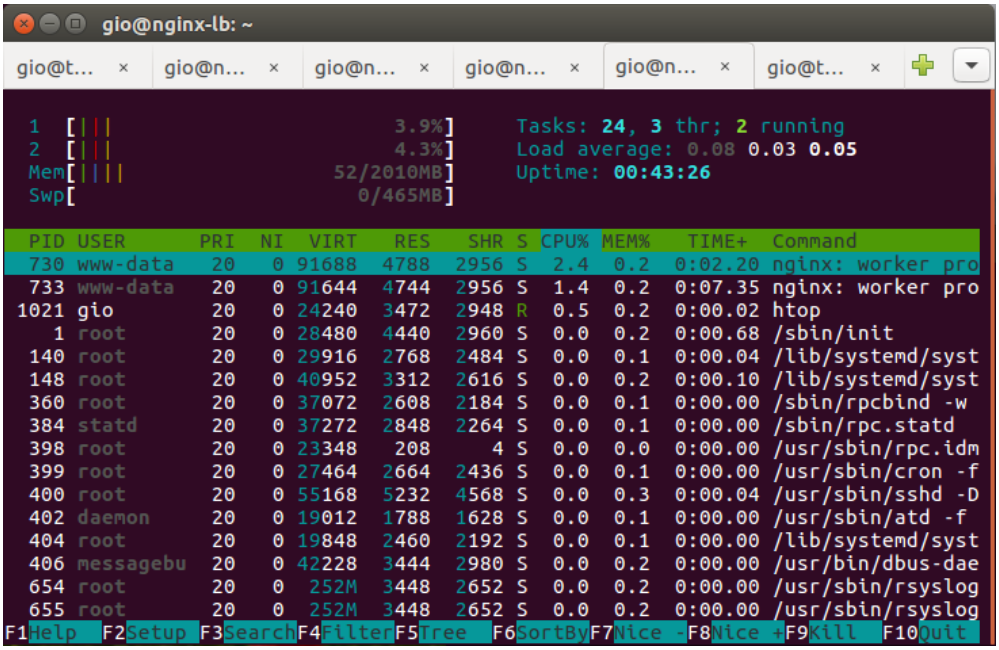
Captura 1 – Servidor 01



Captura 2 – Servidor 02



Captura 3 – Balanceador



Estadísticas generales

Nombre	Tasa más alta	Tasa promedio	Media
Conexión	2.0 /sec	0.38 / sec	0.702 msec
Request	78.3 /sec	71.45 / sec	0.37 sec
Sesión	5.333 /sec	3.37 / sec	1mn 14sec

Rendimiento de red

Nombre	Tasa	Total
Tamaño recibido	25.68 Mbits/sec	246.25 MB
Tamaño enviado	82.11 Kbits/sec	785.55 KB

Código de retorno HTTP

Código	Tasa más alta	Tasa promedio	Total
200	78.3 / sec	71.42 / sec	6000

5.1.6 Escenario de prueba 5 – Implementación de balanceo de carga con 3 nodos

En este escenario se añade un nodo de servidor back-end, y se sigue utilizando el balanceador de carga.

Modelo de la prueba

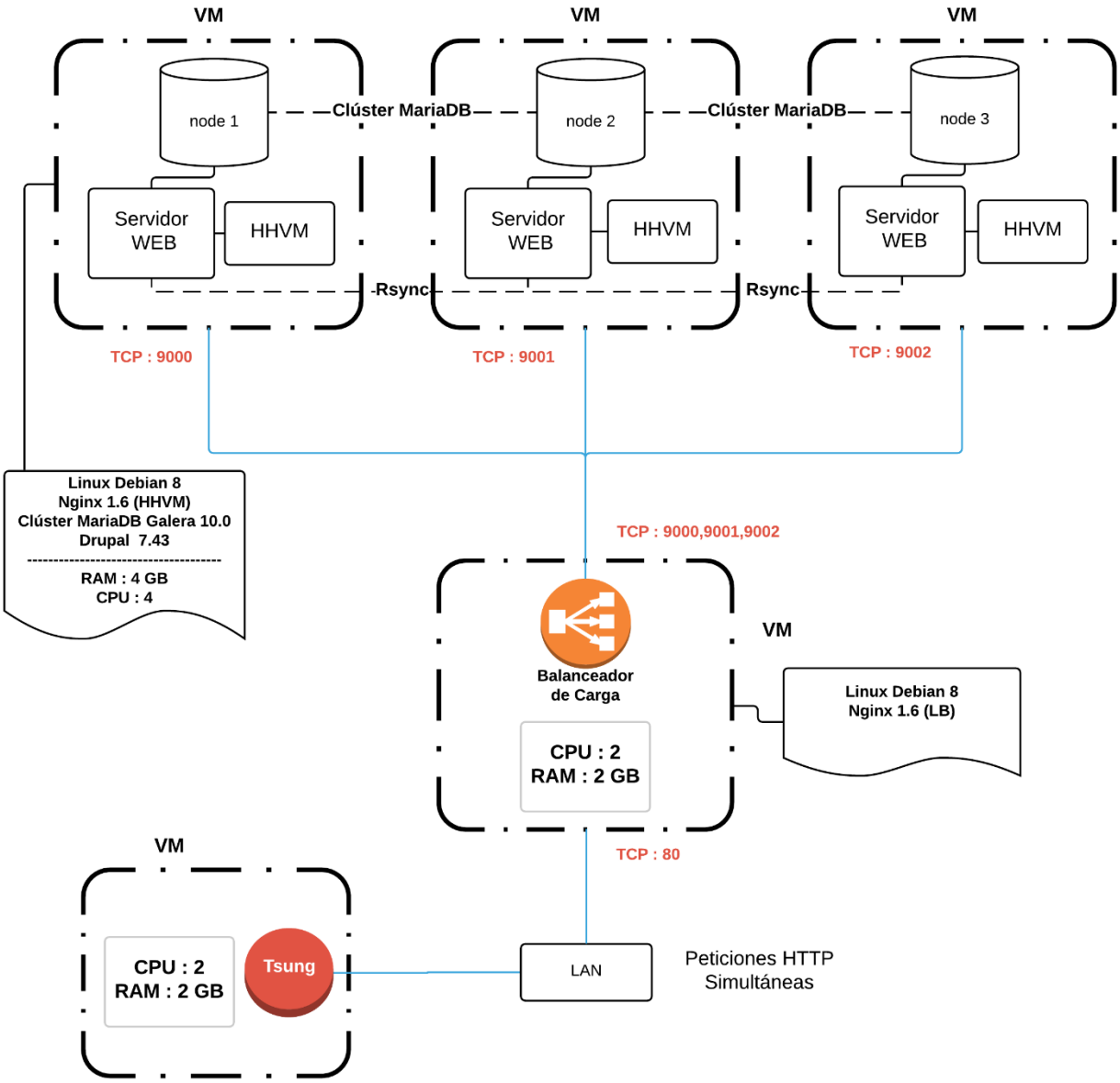
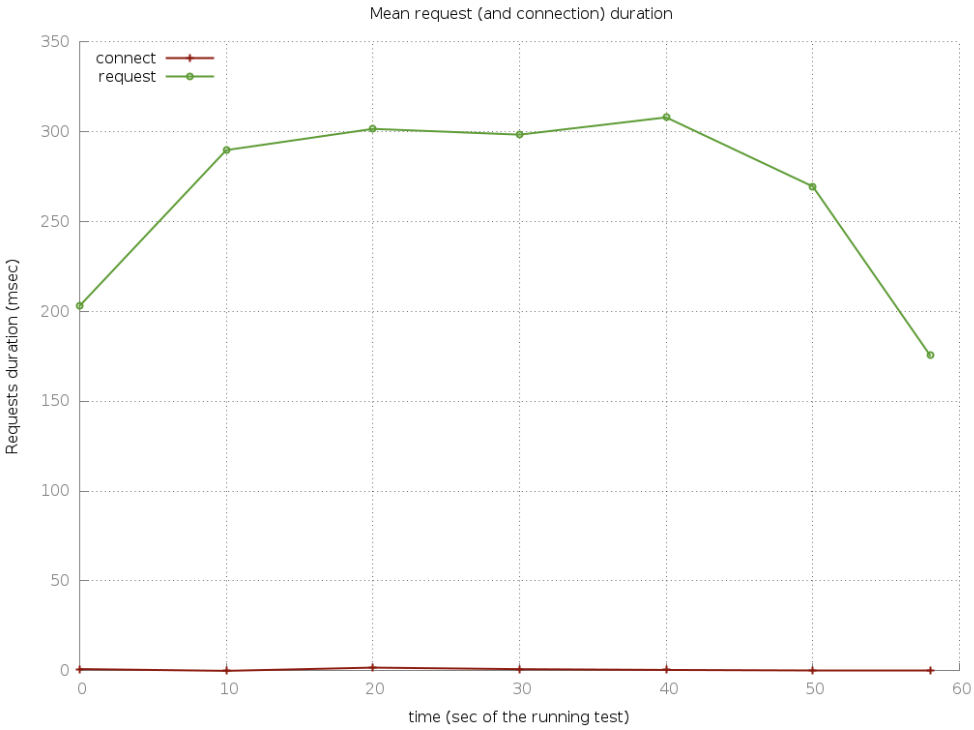


Figura 71: Modelo del escenario de prueba 5

Resultados de la prueba

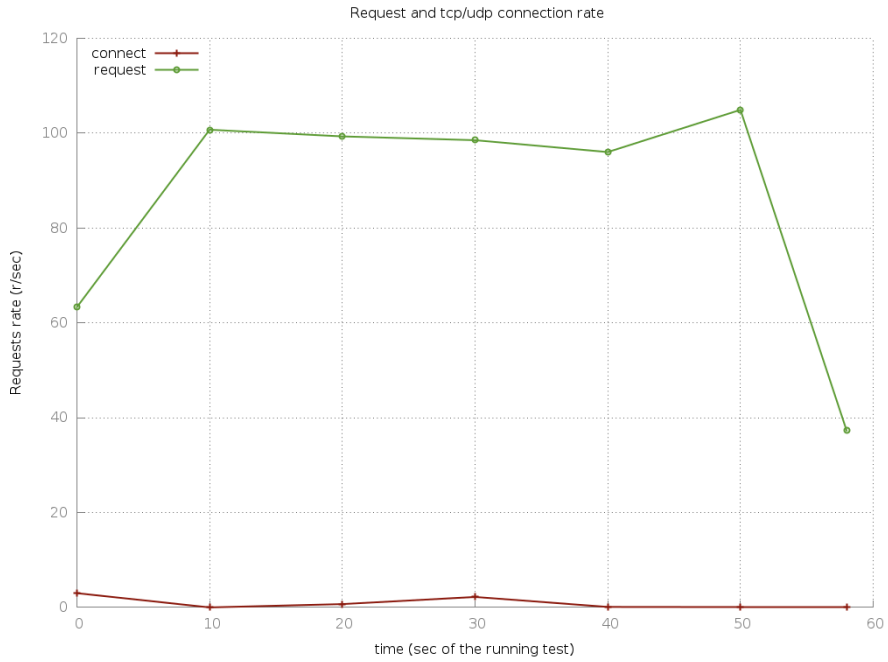
Las siguientes gráficas ilustran los resultados obtenidos.

Tiempos de respuesta



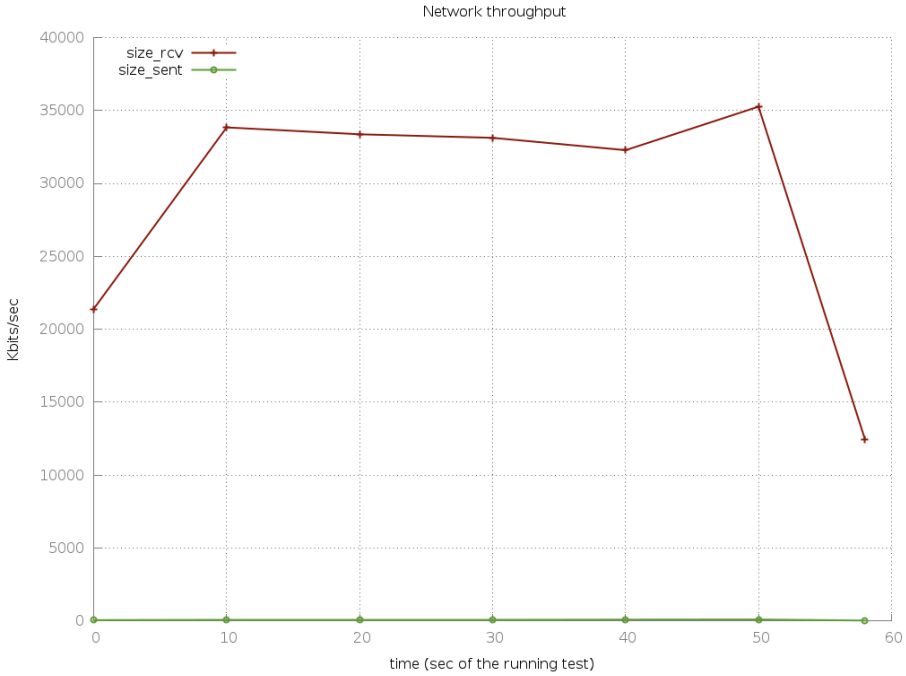
Gráfica 21: Tiempos de respuesta escenario de prueba 5

Tasa de rendimiento



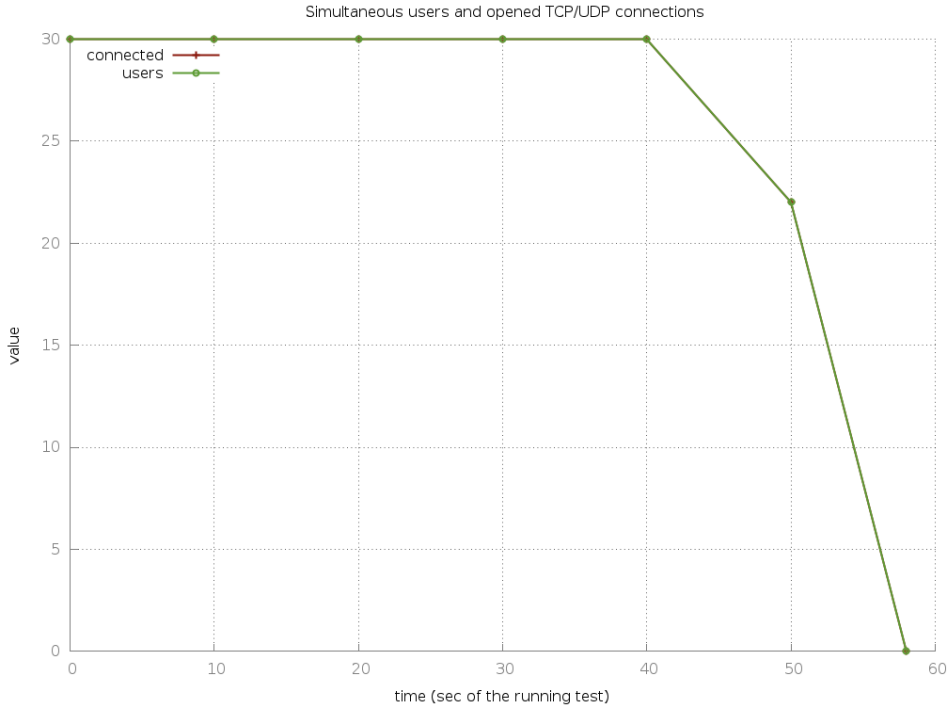
Gráfica 22: Tasa de rendimiento escenario de prueba 5

Tráfico de red



Gráfica 23: Tráfico de red escenario de prueba 5

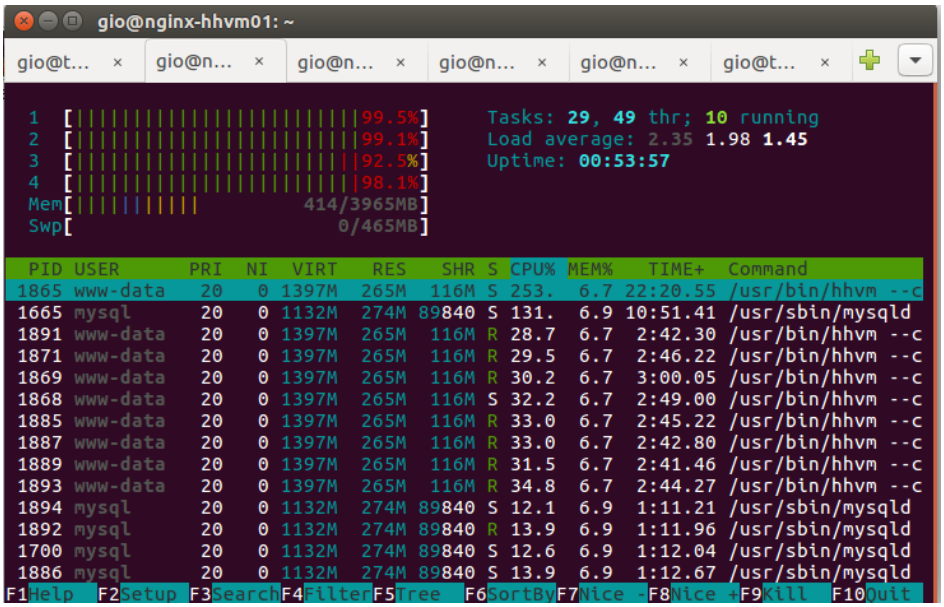
Usuarios simultáneos



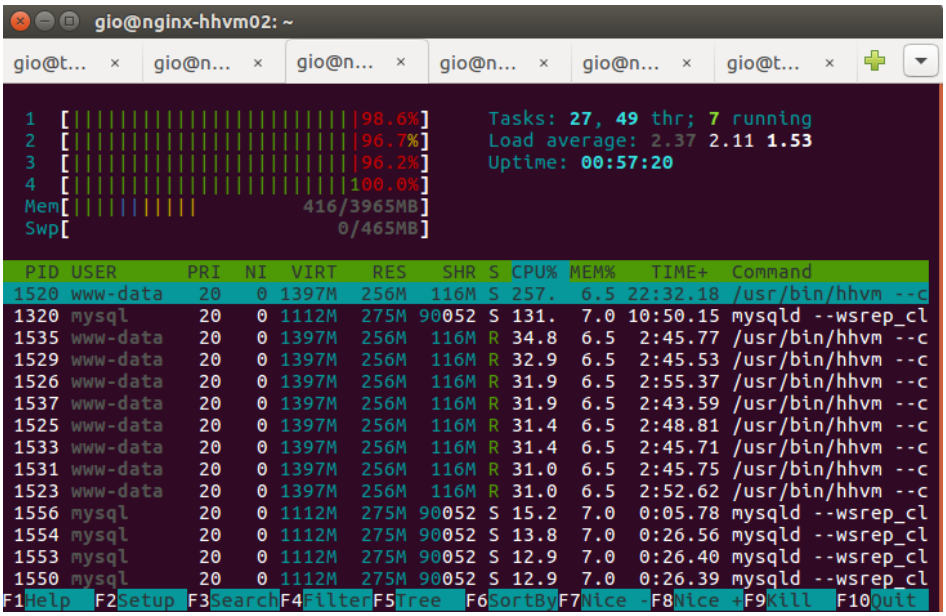
Gráfica 24: Usuarios simultáneos escenario de prueba 5

Utilización de CPU y RAM

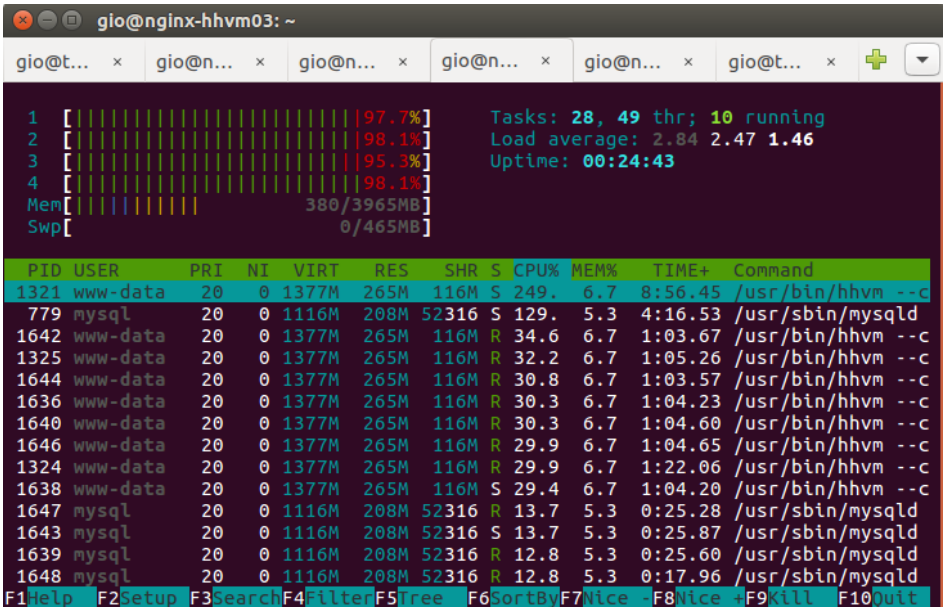
Servidor Web 01



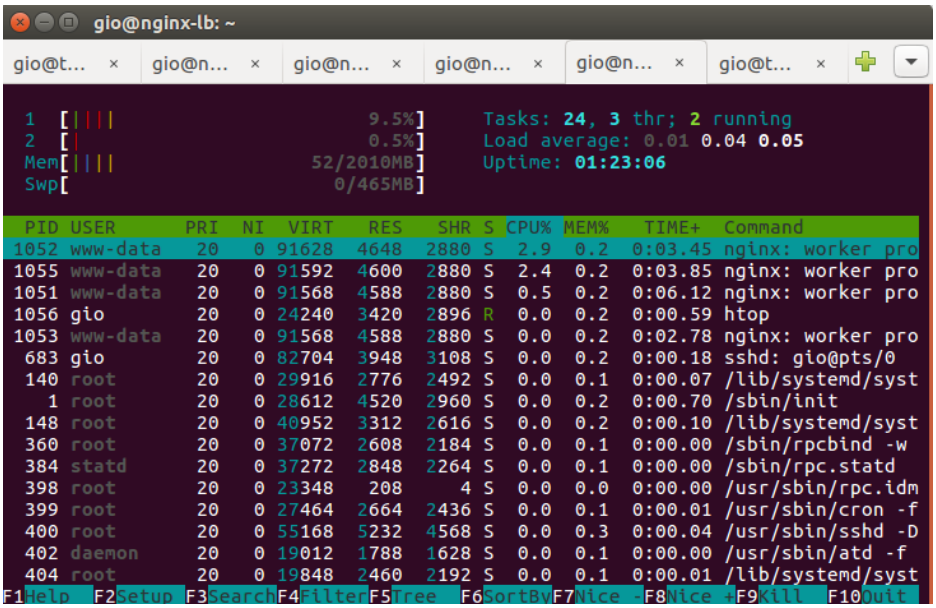
Servidor Web 02



Servidor Web 03



Balanceador



Estadísticas generales

Nombre	Tasa más alta	Tasa promedio	Media
Conexión	2.2 /sec	0.50 / sec	1.06 msec
Request	104.9 /sec	91.45 / sec	0.28 sec
Sesión	2.75 /sec	1.77 / sec	55.50 sec

Rendimiento de red

Nombre	Tasa	Total
Tamaño recibido	34.43 Mbits/sec	246.24 MB
Tamaño enviado	109.21 Kbits/sec	785.55 KB

Código de retorno HTTP

Código	Tasa más alta	Tasa promedio	Total
200	104.9 / sec	91.00 / sec	6000

5.2 Análisis de resultados

Se tomaron las variables: tiempo de respuesta, tasa de rendimiento y tasa de transmisión; de cada uno de los escenarios, para realizar una comparación entre sus valores obtenidos.

La Tabla 9 resumen cada uno de los escenarios de prueba realizados anteriormente.

ESCENARIOS DE PRUEBA

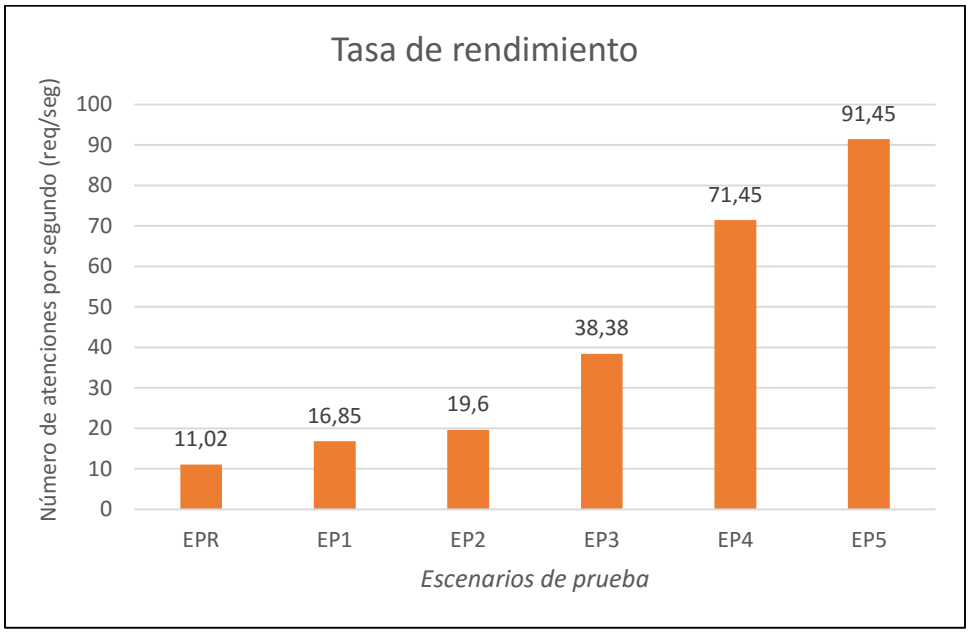
ID	Escenario	Descripción
EPR	Escenario de prueba real	Plataforma web Universidad Tecnológica de Bolívar
EP1	Escenario de prueba 1	Replica local de la Plataforma web Universidad Tecnológica de Bolívar
EP2	Escenario de prueba 2	Implementación de nginx web server y PHP-FPM
EP3	Escenario de prueba 3	Implementación de nginx web server y HHVM
EP4	Escenario de prueba 4	Implementación de balanceo de carga con 2 nodos

EP5	Escenario de prueba 5	Implementación de balanceo de carga con 3 nodos

Tabla 9: Escenarios de prueba

Las siguientes gráficas, contienen los resultandos de cada escenario, desde el escenario de prueba real, hasta, el escenario de prueba 5.

5.2.1 Tasa de rendimiento



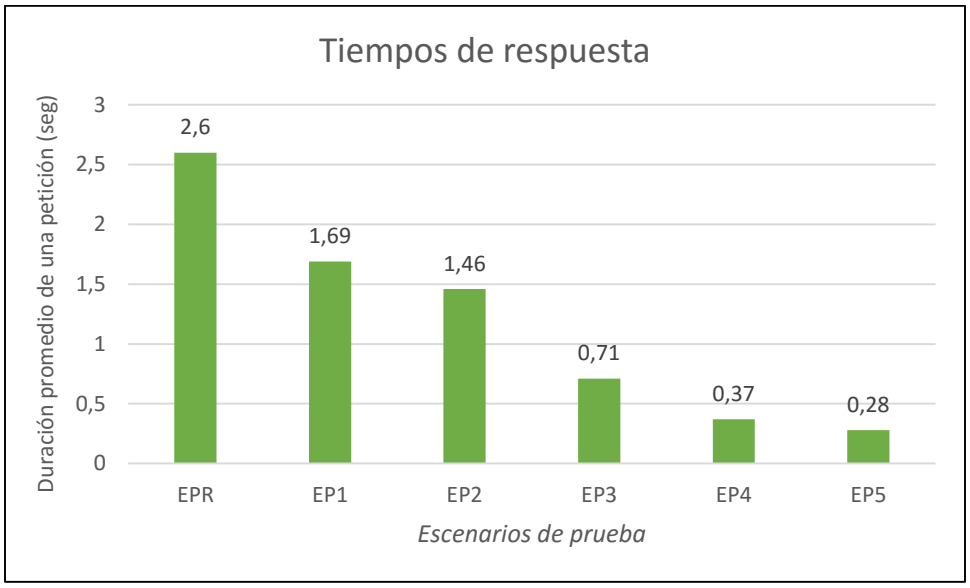
Gráfica 25: Tasa de rendimiento del análisis final

Como se puede apreciar en la anterior gráfica, la tasa de rendimiento del escenario real es de 11,02 req/seg, luego en el escenario de prueba 1, el cual recrea el escenario real, es de 16,85 req/seg; con este último valor se establece la base comparativa para los siguientes entornos. Por lo tanto, el aumento de la tasa de rendimiento durante las pruebas siguiente, con respecto al valor base, es altamente significativo; la tabla siguiente detalla el aumento de cada escenario.

Escenario	Aumento de tasa (%)	Servidor Web + FastCGI	Nodos	Balanceo de carga
EP2	16,32 %	Nginx + PHP-FPM	1	NO
EP3	127,77 %	Nginx + HHVM	1	NO
EP4	324,04 %	Nginx + HHVM	2	SI
EP5	366,58 %	Nginx + HHVM	3	SI

Tabla 10: Aumento de la tasa de rendimiento por cada escenario

5.2.2 Tiempos de respuesta



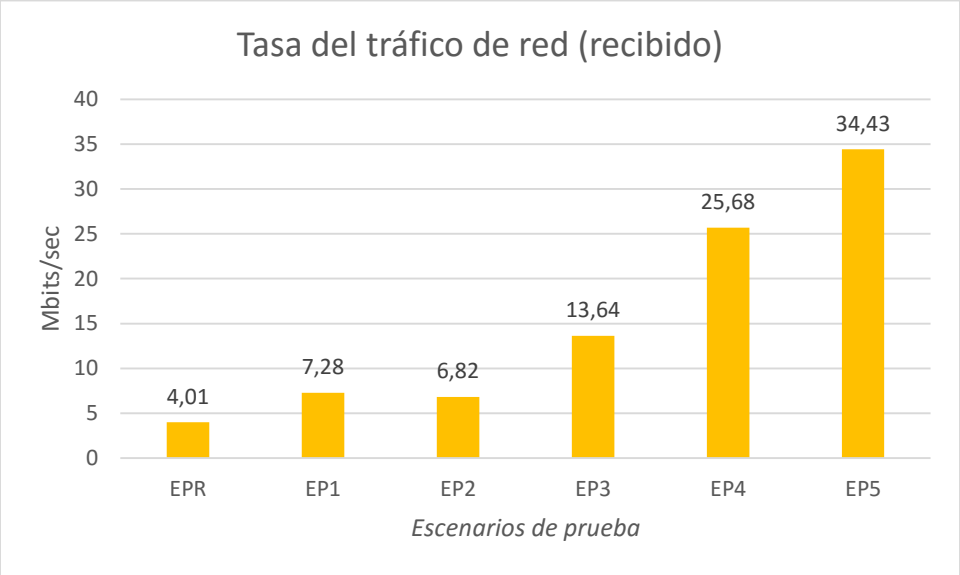
Gráfica 26: Tiempos de respuesta del análisis final

De igual forma que el análisis anterior, el valor base de comparación se obtuvo en el escenario de prueba 1, teniendo un tiempo de respuesta promedio por petición de 1.69 seg. Para medidas de tiempos de respuesta, la disminución del valor indica una mejora en el sistema, por lo tanto, la anterior gráfica determina que existe una mejora significativa en los siguientes escenarios de prueba, la Tabla 11 especifica los porcentajes de mejora con respecto al caso basa del escenario 1.

Escenario	Disminución de los tiempos de respuesta (%)	Servidor Web + FastCGI	Nodos	Balanceo de carga
EP2	13,61 %	Nginx + PHP-FPM	1	NO
EP3	57,99 %	Nginx + HHVM	1	NO
EP4	78,11 %	Nginx + HHVM	2	SI
EP5	83,43 %	Nginx + HHVM	3	SI

Tabla 11:Disminución % de los tiempos de respuesta por cada escenario

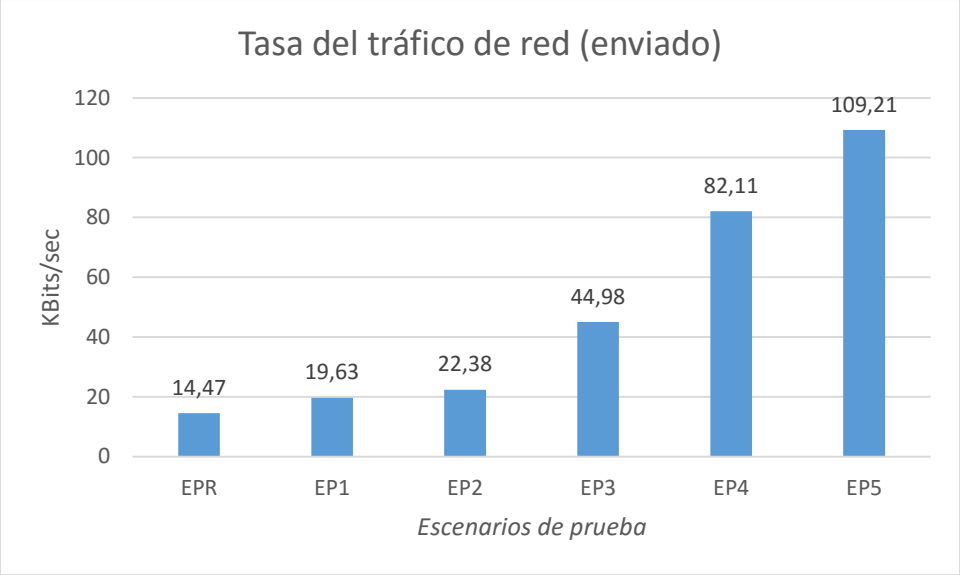
5.2.3 Tráfico de red



Gráfica 27: Tasa del tráfico de red (recibido) del análisis final

Los resultados de la tasa de tráfico de red (recibido), indican el flujo de datos que se están recibiendo desde el servidor, en la anterior gráfica el caso base de comparación tiene como valor 7,28 Mbits/seg perteneciente al escenario 1, el aumento de valor en cuando la tasa de tráfico de red, indica que el servidor está respondiendo más peticiones por segundo, siempre y cuando las pruebas se realicen con el mismo servidor web.

La prueba del escenario 1, fue realizada con el servidor web Apache y la prueba del escenario 2 con el servidor nginx, en el escenario 1 el valor fue de 7.28 *Mbits/seg* y en el escenario 2 fue de 6.82 *Mbits/seg*, aquí existe una disminución de valor, pero esto no indica que en el escenario 1 el servidor este respondiendo más peticiones, debido a que no son los mismos servidores. Como anteriormente se determinó el escenario 2 responde más peticiones que el escenario 1, pero la disminución de la tasa de tráfico se debe a que el servidor nginx maneja respuesta con menor peso (MBits) que el servidor apache, esto se debe a la compresión de los archivos antes de ser transmitidos por la red. En los siguientes escenarios se utiliza el mismo servidor nginx y la tasa de trafico de red aumenta lo cual, si, indica un aumento en la tasa de procesamiento de peticiones.



Gráfica 28: Tasa del tráfico de red (enviado) del análisis final

El tráfico de red (enviado) indica el flujo de datos que se está enviando al servidor, lo que representa las peticiones enviadas al servidor; a medida que el tráfico aumenta, la cantidad de peticiones aumenta también, en la anterior gráfica el caso base de comparación es el escenario 1, el cual presenta el valor de 16,63 Kbits/seg, como se puede apreciar, en los siguientes escenario se muestra un

aumento significativo de tráfico enviado, esto indica que el servidor atiende una cantidad mayor de peticiones, aumentando la capacidad del sistema. Como se determinó anteriormente los escenarios siguientes aumentan en un porcentaje muy alto la tasa de rendimiento (req/seg).

6 CAPÍTULO VI: CONCLUSIONES Y TRABAJOS FUTUROS

6.1 Conclusiones

La presente investigación tuvo como objetivo comprobar que bajo una plataforma web Drupal 7.43, se puede mantener tiempos de respuesta aceptables, mientras aumenta el número simultáneo peticiones HTTP; utilizando técnicas de balanceo de carga HTTP y la optimización de servidores web.

Esto quiere decir que los servidores de una plataforma web donde se utilice técnicas de balanceo de carga HTTP pueden procesar grandes volúmenes de carga sin afectar su calidad de servicio, teniendo tiempos de respuesta promedios, menores a un segundo.

Para demostrar esto, primeramente, se realizó un análisis del rendimiento de una plataforma web Drupal 7.43 sin la utilización de un sistema balanceador de carga HTTP, aquí se estudiaron los componentes de hardware y software, tales como: servidor web utilizado, motor de base de datos, utilización de memoria RAM, utilización de CPU, etc.

Esto demostró que los servidores, no alcanzaban grandes tasas de rendimiento, cuando aumentaba la cantidad de peticiones simultáneas, dando como resultado tiempos de respuesta de 2.6 segundos por petición; el aumento de los tiempos de respuesta, indica que el servidor se congestiona, afectando su funcionamiento normal. Debido a esto, se realizó una investigación de nuevas tecnologías web, con el fin de mejorar el rendimiento y optimizar los tiempos de respuesta.

Por esto, se opta por utilizar nuevos servidores web, los cuales presentan mejoras en cuanto a su diseño y funcionamiento, también se agregaron nuevos componentes como: procesadores de código enfocados en el alto rendimiento, la utilización de un clúster de datos para minimizar los tiempos de consulta, y la utilización de un entorno múltiple de servidores web para permitir la distribución de las cargas.

Con esto se pudo lograr la implementación de un servidor de balanceo de carga HTTP, con lo cual se logra mejorar el manejo de las peticiones HTTP simultaneas de una forma más eficiente e inteligente; lo cual permite disminuir los tiempos de respuesta y aumentar la tasa de rendimiento de una plataforma web CMS Drupal.

Las mejoras significativas se iniciaron, cuando se utilizó un nuevo componente dentro del sistema; el cual fue el procesador de código de alto rendimiento; este componente es un aplicativo totalmente independiente al servidor web, pero su única forma de trabajo es junto a él; esto quiere decir que el procesador de código no tiene la capacidad de procesar peticiones HTTP por sí mismo, y está diseñado únicamente para procesar código proveniente de un servidor web.

Esta separación de responsabilidades, permite optimizar el procesamiento de una petición, ya que mientras el servidor web se encarga de la administración de las peticiones, el procesador de código realiza las operaciones lógicas. En el entorno donde no existe ningún componente de optimización, el servidor tenía la doble responsabilidad de administrar las peticiones y ejecutar el código, por lo tanto, presenta tasas de rendimiento muy bajas, y se muestra en total desventaja frente al nuevo enfoque. La implementación del procesador de código de alto rendimiento mejora la utilización de los recursos de procesamiento, debido a que, no se necesita aumentar la cantidad de RAM, ni la cantidad de CPU para su funcionamiento. Con esta configuración y la utilización de un único host, donde todavía no se utiliza balanceo de carga HTTP, aumento la tasa de rendimiento, de 16.35 req/seg hasta 38.38 req/seg.

La implementación del clúster de datos, permitió tener mejor acceso a la base de datos del sistema CMS Drupal, también la utilización de múltiples servidores web contribuyo a atender muchas más peticiones paralelamente; por lo tanto, estos dos componentes fueron fundamental en la optimización del sistema.

Luego de tener el clúster de datos y los múltiples servidores web, componentes necesarios para la distribución de peticiones, se aplicó la técnica de balanceo de carga, con dos hosts de back-end, esto aumento la tasa de rendimiento de 38.38 req/seg a 71.45 req/seg, teniendo tiempos de respuesta de 0.37 seg, luego la utilización de 3 host de back-end, llevo la tasa de procesamiento a 91.45 req/seg, teniendo tiempos de respuesta de 0.28 seg. Con esto se puede determinar que una plataforma CMS Drupal, puede mantener los tiempos de respuesta aceptables, mientras aumenta la carga de peticiones simultáneas, utilizando un sistema balanceador de carga HTTP.

6.2 Trabajos futuros

El trabajo investigativo cuenta con la facultad de mejoramiento, la cual permite seguir añadiendo nuevas ideas o componentes que ayuden a seguir mejorando el trabajo realizado hasta este momento. En esta investigación se tiene como línea investigativa, los sistemas distribuidos; los cuales actualmente tiene un crecimiento dinámico, donde se adoptan rápidamente nuevos métodos y tecnologías de trabajo.

El balanceo de carga HTTP y la optimización de servidores web, realizada para el mejoramiento del sistema CMS Drupal en esta investigación, ha mostrado grandes mejoras en cuando a escalabilidad y rendimiento, para seguir mejorando este sistema, se piensa en aplicar una nueva solución a nivel de datos. La utilización de un clúster junto con un balanceador de carga de consultas, actualmente en el trabajo realizado se utiliza un clúster de datos el cual mejora el acceso a la base de datos del sistema Drupal, con la utilización del balanceador de base de datos, el acceso al clúster se optimiza dando mejores tiempos de consulta. Los sistemas CMS Drupal tienden a utilizar en gran porcentaje la base de datos para generar el contenido web, por lo tanto, en esta situación es de vital importancia mejorar la capa de acceso de datos.

Para una plataforma en producción que requiera la implementación del sistema de optimización y escalabilidad desarrollado en esta investigación, es recomendable realizar diversas pruebas de rendimiento sobre una plataforma Cloud antes de su implementación final, esto, para obtener mayor confiabilidad en su despliegue. La variedad de resultados que se podrían obtener dependiendo a la estructura de red que tenga el prestador del servicio Cloud, debe ser contemplada para tomar una decisión final.

7 REFERENCIAS BIBLIOGRAFICAS

LIBROS

1. DISTRIBUTED INFORMATION SYSTEM. JAMES POWELL, 1997.
2. WEB PERFORMANCE TUNING, 2ND EDITION. KILLELEA, P., 2002.
3. BUILDING SCALABLE WEB SITES: BUILDING, SCALING, AND OPTIMIZING THE NEXT GENERATION OF WEB APPLICATIONS. HENDERSON, C., 2006.
4. INSTANT NGINX STARTER. MARTIN FJORDVALD, 2013
5. NGINX HTTP SERVER. CLÉMENT NEDELCU,2013
6. SERVER ADMINISTRATION FOR PROGRAMMERS. CHRIS FIDEO, 2015.

ARTICULOS

7. THE C10K PROBLEM. KEGEL D, 1999.
8. DYNAMIC LOAD BALANCING ON WEB-SERVER SYSTEMS. 1999.
9. LOAD BALANCING IN DISTRIBUTED SYSTEMS: A GAME THEORETIC APPROACH. DANIEL GROSU,2003
10. DISTRIBUTED COMPUTING: PRINCIPLES, ALGORITHMS, AND SYSTEMS.
11. FRAGMENT AND LOCALIZED ORBITAL METHODS IN ELECTRONIC STRUCTURE THEORY. DMITRI G. FEDOROV, 2012
12. VALERIA CARDELLINI, MICHELE COLAJANNI, MUKESH SINGHAL, 2012.
13. ADAPTIVE RESOURCE PROVISIONING FOR READ INTENSIVE MULTI-TIER APPLICATIONS IN THE CLOUD. WAHEED IQBALA, MATTHEW N. DAILEYA, DAVID CARRERAB, PAUL JANECEKA, 2010.
14. PERFORMANCE ANALYSIS OF PROCESS DRIVEN AND EVENT DRIVEN WEB SERVERS, PRAKASH P 2015

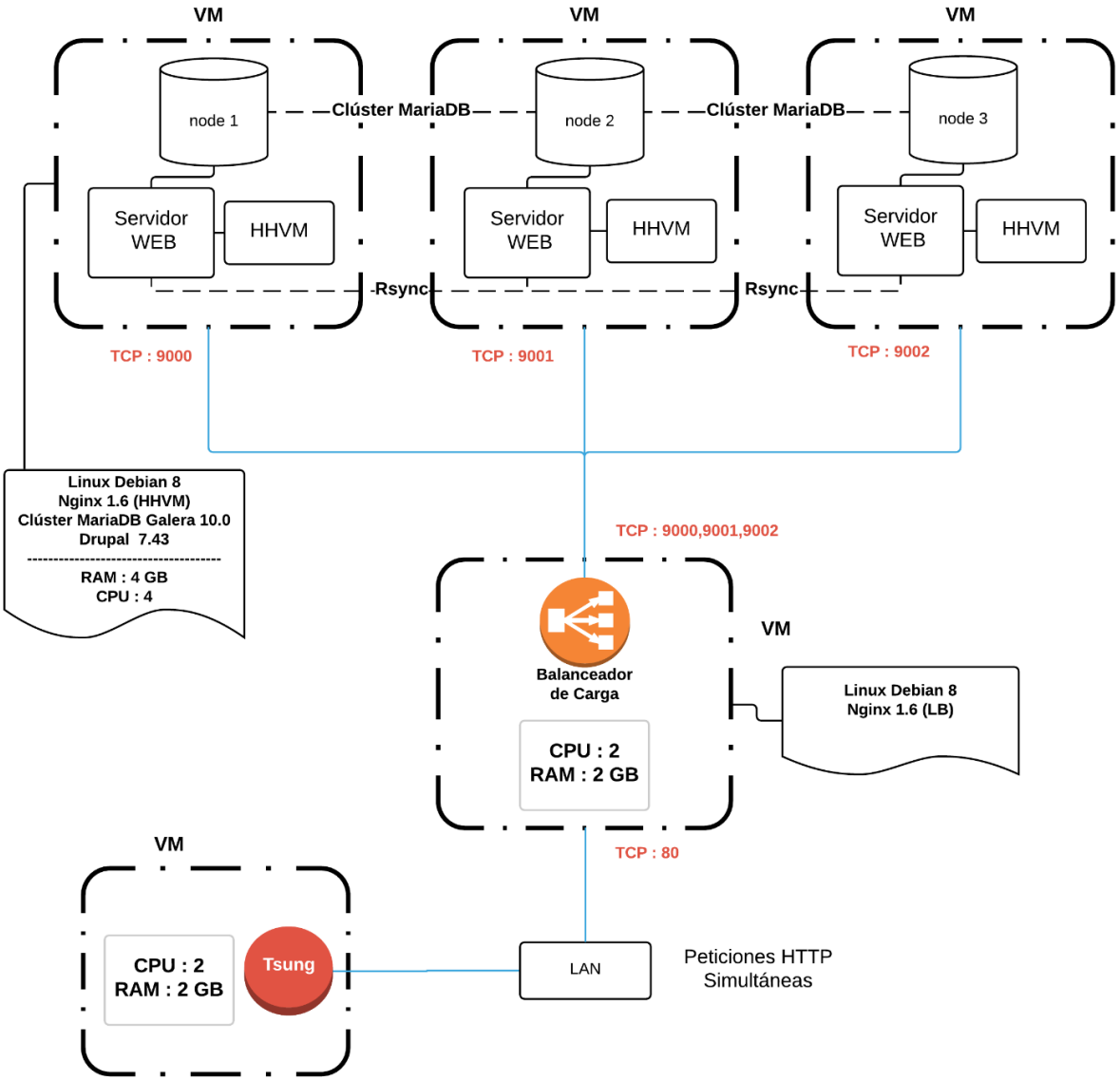
15. IMPLEMENTING A CLOUD BACKED SCALABLE NOTE-TAKING APPLICATION WITH ENCRYPTED OFFLINE STORAGE AND CROSS PLATFORM REPLICATION, ROBERT SMART 2015.
16. ALICE: AVAYA LABS INNOVATIONS CLOUD ENGAGEMENT. JOHN BUFORD, 2015.

SLIDES

17. KATHY Y, 2007. DYNAMIC LOAD BALANCING UNIVERSITY OF BERKELEY.
18. HTTP://ES.SLIDESHARE.NET/DAVEMITZ/7-STAGES-OF-SCALING-WEB-APPLICATIONS.

8 ANEXOS

8.1 ANEXO 1: DIAGRAMA DE LA SOLUCIÓN FINAL



8.2 Anexo 2: CÓDIGO DE CONFIGURACIÓN BALANCEADOR DE CARGA HTTP

****

```
http {

    upstream appDrupal {
        server 172.16.9.50:9000 max_fails=3 fail_timeout=30s;
        server 172.16.9.51:9001 max_fails=3 fail_timeout=30s;
        server 172.16.9.52:9002 max_fails=3 fail_timeout=30s;
    }

    server {

        listen 80;
        access_log /var/log/nginx/drupal-access.log;
        error_log  /var/log/nginx/drupal-error.log error;


      location / {

            include proxy_params;
            proxy_set_header X-Forwarded-Port $server_port;


            proxy_pass http://appDrupal;
            proxy_redirect off;


            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
       }

    }
```

```
}
```

8.3 ANEXO 3: CÓDIGO DE CONFIGURACIÓN SERVIDOR WEB HHVM 01

```
**********************************************************************************************
server {
     listen 9000;
     server_name www.drupal.com.co;
     root /usr/share/nginx/drupal;
     index index.php index.html index.htm;

     location / {
     try_files $uri $uri/ /index.php?$query_string; # For Drupal
>= 7
     }

     location ~ '\.(hh|php)$|^/update.php' {
       fastcgi_keep_conn on;
       fastcgi_pass   unix:/var/run/hhvm/hhvm.sock;
       fastcgi_index  index.php;
       fastcgi_param  SCRIPT_FILENAME
$document_root$fastcgi_script_name;
       include        fastcgi_params;

    }
    location ~ ^/sites/.*/files/styles/ { # For Drpal >= 7
        try_files $uri @rewrite;
```

```
    }


    location @rewrite {
    rewrite ^/(.*)$ /index.php?q=$1;
    }
    location ~ ^/sites/.*/files/styles/ {
      try_files $uri @rewrite;
    }
}
```

8.4 Anexo 4: CÓDIGO DE CONFIGURACIÓN SERVIDOR WEB HHVM 02

```
**********************************************************************************************

server {
     listen 9001;
     server_name www.drupal.com.co;
     root /usr/share/nginx/drupal;
     index index.php index.html index.htm;

     location / {
     try_files $uri $uri/ /index.php?$query_string; # For Drupal
>= 7
     }
     location ~ '\.(hh|php)$|^/update.php' {
       fastcgi_keep_conn on;
       fastcgi_pass   unix:/var/run/hhvm/hhvm.sock;
       fastcgi_index  index.php;
```

```
       fastcgi_param  SCRIPT_FILENAME
$document_root$fastcgi_script_name;


       include        fastcgi_params;
    }
  location ~ ^/sites/.*/files/styles/ { # For Drpal >= 7
        try_files $uri @rewrite;
    }
    location @rewrite {
    rewrite ^/(.*)$ /index.php?q=$1;
    }
    location ~ ^/sites/.*/files/styles/ {
      try_files $uri @rewrite;
    }


}
```

8.5 Anexo 5: CÓDIGO DE CONFIGURACIÓN SERVIDOR WEB HHVM 03

****

```
server {
     listen 9002;
     server_name www.drupal.com.co;
     root /usr/share/nginx/drupal;
     index index.php index.html index.htm;


     location / {
```

```
     try_files $uri $uri/ /index.php?$query_string; # For Drupal
>= 7


     }
     location ~ '\.(hh|php)$|^/update.php' {


       fastcgi_keep_conn on;
       fastcgi_pass   unix:/var/run/hhvm/hhvm.sock;
       fastcgi_index  index.php;
       fastcgi_param  SCRIPT_FILENAME
$document_root$fastcgi_script_name;
       include        fastcgi_params;
    }


    location ~ ^/sites/.*/files/styles/ { # For Drpal >= 7

        try_files $uri @rewrite;

    }


}
```

8.6 Anexo 6: CÓDIGO DE CONFIGURACIÓN CLÚSTER MARIADB NODO 1

****

```
# MariaDB-specific config file.

# Read by /etc/mysql/my.cnf

[client]

# Default is Latin1, if you need UTF-8 set this (also in server
section)

#default-character-set = utf8


[mysqld]

# * Character sets

#

# Default is Latin1, if you need UTF-8 set all this (also in
client section)

#

#character-set-server = utf8

#collation-server      = utf8_general_ci

#character_set_server   = utf8

#collation_server       = utf8_general_ci



query_cache_size=0

binlog_format=ROW

default-storage-engine=innodb

innodb_autoinc_lock_mode=2
```

```
query_cache_type=0

bind-address=0.0.0.0

# Galera Provider Configuration

wsrep_provider=/usr/lib/galera/libgalera_smm.so

#wsrep_provider_options="gcache.size=32G"


# Galera Cluster Configuration

wsrep_cluster_name=mycluster

wsrep_cluster_address="gcomm://172.16.9.50,172.16.9.51,172.16.9.51
"

# Galera Synchronization Congifuration

wsrep_sst_method=rsync

#wsrep_sst_auth=user:pass

# Galera Node Configuration

wsrep_node_address="172.16.9.50"

wsrep_node_name=node1
```

8.7 Anexo 7: CÓDIGO DE CONFIGURACIÓN CLÚSTER MARIADB NODO 2

```
**********************************************************************************************

# MariaDB-specific config file.

# Read by /etc/mysql/my.cnf

[client]


# Default is Latin1, if you need UTF-8 set this (also in server
section)
#default-character-set = utf8


[mysqld]
#

# * Character sets

#


# Default is Latin1, if you need UTF-8 set all this (also in
client section)


#


#character-set-server  = utf8

#collation-server      = utf8_general_ci

#character_set_server   = utf8

#collation_server       = utf8_general_ci
```

```
query_cache_size=0

binlog_format=ROW

default-storage-engine=innodb

innodb_autoinc_lock_mode=2

query_cache_type=0

bind-address=0.0.0.0


# Galera Provider Configuration

wsrep_provider=/usr/lib/galera/libgalera_smm.so

#wsrep_provider_options="gcache.size=32G"

# Galera Cluster Configuration

wsrep_cluster_name=mycluster

wsrep_cluster_address="gcomm://172.16.9.50,172.16.9.51"

# Galera Synchronization Congifuration

wsrep_sst_method=rsync

#wsrep_sst_auth=user:pass

# Galera Node Configuration

wsrep_node_address="172.16.9.51"

wsrep_node_name=node2
```

8.8 ANEXO 8: CÓDIGO DE CONFIGURACIÓN CLÚSTER MARIADB NODO 3

```
**********************************************************************************

# MariaDB-specific config file.

# Read by /etc/mysql/my.cnf


[client]

# Default is Latin1, if you need UTF-8 set this (also in server
section)

#default-character-set = utf8


[mysqld]

#

# * Character sets

#

# Default is Latin1, if you need UTF-8 set all this (also in
client section)

#

#character-set-server  = utf8

#collation-server      = utf8_general_ci
```

```
#character_set_server   = utf8

#collation_server       = utf8_general_ci


query_cache_size=0

binlog_format=ROW

default-storage-engine=innodb

innodb_autoinc_lock_mode=2

query_cache_type=0

bind-address=0.0.0.0


# Galera Provider Configuration

wsrep_provider=/usr/lib/galera/libgalera_smm.so

#wsrep_provider_options="gcache.size=32G"


# Galera Cluster Configuration

wsrep_cluster_name=mycluster

wsrep_cluster_address="gcomm://172.16.9.50,172.16.9.51,172.19.6.52
"


# Galera Synchronization Congifuration

wsrep_sst_method=rsync

#wsrep_sst_auth=user:pass


# Galera Node Configuration
```

```
wsrep_node_address="172.16.9.52"

wsrep_node_name=node3
```