Sistema de distribución de contenido dinámico y en tiempo real

Pere Padial Guiteras

Resumen– Las tecnologías basadas en balanceo de carga y almacenamiento de ficheros temporales en la red se han convertido en una pieza fundamental en la administración de servidores web. El aumento de uso de los dispositivos electrónicos ha provocado un crecimiento masivo del tráfico en internet, con lo que es mucho más fácil provocar una saturación en el entorno si no se utilizan las tecnologías adecuadas. En este trabajo se solucionará un problema de saturación para un proyecto destinado a la educación digital. La solución será hallada mediante la adaptación de la infraestructura para el servicio de ficheros de vídeo en modo progressive download, logrando una reducción en el tiempo de carga y un aumento de un 50% en la velocidad de descarga de datos. Finalmente, se configurará un servidor proxy inverso con la idea de mejorar el rendimiento, pero solo se obtendrá una optimización en el uso de recursos del servidor.

Palabras clave– raspberry, Android, NodeJs, streaming, descarga progresiva, distribución de contenido, proxy inverso, Nginx, procesamiento de ficheros estáticos, TCP, Linux.

Abstract– Technologies based on load balancing and files caching methodologies in the network have become a fundamental property of web server administration. With the increased use of digital devices, the network traffic has increased massively. Because of that, it is easy to overload a server with inadequate use of technologies. This project solves a problem of network overload in the infrastructure of a digital education platform. We solve this problem by setting up a progressive download video file service. As a result, we reduced the load time and increased the download speed. Finally, we install and configure an inverse proxy with the idea of getting a better network performance. However, after all this work we only get a better use of server resources.

Keywords– raspberry, Android, NodeJs, streaming, progressive download, distribution content, reverse proxy, Nginx, static files process, TCP, Linux.

1 INTRODUCCIÓN

PARA el desarrollo de tecnologías de Inteligencia Artificial aplicadas en la educación se desarrolló un proyecto para mejorar la calidad de la enseñanza mediante el análisis del comportamiento de los alumnos, con el propósito de determinar el tipo de contenido en el que se ha prestado mayor atención, para, posteriormente, realizar las clases fomentando las mejores características encontradas.

La infraestructura necesaria para llevar a cabo el proyecto está compuesta por un servidor de bajo rendimiento para realizar las siguientes funciones:Primero, se encarga de la administración y almacenamiento del contenido gráfico perteneciente a las lecciones virtuales. Segundo, es responsable de la monitorización de los usuarios basado en el registro de las actividades realizadas en la aplicación y mostrando los resultados en informes de forma gráfica. Por último, se encarga de la gestión de la seguridad de la red y del control de acceso a la plataforma.

El servidor está compuesto por una raspberry pi 3 con un sistema operativo debian wheezy para procesadores ARM, una interfaz de red de 10/100 base T conectada a un router vía Ethernet, un procesador de 1.2Ghz con cuatro núcleos, un servidor web basado en el lenguaje de programación NodeJS y una base de datos mysql.

El segundo elemento, está compuesto por un conjunto de 30 Tablets Samsung Tab A versión 2016 y tienen la función de mostrar el contenido correspondiente a la lección mediante el uso de una aplicación nativa y personalizada.

• E-mail de contacto: pere.padial@e-campus.uab.cat
• Mención realizada: Tecnologias de la Información
• Trabajo tutorizado por: Andrew Koster (IIIA-CSIC)
• Curso 2017/18

La aplicación se encargará de satisfacer las peticiones de los usuarios, registrar las acciones realizadas y de la gestión de las propiedades del perfil de usuario.
El último elemento es un router que se encargará de posibilitar una vía de comunicación entre la red Local Area Network (LAN) en la que está conectado el servidor y la red Wireless Local Area Network (WLAN) en que se conectan las tablets. Como medida de seguridad, se aplicará un filtro de direcciones MAC con el fin de que solo las tablets pertenecientes a ese filtro sean capaces de interactuar con la red. Además, también se dispone de un firewall para filtrar el tráfico entrante y el tráfico que pasa entre una red LAN a la red WAN. Esta infraestructura de red está adaptada para la intercomunicación de dispositivos de forma local, por lo que no se dispone de acceso a la red global.
Una vez desarrollada la infraestructura necesaria, se descubrió un problema en su funcionamiento: en el momento en que todos los usuarios procedían a la descarga del contenido de una lección con ficheros multimedia, se producía una sobrecarga de red debido a la dimensión del contenido de la lección, con lo que muchos alumnos no eran capaces de utilizar la aplicación por no poder establecer una conexión estable con el servidor.
En este trabajo se realizará un estudio sobre la infraestructura descrita y se implementarán los cambios necesarios con el fin de resolver el problema de sobrecarga. La resolución implica un estudio sobre los tipos de distribución de contenido mediante la red, la elección de la metodología más apropiada para el escenario, la modificación de la programación tanto a nivel de cliente como de servidor para adaptar la nueva metodología de distribución a la infraestructura, la programación y configuración de servidores web basados en eventos y el uso de herramientas para la monitorización del comportamiento de los sistemas y de la red, con el fin de encontrar el cuello de botella y simular el comportamiento del escenario en el momento de carga máxima.
Este proyecto está estructurado de la siguiente forma: Primero, se analizará la metodología de distribución de contenido de vídeo mediante la tecnología progressive download, aplicable gracias a la opción 206 del protocolo Hyper Text Transfer Protocol (HTTP) [1]. Esta tecnología permite reproducir el contenido multimedia a medida que se va descargando el fichero, sin tener que esperar a que se complete su descarga para empezar con su visualización. Seguidamente, se realizará una comparativa entre la metodología de distribución antigua y la nueva para determinar el impacto de las acciones realizadas y valorar si se ha solucionado el problema detectado en el inicio del proyecto. A continuación, con el fin de aumentar el rendimiento del servidor en momentos de mucha carga de red y reducir el tiempo de tratamiento de ficheros estáticos,se realizará un estudio sobre el servidor web Nginx [2] y sobre su uso como proxy inverso. Una vez realizado el estudio, se configurará el servicio. Después, se compararán los resultados obtenidos con el funcionamiento del servidor sin proxy. Finalmente, se realizará una valoración general de los resultados obtenidos en este proyecto para generar una conclusión de todo el trabajo realizado y determinar cuáles son las mejores metodologías de distribución de contenido en la red para servidores de bajo rendimiento.

2 Estado del arte

A medida que avanza la tecnología referente a los dispositivos electrónicos, aumenta la cantidad de productos con capacidad de conectarse a internet y consecuentemente la cantidad de tráfico de red. Debido a este factor, los servidores web han tenido que adaptarse a esta nueva demanda mediante el uso de nuevas tecnologías para poder satisfacer esta nueva demanda. Inicialmente, los servidores estaban basados en hilos, era necesario la asignación de un hilo por cada conexión entrante para poder servir el contenido que se pedía. Cuando esta tecnología no pudo abastecer la crecida de tráfico de internet, se procedió a la creación anticipada de conjuntos de hilos (pools) para poder responder las peticiones más rápidamente, aunque este cambio requería el consumo de muchos recursos a nivel de hardware.
Debido a la necesidad de garantizar una velocidad de descarga a muchas más peticiones sin un consumo de recursos excesivo, en 2004 surgió un nuevo tipo de servidor web, basado en llamadas a funciones asíncronas. En el trabajo de [3] quedo demostrado que estos servidores son más rápidos, consumen muchos menos recursos y puede soportar mucha más carga comparado con los servidores basados en hilos. Esta ventaja se ha logrado gracias al uso de tecnologías de lectura de descriptor de fichero [4] y de la resolución de peticiones de forma asíncrona, sin necesidad de bloquear el proceso del servidor mientras espera respuesta. En este trabajo, se analizará el funcionamiento y la configuración de un servicio web basado en llamadas asíncronas [2] y se demostrará como un servidor de bajo rendimiento puede llegar a resolver muchas peticiones de forma simultánea sin necesidad de un consumo excesivo de los recursos aplicando las metodologías correctas.
Debido al éxito de los servidores basados en llamadas asíncronas, en el año 2009 surgió el lenguaje de programación basado en javascript llamado NodeJS. Esta tecnología permite realizar aplicaciones sin necesidad de configurar ningún servidor web externo para enlazar su funcionalidad con la red. Mientras esta tecnología se iba estabilizando, fueron desarrolladas librerías adicionales que podían realizar acciones de forma asíncrona [5] o bien facilitar la configuración de una nueva metodología de distribución de contenido como el streaming, preloading cache o progressive download.

3 Implementación de descarga de vídeo en progressive download

El método de distribución de contenido en el estado inicial del proyecto se basa en la descarga de un fichero comprimido que contiene todos los recursos referentes al aula. Debido a la dimensión del contenido y a la descarga simultanea por parte de todos los clientes, se produce una sobrecarga en la red que aumenta el tiempo de acceso al contenido o bien no permite la finalización de la descarga.
Las causas principales del problema son la dimensión del fichero de descarga y el método en que este es distribuido por la red, con lo que, tras un debido estudio sobre las posibles metodologías de distribución de ficheros estáticos,

se procedió a la extracción del contenido de mayor ocupación del fichero de descarga inicial formados por archivos de vídeo, imágenes y aplicaciones web para la reproducción de actividades. Con esto, se espera reducir el tiempo de descarga, mejorar el rendimiento de la aplicación y reducir la cantidad de datos de envío.
Se ha elegido servir el contenido de vídeo en modo progressive download por las siguientes razones:

- Reduce drásticamente el tiempo de espera para poder empezar a visualizar el contenido, posibilitando la reproducción instantánea.
- Permite al servidor realizar otras funciones mientras las conexiones de reproducción de vídeo no requieren datos.
- Se puede realizar la reproducción parcial del fichero desde cualquier momento, sin necesidad de descargar todo su contenido.
- La mayoría de sitios web que permiten reproducir contenido multimedia (youtube, Netflix, HBO etc.) utilizan este tipo de tecnología lo que el usuario está familiarizado a este tipo de funcionamiento.
- Se puede adaptar la resolución del vídeo según la velocidad de la red para reducir la cantidad de datos a descargar y evitar tiempos de bloqueo de reproducción.

Para la implementación de esta tecnología se realizarán las siguientes acciones: Primero, se procederá a la extracción del fichero de vídeo al directorio correspondiente, luego se adaptará la infraestructura para que pueda distribuir el contenido multimedia en modo progressive download y finalmente se realizará una comparativa sobre el método de distribución inicial y el que se ha implementado.

3.1 Extracción de ficheros de gran ocupación

Para la inserción del material relacionado con un aula, primero se tiene que rellenar un formulario mediante un navegador, una vez rellenado el formulario, se generará un archivo comprimido en formato epub que contiene todo el material que se utilizará en la lección y un fichero en formato JSON que indica las propiedades de la clase, identificación del profesor que ha generado el archivo, el contenido que contiene y su ruta de acceso.
Seguidamente se procederá a subir el fichero comprimido al servidor NodeJs. Durante el proceso de subida, el servidor registrará en la base de datos las propiedades de la lección y la ruta de imagen previa. En este momento, se realizarán las modificaciones pertinentes para la extracción de los ficheros de mayor ocupación.
La extracción se realizará con el uso de objetos de tipo promise [5] que son una representación de funciónes asíncronas javascript con la peculiaridad de que una vez terminada su tarea, tiene devolver obligatoriamente un objeto del mismo tipo para que la función que se ejecute posteriormente pueda trabajar con el resultado.
El uso de este tipo de objetos permite ordenar el flujo de las funciones asíncronas de forma mucho más ordenada y estructurada como se puede ver en la imagen 14 del apéndice.
Con el fin de trabajar con este tipo de objeto, se usará la librería bluebird que también es capaz de transformar todas las funciones pertenecientes a una librería externa a objetos de tipo promise de forma automática.
Una vez se ha introducido toda la información de la lección en la base de datos, se procederá a la extracción del fichero en un directorio temporal, con un nombre formado por un número aleatorio modificado con una función hash MD5 para prevenir sobreescrituras en caso de subida simultánea.
Una vez descomprimido el fichero, procederemos a la lectura del directorio para mover los ficheros de vídeos y de actividades con el uso del lanzamiento de comandos de sistema mediante funciones javascript.
A continuación, se modificará la ruta de acceso de los ficheros extraídos. Para ello, se realizará una lectura del fichero package.json que es donde se especifica todo el contenido perteneciente a la lección junto con sus propiedades y ruta de acceso. Una vez encontrado el parámetro a modificar, se construirá la nueva ruta de acceso y se escribirá en el fichero.
Por último, se ha configurado el servidor para que trate los recursos multimedia en modo progressive download. Para ello, se ha utilizado la librería express, concretamente la función express.static utilizada para gestionar los ficheros estáticos según la metodología que se utilizará en este proyecto.

3.2 Descarga del vídeo en streaming

Para la reproducción del vídeo se utilizará una página HTML5 ya que tiene incorporada toda la lógica necesaria para la reproducción en modo progressive download. Aun así, es necesario la adaptación de esta página al estilo de la aplicación y también la creación de una nueva actividad con un visor HTML en Android.
El funcionamiento de la reproducción a nivel de protocolo HTTP del vídeo se muestra en la figura 1:

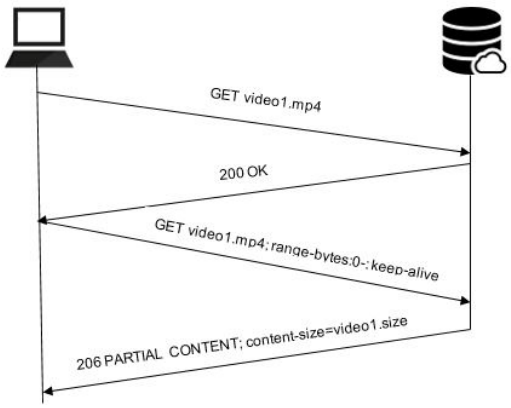

Fig. 1: ejemplo de funcionamiento http 206

Como se puede observar en la imagen 1, la petición que manda la opción range-bytes: 0- indica que se desea leer el fichero de inicio a fin. Seguidamente, el server abre un descriptor de fichero para proceder a la lectura según los limites indicados y responderá con un paquete HTTP 206 que indica que se mandará el fichero por partes y que cada parte puede ser reproducida sin necesidad de esperar a la

descarga completa del fichero. Todos los datos de envío del fichero se realizarán con el uso del protocolo TCP, con la que se generará una conexión keep-alive hasta que se descargue todo el fichero. Una vez se llene el buffer de TCP, se mandarán los datos a la capa de aplicación para que estos puedan ser reproducidos.

3.3 Comparativa de descarga de fichero inicial

Una vez desarrollado los cambios necesarios, se procederá a la comparación de rendimiento entre el nuevo método de distribución y el antiguo. Para la creación del escenario de pruebas, se usará el programa apache-jmeter [6] que se encargará de la creación de peticiones simultaneas y de evaluar el estado de la red mediante la obtención de los tiempos de respuesta. Al mismo tiempo, se usará el comando top en el servidor para hacer una evaluación de su estado según la cantidad de consumo de memoria y de CPU necesarios para satisfacer todas las peticiones de la simulación, en este caso solo se ha medido el consumo exclusivo de las aplicaciones, no del sistema general. Las descargas se realizarán desde 10, 30 y 50 peticiones simultáneamente.

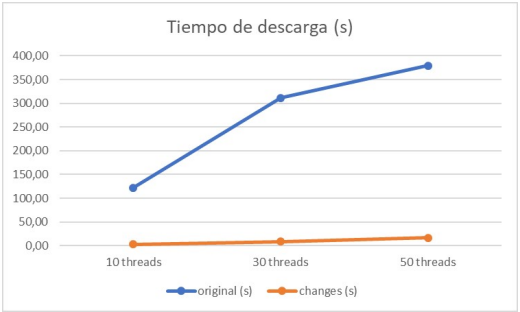

Fig. 2: Tiempo de descarga

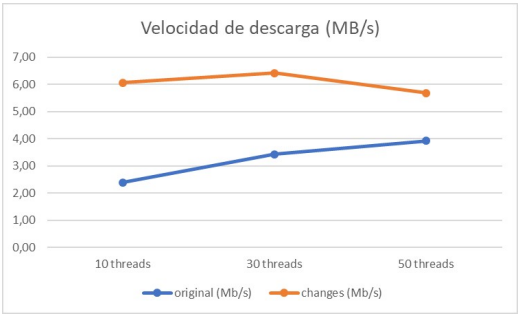

Fig. 3: Velocidad de descarga

En general, el tiempo de descarga (figura 2) se ha reducido se ha reducido de forma drástica (5 minutos aproximadamente), tiempo en que el usuario tenía que estar esperando sin poder realizar ninguna acción. Esta mejora se ha logrado gracias a la reducción del fichero inicial pasando de 48MB a 2MB. Mencionar que no solo cuenta el tiempo de descarga, también hay un tiempo de descompresión del fichero con lo que se tendrían que añadir unos segundos adicionales. En la figura 3 se refleja el aumento la velocidad de descarga en un 50% en los casos de 10 y 30 peticiones simultáneas. Como el fichero a distribuir es mucho más pequeño, se reduce la saturación de la red y es posible almacenar el fichero en memoria temporal, con lo que se reduce el tiempo de tratamiento de las respuestas. Aunque este tiempo se degrada dependiendo del número de usuarios, al solo disponer de 30 clientes en la infraestructura, se puede considerar que el rendimiento es más que suficiente para tener un buen funcionamiento.

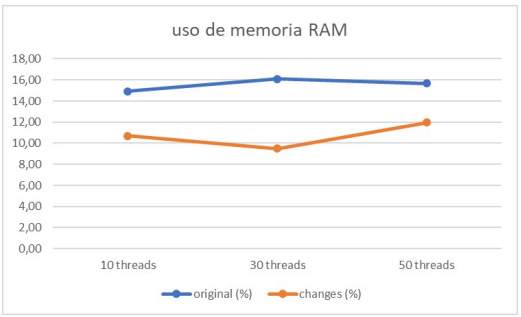

Fig. 4: uso de memoria RAM

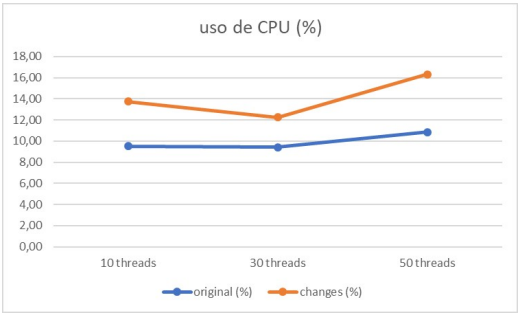

Fig. 5: uso de CPU

El consumo de CPU (figura 5) ha aumentado un 2% ya que es capaz de tratar las peticiones de forma más rápida y el proceso del servicio web no está tanto tiempo bloqueado por lectura de fichero en disco. El uso de memoria (figura 4) se ha reducido al tener que almacenar en memoria un fichero de menor tamaño y no necesitar la sobreescritura de bloques porque el sistema tiene recursos suficientes para el almacenamiento temporal del fichero completo.
En conclusión, los resultados obtenidos han sido los esperados antes de la realización de las pruebas, con lo que, en términos de rendimiento, el problema de acceso al material de cada lección ha sido resuelto. Se ha mejorado sobre todo en el tiempo de carga de la aplicación, el usuario ya no tendrá que esperar durante más de 5 minutos para poder acceder al contenido de la lección. También se ha reducido el uso de la memoria RAM con lo que el servidor dispondrá de más recursos para realizar otras acciones.
Queda pendiente verificar si el servidor es capaz de resolver las peticiones de vídeo en el momento en que los clientes empiezan a reproducirlo para determinar si realmente se ha resuelto el problema de congestión.

3.4 Análisis de reproducción de vídeo en progressive download

En esta sección se va a proceder al análisis del entorno en el momento en que los usuarios empiezan a reproducir el vídeo. Para la realización de esta prueba se procederá a la descarga de vídeo sin el método de distribución a probar, porque la herramienta de simulación no tiene esta funcionalidad, por lo que se realizará un cálculo de la velocidad mínima constante, para compararlo con la velocidad real y determinar si es posible lograr una reproducción sin cortes por falta de datos.
Suponiendo que 30 usuarios quieren reproducir reproducir un vídeo de 5 minutos de duración y 43MB de tamaño sin cortes de reproducción, tienen que descargar a una velocidad de 4.3MB/s constante. Aunque la velocidad de descarga no es lineal, lo tomaremos como un valor de referencia mínimo. En la tabla 1, se muestran los tiempos de descarga del vídeo con 10, 30 y 50 usuarios respectivamente.

TABLA 1: VELOCIDAD DE DESCARGA DEL VÍDEO

Conexiones	**10**	**30**	**50**
Velocidad mbps	5.6	5.532	4.517

En general, se ha logrado una velocidad suficiente para la reproducción sin pausas del vídeo de mayor ocupación de las lecciones disponibles. No obstante, en el caso de 50 peticiones simultaneas, la reproducción podría cortarse por falta de datos en algún cliente.
En conclusión, se ha demostrado que ha quedado resuelto el problema de congestión del servidor mediante el método de distribución de contenido en progressive download. El usuario ya no tendrá que esperar excesivamente para poder iniciar el contenido de la lección ni para la reproducción del vídeo, la red irá menos saturada durante el uso de la aplicación y el servidor dispondrá de más recursos para la realización otras tareas.

4 CONFIGURACIÓN DE UN SERVIDOR PROXY INVERSO

En el apartado anterior, se han obtenido unas métricas de velocidad de descarga muy limitadas en lo que el servicio de ficheros de vídeo se refiere. Para realizar las pruebas de rendimiento se ha utilizado el fichero de vídeo de mayor dimensión disponible, pero en el caso de que fuese necesario servir un fichero mayor dimensión, lo más probable es que la velocidad de descarga se decremente y la probabilidad de provocación de pausas de reproducción debido al bajo ritmo de descarga aumente.
En esta fase del proyecto se ha procedido a la instalación de NGINX: un servidor web muy ligero capaz de resolver muchas peticiones sin necesidad de un consumo excesivo de recursos, gracias a la resolución de peticiones mediante llamadas asíncronas [2].
A comparación con los otros tipos de servidores web, Nginx ha demostrado ser el más rápido en el tiempo de descarga de contenido estático debido a sus cabeceras precompiladas, su método de distribución de carga y realizando un uso eficiente sobre la memoria [7]. Uno de sus usos más habituales es la de servidor proxy inverso, que se encarga de leer todas las peticiones, resolver las que pidan un recurso que ya tiene guardado en memoria, redireccionar las peticiones de contenido dinámico y filtrar la cantidad de paquetes entrantes para evitar ataques de deshabilitación de servicio.
En nuestro entorno, Nginx se ha instalado con fines de proxy inverso, pero con la diferencia de que se encargará de servir los ficheros estáticos de forma directa, y no como cache intermedia, de esta forma, se hará un balanceo de carga de los recursos entre los dos servicios web existentes. Al finalizar esta fase, se espera un aumento de la velocidad de descarga debido a la separación de funcionalidades del servidor y de la capacidad de tratamiento de ficheros estáticos de Nginx. Todo esto a cambio de un aumento de uso de los recursos hardware debido a la coexistencia entre los dos servidores activos.
La prioridad en el funcionamiento de Nginx es eliminar los tiempos de bloqueo, por ello, se basó su funcionalidad en eventos descrita a continuación: Primero, el proceso del servicio está en espera de recepción de peticiones y será despertado en el momento en que reciba una llamada asíncrona conforme ha llegado una nueva petición. Una vez recibido el evento de llegada de una nueva solicitud, creará un socket para establecer la comunicación. A continuación, el servidor enviará una respuesta por el canal y se pondrá automáticamente en modo de espera. En el momento en que recibe un evento de respuesta de cliente, espacio en el disco para realización de escritura o haya datos preparados en el buffer de lectura de fichero, el proceso dejará de realizar la acción que estaba desarrollando para procesar automáticamente la acción correspondiente al evento. Para una descripción gráfica del tratamiento de peticiones, mirar la imagen 15 del apéndice
Otra ventaja que se puede obtener de usar Nginx como servidor inverso es que se puede usar a modo de cortafuegos, añadiendo seguridad extra en el servicio, ya que es capaz de descartar tráfico repetitivo o peticiones excesivamente grandes para evitar ataques de denegación de servicio. Como medidas adicionales se puede restringir el número máximo de conexiones por IP, el tiempo máximo de acceso desde un equipo concreto, asignar un espacio limitado para las conexiones consecutivas o asignar un tiempo límite por conexión keep-alive.

4.1 Configuración del servidor

NGINX permite personalizar su funcionamiento de forma manual para adaptar su forma de trabajo a la máquina en la que está instalado. Tiene muchas opciones disponibles para su configuración relacionados con la gestión de ficheros estáticos, seguridad de la red y de balanceo de carga. Después de un estudio sobre la mayoria de opciones disponibles relacionadas con la distribución de ficheros, el análisis sobre el impacto de mejora en el entorno y la realización de pruebas de rendimiento se ha configurado el servidor de la siguiente forma:
Se ha habilitado el mecanismo epoll [4], que es un conjunto

de llamadas a sistema aplicables en el kernel de Linux con versiones superiores a la 2.6. Esta tecnología permite la administración de los descriptores de fichero de forma asíncrona, ya que tiene constancia de cuál ha sido el descriptor que ha realizado la llamada. A diferencia de los mecanismos más antiguos (poll, set) que cada vez que se lanzaba un evento, el servidor tenía que recorrer la lista de descriptores con el fin de encontrar el origen del evento. Se ha activado esta opción porque las llamadas epoll permiten reducir el tiempo de gestión de peticiones independientemente del número de conexiones abiertas en el equipo, con lo que la complejidad algorítmica se reduce de O(n) a O(1).
Mediante la opción sendfile, habilitamos una llamada de sistema que permite la transmisión de datos entre descriptores de fichero, con lo que no es necesario escribir los datos en memoria temporal para poder compartirlos con otros descriptores de fichero. Esta tecnología hace uso de la llamada de sistema mmap [8] que permite mapear fragmentos de un fichero directamente a memoria RAM, con lo que se pueden realizar operaciones de escritura y lectura de fichero directamente en memoria y así evitar la lectura del disco. Por otra parte, podemos limitar el tamaño de los paquetes de intercambio de datos, en nuestro caso se ha limitado a 512KB evitar sobreescrituras de memoria. El uso de esta métrica permitirá una mejora en la velocidad de gestión de documentos de poca ocupación, como pueden ser ficheros HTML, imagenes, hojas de estilo etc.
El parámetro tcpnopush junto con la opción anterior permite el envío directo de datos independientemente del tamaño del paquete leído. Con ello evitamos esperas para rellenar el paquete hasta la cantidad máxima del MSS (medida máxima del paquete que permiten las redes por las que viajarán los datos sin necesidad de fragmentación, excluyendo el tamaño de cabeceras). Esta funcionalidad será útil cuando lleguemos al final de la lectura del fichero, que al no tener capacidad para llenar el paquete hasta que llegue al MSS el sistema no esperará a que este se llene para enviarlo.
Activando la opción tcpnodelay se deshabilita el algoritmo de Nagle. Este algoritmo se diseñó con el fin de optimizar el rendimiento del protocolo TCP para las conexiones de entorno remoto, en los que se envían paquetes de datos muy pequeños. Esta funcionalidad tiene un factor que degrada la velocidad de descarga de ficheros estáticos, y es que en el momento en que recibe datos leídos de un fichero, espera unos 200 ms (según la implementación en sistemas Linux) para ver si tienen que llegar más antes de la recepción de la confirmación de datos del cliente. Habilitando esta opción, podemos enviar los paquetes de datos sin vernos afectados por los tiempos de espera de este algoritmo. Mediante la activación de esta opción se espera reducir el tiempo de espera en el envío de paquetes definido por el algoritmo de transmisión.
La directiva Directio especifica que se realizarán las lecturas de fichero de forma directa (enviando datos de disco al proceso que realiza la lectura). Como el envío de datos se hace de forma directa, se posibilita la lectura asíncrona del fichero, así el proceso gestor del servidor no queda bloqueado durante el procedimiento de la acción. Se ha configurado está opción para que se active con la lectura de ficheros mayores a 10MB porque esta metodología deshabilita la opción sendfile. Gracias a la activación de esta métrica, evitaremos el bloqueo del proceso gestor durante la lectura del fichero y reduciremos la cantidad de recursos de memoria necesarios para realizar la función de lectura.
Para los ficheros que no sean de vídeo, se realizará una compresión en formato gzip con el fin de reducir el tamaño de los datos a enviar y aumentar la velocidad de descarga.
Desactivando la opción Accept mutex se despiertan todos los procesos trabajadores en el momento en que se recibe un evento de petición web. Este comportamiento mejora el trabajo del servidor porque las peticiones serán recibidas de forma consecutiva, por lo que en ese momento el servidor tiene que estar a pleno rendimiento para minimizar el tiempo de carga.
Con el fin de demostrar la mejora que compone cada métrica descrita anteriormente, se ha medido el impacto que ha realizado cada configuración y demostrado porque es el mejor conjunto de parámetros según la arquitectura de nuestro entorno. Para la realización de las pruebas se realizará la descarga simultanea de un fichero de vídeo, codificado con .mp4, calidad 720p, tamaño de 43MB y duración de 5 minutos. Dicha descarga será llamada por 30 peticiones simultaneas. En la figura 16 del apéndice, se pueden observar los resultados obtenidos.
Por defecto, Nginx crea un proceso por núcleo para maximizar el rendimiento de la capacidad de respuesta, el servidor de nuestro entorno tiene 4 núcleos con lo que cada trabajador estará alojado en un núcleo de la CPU diferente. Si se tienen varios procesos en un solo core, se aumenta la posibilidad de bloqueo entre ellos. En el entorno se dispone de dos servicios web en la misma máquina, con lo que la probabilidad de bloqueo de procesos es superior. Por estas razones, se ha reducido el número de procesos de Nginx a la mitad.
Se ha habilitado la métrica open_file_cache que permite almacenar en memoria los metadatos referentes a un archivo. Esto permite la localización directa del fichero, conocer su disponibilidad y propiedades sin tener que realizar una búsqueda por los directorios. Con esta opción se espera reducir el tiempo de servicio de ficheros estáticos.

4.2 Análisis del cambio

En este apartado se hará una valoración de los cambios implementados durante la configuración del proxy inverso para comparar los resultados con el servidor sin uso de proxy.
Para la realización de las pruebas de rendimiento se utilizará un fichero de vídeo de extensión mp4 de unos 43MB de ocupación, calidad de imagen 720p y de 5 minutos de duración aproximadamente. El proceso de descarga se realizará con 10, 30 y 50 usuarios simultáneamente. Durante el tiempo de descarga se medirá la velocidad, el tiempo y el uso de CPU y memoria.

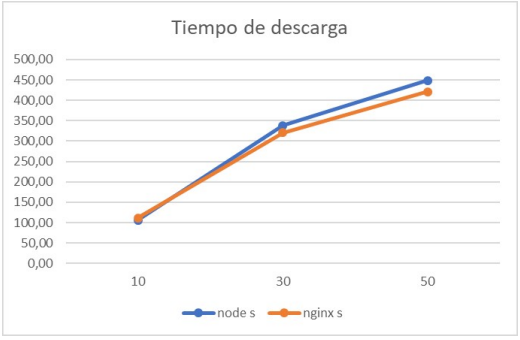

Fig. 6: Tiempo de descarga

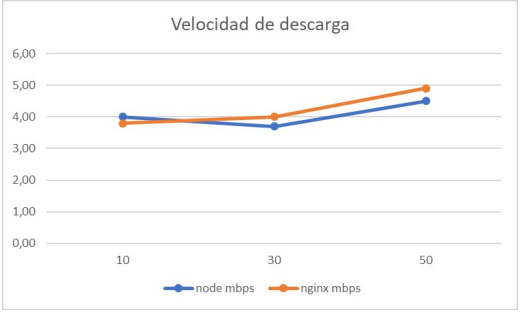

Fig. 7: Velocidad de descarga

Al observar los resultados obtenidos, se puede determinar que contrariamente a lo esperado, no se ha reducido el tiempo y velocidad de descarga. Esto es debido a que ambos servidores utilizan formas similares para resolver las peticiones entrantes, por ejemplo, la librería utilizada para el servicio de ficheros estáticos de NodeJS, utiliza el sistema de lectura de datos epoll y en la lectura de fichero, crea un hilo para no tener que bloquear el proceso durante la acción. Debido a esta metodología se sacrifican más recursos del equipo a cambio de mayor velocidad de descarga tal y como se puede observar en las figuras 8 y 9
También es posible que los valores obtenidos estén limitados por otro factor de la infraestructura, y no deje obtener la diferencia de rendimiento en los dos escenarios.

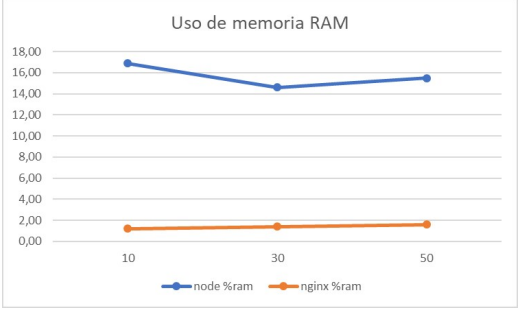

Fig. 8: uso de memoria RAM

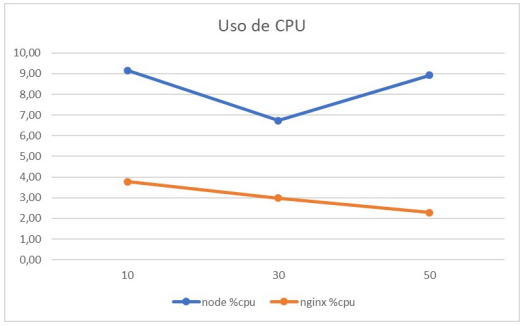

Fig. 9: uso de CPU

Donde Nginx destaca es en el uso de los recursos del servidor, este es capaz de gestionar una gran cantidad de peticiones provocando el menor impacto en el funcionamiento. Esto es debido a que el servicio cuenta con dos procesos para gestionar las peticiones, todos ellos alojados en cada core de la CPU con el fin de minimizar el tiempo de gestión de sus elementos. Contrariamente, Node solo dispone de un proceso y del uso de hilos en caso necesario con lo que provoca un mayor uso de CPU para la creación y el control de los hilos.
En lo que al uso de memoria RAM se refiere, Nginx obtiene una clara ventaja frente a su competidor. La razón es debida a la directiva directio. Como se ha configurado el servicio de realizar una lectura directa de disco a CPU para ficheros mayores de 10MB, los datos transmitidos no pasan por memoria provocando una reducción drástica en su uso.
Durante la realización de estas pruebas, solo se ha usado el servidor para la descarga de ficheros, pero cuando se utilice en el entorno real también tendrá que ser capaz de realizar otras acciones como el control y la monitorización de los usuarios, la gestión de la base de datos, la ejecución de otros servicios etc. Por eso, un correcto uso de los recursos disponibles es muy importante para asegurar el máximo rendimiento posible.
Nginx es un servicio web optimizado para soportar muchas más peticiones de las que se han realizado en las pruebas anteriores y para demostrarlo, realizaremos una prueba de descarga de una página web de 8Kbytes (junto con css y javascript) desde 1000, 5000, 1000, 50000 usuarios consecutivamente.
Los resultados obtenidos se muestran en la figura 10:

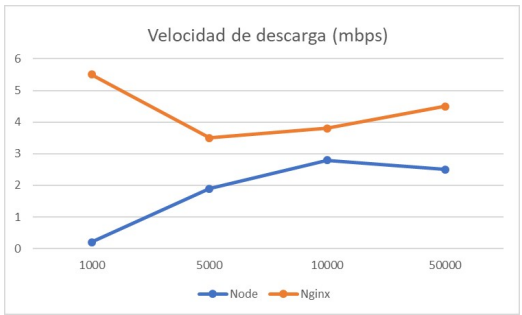

Fig. 10: velocidad de descarga

Una vez analizados los datos obtenidos, queda demostrado que el servidor Nginx está mucho más capacitado que Node en cuanto a la resolución de peticiones masivas gracias a su configuración para el tratamiento de los

ficheros estáticos.
En conclusión, no se ha logrado una mejora en el campo de velocidad y tiempo de descarga tal y como se esperaba, pero se ha reducido el uso de los recursos para que puedan ser utilizados en la realización de otras tareas de monitorización o gestión de servicios y así, asegurar un mejor funcionamiento durante su uso en el entorno productivo.

5 AUMENTO DE LA CAPACIDAD DEL ANCHO DE BANDA

En las pruebas anteriores no se ha logrado utilizar ni un 10% del ancho de banda máximo teórico, que serían unos 100 mbps limitados por la tarjeta de red que tiene el raspberri. Por ello, es necesario realizar un estudio de la infraestructura para detectar el elemento que limita el rendimiento de la red. Este estudio se basará en el análisis de rendimiento de la infraestructura según el uso de memoria, el funcionamiento del router que conecta los dispositivos en ambas redes y análisis de rendimiento del protocolo TCP usado para el intercambio de datos.

5.1 Nginx como caché intermedia

El uso de la memoria RAM es mínimo, con lo que se puede aprovechar estos recursos para la mejora de rendimiento del servicio web. Esta es la razón por la que se ha decidido ha reconfigurar el servidor proxy para que almacene de forma temporal el contenido de las respuestas. Con este cambio se espera una reducción del tiempo de respuesta de cada petición al ya tener almacenados los datos en memoria.
Para configurar nginx como caché intermedia, se ha reconfigurado el servidor eliminando las métricas de optimización de lectura de fichero (sendfile, directio, tcpnopush etc.) y se han añadido nuevas opciones:

- Modificación de la configuración del servidor para que solo redirija y filtre el tráfico entrante.
- Añadir la métrica proxy_cache_lock on; que solo permite una lectura de disco de un fichero independientemente del número de peticiones que lleguen, con lo que estas tendrán que esperar hasta que no esté cargado el fichero en memoria.
- Añadido de parámetros necesarios para el almacenaje de los datos temporales, como el directorio que gestiona la memoria, tamaño máximo del directorio, medida máxima del fichero que va a ser almacenado etc.

Una vez realizados los cambios necesarios, se ha procedido a la realización de pruebas de estrés para determinar su impacto de mejora. Para su realización se ha seguido el mismo patrón que las pruebas anteriores, usando un fichero de vídeo mp4 de 43 MB y descargado por 30 peticiones simultaneas.

TABLA 2: MÉTRICAS DE RENDIMIENTO NGINX COMO CACHE INTERMEDIA

Método	mbps	cpu %	ram %
Proxy cache	2.9	50	120
Cache + sendfile	3.2	47	120
NodeJS	6.4	20	14
Nginx	6.5	3.6	4

Como se puede observar en la tabla 2, el uso de la cache intermedia no ha resultado ser muy efectiva en la velocidad de descarga, debido a la sobrecarga de trabajo al necesitar dos servidores para responder las peticiones. La razón por la cual se hayan obtenido peores resultados es la siguiente: En el momento en que llega una petición, NodeJS usa tecnología de caching para la lectura de fichero, luego Nginx vuelve a almacenar el mismo contenido, con lo que provoca un uso excesivo de memoria (mayor a la que tiene el servidor asignada, con lo que tiene que usar la memoria swap) esto provoca una degradación tanto a nivel de memoria como a nivel de velocidad de descarga. Además, la dimensión del fichero de la petición y la falta de recursos para poder realizar un almacenamiento temporal, son factores que provocan el uso de memoria swap y consecuentemente una pérdida de rendimiento.
En cambio, si realizamos las pruebas con la descarga de un fichero de tipo html, la metodología de caching ha resultado ser más efectiva que la de directa (unos 400kbps de diferencia en la velocidad de descarga), pero con un coste muy mayor de uso de CPU y RAM con lo que no es interesante el uso más excesivo de los recursos para conseguir tan poca mejora en la velocidad de descarga.
Como conclusión, se ha determinado que el uso de Nginx como almacenamiento temporal de ficheros no es productivo en entorno con un solo servidor de bajo rendimiento por el uso excesivo de recursos, con lo que Nginx se seguirá usando para servir los ficheros de forma directa como estaba configurado inicialmente.

5.2 Análisis de funcionamiento del router

En este apartado se procederá a la monitorización del funcionamiento del router, empezando por la medida de la diferencia entre la velocidad de envío y la velocidad de recepción entre los dos tipos de redes para ver si en el traspaso de paquetes entre la LAN y la WAN hay algún tipo de firewall que bloquea la velocidad de descarga. Para la realización de la comparativa, se ha usado una herramienta de monitorización instalada en el router que permite visualizar el tráfico en cada tipo de red.

Fig. 11: velocidad de envío ethernet

Fig. 12: Velocidad de descarga en WAN

Como se muestra en las imágenes 11 y 12, el tráfico de red es medianamente estable considerando una red wifi, con una media de 5 mbps tanto en el envío como en la recepción de datos. Si que tiene una variación muy alta, pero es un comportamiento esperado dentro del ámbito del tráfico de red sin cables. La diferencia de velocidad entre las dos redes es mínima, 460 Kbps considerando la velocidad máxima alcanzada.
Con el fin de reducir la diferencia de velocidad en las dos redes, se ha deshabilitado el firewall y también un filtro de paquetes cuando estos pasaban de una WAN a una LAN. Por otra parte, se ha forzado el uso del estándar de Wireless 802.11n para intentar superar los 5mbps de media. Como resultado, no se ha obtenido una mejora considerable en la velocidad de envío (tan solo unos 100 kbps de mejora) y contando que la velocidad de descarga en una red no es lineal, no se considerará el dato anterior como una mejora. Por lo tanto, se puede descartar la funcionalidad del router como posible cuello de botella.

5.3 Análisis de las conexiones TCP

En este apartado se analizará funcionamiento del protocolo TCP en momentos de mucha carga de datos para determinar si la configuración a nivel de envío penaliza la velocidad de descarga. Para realizar la prueba, se dispone de una herramienta de captura de paquetes de red llamada wireshark, que se activará en el momento en que se proceda a la simulación de 30 peticiones del vídeo utilizado en las pruebas realizadas anteriormente.
Una vez se ha realizada la captura de paquetes, ha sido posible realizar un análisis de la transmisión de datos TCP:

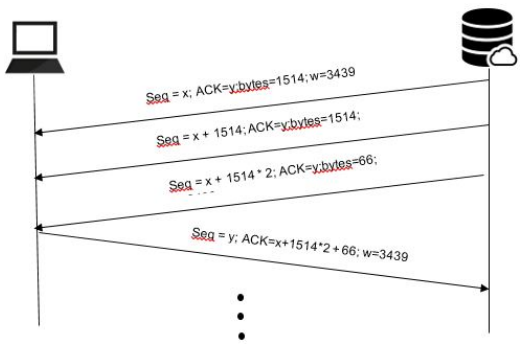

Fig. 13: Velocidad de descarga en WAN

Para medir la calidad de la red, se ha utilizado un método en el que solamente se considera el peor caso. Durante el procedimiento de análisis se han capturado 152.324 paquetes con origen la IP del servidor. Dentro de estos se han detectado 18.114 que piden datos que ya se han enviado previamente gracias a que el programa de monitorización asigna un estado concreto a todas las peticiones que tienen un número de secuencia repetido durante la conexión.
Los valores referentes a la ventana de recepción de datos van variando según la carga de red, en la imagen 13 el valor de la ventana es el asignado al inicio de la conexión (3439 bytes) y consecuentemente su valor mínimo. Considerando que por cada envío fallido se tiene que enviar 3 paquetes de nuevo, teniendo en cuenta que la medida de la window será siempre la mínima, el MSS es de 1514 bytes y siempre se pierde el primer envío (ya que provoca el reenvío de todos los datos enviados sin recibir confirmación de recepción), solo 97.982 (152324 - 18114 * 3) envíos han sido válidos con lo cada uno tiene un 35,67% de que no llegue a su destino correctamente en el peor de los casos.
Como el porcentaje de fallo no es muy grande considerando el uso de redes sin cables, tampoco se puede determinar que este medio sea el cuello de botella.
Como mejora adicional, se ha ampliado el tamaño del buffer usado para el envío de datos de transmisión del protocolo entre la capa de aplicación y la capa de transporte en el modelo TCP/IP. El buffer tiene una medida variable según la cantidad de datos que se envían por cada conexión, esta medida se va adaptando siguiendo las instrucciones del kernel del sistema operativo. El tamaño del buffer es de mínimo 94 KB y de máximo 188 KB, medidas muy pequeñas para el tipo de red que se está usando en el proyecto y para poder servir ficheros de mucha dimensión, con lo que se aumentó la medida máxima del buffer hasta 513 KB, porque es la cantidad de datos leídos por defecto en el momento en que se procede a la lectura directa de un fichero con la directiva de directio, así no será necesario cortar el paquete en trozos más pequeños para poder enviarlo por la red. Estas mejoras no suponen una mejora de forma directa en el funcionamiento de la red, pero reducen el tiempo de envío de datos entra la capa de aplicación y la capa de transporte del modelo TCP/IP.

6 Conclusión

En este trabajo se ha realizado un estudio sobre los métodos de distribución de contenido y se ha decidido que la mejor opción para una infraestructura con un server de bajo

rendimiento es la separación de los ficheros mayor tamaño que contiene todo el material del aula. Posteriormente se han detallado todas las modificaciones necesarias para adaptar esta nueva metodología a nuestro entorno. Se han adquirido conocimientos para la realización de pruebas de estrés mediante el uso del simulador de peticiones apache jmeter y se ha demostrado que el tiempo de descarga se ha reducido en 5 minutos y se ha aumentado la velocidad en un 50%.

Para una mejora de rendimiento, se ha procedido a la instalación y configuración de un servidor proxy inverso, que se encargará de servir los ficheros estáticos, de filtrar las conexiones entrantes para aumentar la seguridad y de redirigir las demás peticiones al otro servicio web. Se ha configurado la gestión de lectura de ficheros asíncrona y las propiedades de las conexiones TCP con el fin de aumentar el rendimiento. Tras los cambios realizados, no se ha producido un aumento de la velocidad de descarga, pero se ha mejorado la gestión de recursos del servidor reduciendo el uso de memoria.

Finalmente, se ha intentado realizar una búsqueda del cuello de botella mediante un análisis de la configuración del router, de las conexiones TCP y de las tecnologías utilizadas para la transmisión de datos con el fin de determinar que el factor que limita más la velocidad de descarga está en el hardware del servidor que utiliza una red de 10 base T logrando un máximo de 10 mbps si se conectan todos los dispositivos en una red LAN. En el entorno real se utilizará una red wifi con una frecuencia de 5Ghz que permite usar más ancho de banda para la transmisión de datos que en las redes de frecuencia 2.4Ghhz. Las pruebas realizadas en este proyecto no se han podido realizar con la red 5G porque el dispositivo que simulaba el entorno de test no tenía capacidad para la utilización de este tipo de frecuencia.

7 Agradecimientos

Antes de finalizar este trabajo, me gustaría mencionar a todas las personas que me han ayudado en la realización de este trabajo, empezando por Andrew Koster por la tutorización y apoyo en este proyecto, a todos los profesores que han pasado durante mi trayecto universitario, en especial a Ian Blanes por ayudarme en el análisis del funcionamiento del HTTP 206, sin olvidarme de todos mis familiares y amigos por el apoyo y la paciencia que han tenido.

Referencias

[1] H. F. The Apache Group, Anselm Baird-Smith. (1999) RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1 - IETF Tools. [Online]. Available: https://tools.ietf.org/html/rfc2616#section-10.2.7

[2] O. Garrett. (2015) Inside NGINX: How We Designed for Performance and Scale. [Online]. Available: https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/

[3] Q. W. Qi Fan, "Performance comparison of web servers with different architectures: A case study using high concurrency workload," *IEEE Std. 1516-2000*, 2015.

[4] L. foundation. (2014) epoll(7) - Linux manual page - man7.org. [Online]. Available: http://man7.org/linux/man-pages/man7/epoll.7.html

[5] M. Hussain, *Mastering Javascript Promises*. Packt Publishing, 2015.

[6] A. S. Foundation. (2001) Apache jmeter. [Online]. Available: https://jmeter.apache.org/

[7] W. Y. Mahendra Data, Muhammad Luthfi, "Optimizing single low-end lamp server using nginx reverse proxy cachingd," *IEEE Std. 1516-2000*, 2017.

[8] R. Love. (2018) Mapping Files into Memory. [Online]. Available: https://www.safaribooksonline.com/library/view/linux-system-programming/0596009585/ch04s03.html

APÉNDICE

```
//ejemplo de lectura de fichero JSON con promises

function readJSON(filename){
  return readFile(filename, 'utf8').then(JSON.parse);
}
```

```
//ejemplo de lectura de fichero JSON con callback

function readJSON(filename, callback){
  fs.readFile(filename, 'utf8', function (err, res){
    if (err) return callback(err);
    try {
      res = JSON.parse(res);
    } catch (ex) {
      return callback(ex);
    }
    callback(null, res);
  });
}
```

Fig. 14: función callback vs función promise

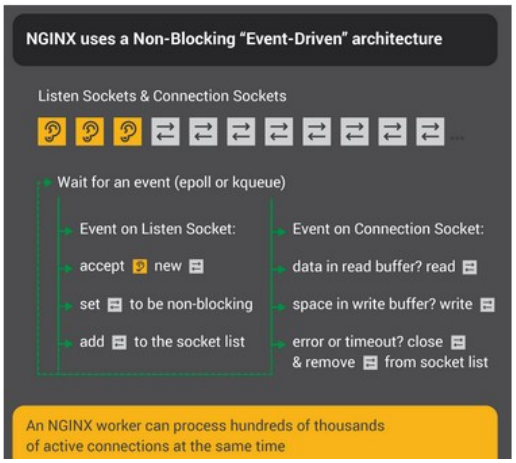

Fig. 15: tratamiento de peticiones en Nginx

use epoll	max chunck 512 Kb	Num workers	Directio 10 MB	Sendfile + push + delay	multi accept	Accept mutex	mbps	Time(s)
	x	4					4.8	263
x		4					4.6	276
x	x	4	x	x			5.04	254
	x	4		x			5.07	248
x	x	4	x	x	x		4.5	281
x	x	1	x	x			3.4	375
x	x	4	x	x	x	x	4.3	292

Fig. 16: Métricas según la configuración Nginx