Varnish http accelerator

- A 2006 software design

Poul-Henning Kamp

phk@FreeBSD.org
Varnish Cheat-Sheet

- Web-acceleration for slow CMS systems
- Narrow focus on server side speedup
  - No FTP etc.
  - Content provider features
- High Performance
  - 32 & 64bit, large RAM, sendfile, accept filters
  - SMP/Multicore friendly architecture
  - 2006 software design
  - 11 syscall + 7 locks per cache hit
dictionary: Varnish

- tr. v. var·nished, var·nish·ing, var·nish·es
  - 1. To cover with varnish.
  - 2. To give a smooth and glossy finish to.
  - 3. To give a deceptively attractive appearance to.
RFC2616 and Varnish

Server domain

HTTP Origin Server

Varnish HTTP cache

RFC2616 origin server

Client domain

HTTP Client

Shared HTTP Cache

Private HTTP Cache

HTTP Client

HTTP Client
Client Cache Situation

- Origin servers are adversarial.
- Anything the origin server says is law
  - ... if we can make sense of it.
- If in doubt: don't cache.
- Be semantically transparent at any cost.
- If origin server does not reply: error.
Server Cache Situation

- Backend (origin server) is on our side
  - More precisely: We are on its side.
- We might be responsible for modifying the origin servers instructions.
  - Change TTL, rewrite URLs etc.
- Whatever happens: protect the backend.
- If backend does not reply: do something!
Content Provider Features

- Instant URL invalidation
  - Regexp matching
  - Easy Integration to CMS system

- Each object checked max 1 time
  - When used next time
  - Many objects will expire without being checked.
Content Provider Features

- Varnish Configuration Language
  - Full expiry time control
  - Load/Situation mitigation
  - Content substitution
  - URL editing
- Prefetching (v2 feature)
  - Inspect object usage stats
  - Compression for bandwidth savings
Varnish Config Language

• Simple domain specific language
  – Compiled via C language to binary
    • Transparantly handled by manager process.
  – Dynamically loaded into cache process
  – Multiple VCLs can be loaded concurrently

• Instant switch from one VCL to another.
  – From CLI or VCL
VCL example

```
acl journalists {
    10.0.0.0/8;
}

if (client.ip ~ journalists) {
    pass;
}

if (req.url.host ~ "cnn.no$") {
    rewrite req.url.host "cnn.no" "vg.no";
}

if (!backend.up) {
    if (obj.exist) {
        set obj.ttl += 10m;
        deliver;
    }
    switch_config "ohhshit";
}
```
Operational Features

- Two process design
  - Automatic restarts on crash
- One binary program
  - Easy installs
- Command Line Interface
  - No need for browser with java
  - Change parameters on the fly
Varnish Architecture

Cluster Controller
CmdLine

Manager
CmdLine
ChildProcMgt
VCL compiler
Params/Args
Initialization
Watchdog

Cacher
CmdLine
Storage
Log/Stats
Accept/herder
Backend
Worker threads
Grim Reaper
Hashing

CLI-interface
Web-interface
CMS-interface
SMS-interface

One binary program
C-compiler
Shared object

Shared Memory
logwriter
stats
ad-hoc
Varnish is fast!

• 1000 cache hits @ 100 Mbit/sec:

<table>
<thead>
<tr>
<th>Sample</th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Average</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>12,9</td>
<td>3001</td>
<td>16,2</td>
<td>71,1</td>
<td>338,5</td>
</tr>
<tr>
<td>90% fractile</td>
<td>12,9</td>
<td>26</td>
<td>15,9</td>
<td>16,3</td>
<td>1,7</td>
</tr>
</tbody>
</table>

(all times are in microseconds)
Varnish Coordinates

- **WWW.varnish-cache.org**
  - Yes, we need to work on the web-page.
- Coded by: Poul-Henning Kamp
  - phk@FreeBSD.org
- Runs on: Any reasonble modern UNIX
  - FreeBSD, Linux etc.
- Open Source
  - BSD license
Why I wrote Varnish

• I was sceptical when VG approached me with the Varnish Project.
  – Not really my specialty
  – Not really my preferred area
• On the other hand...
  – I'm tired of people writing lousy programs and blaming it on ”my” kernel
  – A fine chance to educate by example
• Also, I could use the money.
Performance Programming

- Understand and exploit the envelope of your requirements
- Avoid unnecessary work
- Use few cheap operations
- Use even fewer expensive operations
- Don't fight the kernel
- Use the features the nice kernel guys have given you
"A computer consists of a CPU, internal storage and external storage"
2006 reality

- "A Computer consists of a number of processing units ("cores") connected via a number of caches and busses to a virtualized storage."
A computer consists of a number of cores with private caches, a shared, page granularity, cache for external storage objects.
Squid is 1970 model

- Some objects “in Memory”
- Other objects “on disk”
- Explicit transfer between the two states:
  - Unused memory object gets written to disk
  - Disk object gets read to memory for transmission.
Fight the kernel (1)

- Squid creates a memory object
- Gets used some, then falls into disuse
- Kernel runs short of RAM
- Kernel pages unused pages out to disk
Fight the kernel (2)

- Squid decides memory object is unused and should be moved to disk
- Creates file, issues write(2)
- Page is not in RAM, generates Page-Fault
- Kernel reads data back from disk
- Kernel fixes pagetable, continues squid
Fight the kernel (3)

- Squids write(2) can write object to disk
- Squid can reuse memory for other object
Fight the kernel (4)

- Squid needs first object back
- Push something to disk
- Read in first object back in
Varnish

- Varnish puts object in virtual memory
- Object used some, then falls into disuse
- Kernel pages object out to free RAM for better use.
- Varnish access object in virtual memory
- Kernel page-faults, reads object from disk.
Virtual memory shortage

- On 32 bit systems VM is limited to ~2GB

- If you have a webserver with a larger working set of content, you can afford to buy a new 64 bit CPU.
Performance price List

- char *p += 5;
- strlen(p);
- memcpy(p, q, l);
- Locking
- System Call
- Context Switch
- Disk Access
- Filesystem operation
How Cheap or Expensive?

- CPU clock: 4 Ghz
- Multiple instructions per clock cycle
- Several layers of cache
- Conservative rule of thumb:

  1 Billion instructions per second
Cost of a system call?

- Context switch to kernel is expensive.
  - Thousands of clocks to save/restore registers
  - Argument validation
  - Page Table modifications

- $1 \mu\text{sec} = 0.000001\text{s} = 1,000 \text{ instructions}$
Cost of disk I/O

- Enter kernel
- Navigate disk layout
  - Filesystem, partitions etc.
- Via device driver to hardware
- Busses, handshaking etc.
- Mechanical movement.
- $1 \text{ msec} = 0.001 \text{s} = 1,000,000 \text{ instructions}$
Varnish logging

• Logging to shared memory
  – Practically Free (as in peanuts)
  – No slowdown for the real workload.

• Generate ”real” output in separate processes, which ”tail” the shmlog:
  – Apache format
  – Custom format
  – Realtime views
Logging

• Logging to a file is expensive:

```c
FILE *flog;  // Filesystem operation
flog = fopen("/var/log/mylog", "a");  // Once only

[...]

fprintf(flog, "%s Something went wrong with %s\n", 
        timestamp(), myrepresentation(object));
fflush(flog);  // Disk I/O
```

Millions of calls
Cheap logging

- Shared memory logging is cheap:

  ```c
  char *logp, *loge;
  fd = open(...);
  logp = mmap(..., size);
  loge = logp + size;
  ...
  LOCK(&shmlog);
  logp[1] = LOG_ERROR;
  logp[2] = sprintf(logp + 3,
      "Something went bad with %s",
      myrepresentation(obj));
  logp[3 + logp[2]] = LOG_END;
  logp[0] = LOG_ENTRY;
  logp += 3 + logp[2];
  UNLOCK(&shmlog);
  ```

  Filesystem ops

  Once only

  Memory and arithmetic

  Millions of calls.
But it can be even cheaper

- `strlen(3)` and ASCII/Z format is not cheap
- Keep track of both ends of strings
- Use `memcpy(3)` instead of `strcpy(3)`
  - `memcpy(3)` can move data in 32 or even 64 bit chunks.
  - `strcpy(3)` has to use 8 bit chunks.
But it can be **even** cheaper

- Add a shmlog buffer to worker threads
- Logging to this buffer is lockless
- Batch "commit" local buffer to "real" shmlog at synchronization points.
  - Cost: One lock + one memcpy().
- => Practically no mutex contention.
...and

- Coredumps contains the most recent log records

- We can afford to log much more detail, it does not take up disk-space

- Approx 30 records per request
  - =>100,000's records/sec
"varnishtop"

- Example of ad-hoc shmlog reader:
  - `./varnishtop -i sessionclose`

- Why do sessions close?

<table>
<thead>
<tr>
<th>Duration</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>132232.12</td>
<td>timeout</td>
</tr>
<tr>
<td>39002.59</td>
<td>EOF</td>
</tr>
<tr>
<td>5232.97</td>
<td>not HTTP/1.1</td>
</tr>
<tr>
<td>3069.28</td>
<td>Connection: close</td>
</tr>
<tr>
<td>1383.40</td>
<td>remote closed</td>
</tr>
<tr>
<td>630.93</td>
<td>silent</td>
</tr>
<tr>
<td>15.49</td>
<td>pipe</td>
</tr>
<tr>
<td>9.15</td>
<td>Not in cache.</td>
</tr>
<tr>
<td>6.36</td>
<td>Purged.</td>
</tr>
<tr>
<td>5.41</td>
<td>no request</td>
</tr>
<tr>
<td>0.98</td>
<td>Bad Request</td>
</tr>
</tbody>
</table>
• What is my most popular URL?

<table>
<thead>
<tr>
<th>URL</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/tmv11.js</td>
<td>1304.86</td>
</tr>
<tr>
<td>/sistenytt.html</td>
<td>989.08</td>
</tr>
<tr>
<td>/include/global/art.js</td>
<td>495.05</td>
</tr>
<tr>
<td>/css/hoved.css</td>
<td>491.01</td>
</tr>
<tr>
<td>/gfk/ann/n.gif</td>
<td>490.05</td>
</tr>
<tr>
<td>/gfk/ann/ng.gif</td>
<td>480.08</td>
</tr>
<tr>
<td>/gfk/front/tipsvg.png</td>
<td>468.12</td>
</tr>
<tr>
<td>/css/ufront.css</td>
<td>352.66</td>
</tr>
<tr>
<td>/t.gif</td>
<td>317.75</td>
</tr>
<tr>
<td>/gfk/plu2.gif</td>
<td>306.79</td>
</tr>
<tr>
<td>/css/front.css</td>
<td>298.84</td>
</tr>
<tr>
<td>/gfk/min2.gif</td>
<td>292.84</td>
</tr>
<tr>
<td>/css/blog.css</td>
<td>280.94</td>
</tr>
<tr>
<td>/</td>
<td>279.84</td>
</tr>
</tbody>
</table>
**Where does traffic come from?**

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Referer</th>
</tr>
</thead>
<tbody>
<tr>
<td>33913.74</td>
<td><a href="http://www.vg.no/">http://www.vg.no/</a></td>
</tr>
<tr>
<td>4730.72</td>
<td><a href="http://vg.no/">http://vg.no/</a></td>
</tr>
<tr>
<td>925.62</td>
<td><a href="http://www.vg.no">http://www.vg.no</a></td>
</tr>
<tr>
<td>510.10</td>
<td><a href="http://www.vg.no/pub/vgart.hbs?art">http://www.vg.no/pub/vgart.hbs?art</a></td>
</tr>
<tr>
<td>434.37</td>
<td><a href="http://www.vg.no/export/Transact/menu.html">http://www.vg.no/export/Transact/menu.html</a></td>
</tr>
<tr>
<td>349.55</td>
<td><a href="http://www.vg.no/pub/vgart.hbs?art">http://www.vg.no/pub/vgart.hbs?art</a></td>
</tr>
<tr>
<td>344.66</td>
<td><a href="http://www.vg.no/pub/vgart.hbs?art">http://www.vg.no/pub/vgart.hbs?art</a></td>
</tr>
<tr>
<td>324.06</td>
<td><a href="http://www.vg.no/export/Transact/top.hbs">http://www.vg.no/export/Transact/top.hbs</a></td>
</tr>
<tr>
<td>297.25</td>
<td><a href="http://www.nettby.no/user/">http://www.nettby.no/user/</a></td>
</tr>
<tr>
<td>263.82</td>
<td><a href="http://www.vg.no/sport/fotball/">http://www.vg.no/sport/fotball/</a></td>
</tr>
<tr>
<td>242.55</td>
<td><a href="http://www.vg.no/pub/vgart.hbs?art">http://www.vg.no/pub/vgart.hbs?art</a></td>
</tr>
</tbody>
</table>
Statistics

- Same story
- Stored in shared memory
- Programs can monitor & present data
- No system calls necessary to get up to date numbers
### Real-time statistics

<table>
<thead>
<tr>
<th>Time</th>
<th>Hitrate ratio</th>
<th>Hitrate avg</th>
<th>Client connections accepted</th>
<th>Client requests received</th>
<th>Cache hits</th>
<th>Cache hits for pass</th>
<th>Cache misses</th>
<th>Backend connections initiated</th>
<th>Backend connections recyles</th>
<th>Backend connections unused</th>
<th>N struct srcaddr</th>
<th>N active struct srcaddr</th>
<th>N struct sess_mem</th>
<th>N struct sess</th>
<th>N struct object</th>
<th>N struct objecthead</th>
</tr>
</thead>
<tbody>
<tr>
<td>16:23:13</td>
<td>9 9 9</td>
<td>0.9986 0.9986 0.9986</td>
<td>301.26</td>
<td>2207.26</td>
<td>2201.93</td>
<td>1.44</td>
<td>3.85</td>
<td>5.32</td>
<td>0.01</td>
<td>0.00</td>
<td>0.11</td>
<td>0.03</td>
<td>0.05</td>
<td>0.04</td>
<td>1.11</td>
<td>1.11</td>
</tr>
</tbody>
</table>
Memory Strategy

• How long is a HTTP header?
  – 80 char? 256 char? 2048 char?

• Classic strategy:
  – Allocate small, extend with realloc(3)

• realloc(3) to more memory is expensive
  – Needs memory copy 99% of the times.
Memory Strategy

Special offer!

Virtually **FREE** memory

offer good from 1980 onwards
Memory Strategy

- Allocate enough memory for 99% of the cases up front

- Unused & untouched memory is free
  - as in “peanuts”

- If data is long lived, trim back with realloc(3)
Memory management

- During the handling of a request we need various bits of memory
- Use a private pool
  - Keeps things close together (good caching)
  - Avoids locking against other CPUs
- Free everything with a single pointer assignment at the end
  - As cheap as can be
Memory/Lock interaction

- Do minimum work while holding lock
- Allocate memory before grabbing lock
- If you didn't need it anyway:
  - cache it until next time
Preallocation

LOCK(storage_lock);
TAILQ_FOREACH(...) {
    if (...) 
        break; /* found */
}
if (ptr == NULL) {
    ptr = malloc(sizeof *wp->store);
    assert(wp->store != NULL);
    [...] // fill ptr->stuff
    TAILQ_INSERT_TAIL(...)
}
UNLOCK(storage_lock)

Malloc(3)

May need to synchronize between CPUs with locks
May need to get memory from kernel with syscall
if (wp->store == NULL) {
    wp->store = malloc(sizeof *wp->store);
    assert(wp->store != NULL);
}

LOCK(storage_lock);
TAILQ_FOREACH(...) {
    if (...)
        break; /* found */
}
if (ptr == NULL) {
    ptr = wp->store;
    wp->store = NULL
    [...] // fill ptr->stuff
    TAILQ_INSERT_TAIL(...)
}
UNLOCK(storage_lock)
Avoid unnecessary work

- Don't copy data when we can avoid it
  - Receive headers into one chunk of memory and leave them there

- Don't text-process all HTTP headers
  - We only care about few of them

- Transport headers by reference
  - Use scatter/gather I/O (see writev(2) syscall).
Avoid strlen(3)

- `find_hdr(sp->http, "Host:" )`
  - `find_hdr()` must do `strlen(3)` on literal constant to find out how long it is.

- `find_hdr(sp->http, "Host:" , 5)`
  - Better, but programmer has to get it right.

- `#define HTTP_HOST \005Host:"`
  - Even better, programmer must only get it right once.
  `find_hdr(sp->http, HTTP_HOST );`
Remember the caches

• Reuse most recently used resource.
  – It is more likely to be in cache.

• Process, Thread, Memory, File, Directory, Disk-block etc.

• Avoid round-robin scheduling.
  – Use ”Last-In-First-Out” (ie: stack)
  – Not ”First-In-First-Out” (ie: queue)
Measure & Benchmark

• Understand how often things happen
  – Use tcov(1) or add counters.

• Check your assumptions
  – Use ktrace(8) to see the system-calls.

• Study system statistics
  – Netstat(8), Vmstat(8), sysctl(8)

• Time things if in doubt
  – use clock_gettime(3) or RDTSC()