

Acelere su servidor web con el sistema de caché distribuida memcached

CACHÉ RÁPIDA



Esta práctica herramienta puede reducir la carga de un servidor de base de datos hasta un 90%.

POR TIM SCHÜRMANN

Brad Fitzpatrick estaba frustrado: Aunque la plataforma de blogs LiveJournal.com que había fundado – y en la que había desarrollado la mayor parte del trabajo – estaba funcionando en un sistema con más de 70 máquinas potentes, su rendimiento dejaba mucho que desear. Ni siquiera parecía ayudar el tamaño de 8GB de la caché del servidor de la base de datos. Había que hacer algo y había que hacerlo rápidamente. Las medidas típicas de escenarios como estos se basan en páginas de caché servidas previamente. Por supuesto, estos remedios requieren almacenamiento redundante de cualquier elemento que aparezca en múltiples páginas – una forma de llenar la caché con basura. Si el sistema se queda sin RAM, se realizará el intercambio de memoria a disco, desde luego, pero de nuevo esto sería muy lento.

Según Fitzpatrick, la solución tenía que ser una nueva clase de sistema de caché – una que almacenara los objetos individuales en páginas separadas, evitando así los lentos accesos al disco. Pronto se cansó de buscar una solución adecuada y diseñó su propio sistema de caché. Los servidores que quería utilizar para esta tarea dispo-

nían de suficiente RAM libre. Al mismo tiempo, todas las máquinas tendrían que acceder a la caché de forma simultánea y el contenido modificado tendría que estar disponible para cualquier usuario sin retraso. Estas consideraciones finalmente le llevaron a memcached, que reduce la carga de la base de datos de LiveJournal a la increíble cantidad de un 90%, mientras que se acelera la entrega de las páginas a los usuarios y se mejora la utilización de los recursos de las máquinas individuales.

Memcached [1] es un sistema de caché distribuido de alto rendimiento. Aunque está diseñado para ser independiente de la aplicación en su mayor parte, se usa normalmente para hacer de caché en los accesos a bases de datos de aplicaciones web dinámicas.

Actualmente, pesos pesados como Slashdot, Fotolog.com y, por supuesto, su creador LiveJournal.com se basan en memcached para acelerar el rendimiento web. Desde el desarrollo inicial, LiveJournal.com ha sido adquirida y vendida varias veces y memcached, que se encuentra disponible bajo la licencia de código abierto BSD, es ahora responsabilidad de Danga Interactive.

Ropas Nuevas

Instalar una caché distribuida con memcached es una tarea sencilla. Hay que ejecutar el servicio memcached en cada servidor con RAM disponible para compartir como caché. Si fuera necesario, se pueden activar múltiples áreas de caché en una máquina. Esta opción es particularmente útil con sistemas operativos que sólo dan a los proce-

Listado 1: Consulta de Caché Simple en PHP

```
01 <?php
02 $memcache = new Memcache;
03
04 $memcache->connect('localhost', 11211) or die ('No
05 connection to memcached
06 server');
07
08 $memcache->set('key',
09 'datum', false, 10);
10
11 $result =
12 $memcache->get('key');
13
14 var_dump($result);
15 >>
```

esos accesos a parte del total de la RAM disponible. En este caso hay que iniciar múltiples servicios, de manera que el sistema operativo les asigne tanta memoria como pueda, usando de este modo la máxima cantidad posible de memoria libre para la caché.

Una librería cliente especial proporciona la interfaz con la aplicación del servidor. Acepta los datos a almacenar y los almacena en uno de los servidores existentes utilizando una palabra reservada escogida libremente (Figura 1). La librería cliente aplica un sofisticado método matemático para escoger uno de los servidores memcached para gestionar los datos y pedirle que la deje en su RAM.

Se podría comparar este procedimiento con el guardarropas de un teatro: Se le da el abrigo al empleado tras el mostrador y a cambio le da un número. El empleado toma el abrigo, localiza el lugar adecuado y lo cuelga en la percha correspondiente al número entregado. Al final de la actuación, se repite el mismo proceso pero a la inversa: se le dice al empleado – es decir, a la librería cliente – el número de nuevo y el cliente acude al servidor correspondiente, toma los datos de la “percha” y los entrega a la aplicación.

El modelo completo es muy similar al de una base de datos distribuida o al de un sistema de ficheros distribuido. Pero cuando se está trabajando con memcached, siempre debería recordar que es simplemente una caché. Dicho de otro modo, el empleado del guardarropas no es de confianza y tiene problemas a la hora de recordar las cosas. Si el espacio es insuficiente para los nuevos elementos, uno de los servidores volcará los datos menos solicitados para liberar espacio. Algo similar ocurre si uno de los servidores de memcached fallara – en este caso, cualquier información importante que tuviera almacenada en ese momento desaparecería. Dicho de otro modo, no podría recuperar su abrigo al final de la actuación y la aplicación estaría obligada a comunicarse de nuevo con la base de datos. El sistema memcached no tiene redundancia, pero tampoco hay necesidad para ello: Después de todo, es simplemente una caché, y su trabajo consiste en almacenar información temporalmente y distribuirla tan rápidamente como pueda. En línea con esta filosofía, sería imposible iterar de nuevo por todos los elementos de la caché o volcar el contenido completo de la caché al disco.

Oyente

Danga Interactive tiene el servidor memcached en su página web para que los usuarios puedan descargarlo [1]. Las únicas dependencias que posee son la librería *libevent* y el paquete de desarrollo correspondiente. El servidor se compila fácilmente con los tres pasos normales:

```
configure
make
sudo make install
```

Algunas de las distribuciones principales poseen paquetes memcached precompilados en sus repositorios, pero normalmente están obsoletos. Tras finalizar la instalación, el siguiente comando – o similar – ejecuta el servidor:

```
memcached -d -m 2048 -l 192.168.1.111 -p 11211 -u USERNAME
```

Este comando ejecuta memcached en modo servidor (-d), indicándole que use 2048MB de RAM para la caché distribuida en esta máquina (-m 2048). El servidor escucha las peticiones de los clientes en el puerto 11211 en la dirección IP 192.168.1.111. El servidor también tiene que saber la cuenta que debe utilizar, aunque se puede omitir el parámetro -u para ejecutarlo con la cuenta del usuario conectado.

Los expertos en seguridad probablemente estén echando espuma por la boca: Por defecto, cualquier usuario de un sistema Linux puede ejecutar su propio servidor memcached. Para impedirlo, hay que seguir unos pasos, tales como retirarle privilegios de acceso – tan sólo es uno de los múltiples problemas de seguridad que memcached deliberadamente evita (más adelante se muestran más).

Escogiendo Compañeros

Tras preparar todos los servidores, hay que escoger una de las muchas librerías cliente que se encuentran ahora disponibles para diversos lenguajes de programación y de

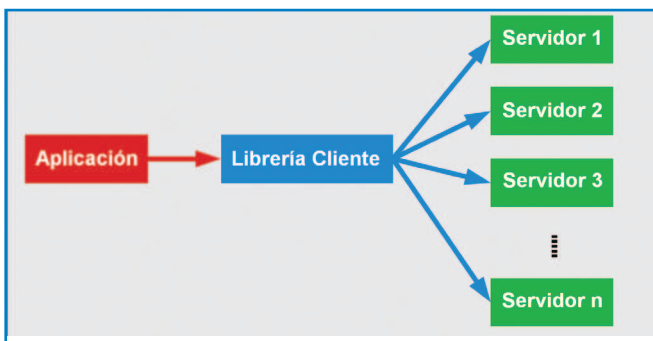


Figura 1: La librería cliente acepta los datos desde las aplicaciones y selecciona un servicio, que a continuación se encarga de almacenar los datos.

script: En algunos casos se tiene incluso una elección de paquetes [2]. Si prefiere crear su propio cliente, puede encontrar una descripción detallada del protocolo subyacente en la wiki de memcached en el sitio Google Code [3].

Como memcached se usa a menudo para acelerar aplicaciones web, la mayoría de la gente opta por el cliente PHP. Para más información sobre cómo usar memcached con C y C++, véase el cuadro titulado “libmemcached”.

La solución básica es la misma para todos los lenguajes: Tras localizar e instalar correctamente la librería cliente, los desarrolladores tienen que incluirla en sus programas. La siguiente línea crea un objeto *Memcached* nuevo usando PHP y el cliente *memcache* desde el repositorio PECL, tal como lo incluye el paquete de Ubuntu *PHP5-memcache*:

```
$memcached= new Memcached;
```

libmemcached

La librería cliente memcached más popular para aplicaciones C y C++ es ahora *libmemcached* [4] – que no debería confundirse con su predecesora *libmemcache* (sin la “d” final). Incluso aunque no seamos programadores en C o C++, es buena idea echarle un vistazo al paquete; contiene un par de herramientas de diagnóstico (servidor) de la línea de comandos interesantes. Por ejemplo, *menstat* recupera los datos para una clave desde la caché y presenta los resultados en la consola, *menstat* consulta el estado actual de un servidor o de varios. Para compilar *libmemcached* necesitamos en nuestro sistema un compilador tanto de C como de C++; aparte de esto, es lo de siempre: `./configure; make; make install`. En el Listado 2 se muestra una consulta simple de caché.

Siguiendo esto, una llamada de función le dice a la librería en qué servidores se encuentran los servicios memcached escuchando:

```
$memcache->connect(
('192.168.2.1', 11211) or die
('No connection to
memcached server');
```

A partir de ahora se pueden usar más llamadas a funciones para rellenar la caché con nuestro propio contenido:

```
$memcache->
set('key', 'test', false, 10);
```

Esta instrucción escribe la cadena *test* en la caché, con *key* como clave, y mantiene la entrada durante 10 segundos. La longitud de las claves está actualmente restringida a 250 caracteres – esta restricción está impuesta por el servidor memcached.

Para obtener los datos se debe pasar la clave a la librería cliente y aceptar los resultados devueltos:

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

El Listado 1 muestra el script PHP completo.

La opción de escribir múltiples conjuntos de datos en la caché mientras se están obteniendo es interesante. La librería

cliente paraleliza automáticamente las peticiones a los servidores memcached. Desafortunadamente, algunas librerías cliente no proporcionan estas funciones; este ejemplo en PHP:

```
$multiple = array(
'key1' => 'wert1',
'key2' => 'wert2',
'key3' => 'wert3'
);
$memcache->setMulti($multiple);
```

sólo es soportado por el cliente memcached (terminado en “d”).

Tiempo de Perfiles

Para las aplicaciones web existentes, la cuestión siempre es dónde se puede desplegar y utilizar mejor memcached. Los perfiles son la solución: Las consultas de la base de datos que realmente estresan al servidor serán mejor que se dirijan por medio de la caché. Los Listados 3 y 4 muestran el aspecto que realmente tiene en la vida real: Antes de consultar la base de datos, el código comprueba si la información solicitada se encuentra disponible en memcached. Una consulta sólo se realiza si la información no se encuentra.

Para evitar la necesidad de otra consulta, los resultados se almacenan en la caché. Para mantener, a la caché actualizada, la información de cada operación de escritura también se almacena en la caché. En el Lis-

tado 4, las claves se construyen combinando la palabra *user* con la ID de la cuenta de usuario – esta es una estrategia común para generar claves únicas.

Esta solución facilita la integración de memcached con sus propias aplicaciones, pero tiene que estar al tanto de los fallos, que no se hacen aparentes hasta que no se mira debajo de la capa.

Diccionario

Los programadores con experiencia probablemente ya se hayan dado cuenta de que memcached utiliza internamente un diccionario; algunos lenguajes de programación lo llaman array asociativo. Como en un diccionario real, esta estructura de datos almacena cada valor bajo una clave (palabra) específica. El sistema memcached implementa su diccionario con dos tablas hash [5]. Primero, la librería cliente acepta la clave y ejecuta una sofisticada función matemática, y a partir de ella, crea un número o hash. El número le dice a la librería con qué servidor memcached tiene que comunicarse. Tras recibir los datos, el servidor usa su propia función hash para asignar una zona de memoria para almacenar los datos. La función matemática está diseñada para devolver exactamente el mismo número para una clave específica. Esta solución garantiza las búsquedas y los tiempos de respuesta extremadamente cortos. Para obtener la información de caché, el sistema memcached simplemente tiene

Listado 2: Una Consulta Simple de Caché

```
01 #include <memcached.h>
02 #include <string.h>
03 #include <stdio.h>
04
05 main()
06 {
07 /* crear una estructura memcached_st (conteniendo
    toda la información básica para los servidores
    memcached) */
08 memcached_st *mcd = memcached_create(NULL);
09
10 /* Añadir un servidor: */
11 memcached_server_add(mcd, "127.0.0.1", 11211);
12
13 /* Colocar el objeto en cache: */
14
15 char *key = "key";
16 size_t keylength = strlen(key);
17 char *value = "information";
18 size_t valuelength = strlen(value);
19 time_t expiration = 0;
20 uint32_t flags = 0;
21
22 memcached_add(mcd, key, keylength, value,
    valuelength, expiration, flags);
23
24 /* Recuperar un objeto de la cache: */
25
26 memcached_return errovariable;
27
28 char *result = memcached_get(mcd, key, keylength,
    &valuelength, &flags, &errovariable);
29
30 /* Imprimir objeto: */
31 printf("Cache: %s\n", result);
32
33 /* Flush: */
34 memcached_free(mcd);
35 }
```

que evaluar dos funciones matemáticas. La transmisión de los datos por la red consume la mayor parte del tiempo de respuesta.

Como la librería cliente decide qué servidor almacena los datos, todas las máquinas implicadas tienen que poseer las mismas versiones de las mismas librerías. Una mezcla de versiones puede dar lugar a que los clientes utilicen diferentes funciones hash y así almacenar la misma información en diferentes servidores, provocando inconsistencia en los datos. Si se utiliza la librería de C y C++ *libmemcached*, hay que prestar una atención especial, ya que ofrece la posibilidad de escoger la función hash de un repertorio.

Encima de esto, cada cliente utiliza un método diferente de serialización. Por ejemplo, Java utiliza Hibernate, mientras que PHP usa *serialize*. Dicho de otro modo, si no se están almacenando cadenas de caracteres, sino objetos, en la caché, el uso compartido basado en diferentes lenguajes es imposible – incluso si todos los clientes utilizan la misma función hash. Las librerías también pueden elegir sus propios algoritmos de compresión.

Pérdidas de Memoria

La caché maneja peticiones en paralelo sin pérdidas de velocidad. En un teatro, varios empleados podrían trabajar al mismo tiempo, colgando los abrigos o devolviéndolos a sus dueños, sin que la gente tenga que esperar en cola. El mismo principio es aplicable a memcached: Cada cliente averigua con qué servidor tiene que comunicarse, y en un mundo ideal, cada empleado iría a un pasillo diferente: por supuesto, nadie puede impedir que un empleado siga a otro por el mismo pasillo. Si se obtienen datos de la caché, se manipulan y se escriben de nuevo, no hay garantía de que una segunda instancia haya modificado los datos mientras tanto. Los comandos *gets* y *cas* presentados en la versión 1.2.5 ofrecen

Listado 3: Consulta a la Base de Datos sin memcached

```
01 function get_user($userid)
02 {
03     $result = mysql_query
04     ("SELECT * FROM users WHERE
05     userid = '%s'", $userid);
06     return $result;
07 }
```

una solución: Los usuarios pueden utilizar el comando *gets* para obtener datos y recibir un identificador único, que pueden enviar de nuevo al servidor, junto con los datos modificados, con el comando *cas*. Luego, el servidor comprueba la ID para ver si los datos se han modificado desde la última consulta y los sobrescribe utilizando el nuevo valor si este fuera el caso.

La forma en que memcached maneja un fallo del servidor también depende del cliente. Por defecto, memcached simplemente actuará como si la información solicitada no estuviera, o no fuera a estar más, en la caché. Así pues, es buena idea monitorizar permanentemente los servidores de caché. Gracias al diseño modular, un servidor se puede reemplazar fácilmente. Lo único que hay que hacer es dar de baja la dirección IP previa y registrar la nueva dirección IP en los clientes. Pero téngase en cuenta que algunas librerías considerarán la caché completa como no válida en este caso.

Queso Suizo

Para impedir la fragmentación de la RAM, el servidor utiliza un asignador slab [6] para la gestión de la memoria. Este método está especializado en reservar y liberar repetidamente pequeños fragmentos de memoria. En el caso de memcached, “pequeños” significa un máximo de 1MB; el servidor no aceptará nada mayor que esta cantidad. Si se quiere almacenar más,

Listado 4: ... y Tras Introducir memcached

```
01 $memcache = new Memcache;
02
03     $memcache->connect('serverna
04     me', 11211) or die ('No
05     connection to memcached
06     server');
07 ...
08 function get_user($userid)
09 {
10     $result = memcache->
11     get("user" + $userid);
12     if(!$result)
13     {
14         $result = mysql_query
15         ("SELECT * FROM users WHERE
16         userid = '%s'", $userid);
17         memcache->add("user" +
18         $userid, $result);
19     }
20     return $result;
21 }
```

habrá que distribuir los datos en múltiples claves o usar un sistema de caché diferente.

Anarquía

Memcached no se hace cargo por sí mismo de la seguridad. Los clientes no tienen que autenticarse en los servidores. Cualquiera que tenga acceso a la red puede acceder a la caché sin problemas. Un atacante que conozca los nombres de usuarios que hay tras las claves puede preguntar sistemáticamente a todos los servidores por estos nombres. Las claves cifradas pueden ayudar a proporcionar alguna protección básica. Para generarlas, hay que aplicarle una función hash a los nombres de usuario dentro de la aplicación y luego utilizar los resultados como claves. Todos los datos de las cuentas deberían borrarse de la caché tras su uso. También es buena idea definir un tiempo de vida limitado para los datos y añadir más niveles de seguridad, comenzando con un cortafuegos para proteger la granja de servidores de los ataques externos.

Conclusiones

Memcached es fácil de configurar e integrar con las aplicaciones existentes, pero esta conveniencia tiene un precio, sus diversas vulnerabilidades. Si se las arregla para solucionar estos problemas, obtendrá una caché distribuida muy rápida – incluso en condiciones extremas. El sistema ha demostrado su valor día a día en LiveJournal.com y Slashdot. Al mismo tiempo, es extremadamente frugal. Como memcached genera principalmente valores hash, la potencia de la CPU no es fundamental, pudiéndose utilizar incluso ordenadores antiguos como proveedores de caché. ■

RECURSOS

- [1] Memcached: <http://www.danga.com/memcached>
- [2] Descripción general de librerías clientes: <http://code.google.com/p/memcached/wiki/Clients>
- [3] Dentro del protocolo: <http://code.google.com/p/memcached/wiki/MemcacheBinaryProtocol>
- [4] Libmemcached: <http://tangent.org/552/libmemcached.html>
- [5] Cómo funciona una tabla hash: http://en.wikipedia.org/wiki/Hash_table
- [6] Cómo trabaja un asignador slab: http://en.wikipedia.org/wiki/Slab_allocator

Adonde vayas...



Lee Linux Magazine desde cualquier sitio con nuestra suscripción digital.
Accede a todos los números en PDF entrando en nuestro sitio.
Encuentra lo que buscas con un sencillo formulario de búsqueda.
Mantén tu propia biblioteca de artículos y leelos desde cualquier dispositivo.

... Linux Magazine va contigo.

<http://www.linux-magazine.es/digital>