Memcached and Perl

Colin Bradford

Professional iT

generating efficiency

Agenda

- **Introduction to caching**
- **What is memcached**
- **Example Uses**
- **Gotchas**

What is a cache?

- **A cache is "a collection of data duplicating original values stored elsewhere or computed earlier" according to Wikipedia**
- **Store data under a key**
- **Often implemented as an associative array**
- **Limited in size, with an eviction policy**

Why cache?

- **Performance increase**
  - Reduce disk IO and wait times
  - Reduce the amount of calculation
- **Load decrease**
  - For databases, reduce the number of queries
  - For servers, reduce the amount of CPU spent on calculations

What is memcached

- **In-memory cache daemon, with a TCP interface**
- **Simple operations:**
  - set(key, data, expiry time)
  - get(key)
  - . . . plus some others
- **Evicts based on LRU (with caveats)**
- **Open source server (BSD)**
- **Clients for multiple languages**
  - Need to ensure data can be read across all

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**
  - Non blocking
- **Simple text based network protocol**
  - Newer releases have a binary protocol as well

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**
  - 50ish page views/second
  - Each is personalised
  - Wide range of products (75,000+) and users (900,000+)
- **Cache uses**
  - Product data
  - Customer data
  - Editorial text

LOVEFiLM: Product data

- **Infrequently changing data**
- **Expensive to compute**
  - multiple tables for actors, directors, related titles
  - Data is manipulated before presentation
- **Frequently accessed**
  - Nearly every page has at least one product
  - Home page has around 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
- **Cache catalogue object**
  - Cache the result of computation, not the result of a single database query

Customer data

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

Editorial text

- **Text on site can be edited by editorial team**
- **Central cache can be updated**
  - Long expiry times, but very quick updates
- **Reduces database load**
- **Can store the results of processing**
  - Turning movie titles into links
  - Checking for trailers

Performance

- More than 10,000 cache fetches per second
- Single threaded daemon
- Gigabit network connection
- 7.5 Gb cache
- > 95% hit ratio (across all keys)

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**

Deployment in code

- **Need to cache at the appropriate level**
  - Caching individual database queries is easy
    - May not give best performance increase
    - Hard to get cache invalidation correct
  - Caching the end result may give lower hit rates
  - Caching partial results, but still doing some computation may yield good results
  - Test and Benchmark!
- **Use a namespace separator in the key, to avoid clashes**

Alternative APIs

- **Use callbacks to calculate data on a miss**

```
Cache->retrieve($key, $expiry, \&callback, @callbackdata);
```

- **Can instrument cache misses by namespace**
- **Can time data calculation, to do a "cost of miss" calculation**
- **Can instrument cache writes that are never read**

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 – silently fails
- **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 didn'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
- **Don't use *set* and *get* as method names**
  - Use *store* and *retrieve*, or similar – less likely to confuse in code


Thank you
Professional iT
generating efficiency