Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Desarrollo Web *Extremo* (63091 lectures)

Per **Ricardo Galli Granada**, ***gallir*** (http://mnm.uib.es/gallir/)

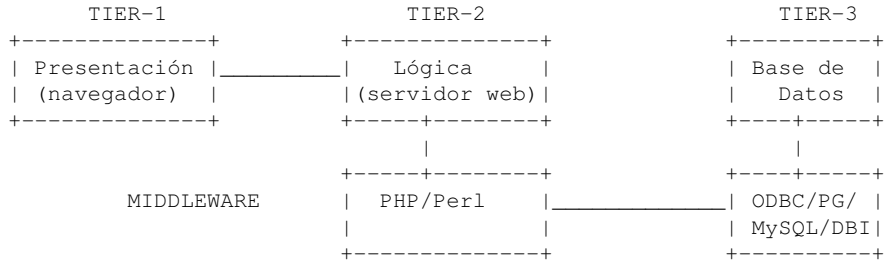
Creado el 16/07/2001 18:26 modificado el 16/07/2001 18:26

Uno de los problemas más importantes a la hora de desarrollar aplicaciones o sitios web dinámicos con base de datos y PHP (o cualquier otro lenguaje de *scripting*) es ¿existe una herramienta como el Dreamweaver (de Windows o Mac) que permita integrar código PHP y consultas en la base de datos? La respuesta es sí (algunas buenas, otras peores). Pero en mi opinión, este tipo de desarrollo de un web, con una herramienta que intenta hacerlo *todo*, debe evitarse. En este artículo explico las razones y una metodología muy *lightweight* que permiten separar el trabajo y desarrollar sitios web de forma más rápida y modular (el título del artículo parafrasea a *Extreme Programming*[1], mi metodología favorita ;-). Aprovecho también para explicar como hacer las pruebas de tiempos del servidor y el cálculo del ancho de banda necesario.

Introducción

Existe una contradicción muy interesante. Cuando se desarrollan aplicaciones web separamos muy claramente las distintas partes del sistema (arquitectura *multi-tier*):

- Presentación,
- Lógica de aplicación,
- Almacenamiento de datos.



ARQUITECTURA MULTI-TIER

Sin embargo a la hora del desarrollo, muchos directores de tecnología o de proyecto pretenden hacerlo **todo** con una sola herramienta, normalmente Dreamweaver Developer, Delphi o FrontPage.

¿Porque hacemos esto? ¿Porque el diseñador de un sitio debe saber como están almacenados los datos o como funciona la lógica de la aplicación?

La separación es necesaria y nos dará muchas ventajas:

- Primera y más importante, evitamos hacer dependiente todo un sitio web (scripts, diseño gráfico y maquetación) de una sola tecnología o aplicación.
- El diseñador no tendrá que preocuparse de aspectos de programación o base de datos.
- Podemos usar cualquier software para el desarrollo del diseño de la página web. Normalmente los diseñadores tienen su herramienta preferida. Puede ser que dicha herramienta no permita trabajar con MySQL, Postgres o

PHP, pero esto no debería significar ningún impedimento.
- Al permitir que un diseñador use su herramienta favorita, aumentaremos la productividad. Tampoco se puede pretender que el diseñador (en su mayor parte free-lance) tengan que cambiar de software cada vez que se desarrolla un web sobre tecnologías distintas (hoy puede ser PHP y MySQL, mañana Python o Postgres).
- No se le puede exigir a los programadores que aprendan a usar un determinado software de diseño o maquetación, que normalmente están orientados a diseñadores y con una curva de aprendizaje bastante elevada.
- Si permitimos que el trabajo de los programadores sea lo más independiente del diseño posible, y además le dejamos que usen sus propias herramientas, aumentaremos considerablemente la productividad.
- Y la última y no menos importante, las herramientas de desarrollo están evolucionando muy rápidamente, lo que es bueno hoy, podría ser muy malo mañana.

El desarrollo de un sitio web tiene por lo menos cuatro fases distintas:

1. Diseño conceptual.
2. Diseño gráfico y árbol de navegación.
3. Desarrollo.
4. Producción.

El problema radica en que los grupos de desarrollo, sobre todos aquellos que trabajan con presupuestos limitados, no suelen tener las fase de desarrollo claramente divididas, ni una especificación de las herramientas de software que se usarán en cada una de dichas fases. Aquí radica una de las barreras para el desarrollo de sitios web con Linux: se justifica la decisión de usar Windows, ASP y SQL Server en el hecho de que existen mejores herramientas de *integración*.

Considero que es un gran error, no sólo les hace dependientes de tecnologías propietarias (y dominantes), sino que hacemos que todo el negocio dependa en realidad de 2 o 3 proveedores de software, ya que siempre tendremos que pagar para poder acceder a las últimas herramientas desarrolladas por esas empresas.

> Es como si un usuario hogareño, para poder escuchar MP3, siempre tenga que estar actualizado a la última versión de Windows, que en 6 años a producido 6 versiones distintas, con precios que no bajan de las 100 - 150 por cada actualización. Llevado éstos a entorno empresariales o gubernamentales hace que los presupuestos se multipliquen enormemente.

Es un error intentar hacer cosas totalmente distintas con las mismos programas o herramientas: no existe la "bala de plata" (*The Mythical Man Month*, la *Biblia* de la ingeniería del software). Para evitarlo, lo que hay que hacer es separar las partes como ya se hace en el desarrollo de software modular o por *tiers*: la presentación, la lógica y la gestión de almacenamiento de datos.

Los *scripts* en PHP, o en cualquier otro lenguaje, son parte de la lógica. No tienen que ser conocidos y modificados por la misma persona que hace el diseño (es el encargado de sólo una parte de la *presentación*), es un trabajo totalmente distinto y no puede producir los mismos resultados.

Tampoco se puede admitir que una herramienta que es apta para diseñadores con pocos conocimientos de programación sea apta para un programador experto. No es sólo aplicar el zapatero a tus zapatos, sino usar la herramienta adecuada para cada trabajo. Los pinceles y el óleo de los artistas son de poca ayuda para un relojero.

Metodología a seguir

Propongo a continuación cuatro pasos básicos a seguir para el desarrollo de un sitio web de forma independiente a las herramientas que usen los diseñadores gráficos. También pongo énfasis en la optimización de las estructuras de las tablas para dar mayor sensación de agilidad al usuario y como estimar y reducir los tiempos de descarga de la páginas.

Estos pasos fueron usados de manera estricta pero no formal en el desarrollo de varios sitios web complejos en los que he dirigido o participado. El más notable quizás sea el web de Última Hora[2] (a mediados de 1998) ya que es un sitio con un diseño relativamente sofisticado, todo los datos están en una base de datos relacional, que además tiene una réplica en la red local de la empresa editora. Todo el web, inclusive la totalidad del diseño y maquetación, fue desarrollado en menos de un mes y partiendo desde cero.

A los 20 días ya estuvo on-line y a los 30 días ya estaba a disposición del todo el público. Desde esa época se ejecuta sobre el mismo servidor, con los mismos programas (CGIs y Perl) y hasta el día de hoy no ha fallado un sólo día. Aunque los diseñadores usaron Photoshop, Illustrator y Dreamweaver, todo el desarrollo de los programas y consultas a la base de datos se hicieron con editores de texto estándares de Linux (principalmente el vi).

1. Diseño conceptual

Se fijan las pautas con el cliente, como será el sitio, que datos tendrá, como se navegará, tipo de cliente al que irá orientado, etc. De esta fase se tiene una idea del diseño y la navegación que tendrá el sitio. El resultado ideal de esta fase es una **prueba de concepto**. La prueba de concepto puede tener dos partes:

1. Diseño gráfico del sitio. Esta parte es preparada por el diseñador principalmente, y suele bastar con los dibujos hechos en aplicaciones de diseño como el Illustrator, FreeHand, KIllustrator e inclusive con software de imágenes como el Gimp o Photoshop.
2. Consulta y carga de datos: Esta parte es preparada principalmente por los programadores y consistirá en unas páginas HTML sencillas generadas por los scripts y que accedan a un prototipo simplificado de la base de datos.

La prueba de concepto no sólo es una buena herramienta para negociar y convencer al cliente, sino también para detectar que información es la que hace falta para completar el diseño de la base de datos y aplicaciones, como así también detectar de antemano los problemas que nos encontraremos en el desarrollo.

2. Diseño gráfico y árboles de navegación

Se hace un diseño detallado de la diferentes páginas que tendrá el web, el diseño gráfico normalmente realizado en una aplicación de dibujo (como el Illustrator), y los árboles de navegación. Si es un sitio dinámico se necesitan tener las pautas y restricciones técnicas de consultas a la base de datos. Aquí es muy importante la discusión entre los diseñadores (gráficos y HTML) y los desarrolladores, los diseñadores no deberían conocer la lógica de la aplicación pero sí las restricciones que esta impone al diseño.

Por ejemplo, si una consulta a la base de datos pueden generar más de 30 o 40 resultados, debe preverse la paginación de la salida, lo que incluye el diseño y enlaces para cada página. También es importante que la longitud y formato de los datos de la base de datos sean adecuados para el diseño que se presenta. Si se generan textos largos, no se pueden poder los mismos en una columna muy angosta y de poca altura.

Lo más importante que hay que tener en cuenta a la hora del diseño conceptual de la parte gráfica y lógica del sitio es el tiempo que tardará el cliente para visualizar el resultado (**tiempo de visualización**). Hay muchos parámetros que influyen, por ejemplo:

- **Tiempo de respuesta**: cuanto tiempo tarda en servidor en empezar a devolver resultados al navegador. Este es el parámetro más importante que debe tener en cuenta el o los programadores. De él depende que se pueda ajustar el diseño para que los usuarios puedan empezar a visualizar la página lo más rápido posible. Si las primeras consultas a la base de datos son muy complejas o tenemos habilitado el *buffering* en el lado servidor (en PHP4 se hace con *ob_start()*), podemos afectar negativamente el resultado. Por el contrario, si el resultado se obtiene muy rápidamente el uso del *buffering* puede ayudar bastante al envío eficiente de los datos a través de la red. Para que la página de una sensación de agilidad, es importante que el tiempo de respuesta no supere los 2 o 3 segundos.
- **Tiempo de retorno**: cuanto tiempo tarda el servidor en terminar de ejecutar los programas en el servidor y entregar todos los datos y será siempre superior al tiempo de respuesta ($T_{retorno} > T_{repuesta}$). El tiempo que tarde el programa en terminar de generar todos los datos no sólo influirá en la conexión con un usuario en particular, sino con el rendimiento de todo el sistema. A mayor tiempo de retorno, menor cantidad de conexiones simultáneas posibles y mayor carga de todo el sistema. Si el tiempo de retorno de un script es superior a un 1 segundo, hay que estudiarlo detenidamente. El primer estudio a hacer es el consumo de CPU. Si ésta es baja, tenemos problemas de latencia, posiblemente con la conexión a la base de datos. Si por el contrario el consumo es elevado, la lógica del programa es muy compleja o usamos muchas llamadas de sistema. En estos casos puede ayuda el uso de sistemas de cache de código[3].
- **Tiempo de descarga**: es el tiempo que tarda el cliente en bajarse todos los datos a su ordenador. Este tiempo es siempre mayor al tiempo de respuesta ($T_{descarga} > T_{retorno}$) y depende de la velocidad de conexión.

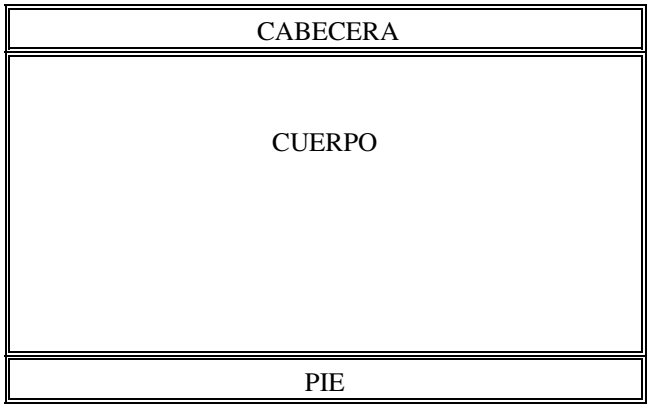
Tiempo de visualización

El primer aspecto a tener en cuenta es la cantidad de código HTML que debe ser bajado para cada página. Esto depende del tipo de conexión que tendrán los visitantes del sitio. Un buen parámetro es que el tamaño máximo del HTML no debe superar los 30-40 KBytes por página genérica. En el caso de imágenes es un poco más flexible, ya que se puede acelerar la previsualización de la página indicando los tamaños en pixels de cada imagen.

Las páginas de sitios dinámicos suelen tener una estructura bien definida:

1. Cabecera: aquí se suelen poner el logo de la empresa, menú de opciones de navegación, *banners* de publicidad.
2. Cuerpo: en estar parte se pone la parte principal de la página. También suele estar dividida en dos a tres columnas que sirven para un menú de navegación y otra para un índice de temas o artículos relacionados
3. Pié: aquí suele ir información para contactar a la empresa, direcciones, teléfonos y otros enlaces corporativos.

Un error común es poner esas tres partes anidadas es una sola tabla:



Diseño erróneo de una página con tres tablas anidadas

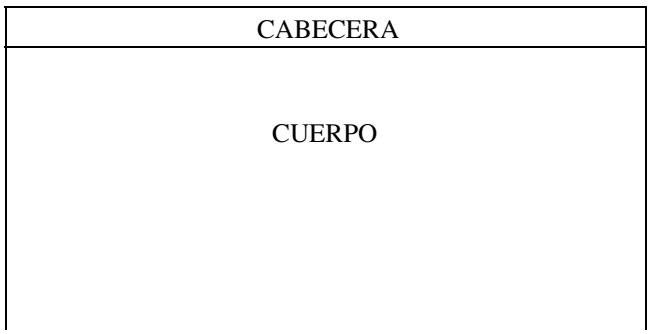
Este tipo de formato hace que el navegador no pueda generar la página hasta que se reciba el último byte (cercano al tiempo de descarga) que define a la tabla (normalmente el cierre de tabla: </table>).

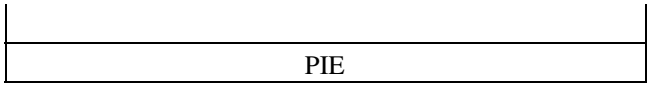
Supongamos que la página tiene 40KB de tamaño (en HTML) , que el tiempo de respuesta del servidor es de 0.5 segundos y que su tiempo de retorno es los suficientemente bajo para alimentar de datos continuamente. Si la conexión del clientes de es un RDSI (64 kbps =~ 7.5 KB/seg) y en condiciones ideales, el tiempo qua tardará el navegador en empezar a generar la página en pantalla será igual al tiempo de descarga:

$$T_{vis} = T_{descarga} = 0.5 \text{ seg} + 40 \text{ KB}/7.5\text{KBseg} = 0.5 \text{ seg} + 5.33 \text{ seg} =\sim \textbf{6 segundos}$$

Aunque este tiempo parezca razonable, no es aceptable, porque en la mayoría de las conexiones no tendremos los 64kbps sólo para bajar el HTML, sino también las imágenes, saturación de la red, etc., por lo que fácilmente **dicho valor se duplica**.

La forma de correcta de solucionar el problema es de intentar anidar la menor cantidad de tablas posibles, y evitar a toda costa tener una sola tabla principal que engloba a toda la página. Para ello lo mejor es hacer una división horizontal de las tablas:



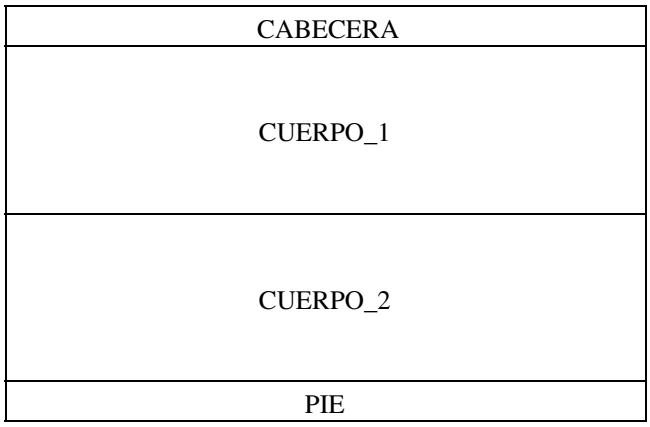


Diseño correcto con tres tablas independientes

Con la forma de especificar las tablas en la figura anterior, se puede obtener los mismos resultados visuales con una mejora sustancial en el tiempo de visualización. Supongamos que la tabla CABECERA está definida por los primeros 3KB (más que suficiente en la mayoría de los casos), por lo tanto el tiempo que esperará el cliente para empezar a pre-visualizar la página será:

$$T_{vis} = 0.5 \text{ seg} + 4 \text{ KB}/7.5\text{KBseg} = 0.5 \text{ seg} + 0.53 \text{ seg} =\sim \textbf{1 segundo}$$

Como se puede ver, obtendríamos una mejora de casi un 700% con sólo estructurar mejora las tablas. Si la estructuras de la páginas son más complejas, conviene seguir con la técnica de sub-dividir horizontalmente a las tablas:



Diseño correcto con cuatro tablas independientes

3. Desarrollo

En esta fase se desarrollan los gráficos, el código HTML y los *scripts* necesarios. Aquí aparece el problema, los responsables de proyectos buscan que todo esto se haga con una sola aplicación.

Es un error buscar ese tipo de integración, no solamente por cuestiones de independencia tecnológica, sino porque el código generado suele ser ofuscado, dependiente de varios ficheros "desconocidos" y muy difíciles de optimizar.

Es mejor dividir el desarrollo en dos partes claramente diferenciadas y que se podrán desarrollar en paralelo.

1. Desarrollo gráfico y HTML
2. Desarrollo de programas

3.a Desarrollo gráfico y HTML

Con las herramientas existentes, normalmente es el mismo diseñador el que se encarga del diseño gráfico y la maquetación (sobre todo en proyectos pequeños). Cada diseñador tiene sus propias herramientas y distintas plataformas (Dreamweaver sobre Windows y Mac principalmente). Esto no debe servir de excusa para diseñar los programas del servidor en aquellos lenguajes y *recordsets* soportados por la herramienta (i.e. ASP y SQL Server, o Cold Fusion).

Lo mejor que se puede hacer es que el o los diseñadores desarrollen todas las páginas con HTML estático y texto *falso* con su herramienta favorita, luego el texto falso se reemplazará por las partes de código (PHP) que generen la salida deseada. Para evitar problemas de integración, es mejor que los diseñadores trabajen asistidos por los programadores para decidir que tipo de texto salida se puede poner en cada página.

Para facilitar la posterior integración con los programas, a la hora de la maquetación en HTML hay que tener en cuenta lo siguiente:

1. Poner comentarios en HTML del tipo de salida o consulta en cada sección.

2. Usar desde el principio nombres de fichero con las extensiones correspondientes que se usarán en la versión definitiva (php, phtml, pl, cgi, etc.). De esta forma se evitarán problemas de enlaces por inconsistencia en los nombres.
3. Usar siempre enlaces relativos, de esta forma se podrá mover fácilmente los programas a distintos directorios o servidores virtuales.
4. Poner todo lo que sea código JavaScript o imágenes en subdirectorios y usar referencias relativas.
5. Grabar todos los ficheros en formato UNIX y con hora GMT.

3.b Desarrollo de programas

Al mismo tiempo que el o los diseñadores preparan el código HTML, el o los programadores deben preparar los programas usando HTML muy simples para probar la base de datos y ejecución de los programas.

Las herramientas que se usen en estas fases serán normalmente aquellas que use el programador y que son básicamente un editor de texto y acceso al servidor por medio de TELNET (o SSH). La ventaja principal es que permiten a los programadores libertad de horarios y de ubicación, ya que la mayoría de este trabajo puede ser realizado remotamente. En caso de trabajo en grupo se recomienda el uso del CVS[4], ya que permitirá el control de concurrencia, cambios, parches y versiones con mucha facilidad.

Lo más importante es tener acabada la mayor parte de la programación antes que se acabe el diseño e integrar paulatinamente los módulos acabados en aquellas páginas ya finalizadas por los diseñadores. Además se debe controlar periódicamente que los enlaces y referencias de las páginas HTML sean válidos y relativos.

Pruebas

Cuando el desarrollo ya está en un estado avanzado se pone *on-line* para que pueda ser consultado por un grupo de gente (cliente y desarrolladores) que evaluará o propondrá cambios y comenzarán las pruebas definitivas de tiempos de respuesta y de descarga.

Además de los cambios inevitables que surgirán cuando el cliente vea como funcionan las páginas hay que ir realizando pruebas para verificar que los tiempos de respuesta, retorno y de descarga son adecuados y controlar que el sistema soporta la cantidad de conexiones simultáneas que pueden haber.

Para realizar estas pruebas no hacen falta herramientas muy sofisticadas, con el ab (del *ABuse Benchmarker*, que viene con la distribución del Apache) podemos realizar la mayoría de las pruebas y obtener datos con una precisión suficiente (para más informacion, ver optimización del A[5]pache[5] y del PHP[6]).

Pruebas de tiempos de respuesta y retorno

En esta prueba nos interesa saber cuanto tiempo tarda el servidor en generar la salida HTML. Podemos tener una aproximación bastante buena si medimos el tiempo total que tardan en bajarse las páginas en una red local (como el ancho de banda es elevado, el tiempo de descarga es prácticamente igual al tiempo de retorno).

El siguiente ejemplo lo haremos sobre el servidor de Bulma[7] y obtendremos el tiempo de retorno para 10 (-n 10) de conexiones en serie (-c 1). Es importante que las pruebas se hagan desde un ordenador distinto al servidor para obtener datos mas fiables.

```
$ /usr/sbin/ab  -c 1 -n 10 http://bulma.net/
...
Document Path:          /
Document Length:        28073 bytes

Concurrency Level:      1
Time taken for tests:   0.433 seconds
Complete requests:      10
Failed requests:        0
Total transferred:      284050 bytes
HTML transferred:       280730 bytes
Requests per second:    23.09
Transfer rate:          656.00 kb/s received
```

```
Connnection Times (ms)
              min   avg   max
Connect:        0     0     0
Processing:    42    42    43
Total:         42    42    43
```

Los resultados nos indican que el tamaño del HTML es de 28.073 bytes, y que el tiempo total para las 10 conexiones ha sido de 0.089 segundos, por lo que el tiempo de descarga de cada conexión es de 0.0433 segundos (si hacéis 1/0.0433 veréis que el valor es muy cercano a 23.09 indicado por el valor *Requests per second*).

Hay otro dato importante, *Transfer rate* nos indica el tráfico que ha sido capaz de generar el servidor en Kilobytes por segundo. Si llevamos eso a kilobits por segundo (multiplicar el valor por 10 para obtener una aproximación debido a los *overhead* del TCP/IP), veremos que el servidor (un Linux PIII, 500 Mhz, con Apache, PHP y MySQL) **es capaz de saturar una línea de ¡6 mbps!**. En este caso no habrá problemas de saturación de CPU a menos que tengamos una línea de Internet con una capacidad superior.

El segundo parámetro a investigar es la cantidad de conexiones simultáneas que puede soportar. La estimación de conexiones simultáneas es compleja e incierta, pero una buena aproximación sería estimar las conexiones promedio por día y multiplicar ese valor por 3 a 5 para obtener los picos que veríamos en el servidor.

Si calculamos que se servirán unas 200.000 páginas diarias (en España es un servidor bastante importante), tendremos que por segundo puede haber unas 2.3 conexiones (200.000/86.400 segundos día). Si multiplicamos ese valor por 5 nos queda unas 11.5 conexiones por segundo de pico.

Con las pruebas hechas anteriormente ya sabemos que el sistema soporta el doble (23.09), pero ahora haremos una mejor aproximación con mayor cantidad de conexiones simultáneas. Lo más sencillo para calcular la cantidad de conexiones simultáneas es usar el mismo valor de conexiones por segundo.

En nuestro caso redondeamos a 12 la cantidad de conexiones simultáneas y subimos la cantidad de pruebas a 100. Así la prueba será:

```
$ /usr/sbin/ab  -c 12 -n 100 http://bulma.net/
...
Concurrency Level:      12
Time taken for tests:   4.450 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      2886876 bytes
HTML transferred:       2852680 bytes
Requests per second:    22.47
Transfer rate:          648.74 kb/s received

Connnection Times (ms)
              min   avg   max
Connect:        0     0     6
Processing:    41   501  1973
Total:         41   501  1979
```

En este caso observamos que el sistema soporta perfectamente esas conexiones simultáneas sin casi degradación en los tiempos de retorno (0.0445 segundos).

Tiempos de descarga con conexiones diversas

La siguiente tabla los resultados de la ejecución de ab -c 1 -10 http://bulma.net/.

Tipo de conexión	Tiempo de descarga (seg)	Solicitudes por segundo	Transferencia de datos (KB/seg)
LAN (100 mbps)	0.04	23.00	656.00
Internet WAN (2mbps)	0.36	2.82	79.88
Internet WAN (2 mbps)	0.39	2.57	72.74
Internet WAN (4 mbps)	0.49	2.07	58.65
ADSL compartida NAT (2 mbps)	0.57	1.74	49.33

ADSL (2 mbps)	0.62	1.60	45.33
ADSL (512 kbps)	0.85	1.17	33.18
ADSL (2 mbps)	0.90	1.11	31.56
Internet WAN (256 kbps)	0.91	1.10	31.55
ADSL (256 kbps)	1.29	0.78	21.95
ADSL (256 kbps)	1.43	0.71	20.16
RDSI (64 kbps, compresión PPP)	3.05	0.32	8.96
RDSI (64 kbps)	3.99	0.25	7.18
Modem (56 Kbps)	4.01	0.25	7.16
Modem (56 Kbps)	6.01	0.17	4.70
RDSI compartida NAT (64 kbps)	11.43	0.10	2.48

Tabla de tiempos de descarga para Bulma[7]

Puede observarse que con la conexión más lenta, el tiempo total de descarga es de 11.43 segundos. Sin embargo, si analizáis el HTML generado por bulma.net[7], veréis que una vez leídos los primeros 1498 caracteres (cabeceras y METAs incluidos), el navegador ya puede mostrar el logo superior, por lo que el tiempo de de pre-visualización para el caso más lento queda en:

$$T_{vis} = (1498 \text{ Bytes}/1024) / 2.48 \text{ KBs} =\sim \mathbf{0.6 \text{ segundos}}$$

Hemos bajado el tiempo de respuesta "percibido" por el usuario **de 11.43 a 0.6 segundos! sólo cambiando la estructura de las tablas** (principalmente evitando una sólo tabla que engloba a todo el contenido de la página).

Si tomamos en cuenta el caso mayoritario hogareño (modem de 56kbps), los tiempos serán:

$$T_{vis} = (1498 \text{ Bytes}/1024) / 4.70 \text{ KBs} =\sim \mathbf{0.31 \text{ segundos}}$$

En este caso se logra una mejora de 6 a 0.31 segundos.

En ambos casos los tiempos son muy buenos, pero una página web, sobre todo si se trata de páginas genéricas, no debería superar los 2 segundos.

> **No confiar en la velocidad de la red local**: la moraleja más importante de la tabla anterior, es que aunque la estructura de las páginas esté muy mal definida, el tiempo de visualización será ínfimo comparado con los mejores tiempos que se puedan obtener a través de Internet. En el caso de pruebas de red local, la página estará descargada en menos de la mitad de una décima de segundo (0.04 seg), que está muy lejos de los 6 segundos del caso "normal". No olvidar que hay que hacer pruebas y estimaciones no sólo en la red local donde se hace el desarrollo, sino con también con una navegador estándar y conexiones con modem. Las diferencias son enormes y pueden generar sorpresas desagradables.

Cálculo del ancho de banda necesario

Hasta ahora hemos hablado de temas de tiempos de respuesta y mejoras en el diseño de las tablas, pero queda pendiente un tema importante y a menudo olvidado, tanto por los clientes como por los desarrolladores, el ancho de banda necesario para un sitio web.

La forma más común (y empírica) para calcular los requerimientos máximos es tomar el promedio de tráfico diario y multiplicarlo por 3 (como mínimo) para obtener el tráfico máximo aproximado. En el ejemplo mencionamos 200.000 páginas por día (2.3 por segundo). Si el promedio de cada página es de unos 28.000 bytes (suelen ser un poco más debido a las imágenes que se descargan una vez), entonces el tráfico promedio y máximo serán:

$$Traf_{promedio} = 2.3 \text{ páginas/seg} * 28.000 * 10 = 644 \text{ kbps}$$

$$Traf_{maximo} = 2.3 \text{ páginas/seg} * 28.000 * 3 * 10 = 1.932 \text{ kbps} =\sim \mathbf{2 \text{ Mbps}}$$

Esto nos da una idea del tipo de línea a Internet (o ancho de banda) que tendremos que contratar para atender adecuadamente los picos de tráfico. Esto significa, en España, entre 600.000 y 1.500.000 de Ptas. mensuales (3.600 y

9.000 respectivamente). El coste del ancho de banda no es nada despreciable en España y Europa en general. Y mucha gente se olvida de hacer estas estimaciones a priori, y luego se encuentran que la calidad que ofrecen es muy mala por falta de previsión.

4. Producción

El sitio ya entra en producción y su acceso es público. Aunque esta parece la más sencilla de las fases, es la más cara y difícil de conseguir. Hay que actualizar los datos para que el sitio sea interesante y visitado y además asegurase que funcione las 24 horas del día, los 7 días de la semana (24x7).

Si habéis usado Linux, Apache, MySQL/Postgres y PHP/Perl oPython (LAMP[8]), no tendréis ningún problema en sobrevivir muy tranquilamente a esta fase. Y quedarás como un gurú super-profesional ;-)

Ricardo Galli
_gallir@uib.es[9]

Lista de enlaces de este artículo:

1. http://www.extremeprogramming.org
2. http://www.ultimahora.es/
3. http://apc.communityconnect.com
4. http://bulma.net/body.phtml?nIdNoticia=664
5. http://httpd.apache.org/docs/misc/perf-tuning.html
6. http://php.weblogs.com/tuning_apache_unix
7. http://bulma.net/
8. http://www.onlamp.com
9. mailto:gallir@uib.es

E-mail del autor: gallir _ARROBA_ uib.es
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=734