Arquitectura de aplicaciones web

Xavier Vilajosana Guillén
Leandro Navarro Moldes

PID_00184783

UOC
Universitat Oberta de Catalunya

Índice

Introducción

En este módulo didáctico se hablará de las maneras de organizar aplicaciones web y de cómo hacer que puedan funcionar a pesar de estar sujetas al comportamiento caótico e imprevisible de Internet.

Primero se caracteriza la demanda de estos servicios y cómo medirla en una situación real. Después se describen las formas de organizar las aplicaciones en servidores web y también se profundiza en su funcionamiento. Seguidamente se presentan formas distribuidas de servicio: servidores intermediarios *proxy-cache*, redes de distribución de contenidos que no dejan de ser extensiones o servicios que facilitan las tareas de los servidores de aplicaciones y que permiten un funcionamiento más óptimo de Internet. Finalmente se presentan las aplicaciones orientadas a servicios y computación bajo demanda que a día de hoy están cambiando el funcionamiento global de Internet.

Objetivos

Las competencias que se alcanzarán fruto del trabajo de este módulo didáctico son las siguientes:

1. Conocer las características de la demanda que tiene que satisfacer un servidor web.

2. Conocer las diversas maneras de organizar una aplicación web y los modelos que existen, según varios criterios.

3. Conocer las características y el funcionamiento de cada modelo.

4. Poder elegir la mejor opción en cada situación y valorar las implicaciones del montaje que hay que hacer.

1. Características de la demanda de páginas web

El tráfico de web es el responsable de un buen porcentaje del tráfico de Internet. Esta tendencia ha ido creciendo gradualmente desde que apareció la web (protocolo HTTP), y hoy día el tráfico HTTP predomina respecto del resto de los protocolos, y hay una gran población de usuarios "navegantes" que pueden generar una cantidad inmensa de peticiones si el contenido es interesante. La organización de un servicio web conectado a Internet requiere tener en cuenta las características de la demanda que pueda tener que atender.

Arrecifes de coral y tráfico en Internet

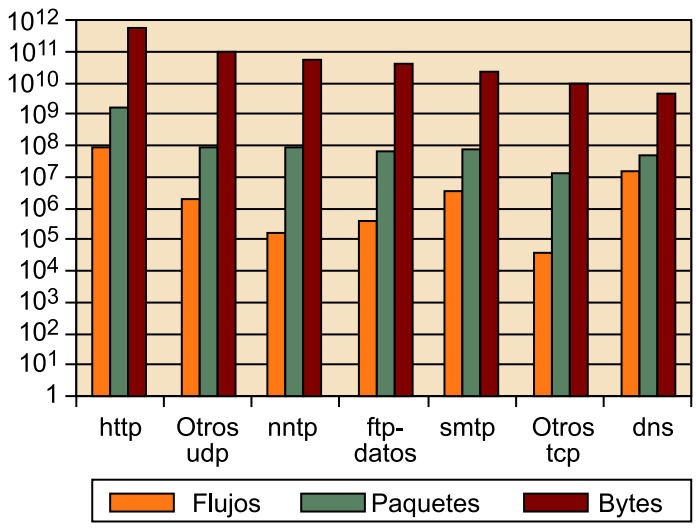
La organización CAIDA se dedica al análisis del tráfico en Internet y ha desarrollado una herramienta denominada *Coral Reef* que toma trazas del tráfico de un enlace. Con ésta, en 1998 hicieron un estudio de tráfico por protocolos en el núcleo de la red del proveedor MCI. El artículo que lo describe se presentó en la conferencia Inet 98, y se titulaba "The nature of the beast: recent traffic measurements from an Internet backbone". Las gráficas adjuntas se han obtenido de estas medidas.

La cronología de Internet

Un calendario de los eventos relacionados con Internet desde 1957 hasta hoy lo podéis ver en la dirección siguiente:

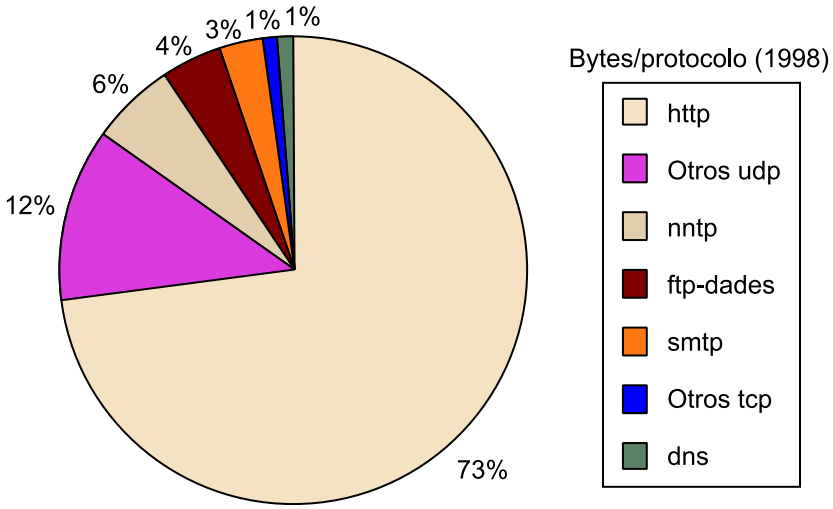
Hobbes' Internet Timeline 10.1

Figura 1



Volumen de tráfico en escala logarítmica de flujos, paquetes y bytes intercambiados durante veinticuatro horas en un enlace del núcleo de la red de MCI/Worlcom (1998), organizado por el protocolo.

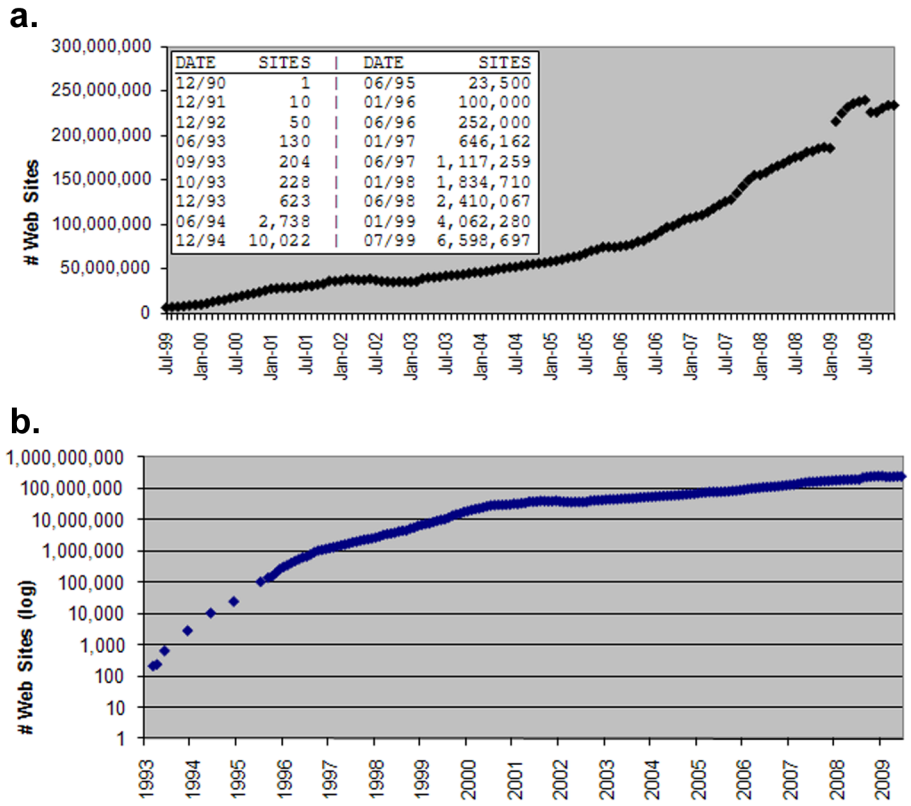
Figura 2



Porcentaje de tráfico en bytes de cada protocolo respecto al total medido. Puede apreciarse mejor que en la figura 1, en escala logarítimica, que el porcentaje de tráfico web domina el resto (73% del total).

Por otro lado, la web (HTTP) es un servicio muy reclamado por todo tipo de organizaciones para publicar información, como puede verse en la tendencia de crecimiento del número de servidores web en Internet, que ha sido exponencial, tal y como muestra la figura 3.

Figura 3

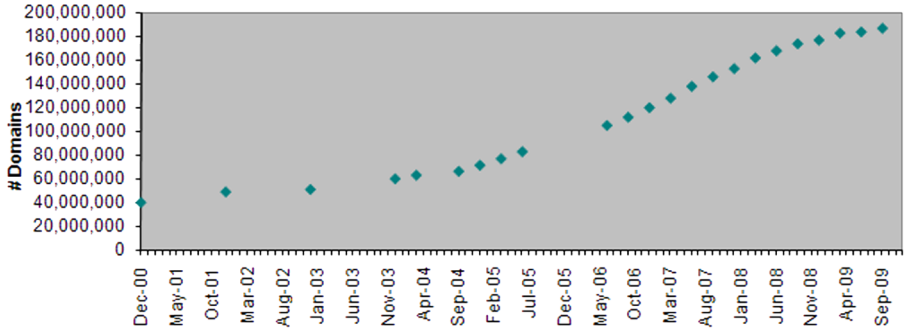


DATE	SITES	DATE	SITES
12/90	1	06/95	23,500
12/91	10	01/96	100,000
12/92	50	06/96	252,000
06/93	130	01/97	646,162
09/93	204	06/97	1,117,259
10/93	228	01/98	1,834,710
12/93	623	06/98	2,410,067
06/94	2,738	01/99	4,062,280
12/94	10,022	07/99	6,598,697

a. Crecimiento del número de sitios web durante los últimos años.
b. Crecimiento del número de lugares web durante los últimos años en escala logarítmica. Podemos ver la tendencia asintótica.
Hobbes' Internet Timeline Copyright © 2010 Robert H. Zakon

La figura 4 nos muestra el crecimiento en la creación de nuevos dominios que también sigue una tendencia exponencial fruto de la adopción de la web como principal media de distribución de la información.

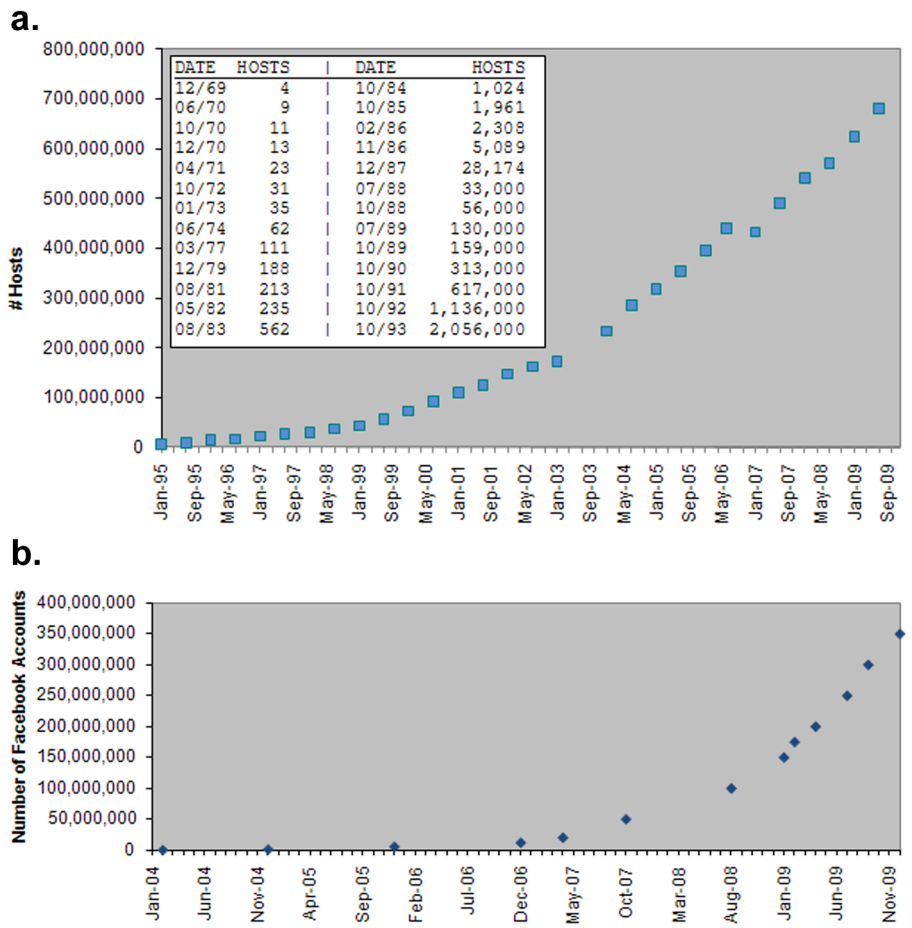
Figura 4



Crecimiento del número de dominios registrados en los últimos años.
Hobbes' Internet Timeline Copyright © 2010 Robert H. Zakon

Para completar este análisis de las tendencias a la Red, las figuras 5.a y 5.b nos muestran por un lado el incremento exponencial del número de huéspedes que acceden a la Red y por otra, el auge de las redes sociales como Facebook con un incremento muy elevado del número de usuarios en muy poco tiempo.

Figura 5

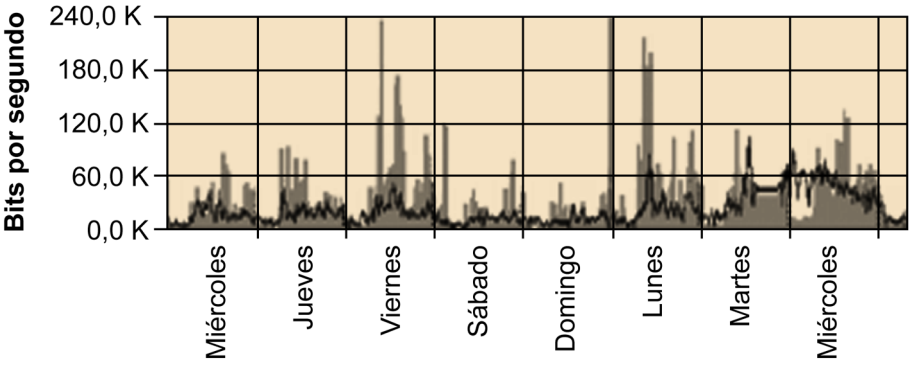


DATE	HOSTS		DATE	HOSTS
12/69	4		10/84	1,024
06/70	9		10/85	1,961
10/70	11		02/86	2,308
12/70	13		11/86	5,089
04/71	23		12/87	28,174
10/72	31		07/88	33,000
01/73	35		10/88	56,000
06/74	62		07/89	130,000
03/77	111		10/89	159,000
12/79	188		10/90	313,000
08/81	213		10/91	617,000
05/82	235		10/92	1,136,000
08/83	562		10/93	2,056,000

a. Crecimiento del número de huéspedes que acceden a la Red en los últimos años.
b. Crecimiento del número de usuarios de la red social Facebook.

La popularidad de los servidores web también es muy variable. Un mismo sitio web puede recibir muy pocas visitas durante mucho tiempo y, de repente, recibir más peticiones de las que puede servir: es un tráfico a ráfagas.

Figura 6



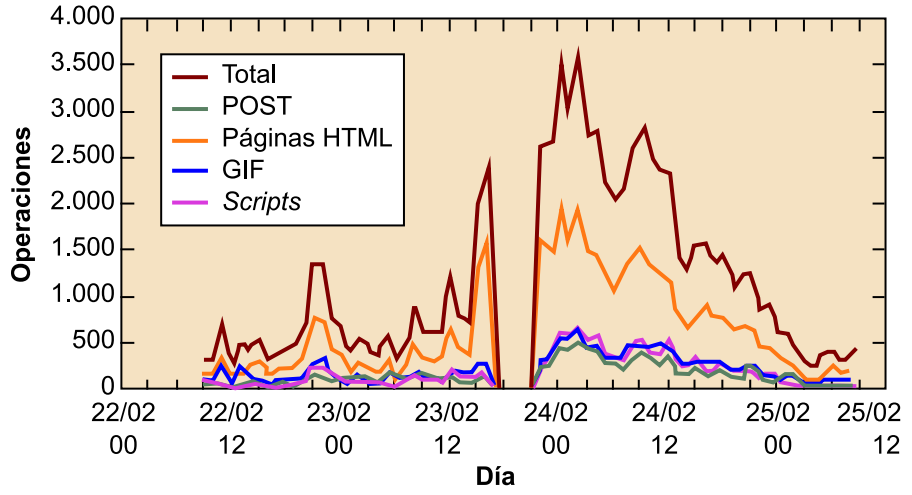
Evolución del tráfico entrante y saliente de un sitio web típico durante una semana. Podéis observar la gran variación horaria y la reducción de tráfico durante el fin de semana.

Flash crowd

Un cuento de ciencia ficción de varias Larry Niven (1973) predijo que una consecuencia de un mecanismo de teletransporte barato sería que grandes multitudes se materializarían instantáneamente en los lugares con noticias interesantes. Treinta años después, el término se usa en Internet para describir los picos de tráfico web cuando un determinado sitio se hace popular de repente y se visita de manera masiva. También se conoce como efecto "slashdot" o efecto "/.", que se da cuando un sitio web resulta inaccesible a causa de las numerosas visitas que recibe cuando aparece en un artículo del sitio web de noticias slashdot.org (en castellano, barrapunto.com

Un servidor puede recibir aludes repentinos de tráfico. Por ejemplo, por las estadísticas del siguiente servidor web sabemos que, después de que se anunciara en la página de noticias salshdot.org, experimentó un exceso de visitas tan elevado que el servidor se bloqueó.

Figura 7



Peticiones web por hora, servidas por http://linuxcounter.net durante tres días. Puede verse que mientras que el número habitual de operaciones (peticiones web) estaba por debajo de 500, subió rápidamente a unas 2.500, lo cual provocó el fallo del sistema. Después de reconfigurarlo, estuvo soportando durante unas doce horas en torno a 3.000 peticiones/hora para bajar posteriormente a valores normales. La historia completa está en la página *The Linux Counter Slashdot Experience*

Un servidor web puede tener miles de documentos y, sin embargo, recibir la mayoría de las peticiones por un único documento. En muchos casos, la popularidad relativa entre distintos sitios web o entre diferentes páginas de un cierto sitio se rige por la ley de Zipf (George Kingsley Zipf, 1902-1950).

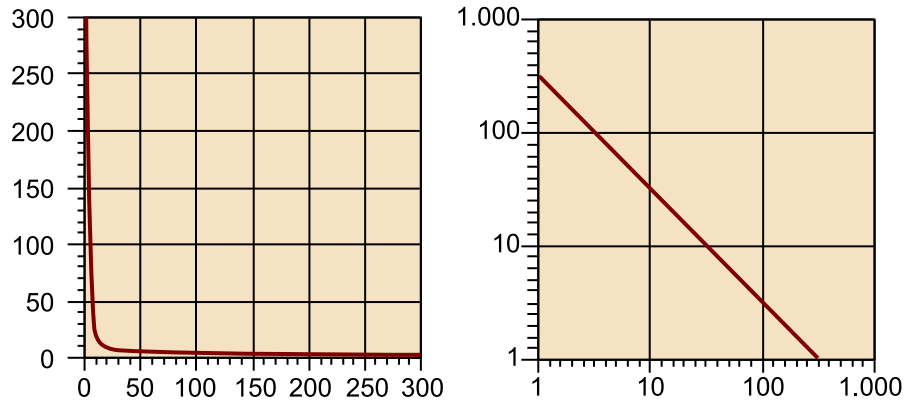
Ley de Zipf

La frecuencia de ocurrencia de un evento concreto (P) como función del rango (i) cuando el rango estás determinado por la frecuencia de ocurrencia es una función potencial $P_i \sim 1/i^a$, con el exponente a cercano a la unidad.

El ejemplo más famoso es la frecuencia de palabras en inglés. En 423 artículos de la revista *Time* (245.412 palabras), *the* es la que más aparece (15.861), *of* está en segundo lugar (7.239 veces), *to* en tercer lugar (6.331 veces), y con el resto forman una ley potencial con un exponente cercano a 1.

Una distribución de popularidad Zipf forma una línea recta cuando se dibuja en una gráfica con dos ejes en escala logarítmica, que resulta más fácil de ver y comparar que la gráfica en escala lineal, tal y como puede verse en la figura 8.

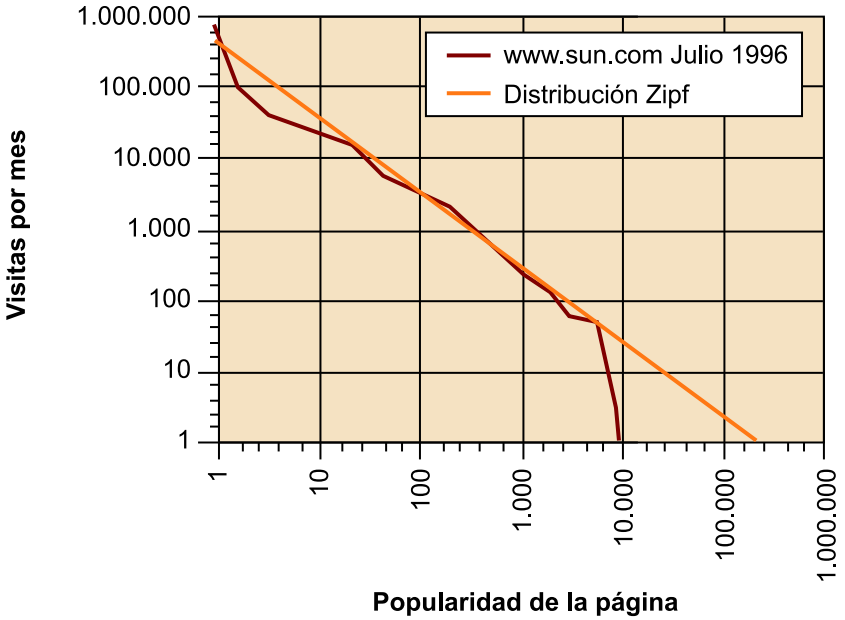
Figura 8



Una distribución de popularidad (casos ordenados por popularidad en el eje *x*, valor de popularidad en el eje *y*) que sigue la ley de Zipf a escala lineal queda "pegada a los ejes": muy pocos casos tienen mucha popularidad y muchos casos tienen muy poca. Por este motivo, se suele representar en escalas logarítmicas (gráfica doble logarítmica: los dos ejes en escala logarítmica).

Muchos estudios muestran que las visitas de páginas web siguen una distribución de Zipf. La figura 9 muestra las visitas en www.sun.com durante un mes de 1996. La página principal recibió prácticamente 1 millón de visitas, mientras que la página de la posición 10.000 de popularidad sólo recibió una visita aquel mes. La gráfica de visitas sigue la curva de Zipf excepto para los valores menos populares, lo cual seguramente se debe al hecho de que el periodo de observación no fue lo bastante largo.

Figura 9



Número de visitas de las páginas de www.sun.com ordenadas por popularidad. Puede verse cómo se ajusta a una distribución de Zipf.

Como resumen, diferentes estudios del tráfico web contribuyen a definir un perfil típico o reglas a ojo de la web –según M. Rabinovich y O. Spatscheck (2002):

- El tamaño medio de un objeto es de 10-15 kbytes, y la media de 2-4 kbytes. La distribución se decanta claramente hacia objetos pequeños, aunque se encuentra una cantidad nada despreciable de objetos grandes (del orden de Mbytes).

- La mayoría de los accesos a la web son por objetos gráficos, seguidos de los documentos HTML. El 1-10% son por objetos dinámicos.

- Una página HTML incluye de media diez imágenes y múltiples enlaces a otras páginas.

- Un 40% de todos los accesos son para objetos que se considera que no se pueden inspeccionar.

- La popularidad de objetos web es muy diferente: una pequeña fracción de objetos es la responsable de la mayoría de los accesos, siguiendo la ley de Zipf.

- El ritmo de acceso para objetos estáticos es muy superior al ritmo de modificación.

- En una escala de tiempo inferior al minuto el tráfico web es a ráfagas, por lo cual valores medidos con medias durante algunas decenas de segundo son muy poco fiables.

- Un 5-10% de accesos a la web se cancelan antes de finalizar.

- Prácticamente todos los servidores utilizan el puerto 80.

Cada sitio web es un poco distinto, y puesto que un servidor web es un sistema complejo y, por lo tanto, difícil de modelar, resulta conveniente hacer experimentos tanto para ver cómo los niveles de carga (peticiones de páginas web) crecientes afectan a nuestro servidor, como para observar periódicamente el comportamiento de un servidor web analizando los diarios (*logs*) que puede generar.

Para probar el rendimiento de un servidor web, normalmente se utiliza algún programa que, instalado en otro computador, genere un ritmo de peticiones equivalentes para medir el efecto de las visitas simultáneas desde distintos clientes. El ritmo de peticiones puede configurarse de una manera ligeramente distinta para cada herramienta, pero el objetivo es ser estadísticamente equivalente a la demanda real que el servidor pueda experimentar durante su funcionamiento normal. Esto recibe el nombre de *carga sintética*.

Hay muchas herramientas. A continuación se describen brevemente tres herramientas populares y gratuitas:

Enlaces de interés

Visitas web sobre generadores de carga:
Apache Jmeter
Surge

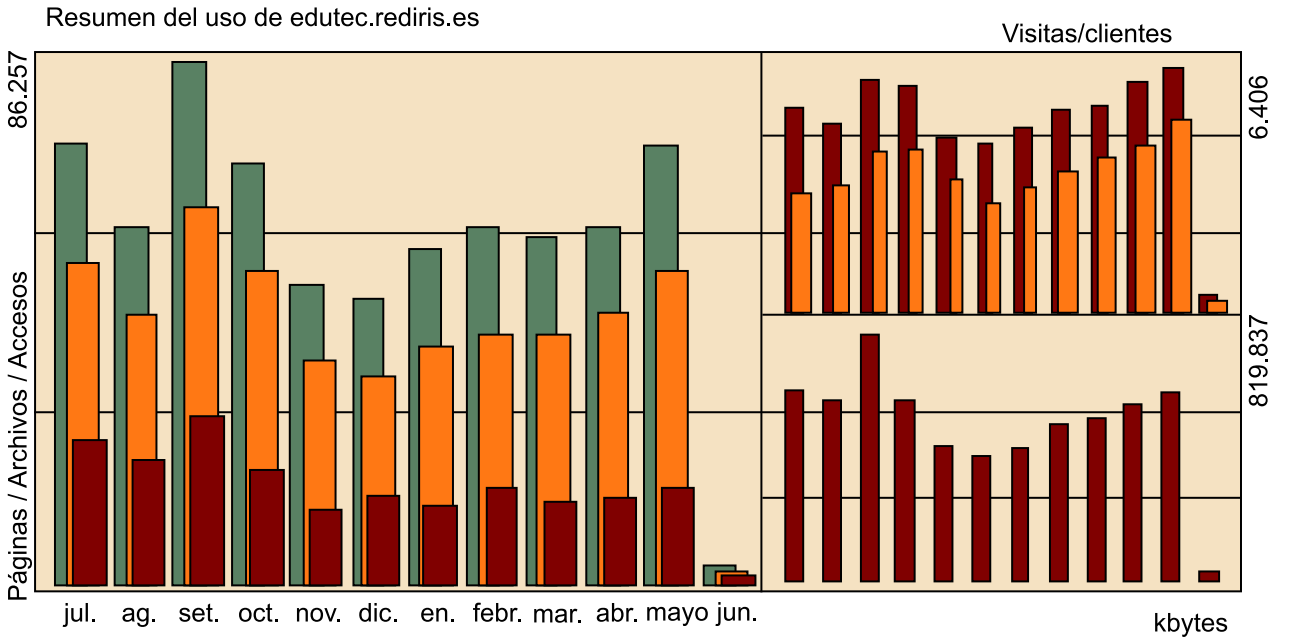
- *Microsoft Web Application Stress* (WAS) es una herramienta de simulación para Windows diseñada para medir el rendimiento de un sitio web. Tiene en cuenta las páginas generadas dinámicamente (ASP) en un servidor web de Microsoft.

- *Apache JMeter*, una aplicación Java para medir el rendimiento de documentos y recursos estáticos y dinámicos (archivos, *servlets*, *scripts* Perl, objetos Java, consultas de bases de datos, servidores FTP, etc.), que simula distintos tipos de carga extrema de la Red, del servidor o de un objeto concreto.

- *Surge* de la Universidad de Boston: una aplicación Java que genera peticiones web con características estadísticas que simulan con mucha precisión la demanda típica de un servidor web.

Durante el funcionamiento normal del servidor, es conveniente supervisar la demanda y el rendimiento del servicio para detectar su degradación (responde muy lentamente por exceso de peticiones o tráfico) o situaciones críticas (sobrecarga: no responde).

Figura 10



Una de las muchas gráficas que genera una herramienta de estadísticas web popular y gratuita denominada *webalizer*, que facilita mucho el análisis de la actividad de un servidor web.

Las herramientas de visualización y análisis de actividad del servidor se basan en el hecho de que prácticamente todos los servidores son capaces de generar archivos históricos de la actividad del servidor (conocidos como *diarios*[1]) en un formato conocido como *common log format* o CLF.

(1)En inglés, *logs*.

En CLF cada línea (a veces denominada *entrada*) registra una petición recibida por el servidor. La línea está formada por distintos elementos separados por espacio:

```
máquina ident usuario_autorizado fecha petición estado bytes
```

Si un dato no tiene valor, se representa por un guión (-). El significado de cada elemento es el siguiente.

- Máquina: el nombre DNS completo o su dirección IP si el nombre no está disponible.

- Ident: si está activado, la identidad del cliente tal y como lo indica el protocolo identd. Puede no ser fiable.

- UsuarioAutorizado: si se pidió un documento protegido por contraseña, corresponde al nombre del usuario utilizado en la petición.

- Fecha: la fecha y hora de la petición en el formato siguiente:

  ```
  día = 2*digit
  mes = 3*letter
  año = 4*digit
  ```

```
hora = 2*digit
minuto = 2*digit
segundo = 2*digit
zona = ('+' | '-') 4*digit
```

- Petición: el URL solicitado por el cliente, delimitado por comillas (").
- Estado: el código de resultado de tres dígitos devuelto al cliente.
- Bytes: el número de bytes del objeto servido, sin incluir cabeceras.

Este formato es adecuado para registrar la historia de las peticiones, pero no contiene información útil para medir el rendimiento del servidor. Por ejemplo, no indica el tiempo transcurrido en servir cada URL.

Para permitir construir un formato de entrada de diario (*log*) que contenga la información necesaria, puede definirse un formato particular que puede contener otra información. A continuación se muestra una lista de las variables que el servidor web Apache puede guardar (*mod_log*).

Nombre	Descripción de la variable
%a	Dirección IP remota.
%A	Dirección IP local.
%B	Bytes enviados, excluyendo las cabeceras HTTP.
%b	Bytes enviados, excluyendo las cabeceras HTTP. En formato CLF: un '-' en lugar de un 0 cuando no se ha enviado ningún byte.
%c	Estado de la conexión cuando la respuesta se acaba: 'X' = conexión abortada antes de acabar la respuesta. '+' = conexión que puede quedar activa después de haber enviado la respuesta. '-' = conexión que se cerrará después de haber enviado la respuesta.
%{NOMBRE}e	El contenido de la variable de entorno NOMBRE.
%f	Nombre del fichero.
%h	Nombre de la máquina remota.
%H	El protocolo de la petición.
%{Nombre}i	El contenido del encabezado o encabezados “Nombre:” de la petición enviada al servidor.
%l	Usuario remoto (de *identd*, si lo proporciona).
%m	El método de la petición.
%{Nombre}n	El contenido de la “nota” “Nombre” desde otro módulo.
%{Nombre}o	El contenido de la cabecera o cabeceras “Nombre:” de la respuesta.
%p	El puerto del servidor sirviendo la petición.

Nombre	Descripción de la variable
%P	El identificador del proceso hijo que sirvió la petición.
%q	El texto de una consulta o *query string* (precedido de ? si la consulta existe, si no, un texto vacío).
%r	Primera línea de la petición.
%s	Estado de peticiones que fueron redireccionadas internamente, el estado de la petición original - %>s para el de la última.
%t	Tiempo (fecha) en formato de LOG (formato estándar inglés).
%{formato}t	El tiempo (hora), en el formato especificado, que debe expresarse en formato strftime (posiblemente localizado).
%T	El tiempo de servicio de la petición, en segundos.
%u	Usuario remoto (de autentificación; puede ser incorrecto si el estado de la respuesta %s es 401).
%U	La parte de camino (*path)* del URL, sin incluir el texto de la consulta (*query string*).
%v	El nombre original o *canónico* del servidor dependiente de la petición.
%V	El nombre del servidor según el valor del orden *UseCanonicalName*.

Según estas variables, el formato CLF sería:

```
"%h %l %u %t \"%r\" %>s %b"
```

El formato CLF incluyendo el servidor web virtual solicitado (un servidor web puede servir a diferentes dominios DNS o servidores virtuales):

```
"%v %h %l %u %t \"%r\" %>s %b"
```

- El formato NCSA extendido/combinado sería:

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
```

Analizar la demanda y rendimiento de un servidor es una tarea necesaria, ya que los servidores web están sujetos a variaciones de demanda muy extrema. Después del análisis de los ficheros de diario del servidor, puede ser necesario limitar, resituar y ampliar los recursos del servidor y la red de acceso a Internet para poder atender aceptablemente el “extraño y voluble” tráfico de peticiones que visita los servidores web.

2. Organización de las aplicaciones en servidores web

En este apartado veremos cómo se organizan las aplicaciones en un servidor web y detallaremos aspectos relevantes del funcionamiento de una aplicación web.

2.1. El servidor web

Un servidor web que se ejecuta en un ordenador se mantiene a la espera de peticiones por parte de un cliente (un navegador web o un programa que hace una llamada a un servicio web). Cuando el servidor recibe una petición, responde adecuadamente mediante una página web que se exhibirá en el navegador, o bien mostrará el mensaje de error correspondiente.

A guisa de ejemplo, al escribir www.uoc.edu en nuestro navegador, este realiza una petición HTTP al servidor de la mencionada dirección (una vez resuelto el nombre mediante la DNS). El servidor responderá al cliente enviando el código HTML de la página, y el cliente –una vez haya recibido el código– lo interpretará y lo presentará por pantalla. Como vemos con este ejemplo, el cliente es el encargado de interpretar el código HTML, es decir, de mostrar las fuentes, los colores y la disposición de los textos y los objetos de la página. El servidor sólo se limita a transferir el código de la página sin llevar a cabo ninguna interpretación.

Además de la transferencia de código HTML, los servidores web pueden ejecutar aplicaciones web. Estas están formadas por código que se ejecuta cuando se realiza alguna petición o respuesta HTTP.

Hay que distinguir entre:

a) Aplicaciones en el lado del cliente: el cliente web es el encargado de ejecutarlas en la máquina del usuario. Son las aplicaciones de tipo Java *applets* o Javascript. El servidor proporciona el código de las aplicaciones al cliente y este, mediante el navegador, las ejecuta. Es necesario, por lo tanto, que el cliente disponga de un navegador con capacidad de ejecutar aplicaciones (también llamadas *scripts*). Normalmente, los navegadores permiten ejecutar aplicaciones escritas en lenguaje Javascript y Java, aunque pueden añadirse más lenguajes mediante el uso de *plugins*.

b) Aplicaciones en el lado del servidor: el servidor web ejecuta la aplicación y esta, una vez ejecutada, genera cierto código HTML y lo devuelve al servidor. Seguidamente, el servidor envía este código al cliente por medio del protocolo HTTP.

Una ventaja de desarrollar aplicaciones web del lado del servidor es que, al ejecutarse en el servidor y no en la la máquina del cliente, no hacen necesaria ninguna capacidad añadida al navegador cliente, como sí que sucede en el caso de la ejecución de aplicaciones de Javascript o Java. Así pues, cualquier cliente dotado de un navegador web básico puede utilizar este tipo de aplicaciones. En contrapartida, la carga del servidor aumenta, y el rendimiento se ve afectado. La llamada Web 2.0 ha cambiado la tendencia en el desarrollo de aplicaciones, que hace unos años se basaban casi completamente en la ejecución en el lado del servidor. Más recientemente, las tecnologías AJAX han traído la computación a los navegadores de los clientes, distribuyendo así la carga que recibían los servidores web entre sus clientes y por lo tanto, consiguiendo un Internet más escalable y con aplicaciones más potentes.

Las aplicaciones web –aplicaciones que van asociadas o son extensiones de un servidor web– pueden necesitar un diseño y ajuste muy cuidadosos para ofrecer un rendimiento adecuado en situaciones de alta demanda, o simplemente para responder rápidamente o aprovechar de manera adecuada los recursos de la máquina en la que están instalados.

En primer lugar, hay que saber cómo está organizado un servidor web para atender peticiones HTTP de la manera más eficiente. En segundo lugar, se debe conocer cómo puede extenderse el servidor web para ofrecer otros servicios gestionados por un código adicional.

2.2. Organización del servidor web

Para caracterizar cómo se organiza un servidor web para atender peticiones de una manera eficiente y económica, es necesario definir algunos términos:

- Proceso: la unidad más "pesada" de la planificación de tareas que ofrece el sistema operativo. No comparte espacios de direcciones ni recursos relacionados con ficheros, excepto de manera explícita (heredando referencias a ficheros o segmentos de memoria compartida), y el cambio de tarea lo fuerza el núcleo del sistema operativo (*preemptivo*).

- Flujo o *thread*: la unidad más "ligera" de planificación de tareas que ofrece el sistema operativo. Como mínimo, hay un flujo por proceso. Si distintos flujos coexisten en un proceso, todos comparten la misma memoria y recursos de archivos. El cambio de tarea en los flujos lo fuerza el núcleo del sistema operativo (*preemptivo*).

- Fibra: flujos gestionados por el usuario de manera cooperativa (*no preemptivo*), con cambios de contexto en operaciones de entrada/salida u otros puntos explícitos: al llamar a ciertas funciones. La acostumbran a implementar librerías fuera del núcleo, y la ofrecen distintos sistemas operativos. Para ver qué modelos de proceso interesan en cada situación, hay que considerar las combinaciones del número de procesos, flujo por proceso y

fibras por flujo. En todo caso, cada petición la sirve un flujo que resulta la unidad de ejecución en el servidor.

Para ver qué modelos de proceso interesan en cada situación, hay que considerar las combinaciones del número de procesos, flujo por proceso y fibras por flujo. En todo caso, cada petición la sirve un flujo que resulta la unidad de ejecución en el servidor.

Lectura complementaria

El apartado "4.4 Server Architecture" del libro *Krishnamurthy, Web Protocols and Practice* (págs. 99-116), describe un estudio sobre el funcionamiento de Apache 1.3.

Los modelos que se pueden construir son (U: único, M: múltiple):

Proceso	Flujo	Fibra	Descripción
U	U	U	Es el caso de los procesos gestionados por *inetd*. Cada petición genera un proceso hijo que sirve la petición.
M	U	U	El modelo del servidor Apache 1.3 para Unix: distintos procesos preparados que se van encargando de las peticiones que llegan. Lo implementa el módulo Apache: *mpm_prefork*.
M	U	M	En cada proceso, una librería de fibras cambia de contexto teniendo en cuenta la finalización de operaciones de entrada/salida. En Unix se denomina *select-event threading* y lo usan los servidores Zeus y Squid. Ofrece un rendimiento mejor en Unix que en MUU. Lo implementa el módulo Apache *state-threaded multi processing*.
M	M	U	El modelo MMU cambia fibras por flujos. Lo implementan los módulos Apache: *perchild* y *mpm_worker_module*. El número de peticiones simultáneas que puede atender es *ThreadsPerChild* x *MaxClients*.
U	M	U	El modelo más sencillo con diferentes flujos. Se puede montar en Win32, OS2, y con flujos POSIX. Lo implementa el módulo Apache: *mpm_netware*.
U	M	M	Probablemente el que proporciona un rendimiento más alto. En Win32 se puede conseguir con los denominados *completion ports*. Se usan los flujos necesarios para aprovechar el paralelismo del *hardware* (número de procesadores, tarjetas de red), y cada flujo ejecuta las fibras que han completado su operación de entrada/salida. Es el modelo con un rendimiento más alto en Windows NT. Lo implementa el módulo Apache: *mpm_winnt_module*. Muchos servidores como Internet Information Server o IIS 5.0 utilizan este modelo con Windows NT. El servidor web interno al núcleo de Linux *Tux* también utiliza este modelo.
M	M	M	Puede ser una generalización de UMM o MUM, y en general la presencia de distintos procesos hace que el servidor se pueda proteger de fallos como consecuencia de que un proceso tenga que morir por fallos internos, como por ejemplo el acceso a memoria fuera del espacio del proceso.

En general, los modelos con muchos procesos son costosos de memoria (cada proceso ocupa su parte) y de carga (creación de procesos). En servidores de alto rendimiento, los modelos con flujos parecen mejores, aunque tienen el problema de la portabilidad difícil y la posible necesidad de mecanismos que anticipan el coste de creación de procesos o flujos y, por lo tanto, son muy rápidos atendiendo peticiones.

TUX: servidor web en el núcleo de Linux

Tux es un servidor web incorporado al núcleo de Linux que usa un *pool* con muy pocos flujos del núcleo (un flujo por procesador), lee directamente de la memoria del sistema de ficheros de dentro del núcleo, usa su propio algoritmo de planificación de fibras y usa el TCP/IP "zero-copy" que minimiza las veces que los datos que vienen de la Red se copian en la memoria (en redes de alta velocidad como Gigabit Ethernet, las copias de

bloques de memoria son el factor limitante). Puede servir entre dos y cuatro veces más peticiones/ segundo que Apache o IIS.

Para más información: http://docs.huihoo.com/tux

Cuando la Red limita el servicio web, lanzar procesos para atender peticiones puede funcionar razonablemente bien, pero en redes de alta velocidad, como ATM o Gigabit Ethernet, la latencia en iniciar un proceso al recibir una petición es excesivo.

En máquinas uniprocesador, los modelos con un solo flujo funcionan bien. En máquinas multiprocesador, es necesario usar múltiples flujos o procesos para aprovechar el paralelismo del *hardware*.

El mayor obstáculo para el rendimiento es el sistema de ficheros del servidor, ya que la función principal del servidor web es "trasladar" ficheros del disco a la Red. Si éste se ajusta, el siguiente obstáculo es la gestión de la concurrencia.

Además, los estudios de tráfico web indican que la mayoría de las peticiones son de ficheros pequeños, por lo cual sería conveniente optimizar el servidor para poder priorizar estas peticiones que son más frecuentes y mejoran el tiempo de respuesta percibido por los usuarios, por ejemplo al cargar páginas web con muchos gráficos pequeños insertados.

Por esta razón, Apache es un servidor web modular, en el cual la gestión del proceso está concentrada en un módulo que puede seleccionarse en la instalación, según las características del sistema operativo, la máquina, los recursos que tendrá que utilizar el servidor, etc.

2.3. Organización de las aplicaciones web

Los servidores web se encargan de atender y servir peticiones HTTP de recursos, que en su forma más simple acostumbran a ser documentos guardados en el sistema de ficheros. Sin embargo, la otra función importante de un servidor web es la de actuar de mediador entre un cliente y un programa que procesa datos. Recibe una petición con algún argumento, la procesa y devuelve un resultado que el servidor web entrega al cliente. La interacción entre el servidor web y los procesos que tiene asociados es otro aspecto que hay que considerar.

Existen distintas maneras de organizar una aplicación web. A continuación, por orden cronológico de complejidad y rendimiento creciente, se presentan distintos modelos de organización.

- CGI: es el modelo más antiguo y simple. Para cada petición HTTP se invoca un programa que recibe datos por las variables del entorno y/o de entrada estándar, y devuelve un resultado por la salida estándar. Consumir un proceso por cada petición genera problemas importantes de rendimiento, que el modelo FastCGI intenta mejorar.

- *Servlets*: es un modelo diseñado para Java, más eficiente y estructurado, que permite elegir distintos modelos de gestión de flujos o *threads*, duración de procesos del servidor, etc. Partiendo de este modelo, se han construido servidores de aplicaciones con múltiples funciones adicionales que facilitan el desarrollo de aplicaciones web complejas.

- Lenguajes de *script*: existen también lenguajes, como por ejemplo PHP, que permiten incluir trozos de código (*scripts*) en el código HTML y que al llegar al servidor son ejecutados como si fueran un CGI, devolviendo así la respuesta al cliente. Las Java Server Pages (JSP) son *scripts* en código Java que al llegar al servidor son ejecutadas como si fueran *servlets*.

3. Servidores *proxy-cache* web

Para la web, se diseñó un protocolo simple (HTTP) de acceso a documentos sobre un transporte fiable como TCP. Un objetivo de diseño era la interactividad: el cliente se conecta con el servidor web, solicita el documento (petición) e inmediatamente lo recibe del servidor. Este esquema es rápido en situaciones de tráfico en la Red y carga de los servidores reducida, pero no es eficiente en situaciones de congestión.

NAT (*network addresstranslation*)

En los paquetes IP salientes: sustituir la dirección IP de máquinas internas (no válidas en Internet) por la suya propia, en los paquetes IP entrantes: sustituir su propia dirección IP por la de una máquina interna y reenviar el paquete hacia la red interna. Para poder saber a quién entregarlo, el intermediario debe asociar cada uno de sus puertos a máquinas internas, pues una dirección de transporte es una pareja (dirección IP, puerto).

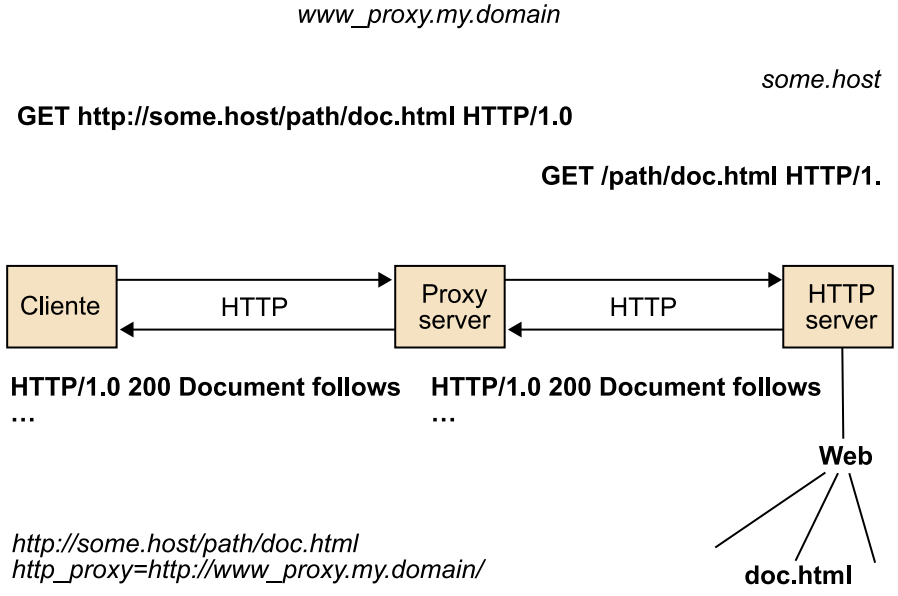
Para todas aquellas situaciones en las cuales la comunicación directa cliente-servidor no es conveniente, se ha introducido un tipo de servidores que hacen de intermediarios entre los extremos.

Un servidor intermediario (*proxy*) es un servidor que acepta peticiones (HTTP) de clientes u otros intermediarios y genera a su vez peticiones hacia otros intermediarios o hacia el servidor web destino. Actúa como servidor del cliente y como cliente del servidor.

Ejemplo de servidor intermediario

La idea del *proxy* se usa en muchos protocolos además del HTTP. En general, se sitúan en una discontinuidad para realizar en la misma una función. Por ejemplo: Cambio de red. Una máquina conectada a la red interna de una organización que usa direcciones IPv4 privadas (por ejemplo, 10.*.*.*), y también en Internet, puede hacer de intermediario para traducción de direcciones IP entre las dos redes (NAT).

Figura 11



Interacción entre cliente-servidor intermediario-servidor tal y como aparece en la publicación original (Luotonen, 94) describiendo el servidor intermediario HTTP del CERN (Centro Europeo de Investigación en Física), en el que fue inventada la web.

Si no se desea usar este mecanismo, que tiene ciertas limitaciones, se puede salvar la discontinuidad al nivel del HTTP poniendo un intermediario HTTP: una máquina conectada a las dos redes que recibiría peticiones de páginas web desde cualquier cliente interno por una dirección interna y volvería a generar la misma petición desde el intermediario, pero hacia Internet, con la dirección externa.

Un servidor intermediario (*proxy*) es un servidor que acepta peticiones (HTTP) de clientes u otros intermediarios y genera a su vez peticiones hacia otros intermediarios o hacia el servidor web destino. Actúa como servidor del cliente y como cliente del servidor.

Como podéis suponer, aprovechando que tanto la petición como el resultado (la página web) pasarán por el intermediario, se puede aprovechar para ofrecer algunas funciones:

a) Control de acceso a contenidos. El intermediario consulta si la página solicitada está o no permitida por la política de la organización. Puede hacerse consultando una tabla de permisos o usando el mecanismo denominado PICS.

b) Control de seguridad. El intermediario genera todas las peticiones que salen a Internet, lo que oculta información y evita ataques directos sobre las máquinas internas de la organización. Además, puede existir un cortafuego que no permita acceder directamente hacia el exterior, con lo que el uso del intermediario es imprescindible.

c) Aprovechar peticiones reiteradas (función caché). El intermediario puede almacenar (en memoria o disco) una copia de los objetos que llegan como resultado de peticiones que han pasado por el intermediario. Estos objetos se pueden usar para ahorrar peticiones hacia Internet y servir peticiones sucesivas de un mismo objeto con una copia almacenada en el intermediario, sin tener que salir a buscarlo a Internet. Los ***proxy-cache*** son el caso más frecuente de servidor intermediario, y son los que detallaremos aquí.

d) Adaptar el contenido. Un intermediario podría también adaptar los objetos que vienen del servidor a las características de su cliente. Por ejemplo, convertir los gráficos al formato que el cliente entiende, o reducir su tamaño para clientes, como teléfonos móviles con reducida capacidad de proceso, comunicación o presentación.

El intermediario puede ser transparente, invisible para cliente y servidor: se obtiene el mismo resultado con el mismo o sin el mismo, aunque en los navegadores web habitualmente el usuario debe configurar expresamente su navegador para usarlo, pues el navegador no tiene un mecanismo para detectarlo y usarlo automáticamente.

PICS: Platform for Internet Content Selection

Figura 12

PICS define mecanismos para que se puedan catalogar sitios y páginas web según ciertos criterios y, de esta manera, controlar el acceso a contenidos no deseados. Es útil en escuelas y hogares para controlar el acceso a Internet de los menores. Control de contenidos: ciertos contenidos que se consideren no apropiados por la organización pueden ser bloqueados o redirigidos. Podéis encontrar más información en la página web de Platform for Internet Content Selection (PICS).

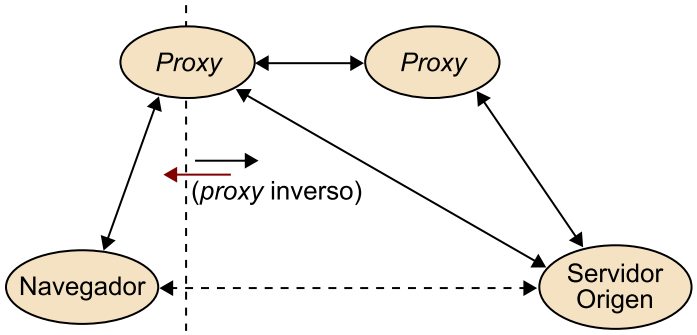
En algunas instalaciones, se puede instalar un *proxy* transparente que son *routers* extensiones del *software* del que hacen que éste intercepte las conexiones TCP salientes hacia el puerto 80 (HTTP) y las redirija a un *proxy-cache*. Tiene el peligro de que el usuario no es consciente de esto, y puede llevar a situaciones equívocas si el *proxy-cache* falla o trata mal la petición.

En la figura 13 podemos ver cómo hay servidores intermediarios para distintos protocolos además de HTTP, y que un *proxy* puede comunicarse con el servidor origen (el que tiene el documento original que el usuario ha solicitado) o pedirlo a otro *proxy*, formando una jerarquía de *proxies*.

Reflexió

El protocolo HTTP 1/1 tiene definido un código de respuesta a una petición: 305 Use proxy. El problema es que no se llegó a un acuerdo al escribir la especificación, y la respuesta hace que simplemente en el navegador aparezca el mismo mensaje de error, en lugar de tratar de buscar un *proxy* y reintentar la conexión mediante el mismo. ¿Cómo arreglarlo?

Figura 13



Un servidor intermediario se suele situar con proximidad a un cliente y puede colaborar con otros servidores intermediarios, pero también puede estar cerca de un servidor (servidor intermediario inverso).

Se usan también *proxies* en la proximidad de un servidor para reducir la carga de peticiones sobre el mismo. Son los *proxies* inversos: puede ser más sencillo y barato colocar uno o varios *proxy-cache* que reciban todas las peticiones: responderán directamente a las peticiones ya cacheadas y al servidor sólo le llegarán unas pocas que no están aún en el *proxy-cache*, o han expirado o son contenidos dinámicos.

a) Petición HTTP 1.0 cliente → servidor intermediario:

```
GET http://www.ac.upc.es/docencia/ HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/jpeg
```

b) Petición HTTP 1.0 servidor intermediario →; servidor. Podemos observar cómo el servidor intermediario introduce un campo "reenviado" (*forwarded*) para notificar su presencia al servidor:

```
GET /docencia/ HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Forwarded: by http://proxy.ac.upc.es:8080
```

Debido a la gran difusión del uso de servidores *proxy-cache*, sobre todo en entornos académicos, la especificación HTTP/1.1 define una nueva cabecera que permite controlar el comportamiento de un servidor *proxy-cache* tanto desde el cliente en la petición como desde el servidor en una respuesta.

Cache-control (petición)	
No-cache	Cliente ↔ origen (las memorias caché se inhiben).
No-store	El *proxy* no debe almacenar permanentemente petición/respuesta.
Max-age = **sgs**	La máxima "edad" aceptable de los objetos en la *caché*.
Max-stale	Se aceptan objetos viejos.
Max-stale = **sgs**	Se aceptan objetos **sgs** segundos viejos.
Min-fresh = **sgs**	Al objeto deben quedarle **sgs** de vida.
Only-If-Cached	Petición si sólo está en el servidor *proxy-cache*.

Cache-control (respuesta)	
Public	Se puede cachear por *proxies* y cliente.
Private	Sólo se puede guardar en la memoria caché del cliente.
Private="**cabc**"	Todos pueden mediar el objeto, excepto la cabecera **cabc**: sólo en la memoria caché del cliente.
No-cache	No se puede mediar ni en servidores intermediarios ni en el cliente.
No-cache="**cabc**"	Combinación de los dos anteriores.
No-store	Nadie puede almacenar permanentemente (sólo en la memoria del navegador).
No-transform	Los servidores intermediarios no pueden transformar el contenido.
Must-revalidate	Revalidar (con origen) si es necesario.
Max-age	Margen de edad en segundos.

Los servidores *proxy-cache* tienen algoritmos para decidir si, cuando llega una petición de un contenido que ya tienen, es o no necesario confirmar que no haya cambiado en el servidor origen, y el campo caché-control permite influir en la decisión. Otro problema grave es la pérdida de privacidad potencial si se guardan contenidos en el *proxy-cache*, más cuando se almacenan objetos en disco. Para esto también sirve el campo cache-control.

Un *proxy-cache* está formado por un almacén de mensajes de respuesta (objetos), un algoritmo de control del almacén (entrar, salir, borrar o reemplazar) y un algoritmo rápido de consulta (*lookup*) de un mapa del contenido; toma la decisión de servirlo del almacén o pedirlo al servidor origen o a otro *proxy-cache*.

Cookies (galletas): estado y privacidad

El protocolo HTTP no tiene estado: cada interacción (petición + respuesta) no tiene relación con las demás y cada una puede usar una conexión TCP distinta. Para poder relacionar varias interacciones HTTP y pasar información de estado entre las mismas, Netscape inventó las *cookies*: un servidor web puede enviar al cliente un objeto de hasta 4096 bytes, que contiene datos que el navegador devolverá a ese mismo servidor en el futuro (o a otros servidores que indique la *cookie*). Las *cookies* han sido muy usadas para cuestiones publicitarias (qué sitios web visita la gente y en qué orden), para guardar contraseñas, identificar usuarios, etc. sin que el usuario sea consciente ni de que está recibiendo estas "galletas", ni de que las está enviando cuando visita la web. Hay quien dice que son un error llevado a la perfección. ¿Qué opináis? ¿Habéis mirado alguna vez qué *cookies* tenéis en vuestro navegador?

Su uso puede producir una reducción de tráfico en la Red y en el tiempo de espera si los objetos que se piden están en la memoria y el contenido se puede mediar. Estudios en distintas organizaciones muestran con frecuencia tasas de acierto en la memoria del 15-60% de objetos, aunque esto pueda variar mucho con los usuarios y el contenido.

El GET condicional

Hemos visto que el *caching* puede reducir el tiempo de respuesta percibido por el usuario, pero la copia que tenga la caché puede ser obsoleta. Es decir, el objeto hospedado en el servidor web puede haberse modificado de manera posterior a que al cliente lo copiara en su caché. Para solucionarlo, el protocolo HTTP incluye un mecanismo que permite a la caché verificar que su copia del objeto está actualizada.

```
GET /elMeuDirectori/pagina.html HTTP/1.1
Host: www.servidor.edu
If-modified-since: Wed, 18 Feb 2009 21:11:55 GMT
```

La cabecera *If-modified-since*: contiene el valor de la cabecera *Last-Modified*: de cuando el servidor envió el objeto. Esta petición GET, que se denomina *GET condicional*, indica al servidor que envíe el objeto sólo si el objeto se ha modificado desde la fecha especificada.

En el caso de que el objeto no se haya modificado desde la fecha indicada, el servidor web contesta un mensaje a la caché como el siguiente:

```
HTTP/1.1 304 Not Modified
Date: Tue, 24 Mar 2009 18:23:40 GMT
(sense cos)
```

304 Not Modified en la línea de estado del mensaje de respuesta indica a la caché que puede pasar su copia del objeto al navegador que ha hecho la solicitud. Hay que destacar que el mensaje de respuesta del servidor no incluye el objeto, ya que esto sólo haría malgastar ancho de banda y la percepción del usuario sobre el tiempo de respuesta.

Los servidores *proxy-cache* son sistemas pasivos que aprovechan para guardar las respuestas a peticiones de usuarios. Automáticamente no intentan guardar contenidos que parecen privados, dinámicos o de pago: los detectan por la presencia de campos en la cabecera como por ejemplo: *WWW-Authenticate, Cache-Control:private, Pragma:no-cache, Cache-control:no-cache, Set-Cookie*.

Se han propuesto mejoras para hacer de los servidores *proxy-cache* intermediarios activos: acumular documentos de interés en horas de bajo tráfico para tener el contenido preparado y de esta manera ayudar a reducir el tráfico en horas de alta demanda, o mecanismos de *pre-fetch* en los que la memoria caché se adelanta a mostrar, antes de que se lo pidan, páginas web que con gran probabilidad el usuario va a solicitar a continuación. Sin embargo, no son de uso común pues no siempre suponen un ahorro o mejora del tiempo de respuesta, sino que pueden llevar a malgastar recursos de comunicación.

Los mecanismos de *proxy-cache* son útiles para que una comunidad de usuarios que comparten una conexión a Internet pueda ahorrar ancho de banda, y reducir transferencias repetitivas de objetos. Sin embargo, ésta es sólo una parte del problema.

El problema en conjunto se conoce humorísticamente como el ***World-Wide Wait***. Varios agentes participan en el rendimiento de una transferencia web:

a) Proveedor de contenidos (servidor web): debe planificar su capacidad para poder dar servicio en horas punta y aguantar posibles avalanchas de peticiones y, de esta manera, tener cierta garantía de que sus clientes dispondrán de buen servicio con cierta independencia de la demanda.

b) Cliente: podría usar un servidor *proxy-cache* para "economizar" recursos de la Red. Cuando un contenido está en más de un lugar, debería elegir el mejor servidor en cada momento: el original o una réplica "rápida" (o traer por trozos el objeto desde varios a la vez).

c) Réplica: si un servidor guarda una réplica de algún contenido probablemente es porque desea ofrecerlo y reducir la carga del servidor original. Por tanto, debe ser "conocido" por sus clientes, pero además el contenido tiene que ser consistente con el contenido original.

¿Una réplica (*mirror*)?

En enero del 2007, el programa HTTPD de Apache, el servidor web más utilizado en Internet, podía bajarse unas **300 réplicas** del sitio web original. La lista de réplicas está en la dirección dirección The status of Apache mirrors y la información del servidor, en la página web del http server project de Apache.

d) Proveedor de red: debería elegir el mejor camino para una petición, vía direccionamiento IP, vía resolución DNS (resolver nombres en la dirección IP más próxima al cliente que consulte, aunque no es lo más habitual), vía servidores *proxy-cache* HTTP y evitar zonas congestionadas (*hot spots*) o *flash crowd* (congestión en el servidor y en la red próxima debida a una avalancha de demandas).

Por tanto, los *proxy-caché* sólo son parte de la solución. Las redes de distribución de contenidos son la solución de otra parte del "W-W-Wait".

4. Contenidos distribuidos

Otra mejora que ayuda a combatir el mal del “W-W-Wait” son los sistemas de distribución de documentos: ¿basta un único servidor para cualquier "audiencia"? La respuesta es que no, si se desea ofrecer una calidad adecuada para demandas tanto muy pequeñas como bastante grandes, para contenidos que pueden llegar a ser muy populares en ciertos momentos, que pueden visitarse desde cualquier lugar del mundo o que pueden tener una audiencia potencial enorme.

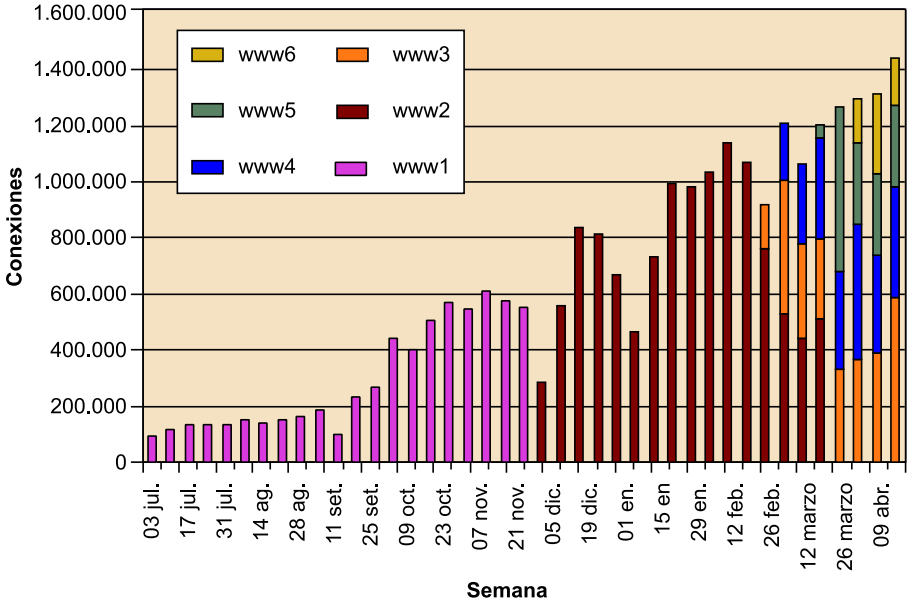
Las aplicaciones que ofrecen contenidos en Internet se enfrentan al reto de la escala: un solo servidor ante eventualmente millones de personas que pueden pedirle sus servicios todos a la vez: el proveedor de información debe poner tantos recursos como audiencia pueda tener.

En consecuencia, debe pagar más quien tiene algo más interesante que contar o quiere llegar a más. La web no funciona como una antena de radio: la potencia de emisión determina la cobertura, pero el número de receptores no la afecta; es más similar al teléfono: la capacidad se mide en número de líneas y esto determina el número de personas que pueden atenderse a la vez.

Por este motivo, puede ser necesario disponer de varios servidores para poder repartir la carga entre los mismos.

La primera página web pública, ahora ya fuera de funcionamiento, fue http://info.cern.ch. Sin embargo, la primera web con una demanda importante fue http://www.ncsa.uiuc.edu. Esta web tuvo que usar cuatro servidores replicados para satisfacer la demanda. En la siguiente gráfica puede verse la evolución del número de peticiones entre julio de 1993 (91.000 peticiones/semana) y abril de 1994 (1.500.000 peticiones/semana).

Figura 14



Crecimiento del número de peticiones semanales durante dos años. Cada tono de gris se corresponde con una máquina distinta. A mediados de febrero de 1994, diferentes máquinas empezaron a servir peticiones al mismo tiempo hasta llegar a cuatro.

Existen varios "trucos" para repartir peticiones entre diferentes máquinas:

- *Mirrors* con un programa que redirige la petición HTTP a la mejor réplica (podéis ver el ejemplo siguiente de Apache).

- Hacer que el servicio de nombres DNS devuelva diferentes direcciones IP (el método *round robin*). De esta manera, los clientes pueden hacer peticiones HTTP cada vez a una dirección IP distinta.

- Redirección en el nivel de transporte (conmutador de nivel 4, *L4 switch*): un encaminador mira los paquetes IP de conexiones TCP hacia un servidor web (puerto80) y las redirige a la máquina interna menos cargada.

- Redirección en el nivel de aplicación (conmutador de nivel 7, *L7 switch*): un encaminador que mira las conexiones HTTP y puede decidir a qué réplica contactar en función del URL solicitado. Muy complejo.

- Mandar todas las peticiones a un *proxy* inverso que responda o con contenido guardado en la memoria o que pase la petición a uno o varios servidores internos.

www.google.com

Si se pregunta al DNS por www.google.com, la respuesta contiene varias direcciones IP: nslookup www.google.com.

Si, además, las réplicas se sitúan cerca de los clientes, el rendimiento será mejor y más predecible. El inconveniente es que es difícil y caro montar un servicio web distribuido.

La Fundación Apache distribuye el servidor HTTPD Apache con la colaboración de más de doscientas réplicas distribuidas en todo el mundo. Aunque las páginas web se pueden consultar en el servidor origen, a la hora de bajar el programa hay un programa que calcula y redirige al visitante al servidor más próximo. De esta manera, las réplicas ayudan a repartir la carga a la vez que ofrecen un mejor servicio al cliente. Aquí el contenido se actualiza periódicamente.

¿Cómo ser *mirror* de Apache?

Apache pide que al menos se actualice el contenido dos veces por semana. Las herramientas para hacerlo son las siguientes:

a. rsync: un programa que compara dos lugares remotos y transfiere los bloques o ficheros que hayan cambiado. Ved http://rsync.samba.org.

b. cvs: un programa pensado para desarrollo de código a distancia, y que permite detectar e intercambiar diferencias. Ved http://www.cvshome.org.

c. Apache como *proxy-cache*: configurar un servidor Apache para pasar y hacer *cache* de las peticiones. Orden: ProxyPass / http://www.apache.org / CacheDefaultExpire.

4.1. Redes de distribución de contenidos

Han surgido empresas que se han dedicado a instalar máquinas en muchos lugares del mundo y algoritmos para decidir qué máquina es la más adecuada para atender peticiones según la ubicación del cliente y la carga de la Red. Estas empresas venden este "servidor web distribuido" a varios clientes que pagan por poder disponer de un sistema de servicio web de gran capacidad que puede responder a demandas extraordinarias de contenidos.

Estos sistemas se denominan redes de distribución de contenidos (*content delivery networks* o CDN), y se pueden encontrar varias empresas que ofrecen este servicio: Akamai es la más importante.

Una CDN es un *software* intermediario (***middleware***) entre proveedores de contenidos y clientes web. La CDN sirve contenido desde varios servidores repartidos por todo el planeta. La infraestructura de la CDN está compartida por varios clientes de la CDN, y las peticiones tienen en cuenta la situación de la Red para hacer que cada cliente web obtenga el contenido solicitado del servidor de la CDN más eficiente (habitualmente, una combinación del más próximo, el menos cargado y el que tiene un camino de mayor ancho de banda con el cliente).

Es como un servicio de "logística": la CDN se encarga de mantener copias cerca de los clientes en sus propios almacenes (no en un punto central, sino en los extremos de la Red). Para esto, debe disponer de multitud de puntos de servicio en multitud de proveedores de Internet (servidores *surrogate*: funcionalidad entre servidor *proxy-cache* y réplica). Por ejemplo, Akamai ofrecía en enero del 2007 unos 20.000 puntos de servicio en todo el mundo.

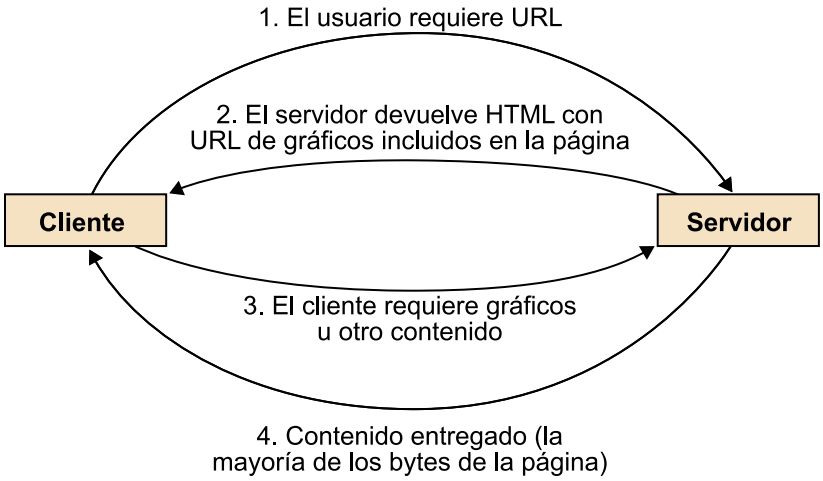
Una propuesta difícil de rechazar

La oferta de una CDN como Akamai a un proveedor de acceso a Internet (ISP) podría ser: "nos dejas en tu sala de máquinas el espacio equivalente para un frigorífico casero, nosotros te mandamos unas máquinas y tú las enchufas a la red eléctrica y a tu red (Internet). Nosotros las administramos". ¿Qué ocurrirá? Todas las peticiones web a varios miles de empresas serán servidas por nuestras máquinas. El ISP ahorra gasto de ancho de banda con Internet, pues sus usuarios no necesitarán ir a Internet para buscar algunos contenidos, que serán servidos por las máquinas de Akamai en el ISP. Akamai cobra del proveedor de contenidos por servir el contenido más rápido y mejor que nadie.

Algunas CDN, además, se sirven de enlaces vía satélite de alta capacidad (y retardo) para trasladar contenidos de un lugar a otro evitando la posible congestión de Internet. Aunque puedan no ser ideales para objetos pequeños, puede funcionar bien para objetos grandes (por ejemplo, *software*, audio, vídeo). Algunos ejemplos son las empresas Amazon 33, Castify y Real Networks.

Por otra parte, la transferencia de una página web normal funciona de la manera como se observa en la figura 15.

Figura 15



Fases de una petición web

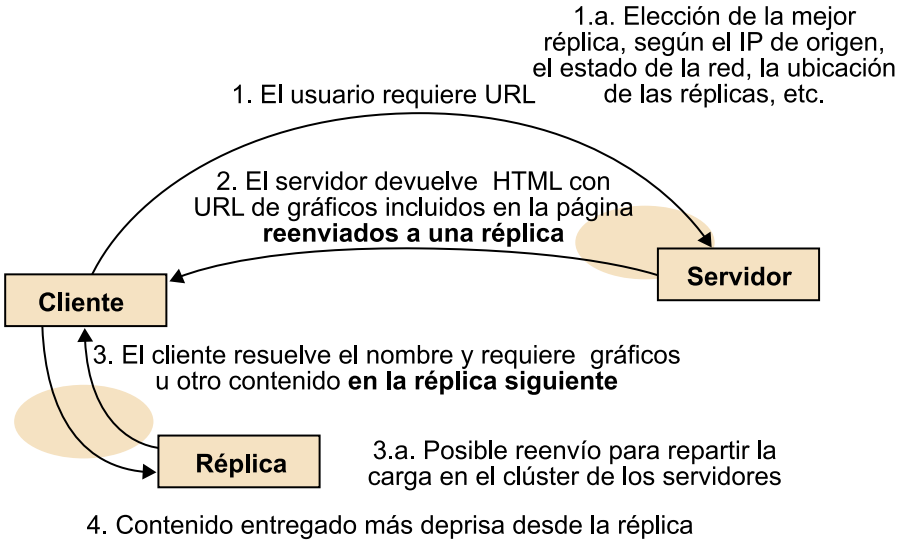
Una petición de una página web (documento HTML + algunos gráficos) hace distintas operaciones: 0.1) resolución DNS del nombre del servidor, 1.1) conexión TCP con el servidor, 1.2) solicitud del URL del documento, 2) el servidor devuelve el documento HTML que contiene referencias a gráficos incluidos en la página, 3) resolución DNS del nombre del servidor donde están los gráficos, que acostumbra a ser el mismo servidor que para la página HTML, 3.1) solicitud de los gráficos necesarios (puede repetirse distintas veces), 4) contenido servido y página visualizada por el cliente (navegador).

Una petición a una CDN como Akamai aprovecha para tomar decisiones en cada paso de la petición:

Enlace de interés

Es interesante visitar la web de Akamai.

Figura 16



Fases de una petición web usando una CDN.

La CDN puede actuar sobre:

- El DNS: cuando se resuelve un nombre, el servidor DNS responde según la ubicación de la dirección IP del cliente.

- El servidor web: cuando se pide una página web, se reescriben los URL internos según la ubicación de la dirección IP del cliente.

- Cuando se pide un objeto a la dirección IP de un servidor de la CDN, el conmutador[2] de acceso a un conjunto de distintos servidores *proxycache*, o réplicas, puede redirigir la conexión al servidor menos cargado del grupo (todo el conjunto responde bajo una misma dirección IP).

(2)En inglés, *switch*.

Los pasos de la petición de una página web con gráficos podrían ser los siguientes:

0. El usuario escribe un URL (por ejemplo, http://www.adobe.com) en su navegador, y su navegador "resuelve" el nombre en DNS (192.150.14.120).

1. El usuario pide el URL a la dirección IP (192.150.14.120).

1.a. El servidor web elige la mejor réplica, según IP origen, estado de la Red y ubicación de las réplicas, o al menos clasifica al cliente en una zona geográfica determinada (por ejemplo, con Akamai decide que estamos en la zona del mundo o región "g").

2. El servidor devuelve el HTML con el URL de los gráficos incluidos en la página (marcas <img>), que se han redirigido a una réplica. De esta manera, los gráficos que constituyen el mayor número de bytes de una página web serán transferidos por la CDN. Por ejemplo:

```
<img          src="http://a1944.g.akamai.net/f/1944/1482/8h/
fondo.jpg">
```

3. El cliente resuelve el nombre y pide gráficos u otro contenido a la réplica más cercana.

3.a. Posible redirección con el DNS o por redirección de la conexión TCP (en el nivel del TCP o el HTTP: un conmutador de nivel 4 o 7), reparto de la carga en un grupo de servidores.

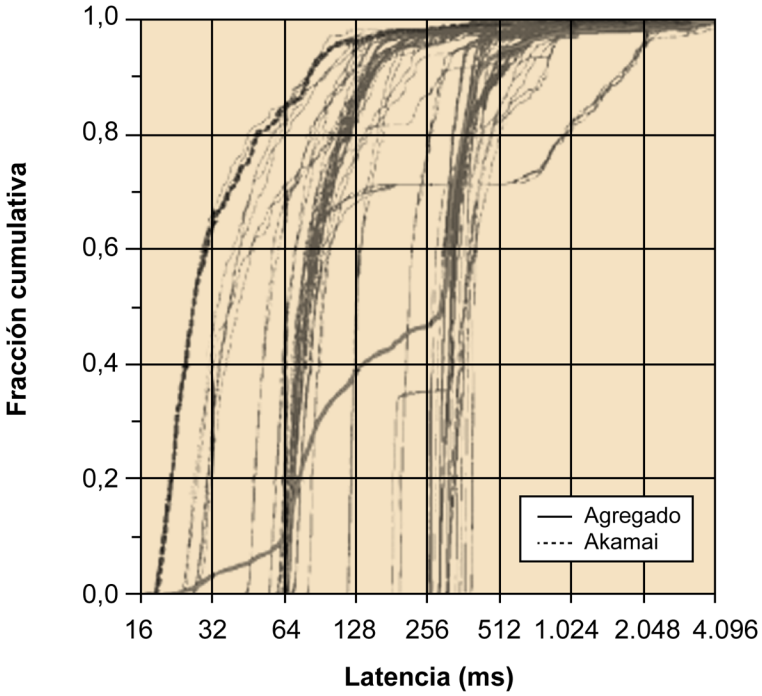
4. Contenido servido más rápido desde una réplica.

La ventaja de todo este montaje es que el usuario sigue viendo en su navegador un URL normal (el de la página HTML), el servidor de la empresa tiene *logs* con visitas a páginas HTML y el departamento de marketing puede hacer estadísticas, pero la mayoría del tráfico (los gráficos) lo sirve Akamai desde la proximidad del cliente.

El servei d'Akamai des de la proximitat del client

Akamai mantiene información del estado de la Red, de los *clusters*, de sus "servidores". No siempre elige el "mejor posible", pero sí uno de los mejores, y en cambio evita los peores servidores en cada momento.

Figura 17



Distribución acumulada de tiempo de respuesta (latencia) de distintos servidores Akamai.

La gráfica anterior muestra en líneas finas el tiempo que se tarda en obtener desde un lugar determinado un objeto de 4Kb pedido a muchos servidores de Akamai.

El eje *x*, en escala logarítmica, mide el tiempo (ms). El eje *y* mide la fracción acumulada de casos en los que el tiempo de descarga es inferior a un cierto valor. La línea gruesa contínua (*agregado*) mide la media de todos los servidores, y la línea gruesa discontínua (*akamai*) muestra el tiempo de respuesta del servidor que selecciona Akamai en cada momento (prácticamente siempre selecciona el mejor).

La línea gruesa continua indica que si se hiciese una elección aleatoria, para el 20% (0,2) de los casos el tiempo de servicio sería inferior a 64 ms, pero para el 80% de los casos el tiempo sería inferior a 400 ms. Con la elección de Akamai, el 80% de las peticiones se sirven en menos de 50 ms. Si eligiéramos uno de los peores servidores, sólo podríamos decir que el 80% de las peticiones se sirven en menos de 1.000 ms.

Consecuencia: la decisión de Akamai es casi ideal, mucho mejor que la decisión aleatoria, y por supuesto que el peor caso (Ley de Murphy).

5. Computación orientada a servicios

Las aplicaciones web devuelven resultados que pueden corresponder al contenido de un archivo (contenidos estáticos) o ser el resultado de la ejecución de un programa (contenidos dinámicos). Las aplicaciones que generan contenidos dinámicos pueden requerir pequeñas o grandes dosis de computación y almacenamiento. Algunos ejemplos son los buscadores web, las tiendas en Internet, los lugares de subastas, los servicios de mapas o imágenes de satélite, que pueden necesitar hacer complejos cálculos, buscas, o servir enormes volúmenes de información. Dentro de las organizaciones también se utilizan sistemas complejos que procesan cantidades ingentes de datos y que presentan sus resultados a través de un navegador. Son ejemplos de ello las aplicaciones financieras, científicas, de análisis de mercados, que tratan con enormes cantidades de datos que hay que recoger, simular, predecir, resumir, estimar, etc., para que unas personas puedan evaluar una situación y tomar decisiones.

En estos sistemas cada unidad autónoma de procesamiento se puede agrupar en un servicio y la forma de interacción entre ellos se suele basar en servicios web. En cuanto a la interacción con el usuario, todo este conjunto de servicios se puede esconder detrás de un servidor web y de una interfaz de usuario que puede ser tradicional (basada en formularios HTML sencillos) o bien ser más interactiva y más parecida a una aplicación centralizada que utilice las capacidades avanzadas de los navegadores como AJAX. En este modelo de aplicaciones o sistemas, el procesamiento está repartido entre el que ocurre en el navegador del usuario (relacionado con la presentación e interacción con el usuario), el servidor web (mediación entre una aplicación en el servidor y el navegador remoto) y otros servidores y servicios localizados potencialmente en otras máquinas, ubicaciones e incluso otras organizaciones, siguiendo un modelo federado. Estos sistemas pueden tener una estructura fija o bien dinámica, en las cuales las herramientas y los recursos se incorporan según la demanda (un modelo de uso llamado *utility computing*).

Utility computing

Se define como el suministro de recursos computacionales, como pueden ser el procesamiento y el almacenamiento, como servicio cuantificado parecido a las infraestructuras públicas tradicionales (como por ejemplo la electricidad, el agua, el gas natural o el teléfono).

5.1. SOA en detalle

Se hace muy difícil encontrar una definición exacta de lo que quiere decir la expresión *arquitectura orientada a servicios* (SOA). El problema es que hay muchas definiciones diferentes que contienen algunas palabras comunes pero que no acaban de converger. Todas las definiciones sin embargo están de acuerdo con el hecho de que el término *SOA*[3] hace referencia a un paradigma que permite una mayor flexibilidad en el desarrollo de aplicaciones web.

[3]SOA es la sigla de *arquitecturas orientadas a servicios*.

SOA no es una arquitectura concreta, sino un paradigma. Para entenderlo tenemos que pensar que es algo que nos lleva hacia una manera de hacer que nos permite desarrollar una arquitectura concreta. Lo podemos denominar *estilo*, *paradigma*, *perspectiva*, *concepto*, *filosofía*, *representación* o *manera de pensar*, que nos lleva a desarrollar aplicaciones de una cierta manera.

Las SOA son adecuadas para el desarrollo de aplicaciones distribuidas, sobre todo a gran escala, puesto que facilitan la interacción entre proveedores de servicios y consumidores de los servicios.

Entonces, las **arquitecturas orientadas a servicios** (**SOA**) se pueden definir como un paradigma de diseño de software que impone el uso de los servicios web para apoyar los requisitos de los usuarios. Así pues, en una aplicación distribuida los recursos se harán accesibles a través de servicios independientes, a los que se podrá acceder sin conocer los detalles de la implementación.

Las arquitecturas orientadas a servicios están basadas en el uso de servicios web estándar, como por ejemplo SOAP o REST, aunque se pueden implementar con cualquier especificación de servicio web. Un requisito deseable –a pesar de que no es obligatorio– de cualquier servicio web es que este sea sin estado, puesto que favorece la escalabilidad y la independencia de ejecución de cada servicio. Este requisito ha favorecido que el uso de los servicios basados en REST haya aumentado mucho en los últimos años.

Las propiedades principales de una arquitectura orientada a servicios son:

- **Distribución**. Las funcionalidades de la arquitectura se ofrecen como servicios que pueden estar ubicados en diferentes lugares, incluso un mismo servicio puede ser ofrecido de forma distribuida.

- **Heterogeneidad**. Los servicios son independientes de la tecnología subyacente; por lo tanto, esta tecnología se puede ejecutar en máquinas diferentes, sobre sistemas operativos diferentes y puede estar implementada con lenguajes diferentes. A la vez, los servicios también pueden ser heterogéneos, se pueden ofrecer diferentes servicios o un mismo servicio implementado de diferentes maneras.

- **Interoperabilidad**. La especificación de los servicios sigue estándares como los de OASIS o los contratos definidos con WSDL. Esto hace que servicios implementados con tecnologías diferentes puedan interoperar porque la interfaz es común.

- **Desacoplamiento**. Los servicios exponen funcionalidades que se pueden ofrecer de forma remota y distribuida. Esto permite desacoplar el servicio

Lectura recomendada

La obra siguiente nos puede servir de referencia para profundizar en las arquitecturas orientadas a servicios:

Nicolai M. Josuttis (2007). *SOA in practice: the art of distributed system design*. O'reilly.

OASIS

OASIS es un consorcio sin ánimo de lucro que guía y regula el desarrollo, la convergencia y la adopción de estándares abiertos por parte de la sociedad de la información.

de la máquina en cuanto que un servicio puede estar replicado o distribuido, o puede migrar a otras máquinas.

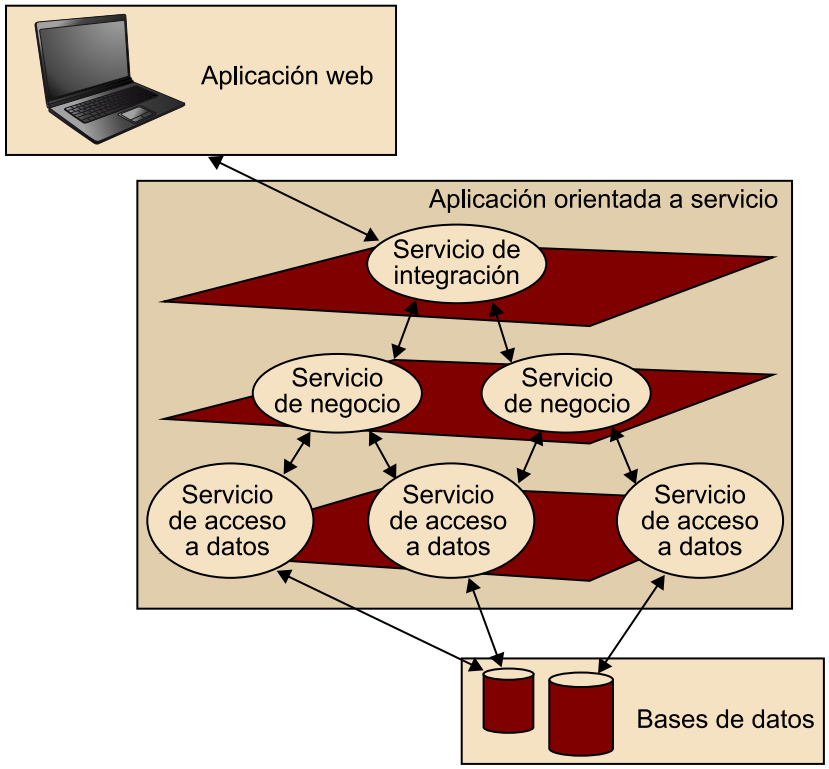
- **Flexibilidad**. Un servicio puede apoyar o ser usado por diferentes aplicaciones para permitir un aprovechamiento de funcionalidades y más flexibilidad en el desarrollo de nuevas aplicaciones. El hecho de que los servicios se expongan con interfaces estándar permite un desarrollo más heterogéneo de las aplicaciones y una elección más adecuada de las tecnologías según cada situación.

- **Escalabilidad**. Los servicios se pueden replicar y distribuir de forma transparente al usuario, que los ve como una única fachada. Esta característica de las arquitecturas orientadas a servicios favorece la implementación de aplicaciones que pueden soportar grandes cantidades de usuarios.

- **Tolerante a fallos**. Gracias al desacoplamiento de la especificación del servicio y de la máquina que lo ejecuta, se pueden desarrollar técnicas para que un fallo de un servicio pueda ser atendido por una réplica de forma transparente al usuario.

No tenemos que pensar que los servicios de una SOA siempre apoyan a un usuario o a una funcionalidad de la arquitectura, sino que muchas veces los servicios se organizan en jerarquías de servicios que constituyen una arquitectura en varias capas formadas por servicios. Este tipo de organización permite desarrollar aplicaciones tan complejas como las aplicaciones de la banca, las que rigen el funcionamiento de grandes cadenas de producción, las de grandes lugares web como Amazon o eBay, etc.

Figura 18



Estructura de una aplicación desarrollada siguiendo una arquitectura orientada a servicios.

5.2. *Grid computing*

Los sistemas de parrilla de cálculo[4] tienen como objetivo la "metacomputación", es decir, capacidades computacionales a escala Internet. No se pueden hacer asunciones sobre el tipo de hardware, sistemas operativos, interconexiones de red, dominios administrativos, políticas de seguridad, de este sistema.

[4] En inglés, *grid computing*.

Cuando se habla de *grid* se hace referencia a una infraestructura que comporta el uso integrado y colaborativo de ordenadores, redes, bases de datos e instrumentos científicos que son de propiedad y están gestionados por diferentes organizaciones. A menudo, las aplicaciones en parrilla de cálculo trabajan con grandes cantidades de datos y/o recursos computacionales, que requieren una compartición segura de recursos que atraviesan diferentes límites organizativos o dominios de administración. Por su parte, e idealmente, el usuario tiene una visión del sistema de cálculo en parrilla como si fuera un único sistema informático, puesto que este le proporciona un acceso uniforme a los recursos.

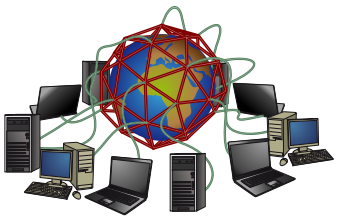


Figura 19
Una parrilla de cálculo puede agregar recursos heterogéneos de todo el mundo.

Como las SOA que hemos visto antes, la definición de *grid* ha sido muy amplia y abierta. A continuación, presentamos una serie de definiciones que pueden ayudar a entender mejor qué se quiere decir cuando se habla de *grid*.

Definiciones

[Sobre la tecnologia *grid*] "la tecnología que posibilita la virtualización de recursos, el aprovisionamiento bajo demanda y la compartición de servicios (recursos) entre organizaciones."

Plaszczak/Wellner

[Sobre el *grid computing*] "la posiblidad, usando un conjunto de protocolos y estándares abiertos, de ganar acceso a aplicaciones y datos, potencia de procesado, capacidad de almacenamiento y un amplio abanico de otros recursos de computación por Internet. Una parrilla es un tipo de sistema paralelo y distribuido que permite la compartición, la selección y la agregación de recursos distribuidos a través de dominios administrativos «múltiples», basados en su disponibilidad (de recursos), capacidad, potencia de ejecución, coste y requisitos de calidad de servicio de los usuarios."

IBM, "IBM Solutions Grid for Business Partners: Helping IBM Business Partners to Grid-enable applications for the next phase of e-business on demand"

[Sobre el *grid*] "un tipo de sistema paralelo y distribuido que permite la compartición, la selección y la agregación de recursos autónomos geográficamente dispersos de manera dinámica y en tiempo de ejecución según su disponibilidad, capacidad, potencia de ejecución, coste y requisitos de calidad de servicio de los usuarios."

Rajkumar Buyya, Srikumar Venugopal, "A Gentle Introduction to Grid Computing and Technologies" (2005)

Se empieza a hablar de parrillas de cálculo a partir de la segunda mitad de los años noventa del siglo XX. Como en el caso de igual a igual, los sistemas de parrilla de cálculo fueron una consecuencia del aumento sustancial en el rendimiento de los ordenadores personales y de las redes en los últimos veinte años. Por otro lado, gracias a la combinación entre el abaratamiento de los ordenadores personales y su aumento de potencia, proliferaron sistemas de altas prestaciones a bajo coste, que permitieron a muchos colectivos disponer de suficiente potencia de cómputo para solucionar problemas que requerían un uso intensivo de recursos sin tener que disponer de superordenadores.

El término *grid*

Esta infraestructura se denominó *grid* por analogía con la red eléctrica (en inglés, *electrical power grid*), que proporciona un acceso universal, fiable, compatible y transparente a la energía eléctrica, con independencia de su origen.

En particular, el mundo científico se ha beneficiado de esta potencia de cómputo para hacer simulaciones y experimentos mucho más exhaustivos y para los cuales antes había que disponer de grandes superordenadores. Las redes más rápidas han permitido compartir datos de los instrumentos y resultados de los experimentos con colaboradores de todo el mundo casi instantáneamente. En este contexto, las parrillas de cálculo nacen como un paso más en este esfuerzo de colaboración y compartición.

En el año 2007 el término *computación en nube*[5] se hizo popular hasta desbancar al concepto de *parrilla de cálculo*. No obstante, y como veremos más adelante, el *grid computing* y el *cloud computing* mantienen cierta relación, cuando

(5)En inglés, *cloud computing*.

menos la que Foster postulaba ya el año 1999 en términos de computación consumida como si fuera electricidad. Incluso el término *grid* se asocia a veces a la capacidad de computación de los sistemas *cloud computing*.

Lectura recomendada

Ian Foster's, Carl Kesselman's (1999). "The Grid: Blueprint for a new computing infrastructure".

5.3. *Cloud computing*

La **computación en nube**[6] es una forma de computación fundamentada en Internet, mediante la cual los recursos compartidos, el software y la información, son ofrecidos a todo tipo de dispositivos con acceso a la Red bajo demanda como servicios ubicados en Internet.

(6)En inglés, *cloud computing*

Se trata de un cambio de paradigma después del paso de estructura de ordenador central (*mainframe*) a estructura de cliente-servidor, que lo precedió en la década de 1980. Los detalles son transparentes para los usuarios que ya no tienen necesidad de tener conocimientos técnicos, ni control sobre la infraestructura de tecnología "en nube" que los apoya. La computación en nube describe un nuevo suplemento, consumo y modelo de prestación de servicios de tecnologías de la información (TI) basados en Internet, y que generalmente implica el suministro dinámico escalable y muchas veces virtualizado de los recursos como servicio a través de Internet. Es consecuencia de la facilidad de acceso a los lugares remotos de computación que ofrece Internet gracias al al incremento en la calidad de las conexiones (ancho de banda, latencia y fiabilidad).

Mainframe

Ordenador central, potente y costoso utilizado para el procesamiento y la gestión de una gran cantidad de datos y/o usuarios. En los inicios de Internet, la mayoría de empresas tenían su *mainframe*, que daba servicio a todos los empleados mediante terminales muy sencillos.

Figura 20
Imagen de un mainframe de los años 70.

Los sistemas de computación en nube han surgido de una evolución lógica de los sistemas de parrilla de cálculo. Mientras que la parrilla procuraba el acceso a recursos bajo demanda, la nube ha pretendido desacoplar la demanda de la infraestructura física ofreciendo la computación no en forma de recurso sino en forma de servicio, haciéndola transparente a la infraestructura subyacente.

La computación bajo demanda se puede encontrar enfocada desde niveles diferentes:

- **SaaS**[7]. Se ofrece el software como servicio, es decir, el proveedor de computación en nube ofrece como un servicio software alojado en sus máquinas a otros que obtienen el acceso a aquel software. Los clientes no se tienen que preocupar del mantenimiento, las actualizaciones ni las licencias del software, y además, pueden acceder a él desde donde quieran.

(7)Acrónimo de *software as a service*.

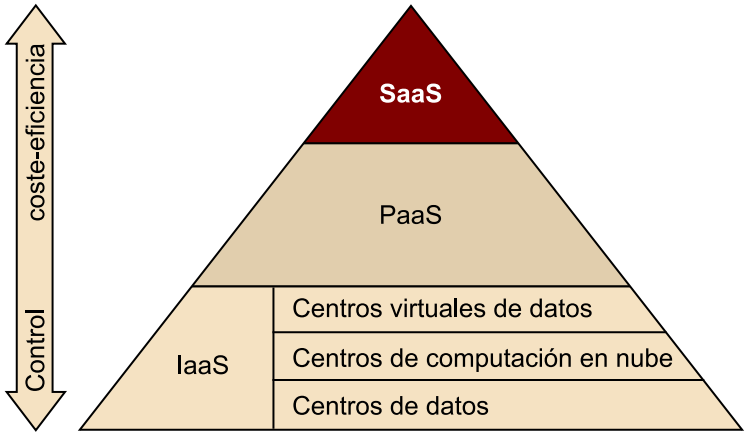
- **PaaS**[8]. Los proveedores de computación en nube ofrecen servicios de acceso y uso de plataformas de software específicas, como servidores de aplicaciones, sistemas operativos, entornos de trabajo específicos y todo en forma de servicio sin que el cliente se tenga que preocupar de la infraestructura necesaria para mantener las plataformas.

(8)Acrónimo de *platform as a service*.

- **IaaS**[9]. Este nivel de computación en nube es lo más parecido al sistema de parrilla de cálculo. La infraestructura como servicio es el ofrecimiento de hardware, la capacidad de comunicación, etc., no limitado y ampliable transparentemente, que evita la complejidad de mantenimiento y agregación de hardware a los clientes.

(9)Acrónimo de *infrastructure as a service*.

Figura 21



Diferentes formas de ofrecer servicios de computación en nube. Cuanto más arriba en la pirámide, mayor es la relación coste-eficiencia de los servicios en detrimento del control que tenemos de ellos. Cuanto más cerca del hardware, los clientes tienen más control del sistema pero a expensas de su mantenimiento.

Resumen

En este módulo se han presentado las formas con las que un servicio web o una aplicación asociada a un servidor web se pueden organizar para atender la demanda que pueden recibir de Internet. Muchos indicadores de Internet continúan creciendo de forma exponencial: el número de personas con acceso a Internet y el tráfico web, que hoy en día predomina sobre el resto de protocolos. La popularidad de los contenidos sigue la ley de Zipf, la ley de la desigualdad extrema: muchas visitas para muy pocos lugares. Esto hace que la demanda que pueda recibir en un momento concreto un servidor web pueda ser exageradamente grande. Conocer las características principales de esta demanda ayuda a prever situaciones en el diseño de un servicio web.

La capacidad para servir peticiones es difícil de prever en un sistema tan complejo en el que influyen, entre muchos factores, la capacidad de la Red y su carga, las características de toda la máquina, el sistema operativo con numerosos subsistemas o el servidor web. Es conveniente conocer la capacidad de servicio de nuestro servidor y observar su comportamiento con niveles de carga elevados usando herramientas de generación de tráfico y medida del comportamiento bajo una carga sintética, así como herramientas de visualización de estadísticas de demanda real a partir de diarios (*logs*), que nos permitan detectar sobrecargas, sintonizar o actualizar el conjunto adecuadamente.

Las aplicaciones en servidores web tienen dos componentes principales: el servidor web y el código adicional que amplía el servidor para ofrecer servicios o contenidos "dinámicos". El servidor web es un sistema que se encarga de servir algunas de las muchas peticiones a la vez, por lo cual, se tiene que optimizar la organización de toda esta actividad simultánea a partir de procesos, flujos y fibras. Las aplicaciones web reciben peticiones del servidor y devuelven un resultado para enviar al cliente.

Otro componente importante son los servidores intermedios entre navegadores y servidores web que desean copias de los objetos que ven pasar desde un servidor hacia un navegador. Principalmente, permiten "acortar" peticiones sirviéndolas a medio camino del servidor, en la memoria, con objetos recibidos recientemente, hecho que reduce la congestión de Internet y la carga de los servidores.

Mientras que los servidores de memoria rápida de trabajo se suelen situar cerca de los lectores, la demanda global de ciertos contenidos o la tolerancia a fallos puede recomendar ofrecer un servicio web desde varias ubicaciones. Hay va-

rias maneras de montar un servicio distribuido: replicación o *mirroring*, DNS *round-roben*, redireccionamiento a nivel de transporte o HTTP, servidores intermediarios inversos.

Otra alternativa es contratar la provisión de servicio a una red de distribución de contenidos o CDN, que es una infraestructura comercial compartida por varios clientes con infinidad de puntos de presencia próximos a la mayoría de usuarios de Internet, que se encargan de repartir y dirigir las peticiones web hacia los servidores menos cargados y más cercanos en su origen de la petición.

La computación bajo demanda es un modelo más general que permite distribuir un sistema en varios componentes distribuidos que se comunican con invocaciones a servicios y el uso de recursos (computación, almacenamiento, servicios de aplicación), que se asignan dinámicamente según cuál sea la necesidad o el volumen de demanda de sus usuarios. Hemos visto las SOA como paradigma de construcción de aplicaciones a escala de Internet, los sistemas de parrilla de cálculo y su evolución hacia sistemas de computación en nube, muy relevantes hoy en día.

Bibliografía

Krishnamurthy, B.; Rexford, J. (2001). *Web protocols and practice: HTTP/1.1, networking protocols, caching, and traffic measurement*. Cambridge: Addison-Wesley (ISBN 0-201-71088-9).

Un libro exhaustivo y detallado sobre el funcionamiento de la web más allá de la presentación a los navegadores. Estudia los detalles de las diversas versiones del protocolo HTTP, su rendimiento, la interacción con los otros servicios involucrados: DNS, TCP, IP, la evolución histórica de los componentes de la web, *scripts*, buscadores, galletas, autenticación, técnicas para recoger y analizar tráfico web, *proxy-caches* web e interacción entre *proxy-caches*.

Luotonen, A. (1998). *Web proxy servers*. Londres: Addison-Wesley (ISBN 0-13-680612-0).

Ari desarrolló el servidor *proxy-cache* del CERN y también el servidor intermediario de Netscape. Habla de los detalles de la estructura interna (procesos) de los servidores, cooperación entre servidores, mediación, filtrado, monitorización, control de acceso, seguridad, aspectos que afectan al rendimiento de un servidor intermediario y otros aspectos.

Luotonen, A.; Altis, K. (1994). "World-Wide Web Proxies". Actas de la Conferencia WWW94.

Es el artículo histórico sobre *proxy-cache* en web, basado en la experiencia con la memoria web del CERN.

Rabinovich, M.; Spatscheck, O. (2002). *Web caching and replication*. Boston (EE.UU.): Addison-Wesley (ISBN 0-201-61570-3).

Un libro exhaustivo y detallado sobre los avances recientes en memorias rápidas de trabajo y replicación para la web. Los capítulos 4: "Protocolos de aplicación para la web", 5: "Apoyo HTTP para *proxy-cache* y replicación", 6: "Comportamiento de la web", ofrecen una buena visión general del problema y los mecanismos que influyen en él. La parte II analiza con detalle los mecanismos de *proxy-cache* y la parte III, los mecanismos de replicación en la web. Hay muchos libros específicos y detallados sobre cada tecnología explicada: sobre las especificaciones, sobre cómo utilizarla, cómo instalarla y administrarla, referencias breves, etc. Entre otros, la editorial O'Reilly tiene muchos libros bastantes buenos y muy actualizados sobre varios temas de la asignatura. Son muy detallados, mucho más de lo que se pretende en la asignatura, pero en la vida profesional pueden ser de gran utilidad para profundizar en los aspectos prácticos de un tema.

Jossuttis, N. (2007). *SOA in practice. The art of distributed system design*. Un libro muy completo y ameno sobre el paradigma de arquitecturas orientadas a servicios. Presenta el paradigma desde un punto de vista práctico y hace referencia a los aspectos más relevantes en el proceso de desarrollo de aplicaciones basadas en servicios web.