



conference
Barcelona 2008
23.02.2008

Optmización de aplicaciones PHP (server side)

por Oriol Jiménez

¿Qué es?

- Optimizar es el proceso de modificar un sistema para que este sea más eficaz.
- En aplicaciones web en concreto perseguimos el objetivo de:
 - Disminuir el consumo de recursos.
 - Aumentar la velocidad de ejecución.

"La Web no va!"

- Normalmente nadie se preocupa por el rendimiento hasta que se convierte en un problema, como por la seguridad.
- PHP es un lenguaje perfecto para montar startups 2.0 en pocos minutos... pero ¿qué pasa si estas startups acaban funcionando?
- Desarrollar sistemas eficientes debería ser una premisa ante un desarrollo no una solución a un problema.

El lado del servidor

- Software servidor
- Sistemas de caché
- Optmizaciones de código (*)

Software servidor
“Lo más simple posible”

Configuración Apache I/II

- Desactivar HostnameLookups y usar siempre IPs en Allow y Deny.
- Desactivar .htaccess con "AllowOverride None".
- Options FollowSymLinks.
- Usar un DirectoryIndex lo más corto posible sin wildcards.
- Desactivar ExtendedStatus (mod_status).

Configuración Apache II/II

- ¿Keepalive or not Keepalive? en todo caso con un timeout bajo (5 segundos).
- Desactivar los logs (`mod_log_config`) si no los usamos, que se encargue el cliente (Google Analytics).
- Desactivar todos los módulos que no usamos.

Configuración PHP I/II

- Relacionadas con los recursos del sistema:
 - `max_execution_time`
 - `max_input_time`
 - `memory_limit`
 - `output_buffering`

Configuración PHP II/II

- Para aumentar el rendimiento desactivar:
 - `register_globals`
 - `magic_quotes_*`
 - `expose_php`
 - `register_argc_argv`
 - `always_populate_raw_post_data`
- Desactivar el log de errores en producción.

Contenido estático I/II

- La combinación Apache+mod_php es una de las mejores y más rápidas para servir contenido dinámico.
- Para servir contenido estático existen otras opciones mucho más ligeras:
 - Lighthtpd, Boa, Tux, Thttpd, LiteSpeed, ...

Contenido estático II/II

- De todos ellos Tux y Lighttpd son ahora mismo los ganadores en velocidad.
- Lighttpd es la mejor opción por posibilidades de configuración, portabilidad, etc. (incluso para ejecutar PHP vía FastCGI).
- Aunque no se tengan varios servidores es una opción igualmente recomendable.

Separar el contenido I/II

- Usando directorios (/images, /css, etc.):
 - Es lo más común y también lo más costoso de separar.
 - Podemos hacerlo con mod_rewrite:

```
RewriteRule ^/(.*\.(png|gif|jpg))
```

```
http://static/$1 [NC,P,L]
```

```
RewriteRule ^/(.*)
```

```
http://dynamic/$1 [P,L]
```

Separar contenido II/II

- Usando subdominios:
 - Lo más fácil y eficaz.
 - Al subdominio se le asigna la IP del servidor de contenido estático.
 - Se puede usar un mismo servidor con varias tarjetas de red o usando un firewall.

Comprimir la salida

- Ventajas:
 - Reduce el tráfico de red (ancho de banda).
 - Reduce el tiempo de carga de la página para los usuarios. La página se reduce hasta 8 veces.
- Desventajas:
 - Aumenta el consumo de CPU dependiendo del nivel de compresión. El nivel 1 ofrece 4X y un consumo aceptable.

¿Cómo comprimir?

- Usando el módulo `mod_deflate` del Apache:

```
AddOutputFilterByType DEFLATE text/  
html text/plain text/xml
```

- Usando PHP:
 - Vía `PHP.ini`:

```
zlib.output_compression=1
```

- Cambiando el handler del output buffer:

```
ob_start('ob_gzhandler');
```

Cambiar el sesión handler

- El manejador de sesiones de PHP por defecto trabaja con disco. Alternativas más rápidas:
 - RAM disk del SO.
 - Memoria compartida:
 - mm
 - apc
 - memcache

Caché

*"¿Porqué hacer lo mismo
más de una vez?"*

Concepto de caché

- Una caché son datos duplicados de otros originales donde los datos originales son costosos de acceder en tiempo y/o recursos.
- Las estrellas invitadas:
 - APC
 - Memcache
 - Squid

Usos de la caché

- En el desarrollo en PHP normalmente nos interesará guardar en la caché:
 - Una página completa para no volver a generarla.
 - Trozos estáticos de webs semi-dinámicas.
 - Resultados de consultas a BBDD u otros datos.
 - El código ya compilado para no volver a procesarlo. (*)

¿Dónde guardar la caché?

- Bases de datos (BBDD)
- Disco duro (HD)
- Memoria compartida (SHM)

Memoria compartida

- Memoria accesible por varios procesos simultáneamente con el objetivo de comunicarse.
- En nuestro caso permite que varios procesos del Apache accedan a los mismos datos.

Configurar la memoria

- Para usar correctamente la memoria compartida necesitamos ajustar el SO:
- FreeBSD:

```
# sysctl kern.ipc.shmmax  
# sysctl kern.ipc.shmall
```

- Linux:

```
# cat /proc/sys/kernel/shmmax  
# cat /proc/sys/kernel/shmall
```

APC

- Caché de opcode para PHP: No tener que procesar el código PHP en cada petición:
- Ahorra accesos a disco al leer de memoria el código "compilado".
- Ahorra la memoria usada en el procesado del código.
- Reduce hasta un 400% el tiempo de respuesta.
- Ofrece una API para trabajar con SHM.

Configuración del APC I/II

- Activación de la extensión:

```
extension=apc.so
```

- Directivas destacables del APC:

```
apc.enabled=1
```

```
apc.shm_segments=1
```

```
apc.shm_size=128
```

Configuración del APC II/II

- Para controlar que archivos guardamos compilados:

```
apc.cache_by_default=On
```

```
apc.filters=""
```

```
apc.max_file_size=1M
```

```
apc.stat=1
```

Usando la API de APC

- Guardar un dato:

```
apc_store( 'key' , serialize( $obj ) ,  
3600 );
```

- Recuperar un dato:

```
$result = unserialize( apc_fetch  
( 'key' ) );
```

- Borrar un dato:

```
apc_delete( 'key' );
```

Administrar APC I/II

- Tenemos algunas funciones para consultar información del APC:
 - `apc_cache_info()`
 - `apc_sma_info()`
- También dispone de una fantástica administración vía web

Administrar APC III/II

APC Admin Log

Refresh Data | **View Real Stats** | System Cache Entries | Per-Directory Entries | User Cache Entries | Version Check | Clear specific Cache

General Cache Information

APC Version	3.0.14
PHP Version	5.2.1
APC Host	olmenec.almpdo.com
Server Software	Apache/2.2.58 (FreeBSD) mod_php/2.0.50 OpenSSL/0.9.7a-pl FreeSSL
Shared Memory	1 Segment(s) with 512.0 MBytes (usage unknown, file backing)
Start Time	20090502 10:43:40
Uptime	1 hour and 15 minutes
File Upload Support	1

File Cache Information

Cached Files	1724 (50.8 MBytes)
Hits	9270
Misses	1832
Request Rate (hits, misses)	2.59 cache requests/second
Hit Rate	2.16 cache requests/second
Max Rate	0.43 cache requests/second
Insert Rate	1.19 cache requests/second
Cache Hit count	0

User Cache Information

Cached Variables	1 (127.8 Bytes)
Hits	5
Misses	0
Request Rate (hits, misses)	0.00 cache requests/second
Hit Rate	0.00 cache requests/second
Max Rate	0.00 cache requests/second
Insert Rate	0.00 cache requests/second
Cache Hit count	0

Runtime Settings

apc.cache_by_default	1
apc.enabled	0
apc.enabled	1
apc.file_upload_protection	2

Real Status Diagram

Memory Usage
 (Multiple slices include fragments)

Free: 442.9 MBytes (86.9%)
 Used: 61.3 MBytes (13.1%)

Hits & Misses

Hits: 9270 (86.9%)
 Misses: 1832 (13.1%)

Detailed Memory Usage and Fragmentation

Fragmentation: 0.85% (2.8 MBytes out of 442.9 MBytes in 7 fragments)

Memcached

- Es un sistema de caché de objetos en memoria distribuida.
- Pensado originalmente para reducir la carga de BBDD de LiveJournal.
- La cache es accesible desde distintos servidores.
- Sigue una estructura LRU con el añadido de tiempos de expiración.
- (Opcional) Tugela: lo mismo pero contra disco.

Instalación de Memcached

- Iniciamos el servidor:

```
# memcached -d -m 512 -l 127.0.0.1  
-p 11211 -u www
```

- Cargamos la extensión PECL:

```
extension=memcache.so
```

Usando memcached I/II

- Conectarse al servidor de caché:

```
memcache = new Memcache;  
  
$memcache->connect('localhost', 11211) or die  
("No se pudo conectar");
```

- Guardar un objeto en la caché:

```
$memcache->set('key', $obj, false, 10) or die  
("Fallo al guardar");
```

- Recuperar el objeto:

```
$result = $memcache->get('key');
```

Usando memcached II/II

- Las keys como máximo son de 250 caracteres y un dato guardado no puede ocupar más de 1 MB.
- Para distribuir la caché entre distintos servidores se ha de implementar un sistema propio.

Caché de usuario

- Ventajas:
 - Reduce el tráfico de red. No se envía nada al cliente
 - No consume recursos de servidor de ningún tipo
 - Reduce la latencia. El navegador responde directamente
- La desventaja es que no la controlamos.

Headers de caché

- Fijar el tiempo de caché deseado:
 - HTTP 1.0

```
header( 'Expires: ' . date() );
```

- HTTP 1.1

```
header( 'Cache-Control: max-age=3600, must-revalidate' );
```

Caché web vía headers

- Utilizar URLs únicas. Una URL siempre sirve el mismo contenido.
- Utilizar librerías comunes de imágenes, css, js, etc.
- Minimizar el uso de SSL.
- Hacer fácilmente variables los enlaces a elementos:

```
<script src="http://phpbsd.net/js/  
func.js?v=123"></script>
```

Acelerador web

- Se instalan delante de los servidores web.
- Cachean las peticiones de los usuarios siguiendo las headers HTTP.
- Es una de las caches más cómodas para un programador pero también muy peligrosa.
- Tenemos varias opciones: Squid, Apache, Lighttpd, Varnish.
- Squid es el más anciano y también el más usado (flickr, wikipedia, etc.).

Caché con handler de 404

- Se trata de generar archivos .html bajo demanda:
- En el HTML siempre enlazamos archivos con la extensión .html.
- El servidor web ha de ejecutar un script en el caso de no encontrar un archivo (404).
- Ese script genera el contenido solicitado.
- Las siguientes peticiones accederán directamente al .html.

Usar la caché con 404 I/II

- Configuramos el Apache:

```
ErrorDocument 404 genera_pagina.php
```

- Mantenimiento de la caché:

```
# find /web -mtime +1h -print0 | xargs  
-0 rm -f
```

Usar la caché con 404 II/II

- Añadir el código necesario:

```
$pagina = array_pop(explode('/', $_SERVER  
[ 'REQUEST_URI' ] ));  
  
ob_start();  
  
echo 'hola'; $contenido = ob_get_contents();  
  
file_put_contents($pagina, $contenido);  
  
ob_end_flush();
```

Otras opciones de caché

- Pre-generación de páginas.
- Directamente desde PHP guardar a disco, bd, o shm (shmop):
 - Mediante el output buffer páginas completas o trozos de ellas.
 - Variables serializadas con cualquier dato.
- PEAR Cache_Lite: serializando a disco cómodamente.

Optimizaciones de código

continuará...

Gracias!!

- Presentada por Oriol Jiménez.
- Todo el material en <http://phpbarcelona.org>
- Patrocinadores:

