Running Wikipedia.org

Varnishcon 2016 Amsterdam

Emanuele Rocca
Wikimedia Foundation

June 17th 2016

1,000,000 HTTP Requests

Outline

- Wikimedia Foundation
- Traffic Engineering
- Upgrading to Varnish 4
- Future directions

Wikimedia Foundation

- Non-profit organization focusing on free, open-content, wiki-based Internet projects
- No ads, no VC money
- Entirely funded by small donors
- 280 employees (67 SWE, 17 Ops)

Alexa Top Websites

Company	Revenue	Employees	Server count
Google	$75 billion	57,100	2,000,000+
Facebook	$18 billion	12,691	180,000+
Baidu	$66 billion	46,391	100,000+
Yahoo	$5 billion	12,500	100,000+
Wikimedia	$75 million	280	1,000+

Traffic Volume

- Average: ~100k/s, peaks: ~140k/s
- Can handle more for huge-scale DDoS attacks

DDoS Example

Source: jimieye from flickr.com (CC BY 2.0)

The Wikimedia Family

WIKIPEDIA
The Free Encyclopedia
Wiktionary
The free dictionary
Wikiquote
WIKISOURCE
WIKIBOOKS
WIKIDATA
WIKINEWS
WIKIVERSITY
WIKISPECIES
free species directory
MediaWiki
WIKIMEDIA
COMMONS
wiki
voyage
WIKIMEDIA
FOUNDATION

Values

- Deeply rooted in the free culture and free software movements
- Infrastructure built exclusively with free and open-source components
- Design and build in the open, together with volunteers

Build In The Open

- github.com/wikimedia
- gerrit.wikimedia.org
- phabricator.wikimedia.org
- grafana.wikimedia.org

Traffic Engineering

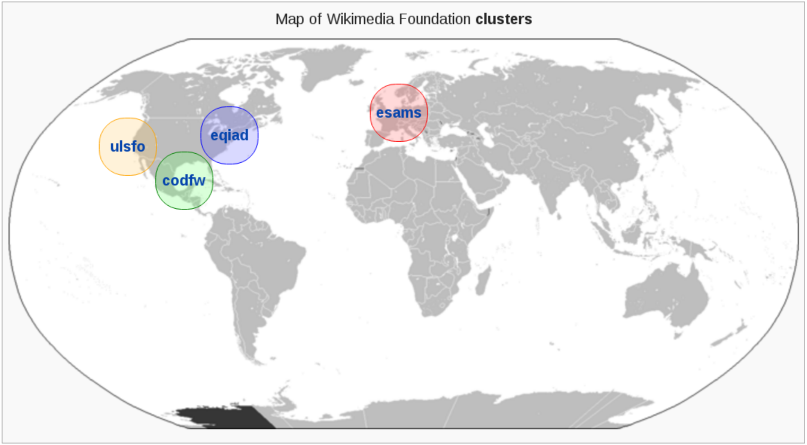
Traffic Engineering

- Geographic DNS routing
- Remote PoPs
- TLS termination
- Content caching
- Request routing

Component-level Overview

- DNS resolution (gdnsd)
- Load balancing (LVS)
- TLS termination (Nginx)
- In-memory cache (Varnish)
- On-disk cache (Varnish)

Cluster Map



eqiad: Ashburn, Virginia - cp10xx
codfw: Dallas, Texas - cp20xx
esams: Amsterdam, Netherlands - cp30xx
ulsfo: San Francisco, California - cp40xx

CDN

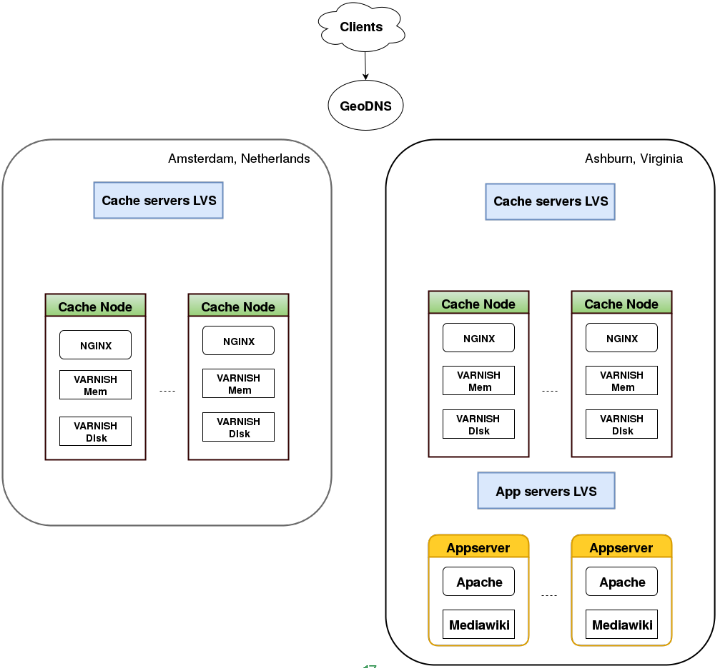
- No third-party CDN / cloud provider
- Own IP network: AS14907 (US), AS43821 (NL)
- Two "primary" data centers
  - Ashburn (VA)
  - Dallas (TX)
- Two caching-only PoPs
  - Amsterdam
  - San Francisco

CDN

- Autonomy
- Privacy
- Risk of censorship

CDN

- Full control over caching/purging policy
- Lots of functional and performance optimizations
- Custom analytics
- Quick VCL hacks in DoS scenarios


Clients
GeoDNS
Amsterdam, Netherlands
Cache servers LVS
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
Ashburn, Virginia
Cache servers LVS
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
App servers LVS
Appserver
Apache
Mediawiki
Appserver
Apache
Mediawiki

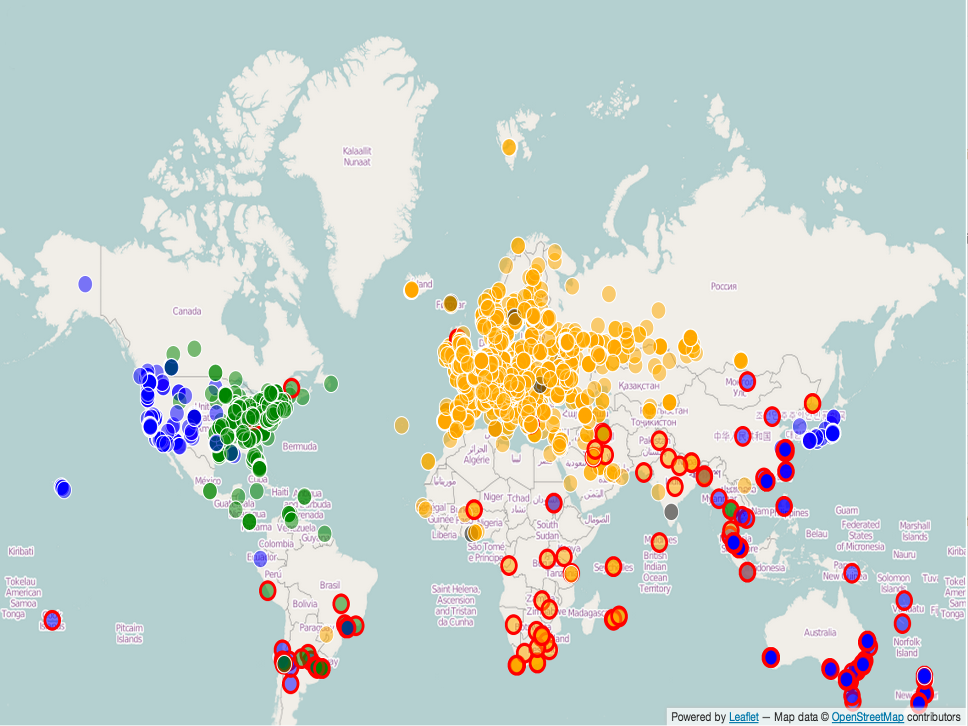
GeoDNS

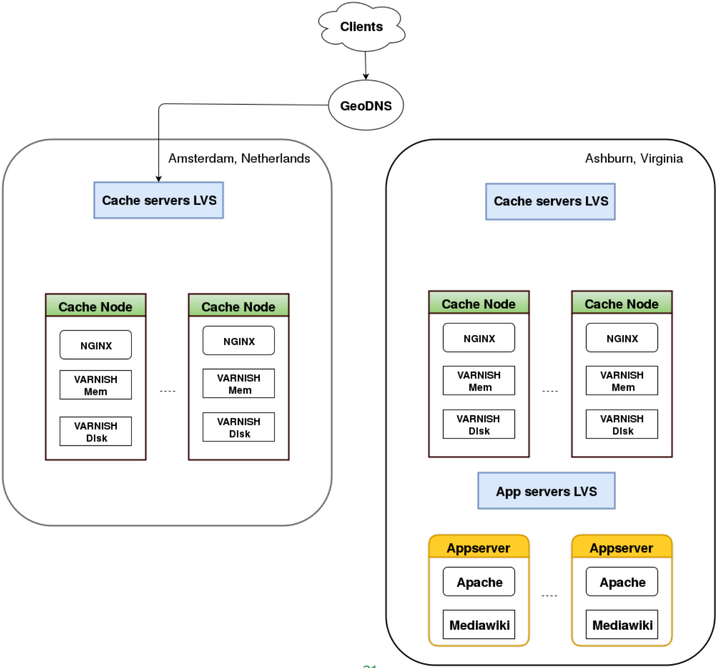
- 3 authoritative DNS servers running gdnsd + geoip plugin
- GeoIP resolution, users get routed to the "best" DC
- edns-client-subnet
- DCs can be disabled through DNS configuration updates

config-geo

```
    FR => [esams, eqiad, codfw, ulsfo], # France
    JP => [ulsfo, codfw, eqiad, esams], # Japan
```

https://github.com/wikimedia/operations-dns/


Kalaallit Nunaat
Canada
Россия
Қазақстан
Тоҷикистон
Bermuda
México
Cuba
Haiti
Colombia
Ecuador
Perú
Brasil
Bolivia
Kiribati
Tokelau
American Samoa
Tonga
Pitcairn Islands
Saint Helena, Ascension and Tristan da Cunha
Algérie
Niger
Tchad
South Sudan
Liberia
São Tomé e Príncipe
British Indian Ocean Territory
Guam
Federated States of Micronesia
Marshall Islands
Belau
Nauru
Solomon Islands
Australia
Norfolk Island
Madagascar
Powered by Leaflet — Map data © OpenStreetMap contributors


Clients
GeoDNS
Amsterdam, Netherlands
Cache servers LVS
Cache Node
NGINX
VARNISH Mem
VARNISH Disk

Cache Node
NGINX
VARNISH Mem
VARNISH Disk
Ashburn, Virginia
Cache servers LVS
Cache Node
NGINX
VARNISH Mem
VARNISH Disk

Cache Node
NGINX
VARNISH Mem
VARNISH Disk
App servers LVS
Appserver
Apache
Mediawiki

Appserver
Apache
Mediawiki

LVS

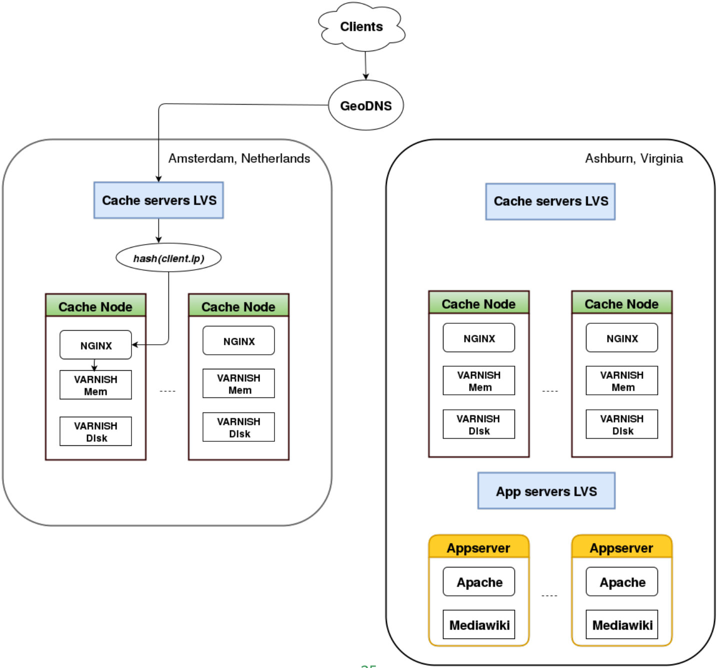
- Nginx servers behind LVS
- LVS servers active-passive
- Load-balancing hashing on client IP (TLS session persistence)
- Direct Routing

Pybal

- Real servers are monitored by a software called Pybal
- Health checks to determine which servers can be used
- Pool/depool decisions
- Speaks BGP with the routers
    - Announces service IPs
    - Fast failover to backup LVS machine

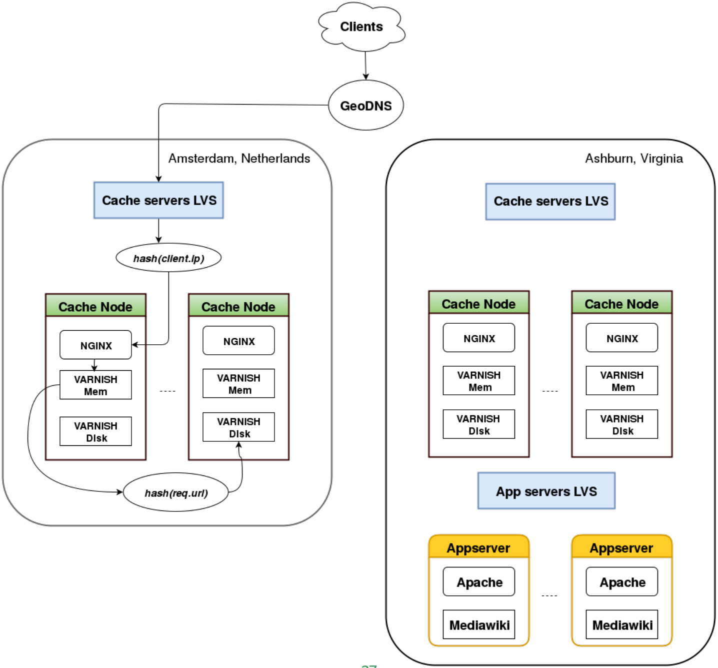
Pybal + etcd

- Nodes pool/weight status defined in etcd
- confctl: CLI tool to update the state of nodes
- Pybal consuming from etcd with HTTP Long Polling


Clients
GeoDNS
Amsterdam, Netherlands
Cache servers LVS
hash(client.ip)
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
Ashburn, Virginia
Cache servers LVS
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
App servers LVS
Appserver
Apache
Mediawiki
Appserver
Apache
Mediawiki

Nginx + Varnish

- 2x varnishd running on all cache nodes
  - :80 -smalloc
  - :3128 -spersistent
- Nginx running on all cache nodes for TLS termination
- Requests sent to in-memory varnishd on the same node


Clients
GeoDNS
Amsterdam, Netherlands
Cache servers LVS
hash(client.ip)
Cache Node
NGINX
VARNISH Mem
VARNISH Disk

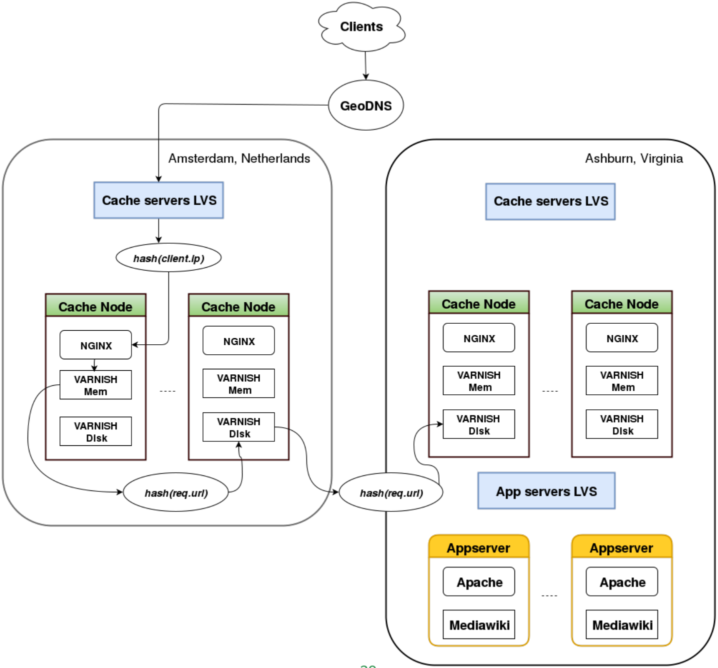
Cache Node
NGINX
VARNISH Mem
VARNISH Disk
hash(req.url)
Ashburn, Virginia
Cache servers LVS
Cache Node
NGINX
VARNISH Mem
VARNISH Disk

Cache Node
NGINX
VARNISH Mem
VARNISH Disk
App servers LVS
Appserver
Apache
Mediawiki

Appserver
Apache
Mediawiki

Persistent Varnish

- Much larger than in-memory cache
- Survives restarts
- Effective in-memory cache size: ~avg(mem size)
- Effective disk cache size: ~sum(disk size)


Clients
GeoDNS
Amsterdam, Netherlands
Cache servers LVS
hash(client.ip)
Cache Node
NGINX
VARNISH Mem
VARNISH Disk

Cache Node
NGINX
VARNISH Mem
VARNISH Disk
hash(req.url)
hash(req.url)
Ashburn, Virginia
Cache servers LVS
Cache Node
NGINX
VARNISH Mem
VARNISH Disk

Cache Node
NGINX
VARNISH Mem
VARNISH Disk
App servers LVS
Appserver
Apache
Mediawiki

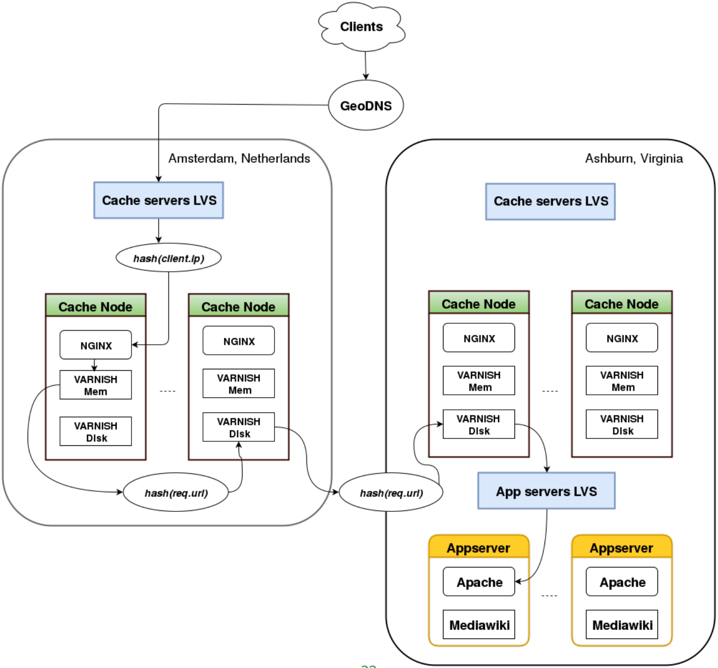
Appserver
Apache
Mediawiki

Inter-DC traffic routing

```
cache::route_table:
  eqiad:  ’direct’
  codfw:  ’eqiad’
  ulsfo:  ’codfw’
  esams:  ’eqiad’
```

Inter-DC traffic routing

- Varnish backends from etcd: directors.vcl.tpl.erb
  - puppet template -> golang template -> VCL file
- IPSec between DCs


Clients
GeoDNS
Amsterdam, Netherlands
Cache servers LVS
hash(client.ip)
Cache Node
NGINX
VARNISH Mem
VARNISH Disk

Cache Node
NGINX
VARNISH Mem
VARNISH Disk
hash(req.url)
hash(req.url)
Ashburn, Virginia
Cache servers LVS
Cache Node
NGINX
VARNISH Mem
VARNISH Disk

Cache Node
NGINX
VARNISH Mem
VARNISH Disk
App servers LVS
Appserver
Apache
Mediawiki

Appserver
Apache
Mediawiki

X-Cache

Cache miss:

```
$ curl –v https://en.wikipedia.org?test=$RANDOM 2>&1 | grep X–Cache
X–Cache: cp1068 miss, cp3040 miss, cp3042 miss
```

X-Cache

Cache miss:

```
$ curl –v https://en.wikipedia.org?test=$RANDOM 2>&1 | grep X–Cache
X–Cache: cp1068 miss, cp3040 miss, cp3042 miss
```

Cache hit:

```
$ curl –v https://en.wikipedia.org | grep X–Cache
X–Cache: cp1066 hit/3, cp3043 hit/5, cp3042 hit/21381
```

X-Cache

Cache miss:

```
$ curl –v https://en.wikipedia.org?test=$RANDOM 2>&1 | grep X–Cache
X–Cache: cp1068 miss, cp3040 miss, cp3042 miss
```

Cache hit:

```
$ curl –v https://en.wikipedia.org | grep X–Cache
X–Cache: cp1066 hit/3, cp3043 hit/5, cp3042 hit/21381
```

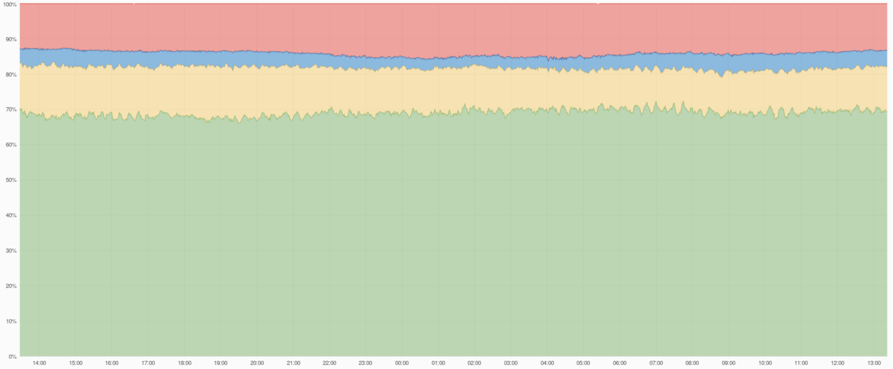
Forcing a specific DC:

```
$ curl –v https://en.wikipedia.org?test=$RANDOM \
  ––resolve en.wikipedia.org:443:208.80.153.224 2>&1 | grep X–Cache
X–Cache: cp1066 miss, cp2016 miss, cp2019 miss
```

Cache clusters

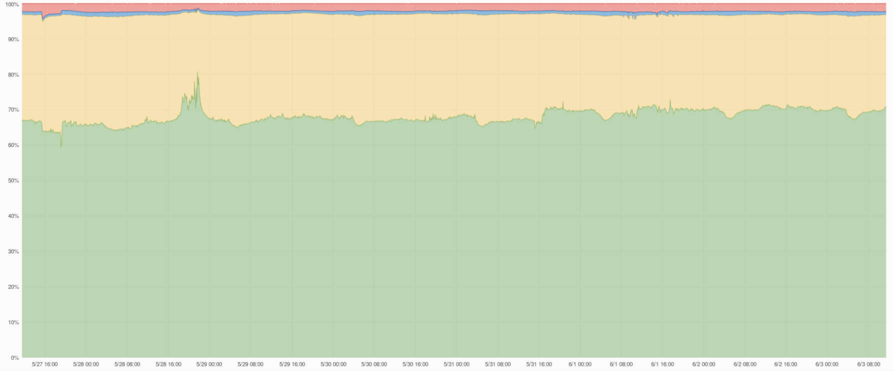
- Text: primary wiki traffic
- Upload: multimedia traffic (OpenStack Swift)
- Misc: other services (phabricator, gerrit, ...)
- Maps: maps.wikimedia.org

Terminating layer - text cluster



- Memory cache: 69%
- Local disk cache: 13%
- Remote disk cache: 4%
- Applayer: 14%

Terminating layer - upload cluster



- Memory cache: 68%
- Local disk cache: 29%
- Remote disk cache: 1%
- Applayer: 2%

Upgrading to Varnish 4

Varnish VCL

- Puppet ERB templating on top of VCL
- 22 files, 2605 lines
- Shared across:
  - clusters (text, upload, ...)
  - layers (in-mem, on-disk)
  - tiers (primary, secondary)
- 21 VTC test cases, 715 lines

Varnish 3

- 3.0.6-plus with WMF patches
  - consistent hashing
  - VMODs (in-tree!)
  - bugfixes
- V3 still running on two clusters: text and upload

Varnish 4 upgrade

- Bunch of patches forward ported
- VMODs now built out-of-tree
- VCL code upgrades
- Custom python modules reading VSM files forward ported
- Varnishkafka

V4 running on two clusters: misc and maps

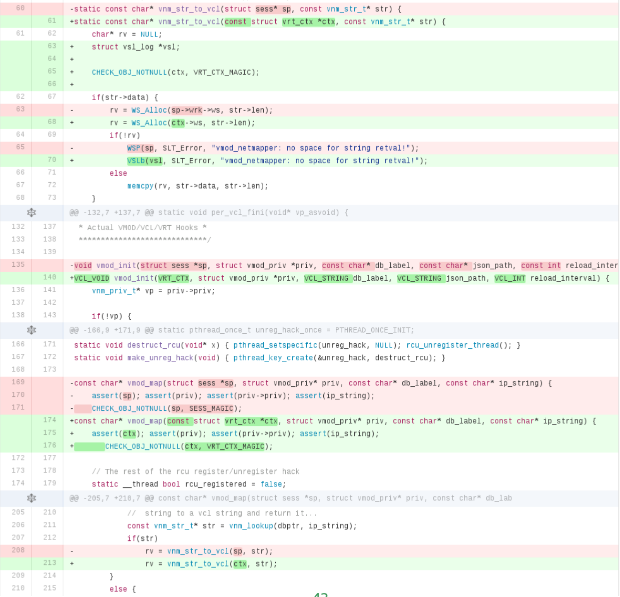
V4 packages

- Official Debian packaging:
  `git://anonscm.debian.org/pkg-varnish/pkg-varnish.git`
- WMF patches:
  `https://github.com/wikimedia/operations-debs-varnish4/tree/debian-wmf`
- Need to co-exist with v3 packages (main vs. experimental)
- APT pinning

VMODs

- vmod-vslp replacing our own chash VMOD
- vmod-netmapper forward-ported
- Packaged vmod-tbf and vmod-header

V4 VMOD porting



```diff
-static const char* vnm_str_to_vcl(struct sess* sp, const vnm_str_t* str) {
+static const char* vnm_str_to_vcl(const struct vrt_ctx *ctx, const vnm_str_t* str) {
     char* rv = NULL;
+    struct vsl_log *vsl;
+
+    CHECK_OBJ_NOTNULL(ctx, VRT_CTX_MAGIC);
+
     if(str->data) {
-        rv = WS_Alloc(sp->wrk->ws, str->len);
+        rv = WS_Alloc(ctx->ws, str->len);
         if(!rv)
-            WSP(sp, SLT_Error, "vmod_netmapper: no space for string retval!");
+            VSLb(vsl, SLT_Error, "vmod_netmapper: no space for string retval!");
         else
             memcpy(rv, str->data, str->len);
     }
@@ -132,7 +137,7 @@ static void per_vcl_fini(void* vp_asvoid) {
  * Actual VMOD/VCL/VRT Hooks *
  *****************************/

-void vmod_init(struct sess *sp, struct vmod_priv *priv, const char* db_label, const char* json_path, const int reload_inter
+VCL_VOID vmod_init(VRT_CTX, struct vmod_priv *priv, VCL_STRING db_label, VCL_STRING json_path, VCL_INT reload_interval) {
     vnm_priv_t* vp = priv->priv;

     if(!vp) {
@@ -166,9 +171,9 @@ static pthread_once_t unreg_hack_once = PTHREAD_ONCE_INIT;
 static void destruct_rcu(void* x) { pthread_setspecific(unreg_hack, NULL); rcu_unregister_thread(); }
 static void make_unreg_hack(void) { pthread_key_create(&unreg_hack, destruct_rcu); }

-const char* vmod_map(struct sess *sp, struct vmod_priv* priv, const char* db_label, const char* ip_string) {
-    assert(sp); assert(priv); assert(priv->priv); assert(ip_string);
-    CHECK_OBJ_NOTNULL(sp, SESS_MAGIC);
+const char* vmod_map(const struct vrt_ctx *ctx, struct vmod_priv* priv, const char* db_label, const char* ip_string) {
+    assert(ctx); assert(priv); assert(priv->priv); assert(ip_string);
+       CHECK_OBJ_NOTNULL(ctx, VRT_CTX_MAGIC);

     // The rest of the rcu register/unregister hack
     static __thread bool rcu_registered = false;
@@ -205,7 +210,7 @@ const char* vmod_map(struct sess *sp, struct vmod_priv* priv, const char* db_lab
             //  string to a vcl string and return it...
             const vnm_str_t* str = vnm_lookup(dbptr, ip_string);
             if(str)
-                rv = vnm_str_to_vcl(sp, str);
+                rv = vnm_str_to_vcl(ctx, str);
         }
         else {
```

V4 VMOD packaging

- Modifications to vmod-tbf to build out-of-tree
  - Header files path
  - Autotools
- vmod-header was done already, minor packaging changes

VCL code upgrades

- Need to support both v3 and v4 syntax (shared code)
- Hiera attribute to distinguish between the two
- ERB variables for straightforward replacements
  - $req_method → req.method vs. req.request
  - $resp_obj → resp vs. obj
  - ...
- 42 if @varnish_version4

varnishlog.py

- Python callbacks on VSL entries matching certain filters
- Ported to new VSL API using python-varnishapi: https://github.com/xcir/python-varnishapi
- Scripts depending on it also ported
  - TxRequest → BereqMethod
  - RxRequest → ReqMethod
  - RxStatus → BereqStatus
  - TxStatus → RespStatus

varnishkafka

- Analytics
- C program reading VSM files and sending data to kafka
- https://github.com/wikimedia/varnishkafka
- Lots of changes:
- 6 files changed, 612 insertions(+), 847 deletions(-)

varnishtest

- Started using it after Varnish Summit Berlin
- See ./modules/varnish/files/tests/
- Mocked backend (vtc_backend)
- Include test version of VCL files
- VCL code depends heavily on the specific server

```
[...]

varnish v1 -arg "-p vcc_err_unref=false" -vcl+backend {
    backend vtc_backend {
        .host = "${s1_addr}"; .port = "${s1_port}";
    }

    include "/usr/share/varnish/tests/wikimedia_misc-frontend.vcl";
} -start

client c1 {
    txreq -hdr "Host: git.wikimedia.org" -hdr "X-Forwarded-Proto: https"
    rxresp
    expect resp.status == 200
    expect resp.http.X-Client-IP == "127.0.0.1"

    txreq -hdr "Host: git.wikimedia.org"
    rxresp
    # http -> https redirect through _synth, we should still get X-Client-IP
    # (same as in _deliver)
    expect resp.status == 301
    expect resp.http.X-Client-IP == "127.0.0.1"
} -run
```

Future plans

Future plans - TLS

- Outbound TLS
- Add support for listening on unix domain socket

Future plans - backends

- Make backend routing more dynamic: eg, bypass layers on pass at the frontend
- etcd-backed director to dynamically depool/repool/re-weight

Future plans - caching strategies

- Only-If-Cached to probe other cache datacenters for objects before requesting from the applayer
- XKey integration to "tag" different versions of the same content and purge them all at once (eg: desktop vs. mobile)

Future plans - bloom filters

Very fast and space-efficient way to find out if something is definitely not in the set

- cache-on-second-fetch: avoid caching "rare" items
- 404 filter with the bloom set representing all legal URLs to help against randomized URL paths from botnets

Conclusions

- One of the most popular CDNs in the world is built in the open using FOSS
- Multi-layered Varnish setup
- Currently upgrading to Varnish 4
- Big plans for the future!

Cache servers

101 bare-metal servers

- 28 Amsterdam
- 27 Virginia
- 26 Texas
- 20 California

edns-client-subnet

```
import dns
import clientsubnetoption

def resolve(client_ip):
    cso = clientsubnetoption.ClientSubnetOption(client_ip)
    message = dns.message.make_query(’en.wikipedia.org’, ’A’)
    message.use_edns(options=[cso])

    # ns0.wikimedia.org
    r = dns.query.udp(message, ’208.80.154.238’)

    for a in r.answer:
        print a

print "United␣States"
resolve(’199.217.118.41’)

print "Italy"
resolve(’151.1.1.1’)
```

edns-client-subnet

```
$ python resolve.py
United States
en.wikipedia.org. 600 IN A 208.80.153.224
Italy
en.wikipedia.org. 600 IN A 91.198.174.192
```