Varnish Cache

Emanuele Rocca

ZenMate DevOps Day 2

Web performance

300x - 1000x speedup

Outline

- Introduction
- Design principles
- Object storage
- Architecture
- Conclusions

Varnish 101

- Web server accelerator AKA caching HTTP reverse proxy
- Really fast. Delivery speedup 300x - 1000x
- Client <-> Varnish <-> Backend

Caching

- Cached responses are two order of magnitude faster
- Maximize cache hits
- Contents are stored in cache according to the backend response (Cache-Control header)
- Caching behavior can be changed using policies written in VCL, the Varnish Configuration Language

Basic config

/etc/default/varnish

```
DAEMON_OPTS="-a :80 \
             -T localhost:6082 \
             -f /etc/varnish/example.vcl \
             -S /etc/varnish/secret \
             -s malloc,256m"
```

Basic config

/etc/varnish/example.vcl:

```
vcl 4.0;

backend example {
    .host = "www.varnish-cache.org";
    .port = "80";
}
```

VCL vs. Settings

- The configuration is written in VCL
- Not switching settings on or off
- It is transformed into C code, built, loaded and executed upon varnish startup
- Writing policies on how incoming traffic should be handled

varnishadm

- Stopping and starting the cache process
- Loading VCL
- Adjusting the built-in load balancer
- Invalidating cached content

varnishlog

- Varnish does not log to disk
- Logs are streamed to a chunk of memory
- varnishlog allows to connect to the stream and inspect the logs

Design principles

- Focus on performance and flexibility
- Design for today

Performance and flexibility

- Multithreaded
- Log to memory to reduce lock-contention between threads
- Binary search tree to quickly store and retrieve cached items

Design for today

- 64-bit architectures, multi-core scalability, advanced OS features
- Leave it to the OS to decide where memory is. Just request a large chunk of memory
- epoll instead of select(2), poll(2)

epoll

- On high loads the one process/thread per connection architecture does not provide good performance
- epoll(7), introduced in Linux 2.6
- O(1) instead of O(n) to monitor n file descriptors
- http://kovyrin.net/2006/04/13/epoll-asynchronous-network-programming/

Object storage

- Objects are stored in memory. References are kept in a tree, not in a hash table. Each node has a key
- Keys are potentially arbitrarily long. Users can choose what to use as a key

Default key

```
sub vcl_hash {
    hash_data(req.url);
    if (req.http.host) {
        hash_data(req.http.host);
    } else {
        hash_data(server.ip);
    }
    return (lookup);
}
```

Problems with long keys

- Storage requirements
- The tree can quickly become unbalanced

Solution

- Keys are cryptographically hashed with SHA256 to ensure compression and randomness
- Anything can be used as a key (user identification, cookies. . .)
- Simple tree implementations can be used without worring about inbalance

Architecture

The varnishd program spawns two processes: manager and worker.

```
root      14730  Ss   17:59   0:00 /usr/sbin/varnishd
nobody    14731  Sl   17:59   0:00  \_ /usr/sbin/varnishd
```

Varnish manager

- Talks to the administrator
- Runs as root in order to open privileged ports
- Compiles the VCL program to be executed by the worker

Varnish worker

- Child of manager with minimal permissions
- Does all the actual work with HTTP traffic
- Restarted by the manager if it dies

VCL programs

- Can be compiled and executed at any time
- No need to restart the worker
- No missed HTTP requests

Shared memory

- One segment of shared memory used to report and log activities and status
- Another segment for statistics and counters. Real-time, down to microsecond monitoring of cache hit-rate, resource usage and performance indicating metrics

Conclusions

- Varnish is a very efficient and flexible web server accelerator
- Configured through a language called VCL. Configuration changes do not require restarts
- Data is stored in virtual memory
- Designed for today
- Next steps: learn VCL and play with it!