Universidad de Granada

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIONES

USO DE MEMCACHED EN SERVIDORES WEB BALANCEADOS

Número de Wiki: 21
Número de horas: 50

Armenteros Soto, David
Melguizo Marcos, Iñaki

Contents

1 Introducción

Memcached es una herramienta desarrollada por la empresa **Danga Interactive** , que mantiene el proyecto bajo licencia BSD y que fue pensada para incrementar la velocidad de aplicaciones web dinámicas, reduciendo la carga de la base de datos. Fue desarrollada inicialmente por Brad Fitzpatrick para su sitio web LiveJournal, el 22 de mayo de 2003.

Memcached es empleado para el almacenamiento de datos en caché u objetos en la memoria RAM, reduciendo así las necesidades de acceso a un origen de datos externo como puede ser una base de datos.

El funcionamiento de este sistema es muy sencillo, hace uso de una tabla hash distribuida a lo largo de varios equipos donde va almacenando los valores asociados a una clave única para cada elemento. Por medio del uso de instrucciones sencillas, se puede almacenar y recuperar información almacenada en la memoria de forma muy rápida, haciendo uso de dicha clave. Conforme la tabla descrita se va llenando, los datos que lleven más tiempo sin ser utilizados se borran para dar espacio a nuevos datos. Usualmente, las aplicaciones que utilizan esta herramienta comprueban primero si pueden acceder a los datos a través de Memcached antes de recurrir a un almacén más lento como una base de datos.

Cabe destacar que hasta ahora hemos estado hablando de que Memcached permite almacenar los resultados devueltos de una consulta a la base de datos, pero además de estos datos, se puede almacenar cualquier información que se nos ocurra, por ejemplo, resultados de cálculo, información de sesión de los usuarios...

Esta herramienta nace como respuesta a la necesidad de incrementar velocidades de respuesta para peticiones web, en sitios de tráfico masivo. En este momento, su uso continúa expandiéndose a medida que el proyecto toma mayor fuerza, con la ayuda de varios desarrolladores de la comunidad Open Source, quienes constantemente revisan y agregan nuevas capacidades.

Actualmente, su uso continúa expandiéndose, a medida que el proyecto toma mayor fuerza con la ayuda de varios desarrolladores de la comunidad Open Source, que revisan y agregan nuevas capacidades al proyecto. La siguiente lista contiene una serie de sitios que actualmente utilizan Memcached para resolver cuestiones de escalabilidad:

- **LiveJournal:** Memcached nace como un desarrollo para el backend de LiveJournal, y como tal, es el primer sitio que lo implementa.
- **Slashdot**
- **Fotolog**
- **Wikipedia**

Después de tener una cierta repercusión y de manejar en promedio, de 35 a 40 millones de 'page views' diarios, lograron una mejora en la prestación de servicios extendiéndose en:

- **HI5**
- **Facebook**
- **Youtube**
- **Reddit**
- **Twitter**

El mayor usuario de Memcached del mundo es, como no es de extrañar, Facebook [1]. En 2008 se informó de que Facebook cuenta con 800 servidores de Memcached con hasta 28 terabytes de datos en su caché y este número no ha parado de subir desde entonces. Utilizan Memcached para aliviar la carga de trabajo de la base de datos, es decir, para reducir el número de peticiones a ésta. Debido al auge de Facebook, en estos últimos años se han disparado el número de accesos a sus servidores y han tenido problemas de escalabilidad. Estos problemas les han obligado a realizar modificaciones tanto en Memcached como en su sistema operativo para proporcionar la mejor experiencia a sus usuarios.

Como tienen muchísimos ordenadores, cada uno de los cuales ejecuta más de cien procesos de Apache, se acababan teniendo cientos de miles de conexiones TCP abiertas a sus procesos de Memcached. Dichas conexiones no eran un problema, pero sí lo era la forma en que Memcached asignaba memoria para cada una de las conexiones TCP. Este problema se debe a que cuando se llega a cientos de miles de conexiones TCP, esto suma gigabytes de memoria, que es memoria que se podía haber utilizado para almacenar los datos de los usuarios. Para recuperar esta memoria para los datos de los usuarios, implementaron una reserva de búferes de conexión compartida por hilo para los sockets TCP y UDP. Con este cambio se permitieron recuperar varios gigabytes de memoria del servidor y se ha conseguido mejorar bastante la escalabilidad de Memcached, pudiendo ahora manejar muchas más peticiones UDP por segundo.

2 Antecedentes

[2] En los últimos años, debido al acceso casi mundial a las tecnologías, ha aumentado enormemente el interés por los sitios web interactivos que ofrecen redes sociales y comercio electrónico. Ambos sitios web generan una gran cantidad de datos dinámicos que normalmente se almacenan en servidores de base de datos, para su posterior recuperación y análisis. Al mismo tiempo, dado que los servicios interactivos (juegos, servicios financieros, etc.) se trasladan cada vez más a la nube y el uso de la computación en la nube sigue creciendo, los centros de datos tienen un gran reto por delante. Las búsquedas en bases de datos con lenguajes como SQL son bastante costosas en cuanto a tiempo se refieren y los centros de datos que alojan sitios web populares deben estar preparados para servir a millones de visitantes del sitio web por lo que se vio necesario un cambio en la arquitectura de los sitios web. Sin embargo, debido a la naturaleza dinámica de los datos, es prácticamente inevitable tener que acceder a los datos almacenados en las bases de datos. Y, es difícil mejorar la velocidad de respuesta de cada consulta de la base de datos manteniendo las famosas garantías ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), especialmente cuando las lecturas se mezclan con las escrituras.

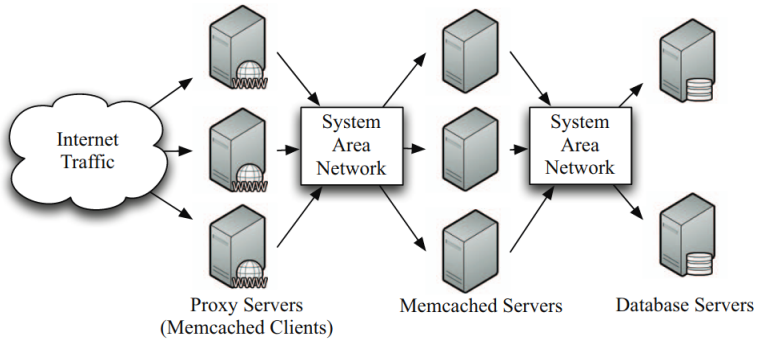


Figure 1: Despliegue de Memcached en centros de datos

Por lo tanto, la solución que se siguió es la implementación de una nueva capa de almacenamiento en memoria [Figura 1], Memcached, para almacenar en caché los resultados de las consultas anteriores a la base de datos. Así, evitamos costosas consultas a la base de datos que requerirían mucho más tiempo con las que podríamos no cumplir las espectativas temporales que tienen los usuarios sobre nuestro sitio web, lo también llamado, mejorar la Calidad de Experiencia (*Quality of Experience*, *QoE*) del usuario final.

Para que veamos la relevancia que tuvo Memcached desde un principio, ya en 2008, en una entrevista [3], uno de los cofundadores de Twitter, Jack Patrick Dorsey,

respondió esto cuando un periodista le preguntó acerca de si realmente tenían solo 3 bases de datos físicas para todo Twitter:

"Hemos mitigado gran parte de este problema utilizando Memcached, como hacen muchos sitios, para minimizar nuestra dependencia de una base de datos. Nuestra nueva arquitectura trasladará nuestra dependencia a un enfoque simple y elegante basado en un sistema de archivos, en lugar de una colección de bases de datos. No planeamos depender nunca de un número masivo de bases de datos en el futuro."

Esto nos da a entender que Memcached llegó para quedarse y era momento de dejar atrás el hecho de tener que acceder a las bases de datos de las compañías punteras para obtener cualquier recurso web. Actualmente hay abundantes proyectos para mejorar más aún el rendimiento y la escalabilidad de Memcached, por ejemplo, MemC3 [4], que presenta mejoras algorítmicas respecto a Memcached pero presenta cuellos de botella en algunos entornos. Por tanto, si a través de uno de estos proyectos de investigación se obtuvieran mejoras significativas respecto al rendimiento de Memcached, ninguna empresa exitosa dudaría en implementar dicha tecnología, al igual que no dudaron en utilizar Memcached en su día.

3 Arquitectura

El sistema usa una arquitectura **cliente-servidor** en el que los servidores mantienen un array asociativo clave-valor y los clientes añaden datos al array y acceden a él. A esta memoria se accede a través de una biblioteca de cliente que forma parte de la aplicación de usuario en el lado del cliente, siendo el puerto de escucha el 11211. Las claves pueden tener una longitud de hasta 250 bytes y los datos pueden tener un tamaño de hasta 1 MB. Cada cliente mantiene una lista de todos los servidores y estos no se comunican entre ellos. Si un cliente desea establecer o leer el valor de una clave, la librería hace cálculos mediante un algoritmo hash para determinar el servidor que va a utilizar. Entonces se pone en contacto con el servidor y este usará otro hash para determinar dónde almacenar o leer el valor correspondiente.

La estructura general de la arquitectura de Memcached es relativamente simple y es similar a un sistema de base de datos distribuida. Consta de la aplicación, una biblioteca cliente y un grupo de instancias Memcached. Se puede instalar cualquier número de estas instancias en la memoria principal de un servidor y se recomienda activar una instancia en cada servidor que tenga memoria de sobra. Estas instancias trabajan juntas para adquirir el espacio libre disponible para la caché . La biblioteca cliente es la interfaz entre la aplicación respectiva y Memcached que toma los datos para ser almacenados y los coloca en un servidor existente. A través de su arquitectura de múltiples subprocesos, Memcached puede usar varios núcleos de proceso simultáneamente.

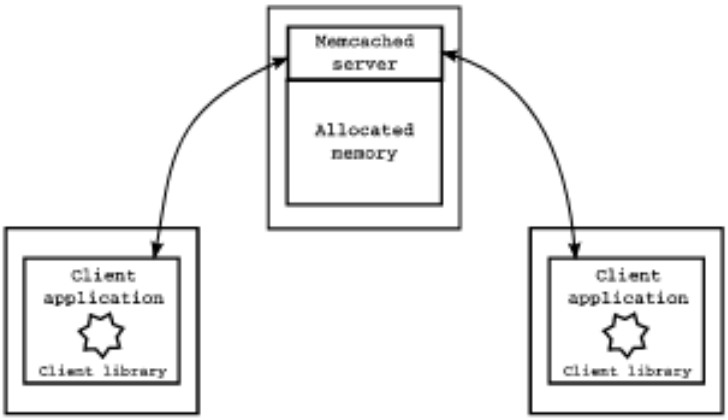


Figure 2: Interacción Clientes-Servidor Memcached

A *Servidor Memcached*

Concretamente las instancias del servidor Memcached se encargan basícamente de tres tareas:

- Gestionan la asignación y restauración de la memoria.
- Mantienen un registro de los objetos almacenados en memoria.
- Atienden las peticiones de los clientes en cuanto a la recuperación y almacenamiento de los objetos.

El demonio no reconoce los tipos de objetos, sino que guarda los paquetes de datos tal y como los recibe en la red, por tanto, es tarea del cliente realizar la serialización en el lado del emisor y la deserialización en el lado del receptor. Desde el punto de vista del cliente, cuando se quieren recuperar datos, el cliente envía una solicitud GET mediante una petición TCP con la clave única descrita anteriormente. Al utilizar esta clave, el servidor siempre sabe qué datos debe devolver. Memcached está diseñado de manera que una operación nunca se bloquea. Incluso si varias conexiones concurrentes están leyendo y escribiendo el mismo objeto, ninguna de ellas se bloqueará. Esto se consigue teniendo múltiples copias de los datos.

B *Bibliotecas de cliente*

Las bibliotecas de cliente están destinadas a proporcionar una forma común de acceso al servidor. Su principal objetivo es serializar (proceso de codificación de un objeto en un medio de almacenamiento con el fin de transmitirlo a través de una conexión en red como una serie de bytes) los objetos que se almacenarán en la caché. Los clientes utilizan el hashing consistente para seleccionar un único servidor por clave, requiriendo sólo el conocimiento del número total de servidores y sus direcciones IP. Esta técnica presenta todos los datos agregados en los servidores como una tabla hash distribuida unificada, mantiene la independencia de los servidores y facilita el escalado a medida que crece el tamaño de los datos. Varios clientes que accedan a los mismos objetos deben tener la misma lista de servidores, ya que la localización de un objeto se determina mediante el hash de la lista de servidores junto con la clave del objeto. En otras palabras, un objeto sólo puede recuperarse utilizando la misma función hash y la misma lista de argumentos que se utilizó cuando se almacenó el objeto. De este modo, un cliente puede calcular la ubicación de un objeto sin ninguna interferencia del cliente que realmente almacenó el objeto en cuestión.

El servidor mantiene los valores en RAM. Si un servidor agota su memoria, descarta los valores más antiguos. Es por esto, por lo que Memcached es una caché transitoria, no se puede asumir que los datos almacenados en Memcached estarán ahí

cuando se necesiten.

Cuando se introdujo Memcached en 2003, había pocos procesadores X86 con múltiples núcleos, y los servidores con varios sockets eran extremadamente caros. Procesadores Intel® Xeon® de dos y cuatro núcleos no se lanzaron hasta 2005 y 2009 respectivamente. Hoy en día, los sistemas de servidor de dos sockets, 8 y 16 núcleos son el componente básico de servicios web. Estos sistemas avanzados brindan una amplia oportunidad para que las cargas de trabajo se ejecuten en paralelo utilizando múltiples subprocesos o procesos, sin embargo, Memcached no puede hacerlo utilizando las estructuras de datos actuales y arquitectura de software.

3.1 Estructura de Datos

Memcached tiene 4 estructuras de datos principales que incluyen:

- Una tabla **hash** para localizar un elemento de caché.
- Lista de menos usados recientemente **(LRU)** para determinar el orden de desalojo de elementos de caché cuando esta llena.
- Elemento de caché para guardar la clave, los datos, las banderas y los punteros.
- Asignador de bloques, que es el administrador de memoria para los elementos de caché.

3.1.1 Tabla Hash

La tabla hash es una matriz de contenedores. El tamaño de la matriz es siempre una potencia de dos. Por ello, podemos encontrar el contenedor de una forma más rápida tomando el valor 2^{k-1} y usándolo como una máscara hash. Los contenedores se construyen como listas enlazadas de elementos de caché.

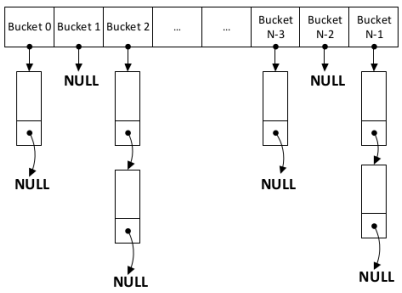


Figure 3: Estructura de datos de la tabla hash utilizada para buscar elementos de caché

3.1.2 LRU

La lista LRU se utiliza para determinar el orden de los elementos de caché, se trata de una lista doblemente enlazada que contiene todos estos elementos en una sola unidad de asignación de memoria. Los punteros de la lista se mantienen en la estructura de cada elemento de la caché y se modifican cada vez que el elemento es manipulado. En caso de desalojo, se comprueba el elemento de caché más antiguo y se retira.

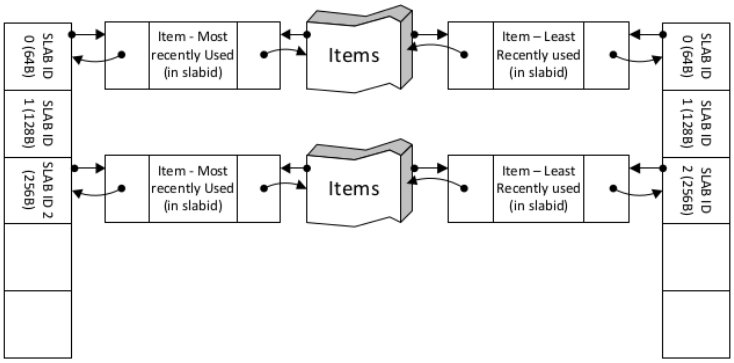


Figure 4: Estructura de datos de la LRU utilizada para determinar el orden de desalojo

3.1.3 Elementos de caché

La estructura de datos del elemento de caché contiene los datos del par **clave-valor**. Es una intersección de los siguientes datos de referencia:

- Punteros de la tabla hash para la lista vinculada.
- Punteros utilizados en la LRU para la lista de enlaces dobles.
- Contador para determinar cuántos subprocesos están accediendo actualmente a un elemento de caché.
- Banderas para el estado del elemento de la caché.
- La clave.
- Longitud del valor en bytes.
- El valor.

3.1.4 Asignador de bloques

El asignador de bloques o *slabs* permite administrar la memoria para elementos de caché. Dado que estos son relativamente pequeños, asignar y liberar estos fragmentos mediante llamadas al sistema sería poca eficiente. Por otro lado, Memcached usa bloques como unidad de asignación de memoria. En este contexto, haremos referencia a una clase de slabs como una pieza de memoria que puede contener muchos elementos de caché en su interior. Por ejemplo, imaginemos 2048 bytes de memoria disponibles para la caché distribuida. Podemos dividir esta memoria en 4 clases de slabs de 512 bytes cada una. La primera clase de slabs está destinada a los slabs de 64 bytes, la segunda a los de 128 bytes y así sucesivamente. El tamaño de los objetos almacenados nunca podrá ser mayor que el tamaño del tamaño máximo de slab en cada clase (64, 128, etc.). En este caso, un objeto de 50 bytes ocupará realmente 64 bytes, ya que 14 bytes quedarán sin utilizar. En el caso de ocupar un objeto 128 bytes, se introduciría en la segunda clase de slabs y no quedaría ningún byte sin utilizar. En cuanto se elimina el objeto, su slab se añade a la lista libre, con lo que siempre se sabe qué slabs de las distintas clases de slabs se pueden utilizar.

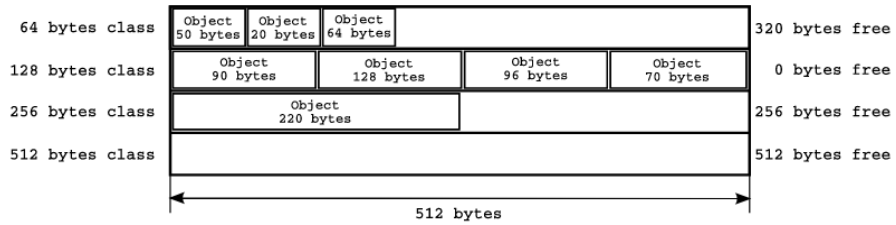


Figure 5: Asignador de slabs utilizado para gestionar la memoria dentro del demonio Memcache

Por tanto el Slab Allocator o Asignador de bloques utilizará las asignaciones de memoria más grandes y mantiene una lista de slabs libres en caché en estas distintas clases de slabs. Cada vez que se solicita un elmento de caché, el asignador de bloques verifica el tamaño del valor almacenado y devuelve un slab de una clase de slabs lo suficientemente grande. En algunos casos, esta solución desperdicia espacio, pero ahorra tiempo. Las llamadas al sistema que nos hemos ahorrado compensan esta desventaja.

4 ¿Cómo funciona Memcached?

El Servidor Memcached, más conocido como **Memcached Server**, almacena los objetos serializados por el cliente (internamente denominados **ítems**), en una gran tabla hash, y los mapea según la clave que le asociemos a dicho ítem. Dicha tabla adopta una estructura de porciones de memoria de tamaño variable, es decir, contendrá diferentes clases de slabs, cada una de ellas, con un tamaño máximo de slab, con el objetivo de optimizar la asignación del espacio de memoria sin necesidad de realizar una llamada al sistema. [5]

Un **slab** es una porción de memoria con un tamaño, que dependiendo de la clase de slab, tendrá un tamaño fijo, siendo éste, por consiguiente, el tamaño máximo que un ítem puede tener si quiere ser almacenado en Memcached Server (aunque este valor puede ser modificado desde el código fuente).

La arquitectura escalable de Memcached, nos permite mantener un **pool** (recursos inicializados) de Memcached Servers, característica que bien puede ser explotada en momentos donde la cantidad de conexiones no puede ser gestionada por un único Server. En ese caso, el pool optará por otro Server, balanceando la carga de conexiones.

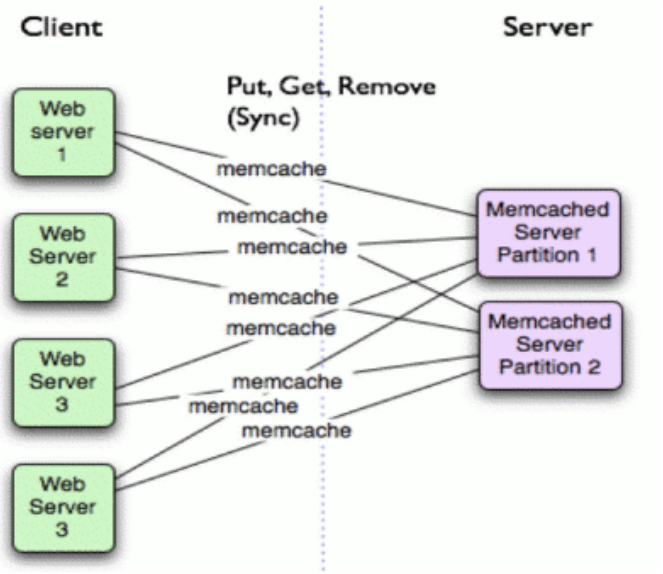


Figure 6: Funcionamiento de la arquitectura de Memcached

La comunicación entre clientes y servidor es muy simple y está basada en comandos. La interfaz de Memcached proporciona todas las primitivas básicas que ofrecen las tablas hash -inserción, borrado y búsqueda/recuperación- así como operaciones más complejas construidas sobre ellas [6]:

- **STORE:** Actualiza el objeto (*clave*, *valor*) si existía anteriormente, o lo agrega en caso contrario.
- **ADD:** Agrega el objeto (*clave*, *valor*) solo si no existe.
- **REPLACE:** Actualiza el objeto (*clave*, *valor1*) por (*clave*, *valor2*).
- **DELETE:** Borra el objeto que le indicamos por medio de una clave .
- **GET:** Recupera el objeto que coincida con la clave indicada.

Las cuatro primeras operaciones son destructivas y tienen el mismo código que la operación STORE. Por tanto, son operaciones de escritura que son siempre transmitidas a través del protocolo TCP para garantizar los reintentos en caso de error de comunicación. Las solicitudes de STORE (las 4 vistas anteriormente) que superan la capacidad de memoria del servidor incurren en un desalojo de la caché basado en el algoritmo *Last Recent Used, LRU.* El proceso exitoso de una operación STORE sería el siguiente:

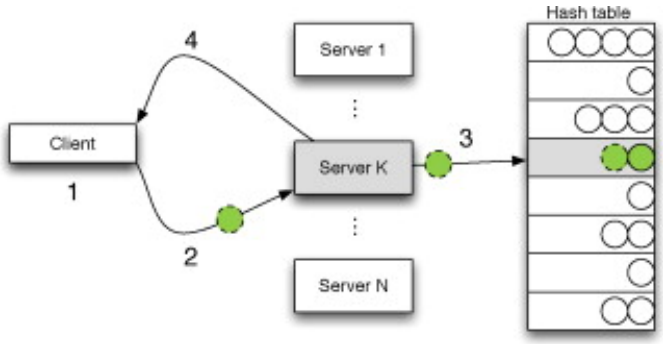


Figure 7: El cliente selecciona un servidor (1) calculando *k1= consistente-hash1(clave)* y le envía (2) el par *(clave, valor)*. A continuación, el servidor calcula *k2 = hash2(clave)*mod M utilizando una función hash diferente y almacena (3) todo el par (clave, valor) en la ranura apropiada *k2* de la tabla hash, utilizando el encadenamiento en caso de conflictos. Por último, el servidor acusa recibo (4) de la operación al cliente.

En el caso de una petición GET, si la clave a recuperar está almacenada en una tabla, lo que se conoce como acierto, el par (clave, valor) es devuelto al cliente. En caso contrario, lo que se conoce como fallo, el servidor notifica al cliente la falta de

clave. Sin embargo, una diferencia notable con respecto a la ruta de escritura es que los clientes pueden optar por utilizar el protocolo UDP, más rápido pero menos fiable, para las solicitudes GET. Podemos ver como sería una petición GET en la siguiente imagen:

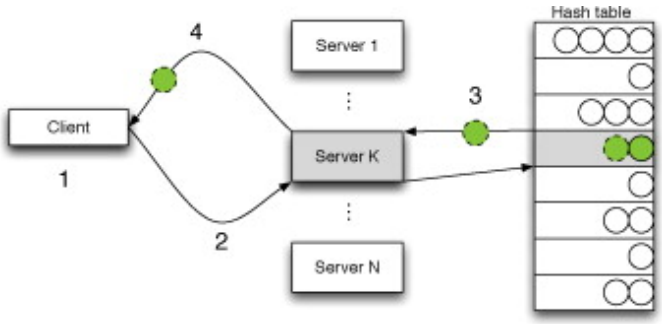


Figure 8: El cliente selecciona un servidor (1) calculando *k1 = consistente-hash1(clave)* y le envía (2) la clave. El servidor calcula *k2 = hash2(clave)*mod M y busca (3) un par *(clave, valor)* en la ranura k2 apropiada de la tabla hash (y recorre la cadena de elementos si hubiera alguna colisión). Por último, el servidor devuelve (4) la *(clave, valor)* al cliente o le notifica el registro que falta.

Cabe destacar que las operaciones GET pueden tomar múltiples claves como argumento. En este caso, Memcached devuelve todos los pares de CV que fueron recuperados con éxito de la tabla. La ventaja de este enfoque es que permite agregar múltiples peticiones GET en un solo paquete de red, reduciendo el tráfico de red y las latencias. Pero para ser eficaz, esta función requiere que los servidores tengan una gran cantidad de RAM, de modo que es más probable que los servidores tengan varias claves de interés en cada solicitud.

Memcached permite controlar el tiempo de vida de un objeto, indicando el tiempo de expiración para el mismo, en el momento de realizar una operación de almacenamiento. Además, el protocolo implementa comandos para recuperar estadísticas, vaciar la caché o utilizar algún tipo de compresión, entre otros.

A diferencia de las bases de datos que almacenan datos en el disco, Memcached guarda sus datos en la memoria. Como no hay necesidad de acceder al disco, los almacenes de clave-valor como Memcached evitan los retrasos y pueden acceder a los datos en cuestión de milisegundos. Memcached también funciona con el modelo distribuido, lo que significa que es fácil escalar con el agregado de nuevos nodos. Memcached es multiproceso, puede escalar la capacidad de computación.

Como resultado de su velocidad y escalabilidad, así como su diseño simple, su eficaz gestión de memorias y la compatibilidad de la API con los idiomas más popu-

lares, Memcached es una opción común para almacenamiento en caché a gran escala y de alto rendimiento.

Memcached nos permite utilizar memoria de partes del sistema donde tenemos más de la que necesitamos y hacer que esta sea accesible a zonas donde tenemos menos de la que necesitamos. De esta forma podemos hacer un mejor uso de la memoria. Si, por ejemplo, consideramos los diagramas de abajo podemos ver dos posibles escenarios:

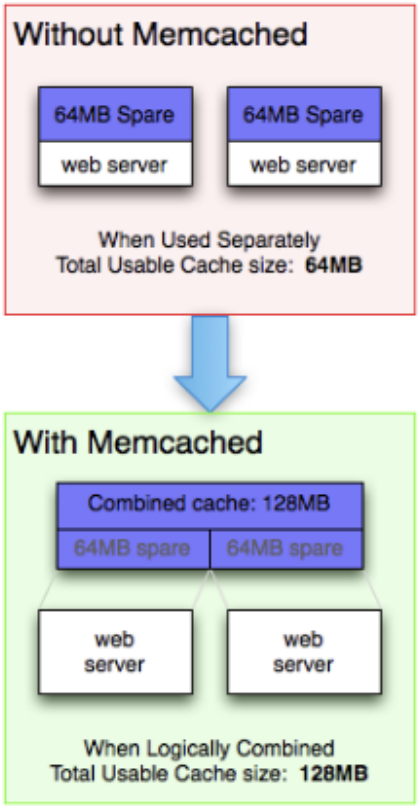


Figure 9: Relación entre el uso de Memcached y los nodos

En la primera estructura de la imagen [Figura 9], tenemos un escenario de implementación clásica, en el que cada nodo es completamente independiente, provocando que la cantidad total de memoria caché sea una fracción del tamaño total de memoria caché de la granja web, haciendo también que la cantidad de esfuerzo requerido para mantener la caché consistente a través de todos estos nodos sea mayor. Por otra parte, en la segunda estructura de la imagen, vemos que utilizando Memcached, cada nodo puede hacer uso de la memoria de los otros nodos. Es decir, podemos ver que todos los servidores están en el mismo pool virtual de memoria. Esto

significa que un elemento determinado siempre se almacena y recupera de la misma ubicación en todo su clúster web.

Además, cuando la demanda de tu aplicación va subiendo hasta el punto en el cual necesitas tener más servidores, normalmente sube también en términos de los datos que tienen que ser regularmente accedidos. Una estrategia de implementación como Memcached en la cual estos dos aspectos del sistema crezcan juntos tiene sentido.

Obviamente, no es obligatorio usar la memoria de los servidores web para caché. Muchos usuarios de Memcached tienen máquinas especializadas que están diseñadas para ser sólo servidores de Memcached. Por ejemplo, en una arquitectura de servicio web, la aplicación Memcached se encuentra entre los servidores web front-end o nivel web y la base de datos back-end como se muestra en la siguiente figura.

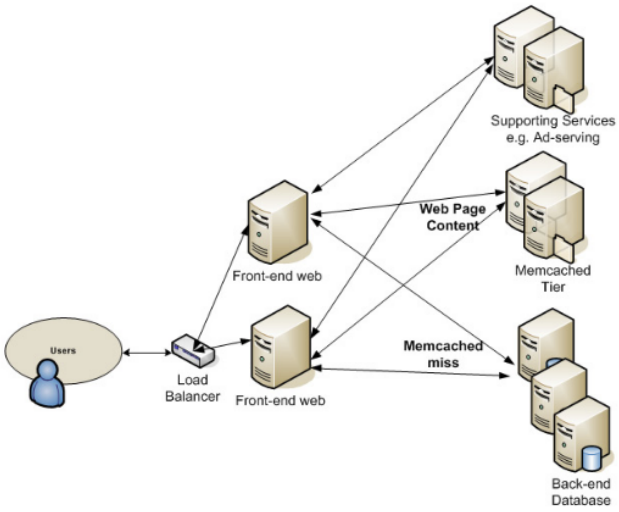


Figure 10: Arquitectura de servicio web con uso de Memcached

El objetivo de Memcached es interceptar peticiones e intentar satisfacerlas cuando sea posible, evitando accesos al almacenamiento en disco adjunto a la base de datos back-end.

También existe la posibilidad de evitar tareas de cálculo intensivo recuperando un valor precalculado de la memoria caché lógica. En ambos casos, se reduce el tiempo empleado en recuperar o calcular los resultados de los datos. Un clúster de servidores forma la caché lógica Memcached, con cada servidor ofreciendo su memoria de sistema como parte del la caché lógica completa.

5 Instalación de Memcached en Ubuntu

Memcached es una herramienta que introduce en memoria swap o ram información de bases de datos y archivos frecuentemente, de manera que acelera el acceso a estos archivos en posteriores consultas.

A continuación vamos a proceder a la instalación de la herramienta Memcached en nuestra máquina de Ubuntu [7]. Lo haremos en Ubuntu18.04 pero se puede realizar en cualquier versión de Ubuntu. Los pasos son los siguientes:

1. Tras actualizar el sistema, procedemos a instalar los paquetes oficiales de **Memcached** ejecutando el siguiente comando:

```
inaki@inaki-N501VW:~$ sudo apt install memcached
```

2. A continuación, vamos a instalar **libMemcached-tools**, que es una biblioteca en la cual se integran herramientas diseñadas para optimizar el uso de Memcached. Esto lo realizaremos con el siguiente comando:

```
inaki@inaki-N501VW:~$ sudo apt install libmemcached-tools
```

3. Una vez completados estos pasos, debemos validar que la instancia de Memcached sea escuchada en la ruta local **127.0.0.1**. Esto lo validamos en el directorio /**etc/Memcached.conf** y es importante aclarar que las nuevas actualizaciones han integrado el parámetro **-l** directamente en la interfaz local, gracias al cual, se previenen ataques de denegación de servicio desde la red externa y esto nos garantiza un mejor nivel de servicio.

```
inaki@inaki-N501VW:/etc$ sudo vi memcached.conf
```

```
# Start with a cap of 64 megs of memory. It's reasonable, and the daemon default
# Note that the daemon will grow to this size, but does not start out holding th
is much
# memory
-m 64

# Default connection port is 11211
-p 11211

# Run the daemon as root. The start-memcached will default to running as root if
 no
# -u command is present in this config file
-u memcache

# Specify which IP address to listen on. The default is to listen on all IP addr
esses
# This parameter is one of the only security measures that memcached has, so mak
e sure
# it's listening on a firewalled interface.
-l 127.0.0.1
```

4. Debido a que hemos modificado el archivo de configuración de Memcached debemos reiniciar el servicio de Memcached, llamado *Memcached.service*, para guardar los cambios que previamente hemos realizado en el archivo de configuración. Para ello ejecutamos el siguiente comando:

```
inaki@inaki-N501VW:/etc$ sudo systemctl restart memcached.service
```

5. Después de realizar estos cambios en el archivo de configuración, es momento de comprobar que Memcached está integrado a la interfaz local y está escuchando por el puerto 11211 tan sólo peticiones realizadas a través del protocolo TCP. Esto lo realizamos mediante el comando *netstat*:

```
inaki@inaki-N501VW:/etc$ sudo netstat -plunt
Conexiones activas de Internet (solo servidores)
Proto  Recib Enviad Dirección local         Dirección remota       Estado       PID/Program name
tcp        0      0 127.0.0.1:3306          0.0.0.0:*              ESCUCHAR     1297/mysqld
tcp        0      0 127.0.0.1:11211         0.0.0.0:*              ESCUCHAR     9019/memcached
```

6. Con el fin de añadir usuarios autenticados a Memcached, podemos hacer uso de la capa simple de autenticación y seguridad **SASL** (Simple Authentication and Security Layer). Esta capa se utiliza, ya que, a menudo Memcached se implementa en redes que no son de confianza o donde los administradores desean ejercer control sobre los los clientes que se conectan. Para esto, Memcached se puede compilar con este soporte de autenticación SASL.

   Para ello, debemos habilitar SASL en el archivo de configuración de Memcached y posteriormente agregar usuarios deseados. Antes de todo vamos a revisar la conectividad de la instancia de Memcached, usando el siguiente comando:

```
inaki@inaki-N501VW:/etc$ memcstat --servers="127.0.0.1"
Server: 127.0.0.1 (11211)
         pid: 9019
         uptime: 214
         time: 1617098443
         version: 1.5.6
         libevent: 2.1.8-stable
         pointer_size: 64
         rusage_user: 0.049678
         rusage_system: 0.024839
         max_connections: 1024
         curr_connections: 1
         total_connections: 2
         rejected_connections: 0
         connection_structures: 2
         reserved_fds: 20
         cmd_get: 0
         cmd_set: 0
         cmd_flush: 0
         cmd_touch: 0
```

Para habilitar SASL vamos a añadir el parámetro **-S** en el archivo /**etc**/**Memcached.conf**, para esto abrimos de nuevo el archivo de configuración.

```
inaki@inaki-N501VW:/etc$ sudo vi memcached.conf
```

```
# Return error when memory is exhausted (rather than removing items)
# -M

# Maximize core file limit
# -r

# Use a pidfile
-P /var/run/memcached/memcached.pid

-S
```

7. Descomentamos ahora la línea -vv con el cual enviamos información de eventos al archivo /var/log/Memcached:

```
# Run memcached as a daemon. This command is implied, and is not needed for the
# daemon to run. See the README.Debian that comes with this package for more
# information.
-d

# Log memcached's output to /var/log/memcached
logfile /var/log/memcached.log

# Be verbose
# -v

# Be even more verbose (print client commands as well)
-vv

# Start with a cap of 64 megs of memory. It's reasonable, and the daemon default
# Note that the daemon will grow to this size, but does not start out holding th
is much
# memory
-m 64
```

Ahora reiniciamos el servicio de Memcached para que se guarden los cambios realizados en el archivos de configuración con el siguiente comando:

```
inaki@inaki-N501VW:/etc$ sudo systemctl restart memcached.service
```

8. Podemos comprobar los registros con el fin de ver que se haya habilitado el soporte SASL en Ubuntu. Allí debemos ubicar la línea de **Initialized SASL** y validamos de nuevo la conectividad con el comando siguiente:

```
inaki@inaki-N501VW:/etc$ sudo journalctl -u memcached
-- Logs begin at Tue 2021-03-30 10:14:39 CEST, end at Tue 2021-03-30 12:08:53 CE
mar 30 11:50:07 inaki-N501VW systemd[1]: Started memcached daemon.
mar 30 11:57:11 inaki-N501VW systemd-memcached-wrapper[7982]: Signal handled: Te
mar 30 11:57:11 inaki-N501VW systemd[1]: Stopping memcached daemon...
mar 30 11:57:11 inaki-N501VW systemd[1]: Stopped memcached daemon.
mar 30 11:57:11 inaki-N501VW systemd[1]: Started memcached daemon.
mar 30 12:06:09 inaki-N501VW systemd-memcached-wrapper[9019]: Signal handled: Te
mar 30 12:06:09 inaki-N501VW systemd[1]: Stopping memcached daemon...
mar 30 12:06:09 inaki-N501VW systemd[1]: Stopped memcached daemon.
mar 30 12:06:09 inaki-N501VW systemd[1]: Started memcached daemon.
mar 30 12:06:09 inaki-N501VW systemd-memcached-wrapper[9285]: Initialized SASL.
mar 30 12:06:09 inaki-N501VW systemd-memcached-wrapper[9285]: slab class   1: ch
mar 30 12:06:09 inaki-N501VW systemd-memcached-wrapper[9285]: slab class   2: ch
mar 30 12:06:09 inaki-N501VW systemd-memcached-wrapper[9285]: slab class   3: ch
```

Como vemos no se genera ningún resultado, allí podemos ingresar lo siguiente para validar el estado de Memcached:

```
inaki@inaki-N501VW:/etc$ memcstat --servers="127.0.0.1"
inaki@inaki-N501VW:/etc$ echo $?
1
```

El hecho de que la ejecución de *echo$?* sea 1 nos indica que ha habido un error en esa consulta.

9. Ahora vamos a instalar **sasl2-bin**, el cual es un paquete donde estarán disponibles las herramientas administrativas para la gestión de la base de datos de usuarios de SASL. Para su instalación ejecutamos el siguiente comando:

```
inaki@inaki-N501VW:/etc$ sudo apt install sasl2-bin
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
El paquete indicado a continuación se instaló de forma automática y ya no es necesario.
  linux-hwe-5.4-headers-5.4.0-66
Utilice «sudo apt autoremove» para eliminarlo.
Se instalarán los siguientes paquetes adicionales:
  db-util db5.3-util
Se instalarán los siguientes paquetes NUEVOS:
  db-util db5.3-util sasl2-bin
0 actualizados, 3 nuevos se instalarán, 0 para eliminar y 56 no actualizados.
Se necesita descargar 173 kB de archivos.
Se utilizarán 680 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]
```

10. Ingresamos la letra **S** para indicar que queremos continuar con la descarga e instalación, y a continuación, vamos a crear el directorio y el archivo que serán usados por Memcached para verificar las configuraciones de SASL. Al directorio lo llamaremos *sasl2* y al archivo *Memcached.conf*:

```
inaki@inaki-N501VW:/etc$ sudo mkdir sasl2
inaki@inaki-N501VW:/etc$ sudo vi /sasl2/memcached.conf
```

Una vez que hemos creado el archivo y el directorio, modificamos el contenido del archivo y añadimos lo que aparece en la captura de abajo. Le indicamos que el nivel de registro va a ser 5 y el fromato plano para poder hacer uso de nuestro archivo de contraseña y verificar la contraseña de texto sin formato. Guardamos los cambios y vemos con el comando *cat* que se ha guardado correctamente:

```
mech_list: plain

log_level: 5

sasldb_path: /etc/sasl2/memcached-sasldb2
```

```
inaki@inaki-N501VW:/etc/sasl2$ cat memcached.conf
mech_list: plain
log_level: 5
sasldb_path: /etc/sasl2/memcached-sasldb2
```

11. El siguiente paso es crear la base de datos SASL, asociando las credenciales de usuarios que deseamos que tengan acceso. Esto lo logramos con el comando **saslpasswd2**. Usaremos la opción **'-c'** para crearlo y la opción **'-f'** para especificar la ruta de la base de datos.

```
inaki@inaki-N501VW:/etc/sasl2$ sudo saslpasswd2 -a memcached -c -f /etc/sasl2/memcached-sas
ldb2 solvetic
Password:
Again (for verification):
```

12. Vamos a otorgar al usuario *memcache* privilegios sobre la base de datos SASL ejecutando el siguiente comando:

```
inaki@inaki-N501VW:/etc/sasl2$ sudo chown memcache:memcache /etc/sasl2/memcached-sasldb2
```

13. Reiniciamos el servicio y ejecutamos **memcstat** con el objetivo de validar si el proceso de autenticación fue exitoso. Lo ejecutaremos con los credenciales de

autenticación:

```
inaki@inaki-N501VW:/etc$ sudo systemctl restart memcached.service
```

```
inaki@inaki-N501VW:/etc/sasl2$ memcstat --servers="127.0.0.1" --username=solvetic --passwor
d=inaki1998
```

14. Ahora vamos a proceder con la instalción de **Apache** y **Php** para poder comprobar que la configuración que hemos realizado de Memcached es correcta:

```
inaki@inaki-N501VW:/etc$ sudo apt-get install apache2 php7.2 libapache2-mod-php7.2 php-memcached php7.2-cli -y
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
apache2 ya está en su versión más reciente (2.4.29-1ubuntu4.14).
libapache2-mod-php7.2 ya está en su versión más reciente (7.2.24-0ubuntu0.18.04.7).
fijado libapache2-mod-php7.2 como instalado manualmente.
php7.2 ya está en su versión más reciente (7.2.24-0ubuntu0.18.04.7).
```

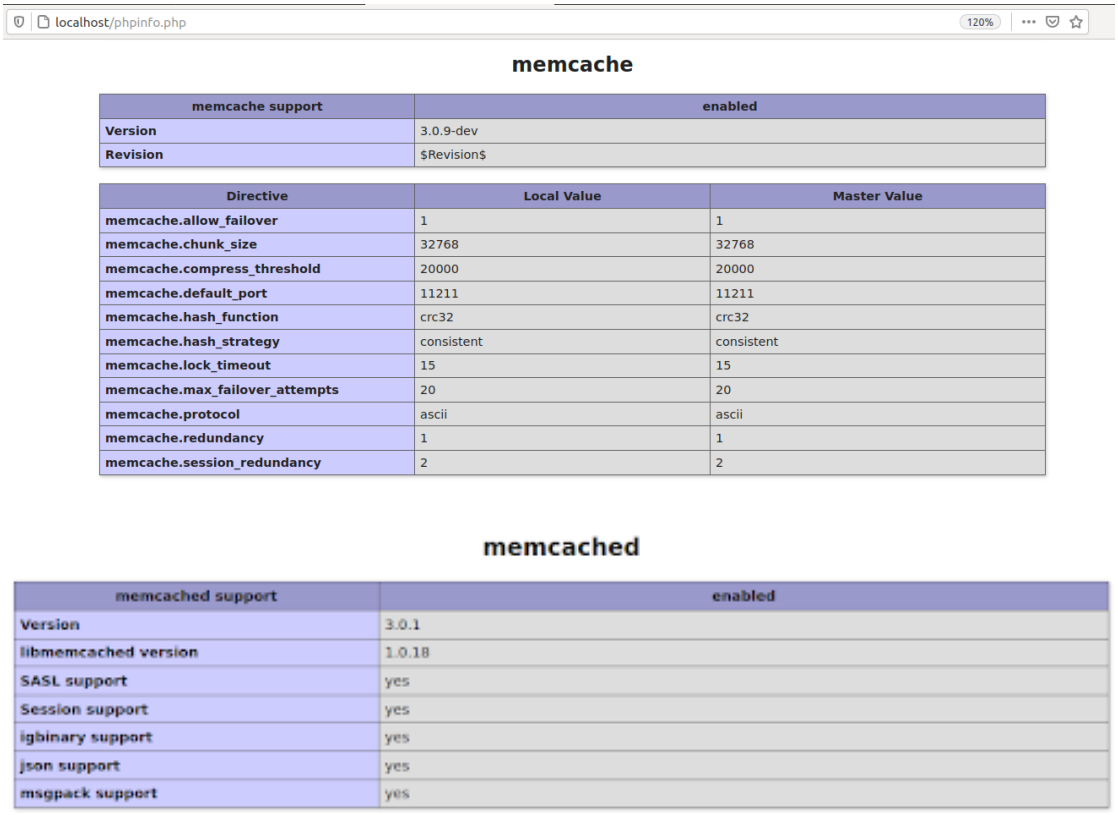
Una vez realizada la instalación de Apache y PHP vamos a crear un archivo en el directorio de Apache (*/var/www/html*) llamado *phpinfo.php*, que llame a la funcion phpinfo(), que se puede utilizar para obtener una gran cantidad de información sobre tu instalación de PHP:

```
inaki@inaki-N501VW:/var/www/html$ cat phpinfo.php
<?php phpinfo(); ?>
```

Reiniciamos los servicios de Memcached y Apache con los siguientes comandos:

```
inaki@inaki-N501VW:/var/www/html$ sudo systemctl restart apache2.service
inaki@inaki-N501VW:/var/www/html$ sudo systemctl restart memcached.service
```

Accedemos a un navegador cualquiera y al buscar el archivo que hemos creado nos aparece la configuración de memcache y de Memcached como se puede ver en las siguientes imágenes:



6 Configuración de Memcached en Ubuntu

Una vez que hemos completado la instalación, ya podemos hacer uso de él, pero podemos modificar los parámetros de configuración y cambiar los valores que trae por defecto tras su instalación. Para modificar estos parámetros, es necesario que editemos el fichero **Memcached.conf**, que está ubicado en el directorio /**etc** [Figura 11]. Dentro de este archivo, nos encontraremos una serie de opciones que podemos configurar según necesitemos.

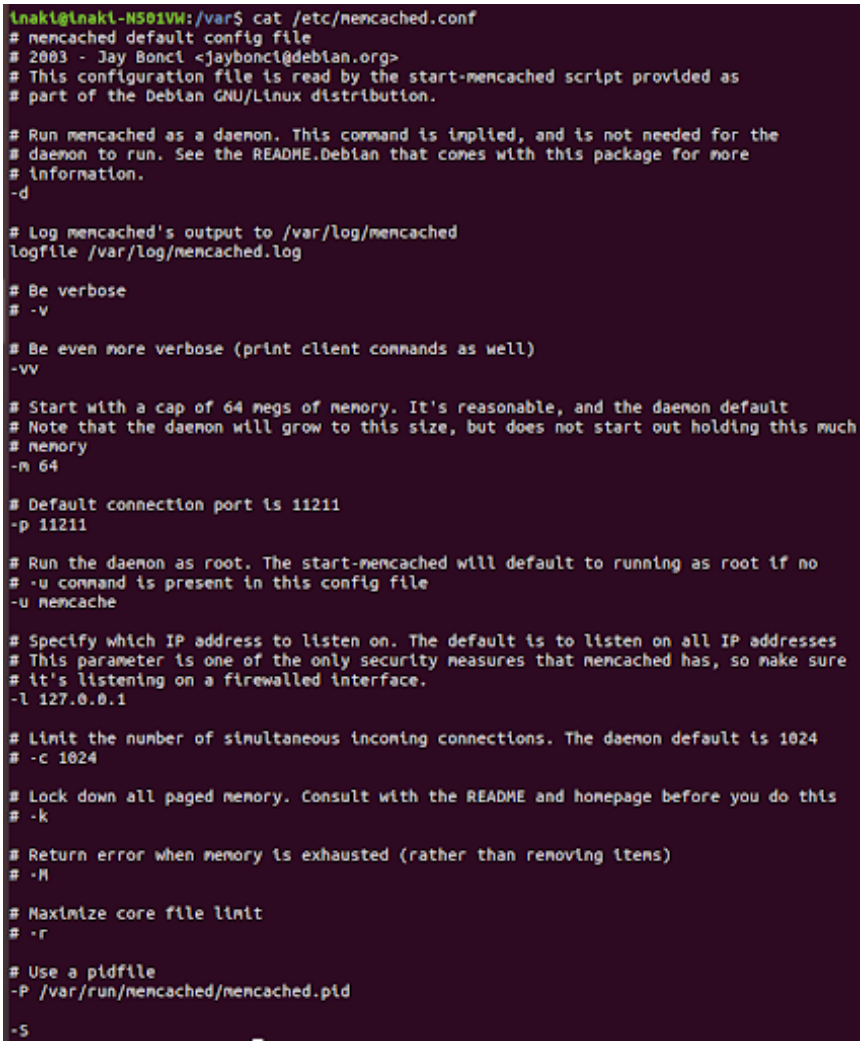


Figure 11: Archivo de configuración de Memcached

Algunas de las opciones que encontraremos en el fichero de configuración son las siguientes:

- **-p:** Indica el puerto por el que nos debemos conectar al servidor Memcached. Por defecto el puerto es el 11211, pero podemos poner cualquier otro que esté libre.
- **-m:** Indica el número de MB de memoria máxima que utilizará Memcached, yendo creciendo esta, hasta el tamaño indicado. El valor predeterminado de esta es 64MB.
- **-l:** Especifica la dirección IP desde la que escuchará Memcached. Se pueden especificar varias direcciones separadas por comas o utilizando -l varias veces.
- **-c:** Indica el número de conexiones simultáneas, por defecto está a 1024.
- **-r:** Indica el límite máximo de los ficheros del núcleo.
- **-s:** Indica la ruta del socket de Unix para escuchar.
- **-d:** Ejecuta Memcached como un demonio.
- **-P 'nombre archivo':** Imprime el pidfile de 'nombre archivo ', en caso de usar la opción -d.

Para ver más opciones, es interesante consultar el manual de Memcached.[8]

7 Ejemplos de uso

Para nuestro primer ejemplo de uso supondremos que tenemos una tabla en nuestra base de datos, donde almacenamos unos clientes. En esta tabla tendremos todos sus datos personales de estos clientes (nombre, apellido, dirección, DNI, . . .).

En primer lugar crearemos un objeto de la clase Memcache. Para ello será necesario indicarle la dirección IP o nombre del servidor donde está instalada la herramienta y el número de puerto donde nos podemos conectar . En nuestro caso, supondremos que está instalado en el mismo servidor con el puerto por defecto (recordemos que el puerto predeterminado era el 11211), por lo que el nombre del servidor que utilizaremos será **localhost** y el puerto **11211**.

```
// Inicializamos y conectamos
$memcache = new Memcache;
$memcache->connect('localhost', 11211) or die("No podemos conectarnos");
```

Ahora creamos una estructura que queremos cachear. A dicha estructura la llamaremos cliente y estará compuesta por dos atributos: nombre y apellido.

```
// Creamos una estructura a cachear
$cliente = new stdClass;
$cliente->nombre = "Daniel";
$cliente->apellido = "Lopez";
```

Almacenamos la estructura con un tiempo de expiración de 10 segundos, que será el tiempo que dicha estructura estará almacenada en la caché. Recordemos que el protocolo de reemplazo de estructuras podía ser el LRU, o el que estamos utilizando aquí, el tiempo de expiración. Le asociaremos como clave el DNI de ese cliente, en este caso ”12345678X”:

```
// Almacenamos la estructura con una expiracion de 10 segundos
$memcache->set("12345678X", $cliente, false, 10) or die ("No podemos guardar la estructura");
```

Ahora vamos a proceder a obtener la estructura mediante la clave, que en este caso era su DNI, es decir, **“12345678X”** y la vamos a guardar en una variable llamada result que mostraremos a continuación:

```php
// Recuperamos la estructura
$result = $memcache->get("12345678X");
echo "Estructura recuperada:<br/>\n";
print_r($result);
```

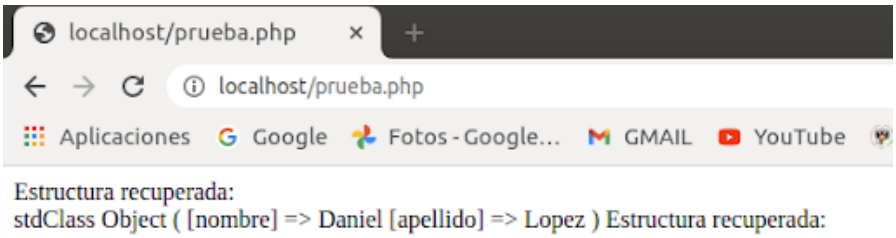
Como habíamos establecido previamente que el tiempo de expiración de esta estructura en caché iban a ser 10 segundos, vamos a probar a obtener esa estructura pasados 11 segundos desde que se introdujo en caché:

```php
sleep(11);
$result2 = $memcache->get("12345678X");
echo "Estructura recuperada: <br/>\n";
print_r($result2);

?>
```

Aquí se muestra el código completo que hemos utilizado:

```php
<?php

// Inicializamos y conectamos
$memcache = new Memcache;
$memcache->connect('localhost', 11211) or die("No podemos conectarnos");

// Creamos una estructura a cachear
$cliente = new stdClass;
$cliente->nombre = "Daniel";
$cliente->apellido = "Lopez";

// Almacenamos la estructura con una expiracion de 10 segundos
$memcache->set("12345678X", $cliente, false, 10) or die ("No podemos guardar la estructura");

// Recuperamos la estructura
$result = $memcache->get("12345678X");
echo "Estructura recuperada:<br/>\n";
print_r($result);

sleep(11);
$result2 = $memcache->get("12345678X");
echo "Estructura recuperada: <br/>\n";
print_r($result2);

?>
```

Como era de prever la primera vez que intentamos recuperar la estructura, se recupera perfectamente, pero ya la segunda vez, al haber trancurrido el tiempo de expiración que le habíamos asociado a dicha estructura, no se puede recuperar:



Como segundo ejemplo, vamos a mostrar como podemos cachear datos alojados en una base de datos. Para ello, creamos una instancia de Memcached y nos conectamos a la base de datos (llamada Memcached). Posteriormente, creamos la consulta para obtener el ID de un usuario. Finalmente, asociamos una clave con la que cachear ese valor obtenido mediante la consuta anterior:

```php
<?php
$memtest = new Memcached();
$memtest->addServer("127.0.0.1", 11211);
$conn = mysqli_connect("localhost", "imm98", "inaki1998") or die(mysqli_error());
mysqli_select_db($conn, "Memcached") or die(mysqli_error());
$query = "SELECT ID FROM tabla WHERE name = 'inaki'";
$retval = mysqli_query($conn, $query );
$result = mysqli_fetch_all($retval, MYSQLI_ASSOC);
$querykey = "KEY" . md5($query);
$memtest->set($querykey, $result);
$result2 = $memtest->get($querykey);
if ($result2) {
print "<p>Data was: " . $result2['ID'] . "</p>";
print "<p>Caching success!</p><p>Retrieved data from memcached!</p>";
}
?>
```

Para la realización de estos ejemplos, hemos hecho uso de los manuales de PHP y MySQL con Memcached [9] [10]

8 Ventajas de Memcached

Las principales ventajas que encontramos al hacer uso de Memcached son las siguientes:

- **Reducir los niveles de carga de los servidores:** Esta es la primera gran ventaja del uso de este sistema, ya que, gracias a él se reducen el número de consultas a la base de datos, debido a que almacena los datos en la caché de los servidores, lo que provoca un rendimiento fulgurante.
- **Flexibilidad de memoria dedicada:** Memcached permite ajustar la cantidad de memoria que va a ser destinada al almacenamiento de información mediante la herramienta descrita.
- **Escalabilidad:** La arquitectura de Memcached es fácil de escalar. Puede dividir sus datos en varios nodos, lo que le permite agregar nuevos nodos a su clúster para aumentar su capacidad. Por tanto si crece el número de consultas y necesitamos añadir nuevos servidores web a la granja podemos añadir dichos nodos al clúster para que la eficiencia de Memcached sea proporcional al número de servidores web.
- **Tiempo de respuesta:** El tiempo de respuesta de Memcached está por debajo del milisegundo.
- **Simplicidad y facilidad de uso:** Memcached está diseñado para ser simple y genérico, de modo que resulta un servicio eficaz y fácil de usar en el desarrollo de aplicaciones. Hay muchos clientes de código abierto disponibles para los desarrolladores de Memcached. Entre los lenguajes admitidos se encuentran Java, Python, PHP, C, C++, JavaScript, Node.js, Ruby, Go y muchos otros.
- **Comunidad:** Memcached es un proyecto completamente desarrollado de código abierto que tiene a una gran comunidad dinámica. No hay limitaciones de proveedores ni tecnología porque Memcached está basado en estándares abiertos, admite formatos de datos de datos de código abierto y cuenta con una base de clientes completa.
- Memcached está configurado para admitir **clústeres** y administrar automáticamente el equilibrio de carga.

9 Desventajas de Memcached

Memcached está diseñado para ser usado en **sistemas que no están conectados a Internet**, pero por desgracia muchos de ellos sí lo están. Esto tiene bastantes desventajas:

- Memcached no requiere **autenticación** y puede tener listeners en TCP y UDP. Al poder hacerse spoofing (usurpar una identidad electrónica para ocultar la propia identidad) con facilidad en UDP, se puede usar como reflector. Indagaremos un poco más en el siguiente punto.
- El espacio de la caché es **limitado**: se dice que la memoria caché de una computadora será inestable si está por encima de 2 GB y los datos se perderán sin ningún motivo.
- **Pérdida de datos** después de un corte de energía: debido a que los datos se almacenan en memoria principal, que es volátil, todos los datos se perderán una vez que se apague la máquina.
- **Volatilidad:** El bloqueo de la instancia del servidor borra todos los datos almacenados dentro de la sesión.
- **Poco soporte de documentación** al ser una herramienta relativamente nueva.
- **Seguridad:** Para una simplicidad eficiente, todas las operaciones Memcached se tratan por igual. Los clientes con permisos de acceso a entradas de baja seguridad dentro de la caché obtienen también acceso a todas las entradas de la caché aún cuando son de mayor seguridad. La falta de mecanismos de seguridad, hace que sea necesario un cortafuegos adicional.
- Dificultad en la **depuración** de los datos.
- No es **redundante**, es decir, no duplica ni realiza copias de seguridad de los datos para protegerlos frente a fallos.
- La longitud de la clave de valores se limita a 250 caracteres (1MB).

10 Memcached attack

Un ataque de denegación de servicio distribuido de Memcached (DDoS) es un tipo de ataque cibernético, en el que un atacante intenta sobrecargar a una víctima con tráfico de Internet. El atacante falsifica solicitudes a un servidor UDP Memcached vulnerable y bombardea a la víctima con tráfico de Internet lo que puede sobrecargar los recursos de la víctima.

Un **"ataque Memcached"** (Memcached Attack) funciona de forma similar al ataque DDoS. El ataque funciona mandando solicitudes falsas a servidores vulnerables, que luego responden con una cantidad de datos muy superior a la petición inicial, aumentando el volumen de tráfico. La amplificación Memcached puede ser pensada en el contexto de un malvado adolescente que llama a un restaurante y dice: "Querría pedir un plato de todo, por favor llámame y cuéntame todo mi pedido". Cuando el restaurante le pregunta por un teléfono de contacto, el número que le da el joven es el de la víctima. Dicha víctima entonces recibe la llamada de vuelta del restaurante que tenía un montón de información que él no había solicitado. [11]

Este ataque funciona porque los servidores Memcached tienen la opción de usar el protocolo UDP. El protocolo UDP es un protocolo de red que permite el envío de datos sin primero haber establecido lo que se conoce como "handshake", que es un proceso de red en el que ambas partes aceptan establecer esa comunicación. UDP es utilizado debido a que el host objetivo no es consultado acerca de si quiere o no recibir los datos, permitiendo que una cantidad masiva de datos sean mandados a la víctima sin su previo consentimiento.

Los pasos para realizar este ataque son:

1. El atacante implanta una gran carga de datos en un servidor Memcached.
2. Después, el atacante realiza una petición **HTTP GET** con la dirección IP de la víctima
3. El vulnerable servidor Memcached que recibe la petición, que realiza su labor de responder la petición, manda una larga respuesta a la víctima.
4. El servidor víctima o su infraestructura es incapaz de procesar esta cantidad de datos mandados por el servidor Memcached, provocando una sobrecarga y una denegación de servicio a peticiones legítimas

11 Memcached Hosting

Memcached se combina mejor con un **host** cuya infraestructura admite velocidades rápidas. Si tiene un sitio web con más de 3000 páginas o una tienda de comercio electrónico, puede beneficiarse de Memcached [12].

Es ideal para sitios con cargas pesadas y por ello está implementado en sitios punteros como Wikipedia y Youtube. Explicaremos que buscar en un host y ofreceremos recomendaciones de alojamiento.

Todos los servidores Memcached miran a un grupo virtual de memoria. Esto significa que sus elementos almacenados siempre se guardan y recuperan de una ubicación coherente dentro de un clúster web. Cuando experimente un crecimiento, Memcached continúa escalando los aspectos de su sistema juntos, asegurando que pueda acceder a cantidades crecientes de datos sin recibir un golpe de rendimiento.

Aunque Memcached generalmente se implementa en sitios web confiables, algunos administradores pueden desear tomar medidas de seguridad adicionales donde les gustaría mantener el control sobre los clientes que se conectan.

En estos casos se puede compilar con autenticación simple y capa de seguridad SASL. Si está ejecutando un sitio y tiene un tráfico creciente, puede usar el alojamiento de Memcached para acomodar la carga del sitio. Es muy probable que obtenga páginas web para cargar más rápido para visitantes. Simultáneamente, un servidor Memcached transferirá sus datos al almacenamiento persistente sin afectar el rendimiento del sitio.

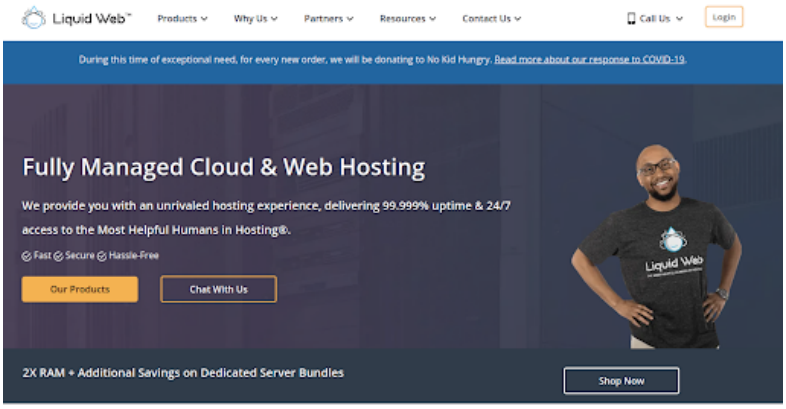
En los paquetes predeterminados, un proveedor de hosting puede reservar una pequeña cantidad de RAM para la caché, dependiendo de la distribución de Linux o del sistema varía entre 64MB-512MB. Con el alojamiento Memcached, puede darle más memoria a Memcached dependiendo del tamaño de su implementación.

Memcached es una característica que en sí misma es gratis y se ofrece junto con otras características como MySQL, cPanel y PHP. Las empresas sólo necesitan configurar un sitio web con un proveedor de alojamiento web que ofrezca Memcached en su paquete.

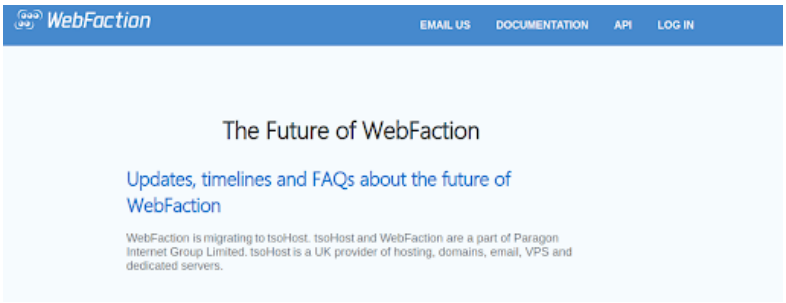
Algunos de los proveedores que debe considerar seriamente al desarrollar un sitio basado en Memcached son los siguientes:

- **SiteGround:**SiteGround brinda amplias instrucciones en su sitio web para usar Memcached con varias aplicaciones web. Su SuperCacher patentado puede usar Memcached u otros motores de almacenamiento de caché para acelerar el rendimiento del servidor web. La atención al cliente es excelente y está disponible 24 horas. Múltiples centros de datos y Cloudflare CDN aumentan la velocidad de entrega de la página y garantizan un tiempo de actividad del 99.9% [13]

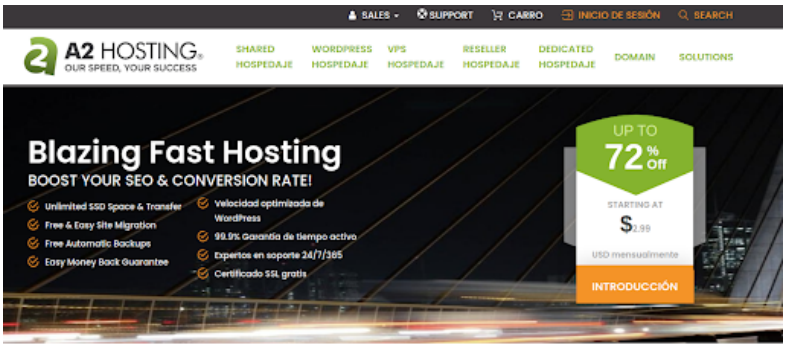
- **Liquid Web:** Admite Memcached en sus planes VPS y planes de servidor dedicado. Los clientes deben instalarlo ellos mismos, pero las instrucciones para todos los servidores compatibles están disponibles en su sitio web. El alojamiento gestionado de LiquidWeb incluye soporte 24/7 con tiempos de respuesta inicial garantizados. No es la opción menos costosa pero ofrece un buen valor. [14]



- **WebFaction:** Cuenta con hosting para desarrolladores y proporciona información técnica detallada sobre el uso de Memcached. Eso proporciona acceso SSH y puede ejecutar cualquier herramienta de marcos de su elección, en muchos lenguajes de programación. El principal inconveniente es la falta de soporte en vivo por teléfono o chat. [15]



- **A2Hosting:** Obtuvo el número 1 en recientes pruebas de velocidad y rendimiento. Tiene un gran soporte en chat o por teléfono. Algunas de las ventajas son: CMS incluídos, CPanel fácil de usar, herramientas web-builder eficiente, soporte Memcached,... [16]



Up To 20X Faster Hosting To Help You Succeed!

12 Memcached vs Redis

Redis se autodefine como un 'almacen de estructura de datos en memoria de código abierto utilizado como base de datos, caché y agente de mensajes'. Almacena pares clave-valor en una selección de diferentes tipos de datos como listas, conjuntos,... Redis mantiene los datos en memoria, lo que significa que es extremadamente rápido a la hora de devolverlos cuando se solicitan. Esta velocidad lo hace perfecto como caché para su aplicación.

A simple vista, Redis y Memcached parecen hacer lo mismo. Tanto Redis como Memcached:

- Almacenan datos en la memoria para una recuperación rápida, lo cual es perfecto para el almacenamiento en caché.
- Son un almacenamiento de datos NoSQL, que mantienen los datos como pares clave-valor.
- Ambos son de código abierto y bien documentados.

Sin embargo, se puede ver que los conjuntos completos de funciones son diferentes entre los dos, y que se requieren ciertas consideraciones antes de decicir cuál elegir.

Hay una serie de diferencias clave entre Redis y Memcached que vamos a mencionar. Estas diferencias se reflejan en cómo Redis y Memcached manejan los datos que almacenan en caché.

- **Tipos de datos:** Redis almacena los datos como tipos específicos, mientras que Memcached lo hace como cadenas. Debido a esto, Redis puede cambiar los datos en su lugar sin tener que volver a cargar todo el valor de los datos. Esto reduce la sobrecarga de la red. A continuación, se explicarán los distintos tipos de datos que puede manejar Redis.
  - **Cadenas:** Las cadenas son un valor básico para almacenar datos. Son binarias seguras, lo que significa que las cadenas pueden almacenar algo como una imagen pequeña. Los valores de cadena pueden tener hasta 512MB de longitud.
  - **Lista:** Las listas en Redis son una forma de almacenar datos desordenados, a lo que luego se puede acceder mediante índices. Redis implementa una lista vinculada, que es una forma diferente de implementar listas en una matriz. Esto significa un mejor rendimiento en cuanto a velocidad y mayor consistencia en las operaciones de insertar y eliminar un valor al principio o al final de la lista.

  - **Hash:** Los hash permiten el almacenamiento de datos estrechamente relacionados en una serie de campos con sus propias relaciones de clave-valor. Aunque es muy similar a las estructuras de datos comunes en los lenguajes de programación, no siempre hay mapeo 1:1 ya que la versión de Redis es plana.
  - **Conjuntos:** Los conjuntos son colecciones desordenadas de elementos únicos. Los conjuntos pueden agregar, eliminar y probar la existencia de datos. El beneficio de su uso sobre las listas es que no permite duplicados, lo que significa que para los datos en los que solo está almacenado una clave única, puedo simplemente agregar la clave y saber que será única dentro del conjunto.
  - **Conjuntos ordenados:** Los conjuntos ordenados son una expansión del conjuntos de Redis, son una colección de cadenas únicas. Sin embargo, cada miembro del conjunto ordenado de Redis recibe una puntuación para ayudar a ordenar el conjunto. Los conjuntos ordenados permiten acceder a rangos según la puntuación que otorga Redis.

- **Persistencia:** La memoria principal es fantástica para la velocidad, sin embargo tiene como inconveniente que si el servidor deja de funcionar, los datos de la memoria se pierden. Sin embargo, Redis permite la persistencia en disco, esto significa que los datos se pueden almacenar y recuperar en caso de que el servidor falle o se reinicie. En cambio, Memcached no tiene la capacidad de persistir los datos en disco de forma nativa.

  Hay dos formas de conservar los datos en Redis:

  - **Registro AOF:** AOF son las siglas de Append Only File, funciona agregando todas las operaciones de escritura recibidas por el servidor. Este registro almacena todos los comandos en el mismo formato que los recibe Redis, lo que significa que se pueden volver a reproducir en la instancia del arranque para reconstruir el estado actual.
  - **Instantánea de RDB:** RDB son las siglas de Database Backup File, que es una forma de tomar una instantánea completa del estado actual. El RDB es una forma de almacenar el estado actual en un archivo que se puede transferir fuera del sitio para la recuperación en caso de desastre. RDB funciona creando un proceso hijo y dejando que este complete la copia de seguridad para que el proceso principal no esté ocupado creando el archivo RDB y escribiendo en el disco.

  La documentación de Redis aconseja usar una combinación de ambos.

Otras de las peculiaridades que tiene Redis, comparándolas con las de Memcached, son:

- **Longitud de datos:** Las claves y cadenas de datos de Redis pueden tener hasta 512MB de longitud. Como binarios seguros, tambien pueden almacenar cualquier tipo de datos. Memcached admite una clave de solo 250 bytes y los valores un límite de 1 MB.
- **Políticas de desalojo de datos:** La memoria es finita en los servidores, los datos obsoletos en caché pueden ser desalojados para dar paso a nuevos datos. Esto se maneja a través de un proceso llamado desalojo de datos. Tanto Redis como Memcached tienen formas de manejar esto, sin embargo, no siempre son iguales. Una forma de llevar a cabo esto, es desalojando los datos que no se han visto afectados recientemente. Si los datos se ven afectados, se indica y por tanto no será un candidato para el desalojo. Esta es la única forma en que funciona Memcached, sin embargo, Redis ofrece varias otras formas de lidiar con el desalojo de datos. Redis deja que la memoria se llene y luego no tome más claves y con un TTL (Time to live) volátil donde Redis intentará eliminar claves con un TTL establecido.
- **Replicación:** La replicación es una forma de copiar datos de una instancia a otra creando una réplica. La intención es garantizar que se mantenga una copia duplicada de los datos. Redis utiliza técnicas de maestro-esclavo para realizar sus replicas. Las instancias pueden tener más de una réplica para mayor redundancia. Memcached no admite la replicación.
- **Agrupación:** La agrupación en clústeres es una forma de garantizar la alta disponibilidad de un servicio. La creación de clústeres puede ser difícil de configurar y administrar cuando el software no esta diseñado para eso. Como solución para el almacenamiento en caché, Redis ofrece **Redis Cluster** que habilita funciones de agrupación en clústeres. Algunos de los beneficios de la agrupación son:
  - **Actuación:** Redis experimentará un mejor rendimiento a medida que la carga de los clientes se distribuya en varios clústeres.
  - **Disponibilidad:** El uso de una combinación de replicación junto con la agrupación en clústeres permite una alta disponibilidad para las aplicaciones. Redis cluster se puede configurar con una selección de instancias maestras y réplicas conectadas.
  - **Precios y escalabilidad:** Como Redis consume mucha memoria, la agrupación en clústeres permite la escalabilidad horizontal donde la carga se distribuye en varias instancias de memoria más pequeñas en lugar de una única instancia con mucha memoria.

- **Subprocesos múltiples:** Memcached anteriormente tenía la ventaja sobre Redis en velocidad de subprocesos múltiples. Esto significaba que en determinadas situaciones, Memcached superaba a Redis. Sin embargo, a partir de Redis 6, el subproceso es compatible con Redis y por esto se ven unas mejoras en las cargas de trabajo de subprocesos múltiples.

En la siguiente imagen [Figura 12] se muestra un resumen de las diferencias y similitudes entre Memcached y Redis comentadas anteriormente.

Feature	Redis	Memcached
Store key value pair in memory	√	√
Open source	√	√
Supports different data types natively	√	x
Supports native persistence	√	x
Replication	√	x
Clustering	√	x
Supports multithreading	√	√

Figure 12: Comparativa entre Memcached y Redis

Es interesante utilizar Memcached cuando se tiene un aplicación muy simple con pocos servidores y que solo requiera una interpretación de cadena simple. Sin embargo, incluso con cargas de trabajo pequeñas, se recomienda el uso de Redis. Debería utilizar Redis cuando:

- Necesita acceso a un conjunto más amplio de estructura de datos y capacidad de procesamiento de secuencias.
- Se requiere capacidad de modificar claves y valores.
- Se requieren políticas de desalojo de datos personalizadas.
- Debe conservar datos en el disco para realizar copias de seguridad y reinicios.
- Necesita tener alta disponibilidad o escalabilidad mediante uso de réplicas y agrupación en clústeres.

13 Conlusiones

Memcached es una herramienta relativamente nueva que tuvo un éxito fulgurante entre los sitios web más destacados del panorama internacional desde el primer momento. Mejoró mucho la velocidad de los sitios web que la implementaron, una mejora necesaria ante el incesante crecimiento en la demanda web y el aumento de las exigencias de velocidad de los usuarios que hubo a final de la década de los 2000. Estas exigencias continuan hasta nuestros días, con un acceso cada vez más globalizado a Internet y un abaratamiento de los dispositivos que utilizan esta tecnología.

Además, como hemos visto, es una herramienta bastante sencilla de usar, que además de reducir la carga de los servidores web, debido a que no todas las consultas son redirigidas a la base de datos, aumenta la velocidad de servicio de los recursos web. El éxito de esta tecnología se debe también en gran parte a la escalabilidad de esta, una propiedad muy importante, ya que la eficiencia en el funcionamiento de esta herramienta crece proporcionalmente al número de servidores que tengamos en nuestro clúster web.

En un sistema como el actual, donde hay tanta cometencia tecnológica y los usuarios finales tienen tantas opciones donde elegir, la velocidad de servicio web se ha convertido en un factor crucial en cualquier proyecto web, razón por la cual, herramientas como Memcached o Redis, son necesarias.

Al ser Memcached una herramienta open-source, hay en desarrollo actualmente muchos proyectos de investigación para añadir mejoras a esta tecnología, que si logra solventar algunos de los problemas de seguridad mencionados durante el trabajo, pocas empresas dudarán en usarla.

References

[1] Facebook Engeniiring. Scaling memcached at facebook. https://www.facebook.com/note.php?note_id=39391378919&ref=mf, 2008.

[2] Jure Petrovic. Using memcached for data distribution in industrial environment. In *Third International Conference on Systems (icons 2008)*, pages 368–372. IEEE, 2008.

[3] Biz Stone. It's not rocket science, but it's our work. https://blog.twitter.com/official/en_us/a/2008/its-not-rocket-science-but-its-our-work.html, 2008.

[4] Bin Fan, David G Andersen, and Michael Kaminsky. Memc3: Compact and concurrent memcache with dumber caching and smarter hashing. In *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 371–384, 2013.

[5] Funcionamiento Memcached. https://pressroom.hostalia.com/white-papers/memcached/. [Online; accessed 1-Mayo-2021].

[6] Mateusz Berezecki, Eitan Frachtenberg, Mike Paleczny, and Kenneth Steele. Power and performance evaluation of memcached on the tilepro64 architecture. *Sustainable Computing: Informatics and Systems*, 2(2):81–90, 2012.

[7] Solvetic Sistemas. Cómo instalar memcached ubuntu 19.04 y ubuntu 18.04. https://www.solvetic.com/tutoriales/article/7706-como-instalar-memcached-ubuntu-1904-y-ubuntu-1804/, 2019.

[8] memcached - high-performance memory object caching system. https://linux.die.net/man/1/memcached. [Online; accessed 12-Mayo-2021].

[9] PHP-Memcached. https://www.php.net/manual/es/book.memcached.php. [Online; accessed 27-Abril-2021].

[10] PHP-MySQL. https://dev.mysql.com/doc/refman/5.6/en/ha-memcached-interfaces-php.html. [Online; accessed 27-Abril-2021].

[11] Memcached Attack. https://www.adslzone.net/2018/03/02/que-es-memcached-ddos/. [Online; accessed 3-Mayo-2021].

[12] Gary McGath. The best memcached hosting: Who's the best for your site? https://www.whoishostingthis.com/compare/memcached/, 2019.

[13] SiteGround. https://www.siteground.es/. [Online; accessed 3-Mayo-2021].

[14] LiquidWeb. https://www.liquidweb.com/. [Online; accessed 3-Mayo-2021].

[15] WebFaction. https://www.webfaction.com/. [Online; accessed 3-Mayo-2021].

[16] A2Hosting. https://www.a2hosting.com/wordpress-hosting?aid=zar&cid=edae5de3. [Online; accessed 3-Mayo-2021].