Universidad Nacional de Luján

Proxy Adaptativo Para Protocolos Web Avanzados

Trabajo Final presentado por Pablo Maximiliano Lulic
para obtener el grado de Licenciatura en Sistemas de Información

2014

Agradecimientos

...

Índice general

Capítulo 1

Introducción

El protocolo HTTP tuvo su primera versión en mayo de 1996, culminando en 1999 con el estándar actual que es HTTP 1.1. Es un protocolo sin estado, lo que conlleva a que se necesite realizar una conexión nueva por cada recurso que se necesite en un sitio.

Los sitios web de la actualidad difieren de las páginas de hace más de 10 años, tanto en tamaño como en cantidad de recursos. Este fenómeno hace que el protocolo ya no tenga el mismo rendimiento que en épocas anteriores. A pesar del avance de la tecnología en cuanto a mejoras en las velocidades de los enlaces de red, un ancho de banda grande no es el único factor que interviene en la performance de la carga de las páginas. Google propuso un protocolo llamado SPDY que busca mejorar la performance de HTTP. Permite múltiples streams en una conexión simple de TCP, además de otras características como push y hint. Aprovechando los resultados favorables de las mediciones de este nuevo protocolo, se busca realizar un proxy que maneje SPDY hacia los nodos interiores, y que hacia la world wide web maneje un algoritmo inteligente que pueda determinar según el perfil del sitio que método (HTTP, HTTPS ó SPDY) elegir para mejorar la performance.

Capítulo 2

Desarrollo de la Web Actual

2.1. Introducción

Internet es la interconexión global de redes individuales alrededor del mundo. Originalmente fué utilizada para interconectar laboratorios dedicados a investigación gubernamental. Desde 1994 se expandió para millones de usuarios de todo el mundo que la utilizan con múltiples propósitos. A medida que iba creciendo, cambiaba la forma de hacer negocios y de comunicarse. Hasta llegar a ser una fuente de información Universal para millones de personas [6].

Internet continúa creciendo día a día. Hacia 1995 la cantidad de usuarios promedio era de 16 millones, 1 año después, la cifra ascendió a más del doble. Para el ao 2000 se incrementó 10 veces la cantidad de usuarios. Esta información puede verse en el Cuadro 2.1.

Fecha	Usuarios(mill)	% Población Mundial
Diciembre, 1995	16	0.4
Diciembre, 1996	36	0.9
Diciembre, 1997	70	1.7
Diciembre, 1998	147	3.6
Diciembre, 1999	248	4.1
Diciembre, 2000	361	5.8
Agosto, 2001	513	8.6

Sigue en la página siguiente.

Fecha	Usuarios(mill)	% Población Mundial
Septiembre, 2002	587	9.4
Diciembre, 2003	719	11.1
Diciembre, 2004	817	12.7
Diciembre, 2005	1018	15.7
Diciembre, 2006	1093	16.7
Diciembre, 2007	1319	20.0
Diciembre, 2008	1574	23.5
Marzo, 2009	1596	23.8
Junio, 2009	1669	24.7
Septiembre, 2009	1734	25.6
Diciembre, 2009	1802	26.6
Junio, 2010	1966	28.7
Septiembre, 2010	1971	28.8
Marzo, 2011	2095	30.2
Junio, 2011	2110	30.4
Septiembre, 2011	2180	31.5
Diciembre, 2011	2267	32.7
Marzo, 2012	2336	33.3
Junio, 2012	2405	34.3
Septiembre, 2012	2439	34.8
Diciembre, 2012	2497	35.7
Marzo, 2013	2749	38.8

Cuadro 2.1: Crecimiento de la cantidad de usuarios de Internet, información extraída de [6]

También puede observarse en el Gráfico 2.1 el increíble crecimiento que tuvo la cantidad de usuarios de Internet, y es clara la tendencia de seguir creciendo.

Además de la cantidad de usuarios, en paralelo iba creciendo la cantidad de Dominios. En sus inicios, la cantidad de dominios era de 19.732 (medición realizada

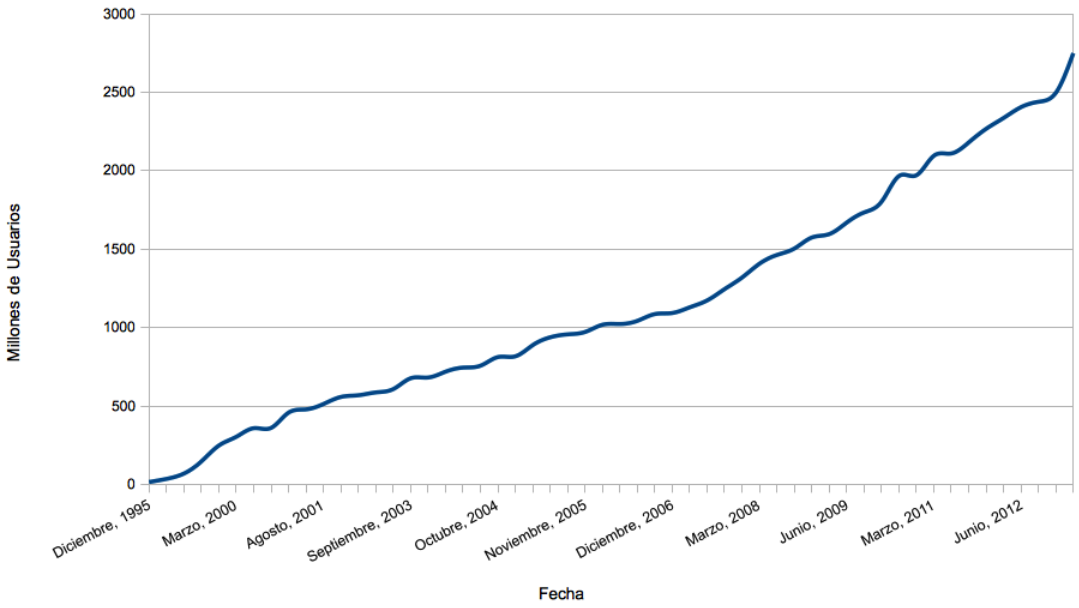

Figura 2.1: Crecimiento de la cantidad de usuarios de Internet, información extraída de [6]

en Agosto de 1995), la medición más actual (Febrero 2014) realizada por Netcraft[1] da un total aproximado de sitios de 920.102.079 [3] (58 millones más que el mes anterior). Esto se puede ver en el Gráfico 2.2.

También fué aumentando el tamaño promedio de los sitios así también como la cantidad de recursos que poseen los mismos. En el ao 1997, el tamao promedio de un sitio era de 60Kb [20], prácticamente era todo texto, pocas imágenes y poca interactividad. Esto fué cambiando con el tiempo, con el avance de la tecnología y de los recursos que se podían compartir en la red. El crecimiento a lo largo del tiempo puede verse en el Gráfico 2.3

Actualmente, el promedio es de 1687Kb[2], su contenido es mucho más variado e interactivo. Hoy los sitios se componen de diversos tipos de recursos, scripts, hojas de estilo, diferentes tipos de imágenes, video, contenido Flash[3], etc. El promedio

[1] NetCraft - http://news.netcraft.com/
[2] Medición extraída de [5] el 1 de Febrero 2014
[3] http://www.adobe.com/

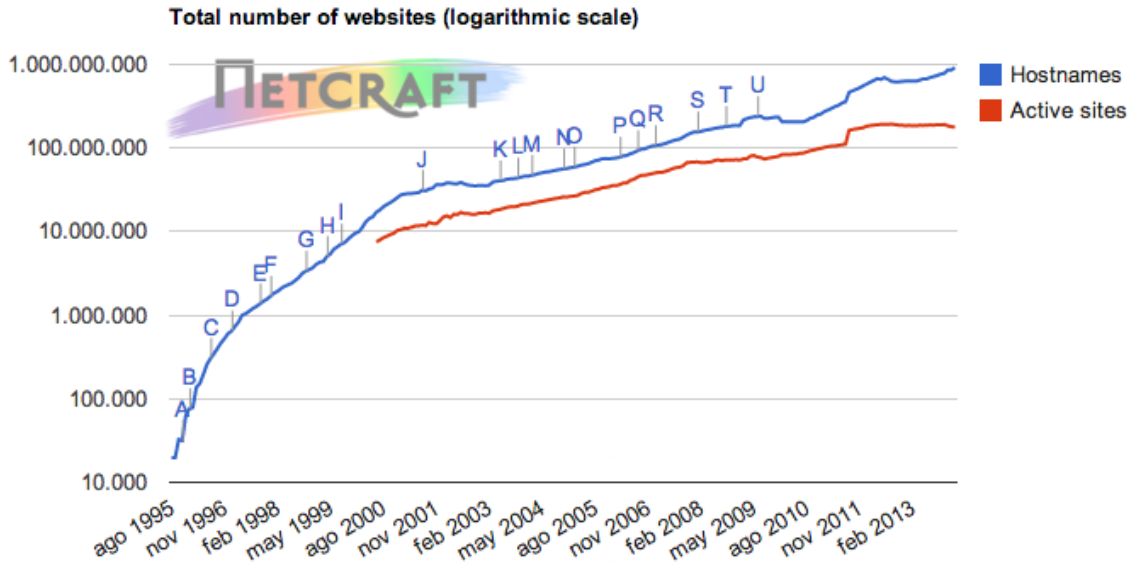

Figura 2.2: Crecimiento de la cantidad de sitios en Internet, extraído de [3]

detallado por contenido de los sitios se pueden ver en el Gráfico 2.4

FALTA: UTILIZACION HTTP HTTPS

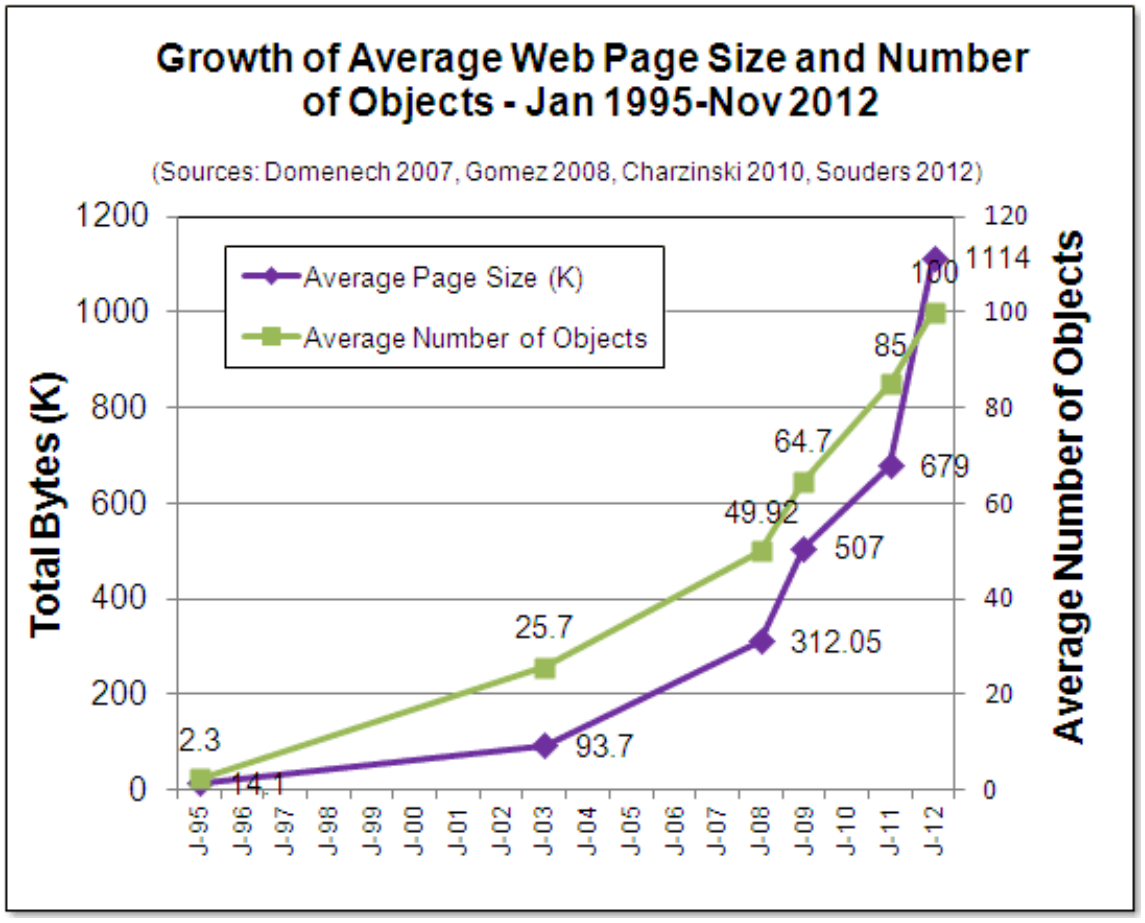

Figura 2.3: Crecimiento del tamaño de los sitios en Internet (1995 a 2012), extraído de [17]

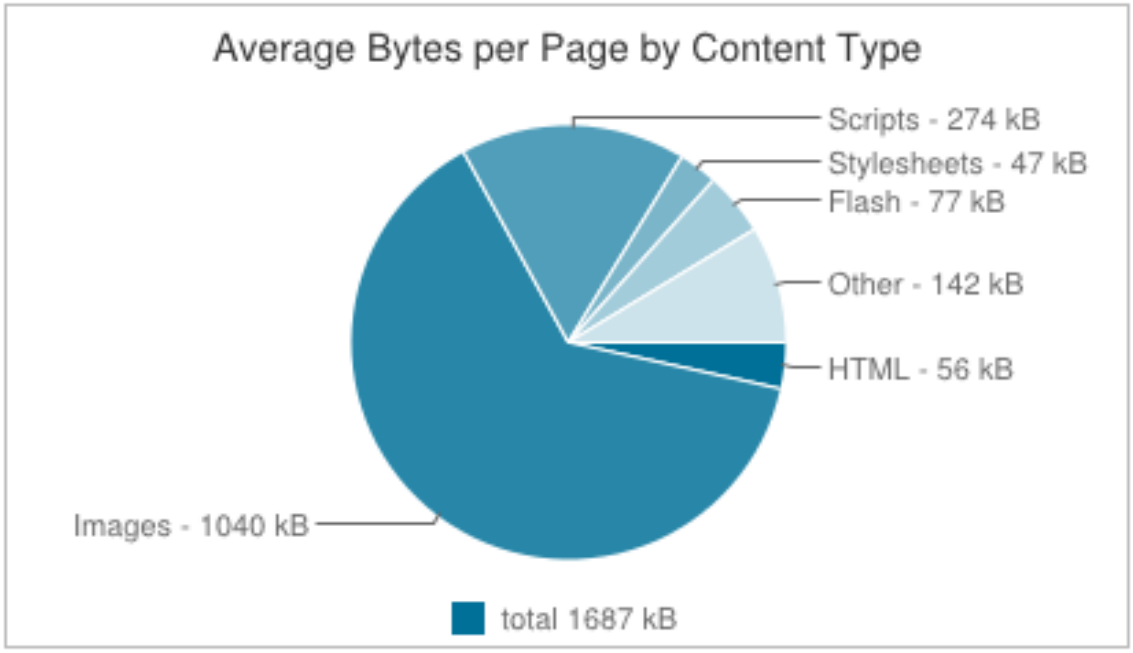

Figura 2.4: Tamaño promedio de los sitios en Internet, detallado por contenido, extraído de [5]

Capítulo 3

Protocolos Web

3.1. Definición

FALTA: INTRODUCCION, DEFINICION DE PROTOCOLOS WEB, MODELO OSI, ETC

3.2. HTTP

Las siglas de este protocolo son por HyperText Transfer Protocol (Protocolo de Transferencia de Hipertexto). Es un protocolo de capa de aplicación[1] que sirve para distribuir información. Fué utilizado en la Web desde el año 1990 en su primera versión (0.9), en la que simplemente se podía transferir texto plano. Su evolución fue el estándar 1.0 en el que se mejoró el protocolo permitiendo que los mensajes usen el formato MIME - Multipurpose Internet Mail Extensions, se incorporaron metadatos acerca de la información transferida y modificadores en la semántica de petición/respuesta. La revisión del protocolo que se usa actualmente es la 1.1, definida en la RFC1626[7]. Contiene nuevos metodos, headers y otras características. Las principales diferencias de esta última definición se pueden ver en [14].

El contenido web reside en Servidores, estos son los que se comunican utilizando este protocolo entre otros. Sirven RECURSOS, estos recursos pueden ser páginas HTML, imágenes, PDF's, video, etc, tanto contenido estático como dinámico (gene-

[1] PONER ACA ALGO DEL MODELO OSI

rado a demanda). Debido a la gran diversidad de contenido que provee un Servidor, es necesario identificar el tipo de recurso que se está enviando. Esto se hace utilizando una etiqueta llamada MIME-Type, que define el tipo de contenido a transferir.

Cada recurso del servidor tiene un nombre, para que los clientes puedan apuntar directamente al recurso deseado. Se nombra con una URL, que tiene el siguiente formato:

PROTOCOLO://SERVIDOR/PATH_AL_RECURSO/RECURSO

El funcionamiento básico es, el cliente envía una petición al Servidor (al puerto 80 por defecto) y este le responde. Esta comunicación se realiza a travéz de mensajes HTTP. Existen diferentes métodos que se pueden utilizar cuando se envía una petición al servidor, tales como

1. GET - El cliente solicita un recurso específico del servidor.
2. POST - El cliente envía datos que van a ser utilizados por el servidor.
3. HEAD - El cliente solicita sólo los Headers (se detallarán más adelante).

Estos son algunos de los métodos, hay otros tales como PUT, DELETE, etc. Según el método, el servidor opera de manera diferente. En la petición se envía el método, el recurso solicitado, la versión del protocolo utilizado, el host, el user-agent[2], entre otros. El Servidor, responde a la petición con una respuesta, que contiene un código de estado de 3 dígitos que le dice al cliente que la petición fue exitosa u otras, por ejemplo 200 (OK) o 404 (Documento no encontrado) de los más comunes.

Los mensajes de HTTP consisten en peticiones y respuestas, sus formatos son similares. Consisten en 3 partes:

1. Linea Inicial - Se indica que hacer en la petición o que fue lo que pasó en la respuesta.
2. Headers - Acá se pueden definir diferentes parámetros por cáda línea con la sintaxis "nombre:valor".

[2]Quién está generando la petición, por ejemplo Mozilla o Safari (browser, proxy, etc.).

3. Cuerpo - Esta parte contiene los datos enviados, ya sea del cliente al servidor o viceversa.

El formato de una petición es el siguiente:

<método> <url del recurso> <versión>

<headers>

<cuerpo>

El formato de la respuesta es el siguiente:

<versión> <estado> <descripción del estado>

<headers>

<cuerpo>

3.2.1. HEADERS

Los Headers, añaden informacion adicional a las peticiones y respuestas. El protocolo define varios Headers, pero se pueden inventar también, los servidores y clientes son libres de hacerlo. Hay diferentes tipos de Headers, entre los que se encuentran:

1. Headers Generales

   Pueden aparecer en peticiones y respuestas.

2. Headers de Peticiones.

   Proveen más información acerca de las peticiones.

3. Headers de respuesta.

   Proveen más información acerca de las respuestas.

4. Entity Headers (ENTIDAD?)

   Proveen información acerca del recurso del mensaje.

5. Headers de Extensión

   Permite agregar nuevos headers que no estén dentro de la especificación.[3]

La definición completa de los Headers se encuentra en la Secci"on 14 de [7].

3.3. HTTPS

Este protocolo, es la versión "segura" de HTTP, a diferencia de este protocolo, añade una capa de cifrado utilizando SSL/TLS sobre TCP. Esto permite que los datos que viajen entre cliente y servidor vayan encriptados, esto se hace antes de enviar los datos por la red. Se distingue fácilmente porque el formato de la URL empieza con https:// y la conexión se hace por el puerto 443 por defecto. Es decir, cuando el browser hace una petición a un servidor, si el esquema es https, inicia la conexión segura con el servidor.

La conexión se hace con otro puerto diferente al de HTTP ya que SSL es un protocolo binario, completamente diferente. Si ambos llegaran al mismo puerto, los servidores interpretarian SSL como HTTP erróneo y cerrarían la conexión.

El procedimiento para iniciar la conexión es el siguiente: El cliente abre una conexión con el puerto 443 al servidor. Una vez que la conexión TCP está establecida, el cliente y el servidor inicializan la capa SSL negociando algunos parámetros criptográficos e intercambiando llaves. Una vez concluida esta negociación, ya pueden empezar a intercambiar mensajes encriptados.

3.4. SPDY

Es un protocolo de aplicación que añade una capa de sesión sobre SSL que permite la transmisión de múltiples Streams[4] sobre una conexión TCP. Especifica un nuevo formato de trama para codificar y transmitir datos. Su especificación se puede ver en [8] y su draft se puede encontrar en [9].

El protocolo HTTP no tiene estado, y, por cada recurso existe la necesidad de abrir una conexión nueva y cerrarla. Esto trae varios problemas. Por cada conexión

[3]PONER LA ESPECIFICACION
[4]Flujo de Datos.

nueva que se hace, se necesitan varios mensajes para establecer la conexión TCP, lo que trae varios RTT adicionales a la comunicación. Retrasos debido al "Slow Start"[5] de TCP. Clientes que evitan realizar múltiples conexiones con el mismo servidor (hasta 6 actualmente PONER REF). A su vez, los servidores crean varios subdominios para almacenar el contenido para que los clientes puedan realizar las peticiones sin tener que evitar las múltiples conexiones a un mismo dominio.

SPDY ofrece por sobre HTTP las siguientes mejoras:

1. Peticiones Multiplexadas. No existen límites de peticiones que se pueden realizar en una sesión de SPDY. A causa de que las peticiones son multiplexadas aumenta la eficiencia del protocolo TCP.

2. Priorización de Peticiones. Los clientes pueden solicitar al servidor cuáles recursos quiere obtener antes que otros. Esto evita la congestión de recursos que no son críticos cuando todavía está pendiente el envío de algun recurso que tiene una prioridad mayor.

3. Compresión de Headers. A causa de que hoy en día los clientes envían mucha información redundante en forma de Headers, como la cantidad de peticiones para obtener un sitio promedio va desde 50 a 100, esta cantidad de información es relevante. Comprimir los Headers reduce el ancho de banda utilizado.

4. Server Push. Al permitir la comunicación bi-direccional a travéz de streams, cualquiera de los 2 (cliente o servidor) puede iniciar un stream hacia el otro. El servidor puede enviar un recurso al cliente antes de que este lo pida[6], esto reduce el tiempo de carga del sitio y disminuye la cantidad de peticiones del cliente.

5. Server Hint. El servidor puede "sugerirle" al cliente que pida un recurso en particular, ya que lo va a necesitar. De todas maneras, el servidor espera a que el cliente peticione el recurso en cuestión antes de enviarlo. Esto reduce el tiempo que tarda el cliente en descubrir cuales son los recursos que tiene que pedirle al servidor.

[5] Se comienza enviando un volumen de datos pequeo hasta alcanzar cierto valor llamado Umbral de Congestión

[6] El servidor conoce de antemano que el cliente va a necesitar el recurso en cuestión.

SPDY se enfoca en la manera en la que se transmiten los datos por la red, preserva toda la semántica del protocolo HTTP. De esta manera, para las aplicaciones se implementa de manera transparente, ya que reside entre la capa de aplicación y la de transporte. Esta sesión es similar al par petición-respuesta de HTTP. Es obligatoria la compresión del mensaje.

Capítulo 4

Problemática

Debido al crecimiento de Internet visto en el Cápitulo 2 y a los problemas que presenta el protocolo HTTP en su implementación (visto en el Cápitulo 3), se busca mejorar los tiempos de carga de los sitios web.

4.1. Técnicas de optimización

Es importante conocer dónde es que el usuario pasa el tiempo esperando en la carga de un sitio web. Según el estudio de Steve Souders en su libro [19], el cliente tarda menos del 20 % para obtener el documento HTML, y el tiempo restante para recibir el resto de los componentes del sitio. Es importante enfocarse en el 80 %, 90 % restante, ya que el tiempo no se desperdicia en descargar el documento HTML ni en el procesamiento que realiza el servidor antes de enviarnos la petición. Esto se resume en la "Regla de Oro de la Performance" de dicho libro:

> *Solo el 10-20 % del tiempo de respuesta del usuario es consumido descargando el documento HTML. El otro 80-90 % se consume descargando todos los componentes de la página.*

Plantea 14 reglas para la optimización de los sitios que se describen a continuación.

1. **Minimizar la cantidad de peticiones.**

La idea básica es eliminar las peticiones al servidor, esto se hace utilizando varias técnicas enumeradas a continuación.

a) Mapa de Imágenes - Permite asociar múltiples áreas en una imagen para que se puedan cliquear y realizar cierta acción (el ejemplo más básico es el de navegar un hipervínculo). En vez de utilizar múltiples imágenes para realizar un menú por ejemplo, se utilizaría una sola mapeada. De esta manera se ahorran las peticiones de las imágenes por 1 sola petición.

b) CSS Sprites - Permite combinar várias imágenes en una sola, luego en el sitio, se define que parte de la imagen combinada desea mostrarse. Por ejemplo, en el sitio de Google cuando se realiza una búsqueda, se utiliza el Sprite que se ve en la Figura 4.1.

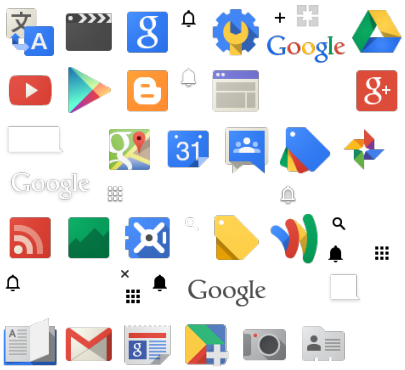

Figura 4.1: Sprite del sitio www.google.com

A simple vista se ven 44 imágenes combinadas. La utilización de este Sprite reduciría todas estas peticiones a 1 sola.

c) Imágenes Inline - Se pueden incluir imágenes en un sitio web sin necesidad de realizar una petición al servidor definiendolas con el esquema *data:* **??** dentro del código HTML de la página.

d) Combinar Scripts y Hojas de Estilo - Como la mayoría de los sitios actuales utilizan Javascript y CSS, es preferible que los Scripts se combinen en uno solo al igual que las Hojas de Estilo. En el caso de tener varios archivos, el navegador va a generar una petición por cada uno de ellos de no tenerlo almacenado en una copia local.

2. **Utilizar un CDN**

   La proximidad del cliente al servidor web tiene un impacto en el tiempo de respuesta de las peticiones. No es lo mismo solicitar un recurso localizado China estando en Argentina que uno ubicado dentro del mismo país. Por ende, si los contenidos están cerca[1] el tiempo de respuesta es menor. Debido a que solo el 10-20 % del tiempo de respuesta se dedica al HTML (visto al inicio de este capítulo), si el resto de los recursos del sitio se encuentran cerca del cliente, se mejorarían los tiempos de respuesta. Para esto es necesario dispersar estos recursos geográficamente.

   Un CDN[2] es una red de distribución de contenidos. Son servidores dispersados geográficamente que ofrecen réplicas de los recursos de un sitio particular para brindarlos al cliente desde el más cercano a su locación.

   Utilizar el servicio que brinda un CDN mejora los tiempos de respuesta de los usuarios.

3. **Añadir Headers para Cachés**

   Cuando un usuario visita el sitio por primera vez, realiza tantas peticiones HTTP como recursos tenga la página. Utilizando Headers Expires o Cache-Control[3] (Capítulo 3, Sección 3.2.1) se puede hacer que los recursos puedan ser almacenados en Cachés (Capítulo 6). Esto disminuye la cantidad de peticiones en una posterior visita a la página del mismo usuario.

   La performance del tiempo de respuesta del sitio mejora según la cantidad de "Hits"[4] que el usuario tenga de los componentes del sitio.

4. **Comprimir los componentes (Gzip)**

   Se puede reducir el tiempo de respuesta, disminuyendo el tamaño de la respuesta HTTP. Esto se puede realizar comprimiendo el recurso que se está solicitando. La reducción del tiempo es mayor en ambientes en donde el ancho

[1] En términos de distancia física.
[2] Content Delivery Networks
[3] Desde HTTP 1.1
[4] Necesidad de solicitar un recurso que ya tengo almacenado en la Caché

de banda es bajo. El formato *Gzip*[5] [4] es el más popular y el más efectivo, el otro formato que se usa con menos frecuencia es *deflate* [2].

El usuario tiene que enviar en la petición un Header *Accept-Encoding* indicando los métodos aceptados. Cuando el servidor recibe este Header, puede comprimir el recurso utilizando alguno de los métodos indicados por el usuario. Este devuelve la respuesta con un Header *Content-Encoding* indicando el tipo de compresión utilizado en el recurso que se está enviando.

Los tipos de archivos que se deberían comprimir son aquellos de texto, tales como HTML, Scripts, CSS, etc. Aquellos formatos que ya se encuentran comprimidos como las imágenes o los archivos PDF no deberían comprimirse ya que se desperdicia tiempo de CPU del servidor[6] y además puede incluso incrementar el tamaño del archivo.

5. **CSS en la parte superior del HTML**

   Las Hojas de Estilo deben incluirse en la parte superior del HTML. De esta manera, los navegadores pueden ir renderizando la página a medida que van llegando las respuestas de las peticiones. Esto es importante en términos de usabilidad, para brindarle un medio visual [7] al usuario que está esperando el sitio.

6. **Scripts en la parte inferior del HTML**

   Cuando se realizan las peticiones al servidor, las descargas pueden hacerse en paralelo con ciertos límites[8], eso claramente es beneficioso, ya que al paralelizar las descargas, el tiempo es menor comparado con descargas secuenciales. En el caso de los Scripts, esta característica se deshabilita por 2 motivos, uno es que si es script altera contenido de la página, el navegador debe esperar a recibirlo para mostrar el contenido correctamente. El otro motivo es que el navegador debe respetar el orden de ejecución de los scripts, si estos vinieran en paralelo, no se puede asegurar que su ejecución tenga el mismo orden en el que se solicitaron.

[5] http://www.gzip.org/
[6] Para realizar la compresión.
[7] Carga progresiva del sitio.
[8] 2 según la especificación HTTP 1.1, hasta 6 según el navegador

7. **Evitar expresiones en CSS**

   Las expresiones en CSS se encuentran deprecadas[9] en los navegadores modernos. Afectaban directamente a la performance de la renderización del sitio una vez que todos los componentes eran recibidos.

8. **Utilizar JavaScript y CSS de manera externa (no embebido en el HTML)**

   Al embeber los scripts y el estilo en el HTML, se minimiza la cantidad de peticiones al servidor, pero se aumenta el tamao del archivo HTML. Incluir archivos externos al HTML, permite a los navegadores y proxys que puedan almacenar en su caché el objeto. Esto es útil ya que si en todas las páginas del sitio se utilizan el mismo Javascript y CSS, al ir navegando las diferentes páginas del sitio, estos recursos ya están almacenados en el navegador. También disminuye el tiempo de respuesta en visitas posteriores.

9. **Reducir las búsquedas de DNS**[10]

   Cada servidor tiene una dirección IP asignada, esta dirección se encuentra asociada a un nombre de Dominio, esto se almacena en los DNS. Cuando se ingresa un nombre de un sitio en el navegador, este necesita la dirección IP asociada a ese Dominio. Para ello necesita solicitar esa asociación a un DNS Resolver, antes de empezar a solicitar los recursos debe esperar la respuesta del DNS. Esto al igual que los recursos se almacenan en una caché local del navegador. A raíz de esta cuestión, tener menos dominios en las URL's de los componentes de una página, requiere menos peticiones (DNS lookups) a los servidores que almacenan los dominios para solicitar las direcciones IP que corresponden a esos dominios.

10. **Minificar el JavaScript**

    La Minificación, es la práctica de remover caracteres innecesarios del código para reducir el tamaño del archivo. Se quitan los comentarios, los espacios en blanco (tabulaciones, espacios, saltos de línea). Una optimización alternativa

[9] http://blogs.msdn.com/b/ie/archive/2008/10/16/ending-expressions.aspx
[10] Domain Name System.

tiene el nombre de Ofuscación, además de lo que hace la Minificación, renombra las variables del codigo por cadenas de texto más pequeñas. Este método disminuye aún más el tamaño de los archivos, que el método visto anteriormente. Con esta optimización se mejora el tiempo de respuesta ya que los recursos tienen menor tamaño.

11. **Evitar redirecciones**

    Una redirección es utilizada para enrutar a los usuarios de una URL a otra. Generalmente se utilizan para documentos HTML, pero, ocasionalmente, también cuando se peticionan componentes de la página. Los retrasos ocasionados por una redirección retrasan directamente el documento HTML completo. Insertar una redirección entre el usuario y el HTML retrasa todo en el sitio.

12. **Remover Scripts duplicados**

    Incluir archivos Javascript más de una vez en un sitio, afecta directamente la performance por 2 factores, peticiones innecesarias y tiempo de procesamiento innecesario del Script.

13. **Configurar Etags**

    Los Etags son cadenas de texto únicas que identifican una versión específica de un componente (ver Capítulo 6, Sección 6.4) . Provee un mecanismo para poder realizar una petición condicional al servidor, comparando el Etag de la copia local con el del servidor. En el caso de que el valor coincida con el del servidor, la respuesta devuelve un código (ver 3.2 REFERENCIAR LOS CODIGOS HTTP CUANDO ESTÉN) que indica que el recurso está fresco (ver 6), de esta manera, se reduce notablemente el tamaño de la respuesta y el tiempo ya que viaja por la red solamente los Headers.

    Hay ciertas cuestiones referidas a la utilización de los Etags. Por un lado, en el caso de un cluster de servidores, los Etags generados por los mismos, no son iguales entre ellos. En este caso, si se realiza una petición condicional, a pesar de que el componente es igual, al no coincidir los Etags, el recurso es devuelto al usuario. Otra cuestión es que la petición condicional *If-None-Match* toma precedencia ante *If-Modified-Since*, según la especificación del protocolo

HTTP 1.1 si ambos Headers se encuentran en la petición, ambas condiciones se tienen que cumplir para devolver un código 304. Por todas estas cuestiones, se recomienda, o bien configurar los Etags de manera que las peticiones sean precisas, o quitarlos.

14. **Hacer que Ajax pueda ser almacenado en Caché**

    SEGUIR

Capítulo 5

Proxy

5.1. Definición

Los servidores proxy son Intermediarios. Se encuentran entre el cliente y el servidor, actúa enviando los mensajes del cliente al servidor y viceversa. En una comunicacin normal, le cliente se comunica directamente con el servidor, en el caso de que en la red haya un proxy presente, el cliente se comunica con el proxy y este es el que se comunica con el servido.

El proxy es un web server y también es cliente. Para recibir los pedidos de los clientes, tiene que actuar como un servidor y manejar correctamente las peticiones, conexiones y respuestas. A su vez, para poder conectarse a los destinos finales y recuperar los recursos que le son pedidos por el usuario del proxy, tiene que actuar como un cliente, enviando peticiones y recibiendo las respuestas. Se puede ver el funcionamiento de un Proxy en la siguiente Figura.

5.2. Usos de un Proxy

Hay diversas utilizaciones de los servidores proxy, entre ellas se destacan:

1. Filtro

   El Proxy al recibir las peticiones de los clientes, puede permitir o denegar esas peticiones según ciertas políticas definidas. Por ejemplo, en una Institución

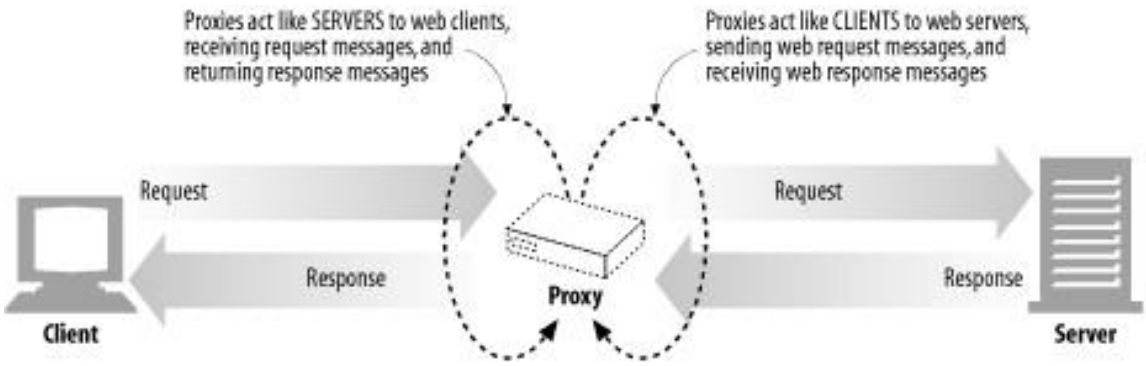

Figura 5.1: Esquema del funcionamiento de un Proxy, extraído de [12]

Educativa se podría bloquear el acceso a ciertas redes sociales o sitios para adultos.

2. Cortafuegos[1]

   Al ser un intermediario, y muchas veces como puerta de acceso hacia internet, se pueden utilizar para aumentar el nivel de seguridad restringiendo ciertos protocolos o filtrando cierto contenido inseguro por ejemplo.

3. Web Caché

   Puede mantener copias locales de los recursos peticionados por los clientes, al servir varios clientes esto incrementa la velocidad de respuesta. Esto se verá detalladamente en el Capítulo 6.

4. Proxy Reverso / Surrogate Server

   Puede actuar delante de un Servidor Web, atendiendo peticiones de los clientes como si fuera él Servidor mismo. Esto permite por ejemplo, generar una Red de Distribución de Contenidos[2]. Recibe una petición de un cliente, y, en vez de devolver el contenido directamente, inicia una comunicación con otros servidores para localizar el recurso pedido de manera más eficiente. FALTA: REVISAR ESTO.

5. Content-Router

[1] Firewall
[2] CDN

Pueden redirigir las peticiones a Servidores basados en las condiciones del Tráfico de Internet y el tipo de contenido. REVISAR

6. Transcoder

   Puede modificar el contenido de los recursos antes de hacer el envío a los clientes. Por ejemplo, transformar imágenes BMP a JPEG para reducir el tamaño, comprimir archivos de texto, e incluso hacer una traducción del contenido a otro lenguaje.

7. Anonimizador

   Permite navegar de manera privada y anónima removiendo ciertos datos de los mensajes HTTP (Dirección IP, From Header, cookies, etc).

FALTA: ARCHIVOS PAC

5.3. SPDYProxy

FALTA: EXPLICAR FUNCIONAMIENTO DEL SPDYPROXY

Capítulo 6

Caché Web

6.1. Definición

Un caché web es un intermediario entre un cliente (o varios) y un Servidor (o varios). Observa las peticiones que se van realizando y va almacenando los recursos solicitados, de manera que, si hay otra petición de la misma URL, el intermediario puede brindar el recurso sin tener que solicitarlo al destino original [15].

FUNCIONAMIENTO BASICO?

AGREGAR HITS,MISS

Posee ciertos beneficios al proveer un mecanismo eficiente para la distribución de Información en la Web [22, p. 20].

1. Latencia

   Un Caché Web cercano a los clientes, reduce la latencia para los aciertos de caché[1]. Los retrasos en la transmisión son menores por la cercanía entre los dos puntos de interacción. Adicionalmente, se reducen los retrasos por retransmisión y las esperas en los routers a causa de que hay menos enlaces intermedios involucrados.

2. Ancho de Banda

   Cuando se utilizan las copias de los recursos almacenados en la caché, se ahorra ancho de banda, ya que no hay que hacer la petición del recurso al destino final.

[1] Cuando un recurso solicitado por el cliente, efectivamente se encuentra almacenado en la caché.

3. Carga del Servidor

   Reduce la carga de los servidores al disminuir la cantidad de peticiones que se realizan. Un servidor que esta con poca carga de trabajo es más rápido que uno que está muy ocupado, es decir, atendiendo gran cantidad de peticiones de diferentes clientes.

4. Si un servidor remoto no está disponible temporalmente, una copia del objeto pedido se puede recuperar de la caché.

6.2. Tipos

Hay diferentes tipos de Cachés Web, entre ellos se encuentran los siguientes:

1. Caché de Navegador

   Los navegadores tienen un caché incluido, la información se almacena de manera local en el dispositivo del usuario. Esta limitado a 1 solo usuario, y se obtiene un hit de caché, solamente cuando se visita nuevamente algun sitio que ya se haya visitado.

2. Proxy Cachés

   A diferencia de los de navegador, estos sirven a un número grande de usuarios. Al crecer la cantidad de usuarios, también crece la tasa de aciertos de la caché, ya que los usuarios suelen acceder a los mismos sitios (sitios de gran popularidad) [11].

3. Surrogates Servers

   Son intermediarios que actúan con la autoridad del servidor original para servir contenidos como si fueran el servidor mismo [16]. Generalmente se encuentran cerca de los servidores originales, sirviendo el contenido de los mismos, generalmente desde una caché interna.

   Se usan como Redes de Distribución de Contenidos (CDN), para tener replicaciones de los recursos de un servidor en diferentes lugares. Los recursos peticionados por los clientes, son revueltos por el CDN más cercano fisicamente.

También se usan como aceleradores, simplemente almacenando en la caché las respuestas del servidor.

6.3. Politicas de Reemplazo / Algoritmos de Caché

Para el buen funcionamiento de la caché, hay que definir de qué manera se van reemplazando los contenidos almacenados en la misma. Para esto se utiliza una Política de Reemplazo de objetos, orientado a ciertas características de los objetos, tales como el tamaño, cantidad de peticiones de dicho objeto, tiempo almacenado en la caché, etc. Hay diversos algoritmos que se detallan a continuación.

1. Primero en Entrar Primero en Salir (FIFO[2])

   Este método es el más sencillo, se trata de una cola en la que el primer elemento en entrar es el primer elemento en salir, es decir, aquel que más tiempo haya estado en la caché es el que más probabilidades tiene de salir.

2. Aleatorio

   Realiza el reemplazo buscando aleatoriamente un objeto almacenado en la caché.

3. Menos Usado Recientemente (LRU[3])

   Este método, busca reemplazar al objeto que más tiempo haya estado en la caché sin ser peticionado, es decir, aquel objeto que menos peticiones haya tenido en un lapso de tiempo amplio, es el que tiene más posibilidades de ser reemplazado.

4. Menor Menos Usado Recientemente (LRU-MIN)

   Es una optimización del algoritmo LRU, en el cual se busca reemplazar los objetos de mayor tamaño de la caché. quedando así objetos de menor tamaño almacenados.

[2]First In First Out
[3]Last Recently Used

5. Menos Frecuentemente Usado (LFU[4])

   Se evalúa la frecuencia con la que un elemento es solicitado, el que es menos solicitado es el que tiene más posibilidades de ser reemplazado.

6.4. Control de Caché

Cuando el cliente realiza una petición, y esta coincide con un elemento que tenemos almacenado en la Caché se debe evaluar si es viable enviar la copia local, o hacer la petición nuevamente, a esto se le llama Revalidación. Para esto se realizan ciertos controles (extraídos de [15]) en base a los Headers de HTTP (vistos en 3.2.1).

1. El Header *Expires*, nos indica cuanto tiempo va a estar "fresco"[5] el recurso. Si en el momento de la petición, todavía no expiró el recurso, puedo servirlo de la copia local. Tiene algunas limitaciones, como por ejemplo que el valor permitido es una fecha en GMT, lo que lleva a que los relojes del Servidor y del caché estén sincronizados, de otra manera no tiene sentido una comparación de fechas. Otro problema que se encuentra es que si se envía este header, pero no está actualizado, siempre va a enviar una fecha anterior a la actual lo que llevaría a generar siempre la petición del recurso.

2. Se puede utilizar el Header *Cache-control*[6] que brinda información variada. El formato es: **Cache-control: especificación1, especificación2, especificaciónN**.

   a) *max-age* - Especifica en tiempo en el que el recurso puede ser considerado "fresco" en segundos. Es el similar a *Expires* pero con un valor específico a diferencia de una marca de tiempo.

   b) *s-maxage* - Similar al anterior pero para los cachés compartidos[7].

   c) *public* - Indica que las peticiones que requieren autenticación pueden almacenarse en la caché.

[4] Least Frequently Used
[5] No va a tener cambios.
[6] Implementado a partir de la versión 1.1 de HTTP
[7] Por ejemplo un Proxy.

d) *private* - Permite que el recurso puede almacenarse en aquellos cachés que son solo para 1 solo usuario[8] y no en una caché compartida.

e) *no-cache* - Fuerza a que el recurso tenga que revalidarse antes de devolver la copia local de la caché al cliente.

f) *no-store* - Indica que el recurso no debe quedar almacenado en la caché bajo ninguna circunstancia.

g) *must-revalidate* - Si el recurso luego de validarlo en la caché no esta "fresco", debe revalidarse.

h) *proxy-revalidate* - Igual que *must-revalidate* pero aplicado a Proxies.

Cuando ambos Headers, *Expires* y *Cache-control* están presentes en la respuesta, *Cache-control* es el que toma mayor importancia.

Para realizar la Revalidación se utilizan ciertos Headers en la petición. Los Servidores utilizan 2 Headers para realizar las validaciones, uno es *Last-Modified*, que indica cuándo fué modificado por última vez el recurso. Cuando éste está presente, el cliente puede realizar una petición con el Header *If-Modified-Since*:fecha, para que el servidor devuelva el recurso unicamente si se cumple con esta condición, de no ser así, devuelve un mensaje con el código `304 Not Modified`, que indica que el recurso no fue modificado y se puede servir la copia local. Por otro lado, se utiliza un identificador llamado *ETag* que se genera cada vez que el recurso se modifica. Es una cadena de texto que identifica una versión específica del recurso. En este caso, el cliente puede utilizar el Header *If-None-Match*:ETag_local, que el servidor responde con el recurso en el caso de que el identificador enviado por el cliente no coincida con el del objeto peticionado.

6.5. En el Proxy

Se implementó como un módulo que se puede agregar al proxy original, posee las poíticas de reemplazo LRU, LRU-MIN y LFU, pudíendose configurar cuando se inicia el Proxy.

[8] Por ejemplo la caché de un Browser.

Capítulo 7

Node

Capítulo 8

Desarrollo de spdyProxy

8.1. Que voy a hacer

8.2. Features a agregar

8.3. Metodos de recuperacion de las paginas (3)

8.4. pseudocodigo de las features

Capítulo 9

Pruebas del Proxy desarrollado

9.1. como se va a probar

9.2. resultados

3 metodos comparados

9.3. resultados

Capítulo 10

conclusiones

Glosario

A

Ancho de Banda

Retraso en la transmisión de datos desde un punto a otro. 28

L

Latencia

Retraso en la transmisión de datos desde un punto a otro. 28

M

MIME - Multipurpose Internet Mail Extensions

Son una serie de convenciones o especificaciones dirigidas al intercambio a través de Internet de todo tipo de archivos (texto, audio, video, etc.) de forma transparente para el usuario. Una parte importante del MIME está dedicada a mejorar las posibilidades de transferencia de texto en distintos idiomas y alfabetos.. 10, 28

N

Netcraft

Es una compañía de servicios de Internet basada en Bath, Inglaterra. Netcraft ofrece análisis de cuota de mercado de servidores y alojamiento web, incluyendo la detección del tipo de servidor web y de sistema operativo.. 28

R

RFC - Request For Comments

. 28

U

URL - Localizador de Recursos Uniforme

Secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación, como por ejemplo documentos textuales, imágenes, vdeos, presentaciones digitales, etc. El formato general de un URL es: esquema://máquina/directorio/archivo. 28

Bibliografía

[1] The "data" url scheme. . URL http://tools.ietf.org/html/rfc2397.

[2] Deflate compressed data format specification. . URL http://www.ietf.org/rfc/rfc1951.txt.

[3] February 2014 web server survey. URL http://news.netcraft.com/archives/category/web-server-survey/.

[4] Gzip file format specification. . URL http://www.gzip.org/zlib/rfc-gzip.html.

[5] httparchive - the http archive tracks how the web is built. URL http://httparchive.org/.

[6] Internet growth statistics. URL http://www.internetworldstats.com/emarketing.htm.

[7] Rfc: Hypertext transfer protocol – http/1.1. . URL http://www.ietf.org/rfc/rfc2616.txt.

[8] Spdy: An experimental protocol for a faster web. . URL http://www.chromium.org/spdy/spdy-whitepaper.

[9] Spdy protocol - draft 1. . URL http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft1.

[10] Web caching: Making the most of your internet connection. URL http://www.web-cache.com.

[11] Duska, Bradley, David Marwood, y Michael Feely. The measured access characteristics of world-wide-web client proxy caches. Diciembre 1997.

[12] David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, y Sailu Reddy. *HTTP: The Definitive Guide*. O'Reilly Media, 2002.

[13] Ilya Grigorik. *High Performance Browser Networking*. O'Reilly Media, 2013.

[14] Balachander Krishnamurthy, Jeffrey C. Mogul, y David M. Kristol. Key differences between http/1.0 and http/1.1. 1999.

[15] Mark Nottingham. Caching tutorial for web authors and webmasters. URL http://www.mnot.net/cache_docs/.

[16] Mark Nottingham y Xiang Liu. Edge architecture specification. URL http://www.w3.org/TR/edge-arch.

[17] Website Optimization. Average web page size triples since 2008. URL http://www.websiteoptimization.com/speed/tweak/average-web-page/.

[18] Carlos E. Quesada Sánchez y Esteban Meneses. Polticas de reemplazo en la caché de web. Diciembre 2006.

[19] Steve Souders. *High Performance Web Sites*. O'Reilly Media, 2007.

[20] Terry Sullivan. How much is too much? URL http://www.pantos.org/atw/35654.html.

[21] W3C. Http - hypertext transfer protocol. URL http://www.w3.org/Protocols/.

[22] Duane Wessels. *Web Caching*. O'Reilly Media, 2001.