Memcached and Perl

Colin Bradford

Professional iT

generating efficiency

Agenda

- **What is memcached**
- **Example Uses**
- **Deployment**
- **Gotchas**

What is memcached

- **In-memory cache daemon**
- **Simple operations:**
  - set(key, data, expiry time)
  - get(key)
  - . . . plus some others

Key features

- **Client based key distribution to multiple servers**
  - Servers do not need to communicate with each other
- **Server written in C, runs on most Unix platforms**
- **Very fast (all operations in O(1))**
- **Simple text based network protocol**

Accessing from Perl

```
use Cache::Memcached;

#Connect
my $cache = Cache::Memcached->new(servers => [ “10.0.0.1:11211”,
    “10.0.0.2:11211” ]);

# Set some data - $data can be a ref, as long as Storable can nfreeze it
$cache->set($key, $data, 3600); # 1 hour expiry

# Get the data back
my $x = $cache->get($key);
# or get multiple pieces simultaneously
my $hashref = $cache->get_multi($key1, $key2, $key3);
```

Example uses

- **LOVEFiLM**
  - Product data
  - Customer data
  - Editorial text

LOVEFiLM: Product data

- **Infrequently changing data**
- **Expensive to compute**
  - multiple tables for actors, directors, related titles
- **Frequently accessed**
  - every page has at least one product
  - home page has 8
  - some pages have more than 30

Advantages of memcached

- **Central store of product data**
  - no duplication of data in memory on multiple servers
- **Fast access to perl data structure**
- **Long expiry time (days)**
  - product updates can be pushed to the central cache

Customer data

- **Store regularly used data**
  - eg Customers rental list, account data
- **Each page view may go to a different web server, so local caches have poor hit rate**
- **Updates to the cache data can be pushed to the central cache; a local cache would be stale**
- **Short expiry times – only needed for the length of a visit**

Editorial text

- **Text on homepage can be edited by editorial team**
- **Central cache can be updated – so long expiry times, but very quick updates**
- **Reduces database load**

Deployment

- **“Typical” deployment puts memcached on web hosts, using spare memory**
- **Very low CPU, can run anywhere that has spare memory**
- **Client hashing algorithm determines server to use for a specific key**

Gotchas

- **Can't store undef**
  - get returns undef on a miss, so you can't store that a key doesn't exist
- **1Mb limit (as standard) on objects**
  - data size (once frozen) must be under 1Mb. ->set doesn't warn you if it's bigger
- **Storable.pm problems with mixed 32/64bit environments**
  - Older versions of Storable didn't cope with a mixed environment

Gotchas

- **A failed cache will timeout (eventually)**
  - The timeout is configurable, but Cache::Memcached doesn't correctly mark servers that are down.
- **Beware of the cost of computing data if the cache is down**
  - A down cache will cause all data that it stores to be recalculated every time it's needed.
- **Beware of context switches**
  - Running memcached on web servers causes context switches between Apache and memcached
  - This slows page build times (measurably for LOVEFiLM)

Gotchas

- **"Stale Slab" problem**
  - memcached allocates a slab of memory as needed
  - A slab holds a single size of object and are not reclaimed
  - memcached eventually hits the configured limit of slabs
  - If the size of your objects changes, memcached many not have the right number of slabs for that size of object – and the hit rate goes down

Tips

- **It's a cache – it will lose data**
  - so make sure you can recreate anything in the cache
- **Think about and test cache failures**
  - Maybe a main/standby method works better for a use case
- **Monitor hit rates**
  - track hit rates (ganglia), and investigate changes


Thank you
Professional iT
generating efficiency