INTRODUCTION TO MEMCACHED

jurriaanpersyn.com
lead web dev at Netlog
since 4 years
php + mysql +
frontend
working on Gatcha

What is caching?

NETLOG

- From Wikipedia: "A cache is a collection of data duplicating original values stored elsewhere or computed earlier, where the original data is expensive to fetch (owing to longer access time) or to compute, compared to the cost of reading the cache."
- Term introducted by IBM in the 60's

The anatomy

NETLOG

- simple key/value storage
- simple operations
  - save
  - get
  - delete

Terminology

NETLOG

- storage cost
- retrieval cost (network load / algorithm load)
- invalidation (keeping data up to date / removing irrelevant data)
- replacement policy (FIFO/LFU/LRU/MRU/RANDOM vs. Belady's algorithm)
- cold cache / warm cache

Terminology

NETLOG

- cache hit and cache miss
- typical stats:
  - hit ratio (hits / hits + misses)
  - miss ratio (1 - hit ratio)

- 45 cache hits and 10 cache misses
  - 45/(45+10) = 82% hit ratio
  - 18% miss ratio

When to cache?

NETLOG

- caches are only efficient when the benefits of faster access outweigh the overhead of checking and keeping your cache up to date
- more cache hits then cache misses

Where are caches used?

NETLOG

- at hardware level (cpu, hdd)
- operating systems (ram)
- web stack
- applications
- your own short term vs long term memory

Where to cache on the server-side?

NETLOG

- As PHP backend developer, where to store cache results?
  - in database (computed values, generated html)
    - you'll still need to access your database
  - in static files (generated html or serialized php values)
    - you'll still need to access your file system

in memory!

memcached

About memcached

NETLOG

- Free & open source, high-performance, distributed memory object caching system
- Generic in nature, intended for use in speeding up dynamic web applications by alleviating database load.
- key/value dictionary

About memcached (cont'd)

NETLOG

- Developed by Brad Fitzpatrick for LiveJournal in 2003
- Now used by Netlog, Facebook, Flickr, Wikipedia, Twitter, YouTube ...

Technically

NETLOG

- It's a server
- Client access over TCP or UDP
- Servers can run in pools
  - eg. 3 servers with 64GB mem each give you a single pool of 192GB storage for caching
  - Servers are independent, clients manage the pool

What to store in memcache?

NETLOG

- high demand (used often)
- expensive (hard to compute)
- common (shared accross users)
- Best? All three

What to store in memcache? (cont'd)

NETLOG

- Typical:
  - user sessions (often)
  - user data (often, shared)
  - homepage data (eg. often, shared, expensive)

What to store in memcache? (cont'd)

NETLOG

- Workflow:
  - monitor application (query logs / profiling)
  - add a caching level
  - compare speed gain

Memcached principles

NETLOG

- Fast network access (memcached servers close to other application servers)
- No persistency (if your server goes down, data in memcached is gone)
- No redundancy / fail-over
- No replication (single item in cache lives on one server only)
- No authentication (not in shared environments)

Memcached principles (cont'd)

NETLOG

- 1 key is maximum 1MB
- keys are strings of 250 characters (in application typically MD5 of user readable string)
- No enumeration of keys (thus no list of valid keys in cache at certain moment, list of keys beginnen with "user_", ...)
- No active clean-up (only clean up when more space needed, LRU)

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
get foo
VALUE foo 0 2
hi
END
stats
STAT pid 8861
(etc)
```

Client Access

NETLOG

- both ASCII as Binary protocol
- in real life:
  - clients available for all major languages
    - C, C++, PHP, Python, Ruby, Java, Perl, Windows, ...

Output caching

NETLOG

- Pages with high load / expensive to generate
- Very easy
- Very fast
- But: all the dependencies ...
  - language, css, template, logged in user's details, ...

Query Caching (cont'd)

NETLOG

- queries with JOIN and WHERE statements are harder to cache
  - often not easy to find the cache key on update/change events
  - solution: JOIN in PHP

Query Caching (cont'd)

NETLOG

- queries with JOIN and WHERE statements are harder to cache
  - often not easy to find the cache key on update/change events
  - solution: JOIN in PHP
  - In following example: what if nickname of user changes?

So?

NETLOG

- Pro's:
  - speed, duh.
  - queries get simpler (better for your db)
    - easier porting to key/value storage solutions
- Cons:
  - You're relying on memcached to be up and have good hit ratios

Multi-Get Optimisations

NETLOG

- We reduced database access
- Memcached is faster, but access to memcache still has it's price
- Solution: multiget
  - fetch multiple keys from memcached in one single call
  - result is array of items

Consistent Hashing

NETLOG

- client is responsible for managing pool
  - hashes a certain key to a certain server
- clients can be naïve: distribute keys on size of pool
- if one server goes down, all keys will now be queried on other servers > cold cache
- use a client with consistent hashing algorithms, so if server goes down, only data on that server gets lost

jurriaan@netlog.com

Resources, a.o.:

- memcached & apc: http://www.slideshare.net/benramsey/caching-with-memcached-and-apc
- speed comparison: http://dealnews.com/developers/memcachedv2.html
- php client comparison: http://code.google.com/p/memcached/wiki/PHPClientComparison
- cakephp-memcached: http://teknoid.wordpress.com/2009/06/17/send-your-database-on-vacation-by-using-cakephp-memcached/
- caching basics: http://www.slideshare.net/soplakanets/caching-basics
- caching w php: http://www.slideshare.net/JustinCarmony/effectice-caching-w-php-caching