TINY CHANGE

huge improvement*

*maybe not that huge

{introduction}

This is the result of several years researching how to best express HTTP caching rules in VCL. The focus of the research was not Varnish internals, but desirable behavior. This session is backed only by prior knowledge (or misunderstanding) via my personal Varnish and HTTP experience and you are most welcome to challenge suggestions made on shaky ground.

{disclaimer}

This is a high level overview of Varnish’s caching model, made with the assumption that attendees can follow. Still, I tried to make it accessible.

Agenda

- Warm up
  - Cache-Control quizz for cache wizzards
- New paradigm
  - Waiting list today
  - Waiting list tomorrow?
- Cool down
  - Cache-Control redux
- Consequences
  - RFC compliance revisited (focus on behavior)
  - Built-in VCL (promote and use more caching criteria)

{warm up}

Cache-Control pop quiz

- Which directives tell us not to cache?

- Cheat sheet
  - `public`
  - `private`
  - `no-cache`
  - `no-store`
  - `must-revalidate`
  - `proxy-revalidate`
  - `stale-while-revalidate`
  - `max-age`
  - `s-maxage`

{new paradigm*}

*allegedly a tiny change

Waiting list today (1/3)

- Lookup is serialized
- Possible outcomes (plural)
  - Straight miss => trigger fetch and wait for it
  - Straight hit => probably deliver a cached copy
  - Grace hit => trigger a bgfetch and probably deliver a cached copy
  - Hit-for-miss => trigger a fetch and wait for it
  - Hit-for-pass => trigger a private fetch and wait for it
  - Busy hit => disembark into a waiting list

Waiting list today (2/3)

- Waiting list outcome (singular)
  - Reembark at the lookup step
  - Find the new lookup outcome (plural)
  - It keeps the waiting list “stateless”
- Lookup is serialized
  - A property inherited by the waiting list
- When the new object is already expired
  - A cache miss “starts” a new waiting list
  - Difficult to diagnose but usually easy to remedy

Waiting list today (3/3)

- Moving parts
  - Lookup finds (or creates) an objhead
  - Requests enter objhead waiting lists (with a ref)
  - A rush is triggered on the objhead
  - Releasing an objcore ref triggers a rush too
    - The objhead may trigger spurious rushes
  - A cacheable object may not be compatible (Vary)

Waiting list tomorrow? (1/3)

- Validation straw man
  - Getting an expired object is valid if the origin server generated it
  - The origin server could be another Varnish tier in grace mode
  - What prevents the resource from being cached is the built-in VCL
    - Core code contains a mitigation for `vcl_backend_error`
  - Currently a small change can help, ideally we do nothing

```
 sub vcl_beresp_stale {
-        if (beresp.ttl <= 0s) {
+        if (beresp.ttl + beresp.grace <= 0s) {
                 call vcl_beresp_hitmiss;
         }
 }
```

Waiting list tomorrow? (2/3)

- Change of paradigm
  - Move the waiting list from objhead to busyobj
    - Keep a busyobj ref instead of an objhead ref in a parked req
  - Inspect the busyobj on reembark
    - Incompatible variant => perform a new lookup (serialized loop back)
    - `OC_F_HFM` => miss transition
    - `OC_F_HFP` => pass transition
    - `OC_F_FAILED` => `stale-if-error` (discussed later)
    - None of the above => valid object, hit transition
  - Waiting list serialization almost gone
    - Shifted from expired objects to variants proliferation
    - We have `vary_notice` for diagnostics
    - We could have a `vary_limit` too

Waiting list tomorrow? (3/3)

- Walk away from the waiting list
  - Honorable mention for today
  - Already implemented in Varnish Enterprise
  - Already submitted to Varnish Cache
    - Not in a merge-able state
  - Limited to h2 connections
  - Maybe best postponed until after this change, if applicable

{cool down}

Cache-Control Rosetta stone (1/3)

Cache-Control directive	VCL
public	# override uncacheable criteria
private	set beresp.uncacheable = true; # should be overridable by user VCL

no-cache	set beresp.ttl = 0s;
	set beresp.grace = 0s;
	set beresp.keep = param.default_keep;

no-store	set beresp.uncacheable = true;
{must,proxy}-revalidate	set beresp.grace = 0s;
stale-while-revalidate=X	set beresp.grace = Xs;
s-maxage=X max-age=X	set beresp.ttl = Xs – beresp.age; # shared max age takes precedence

Cache-Control Rosetta stone (2/3)

Cache-Control directive	VCL
`private=X`	`set beresp.private = “X”;`
`no-cache=X`	`set beresp.nocache = “X”;`

- New HTTP header flags
  - `HTTPH_A_PRIVATE` and `HTTPH_A_NOCACHE`
  - Private and no-cache headers only served to the original request
    - But waiting list hits can deliver no-cache headers
  - Private and no-cache headers not merged with 304 headers
  - Tolerate private and no-cache headers in persistent caches
    - Maybe allow stevedores to repack headers to disk, omitting them

Cache-Control Rosetta stone (3/3)

Cache-Control directive	VCL
`immutable`	`set beresp.ttl = param.immutable_ttl;`
`no-transform`	`# fail depending on beresp.filters?`
`must-understand`	`# hitmiss depending on beresp.status?`
`stale-if-error=X`	`set beresp.mercy = Xs; # for stale_oc`

- New parameters to consider
  - `immutable_ttl` parameter (default to 1 week for example)
  - `uncacheable_ttl` parameter (default to 2 minutes)
  - Expose certain parameters in VCL
    - Use the `param.uncacheable_ttl` symbol in `vcl_beresp_hitmiss`

{consequences}

Default parameters

- Currently we cache by default
  - `default_ttl=2m`
  - `default_grace=10s`
  - `default_keep=0s`
- Instead we could validate by default
  - `default_ttl=0s`
  - `default_grace=0s`
  - `default_keep=2m`
  - On the condition that waiting lists don’t serialize (modulus `Vary`)
  - Make sure we only *keep* revalidation candidates
  - Potentially a lower hit ratio, but overall safer

Parameter symbols in VCL

- Survey existing parameters
  - Expose some as read-only `param.*` symbols
  - Start with DURATION values (timeouts etc)
  - Consider new parameters for magic values

```
 sub vcl_beresp_hitmiss {
-        set beresp.ttl = 120s;
+        set beresp.ttl = param.uncacheable_ttl;
         set beresp.uncacheable = true;
         return (deliver);
 }
```

To cache or not to cache (1/9)

- RFC9111 Section 3 gives a nice (but not comprehensive) breakdown
  - We had it already in RFC7234 Section 3
- This could be implemented in VCL
  - We begin `vcl_backend_response` with a fully computed `beresp`
  - We would move some checks to `vcl_backend_response`
  - This is <u>more expensive</u> than a `pass` transition from `vcl_recv`
- How to deal with conflicting directives?
  - `Cache-Control: stale-while-revalidate=42, must-revalidate`
  - According to the Rosetta Stone we have Schrödinger's grace
  - The sane solution is to iterate: <u>later directives take precedence</u>
  - Precedence requires the promotion of more caching directives to new `beresp` fields

To cache or not to cache (2/9)

- `Cache-Control: public`
  - Raises `beresp.is_public`, cleared by the `private` directive
  - Overrides the Authorization header
  - Same treatment for the Cookie header (as we do today)

```
sub vcl_bereq_authorization {
        if (bereq.http.Authorization && !beresp.is_public) {
                call vcl_beresp_hitmiss;
        }
}
```

To cache or not to cache (3/9)

- `Cache-Control: private`
  - Raises `beresp.is_private`
  - Cleared by the `public` directive
  - Also cleared by the `private=X` directive
    - Which implies only a subset of the response is private

To cache or not to cache (4/9)

- `Cache-Control: private=X`
  - Map to `beresp.private` (contains a list of header names)
  - Set or clear `HTTPH_A_PRIVATE` flags
  - New `HEADER.is_private` property

```
 sub vcl_beresp_cookie {
-        if (beresp.http.Set-Cookie) {
+        if (beresp.http.Set-Cookie &&
             !beresp.http.Set-Cookie.is_private) {
                 call vcl_beresp_hitmiss;
         }
 }
```

To cache or not to cache (5/9)

- `Cache-Control: no-cache=X`
  - Map to `beresp.nocache` (just like `beresp.private`)
  - Set or clear `HTTPH_A_NOCACHE` flags
  - New `HEADER.is_nocache` property
  - Probably not used by the built-in VCL
    - But required for waiting list hits

To cache or not to cache (6/9)

- `Cache-Control: stale-if-error=X`
  - Map to `beresp.mercy`
  - Mercy period capped to grace period
  - Applies when there is a stale objcore

To cache or not to cache (7/9)

- `Cache-Control: no-transform`
  - Map to a `beresp.can_transform` field?
  - Fail if a content-altering filter is set up?
  - We don’t know what filters do the `beresp.body`
    - VRG is probably fine
  - What about `resp.body`?
  - Probably best ignored

To cache or not to cache (8/9)

- `Cache-Control: must-understand`
  - Map to a `beresp.must_understand` field?
  - Overrides `no-store`, but requires a status code check
    - This could be moved from core code to the built-in VCL

```
sub vcl_beresp_status {
        if (beresp.status == <custom>) {
                # skip must-understand check
                return;
        }
}
```

To cache or not to cache (9/9)

```
 beresp

         Type: HTTP

         Readable from: vcl_backend_response, vcl_backend_error

+        Resettable from: vcl_backend_response, vcl_backend_error
+
         The entire backend response HTTP data structure, useful as
         argument to VMOD functions.
+
+        When ``beresp`` is reset, the following fields are recomputed
+        based on the current values of ``beresp.http.*``:
+
+        - beresp.ttl
+        - beresp.grace
+        - beresp.mercy
+        - beresp.keep
+        - beresp.uncacheable
+        - beresp.is_public
+        - beresp.is_private
+        - beresp.private
+        - beresp.nocache
+        - beresp.must_understand
```

RFC9111 Section 3 washed

- the request method is understood by the cache
- ~~the response status code is final~~*
- if ~~the response status code is 206 or 304, or~~ the must-understand cache directive is present: the cache understands the response status code
- ~~the no-store cache directive is not present in the response~~
- if the cache is shared: the private response directive is either not present or allows a shared cache to store a modified response
- if the cache is shared: the Authorization header field is not present in the request or a response directive is present that explicitly allows shared caching
- the response contains at least one of the following:
  - a public response directive
  - ~~a private response directive, if the cache is not shared~~
  - an Expires header field
  - a max-age response directive
  - if the cache is shared: an s-maxage response directive
  - ~~a cache extension that allows it to be cached~~*
  - a status code that is defined as heuristically cacheable

Built-in vcl_recv

```
 sub vcl_recv {
         call vcl_req_host;
         call vcl_req_method;
-        call vcl_req_authorization;
-        call vcl_req_cookie;
 }
-
-sub vcl_req_authorization {
-        if (req.http.Authorization) {
-                # Not cacheable by default.
-                return (pass);
-        }
-}
-
-sub vcl_req_cookie {
-        if (req.http.Cookie) {
-                # Risky to cache by default.
-                return (pass);
-        }
-}
```

Built-in vcl_backend_response

```
 sub vcl_backend_response {
         if (bereq.uncacheable) {
                 return (deliver);
         }
+        call vcl_bereq_authorization;
+        call vcl_bereq_cookie;
+        call vcl_beresp_status;
+        call vcl_beresp_private;
-        call vcl_beresp_stale;
         call vcl_beresp_cookie;
-        call vcl_beresp_control;
         call vcl_beresp_vary;
 }
```

Built-in vcl_bereq_*

```
sub vcl_bereq_authorization {
        if (bereq.http.Authorization && !beresp.public) {
                # Not cacheable unless stated explicitly
                call vcl_beresp_hitmiss;
        }
}

sub vcl_bereq_cookie {
        if (bereq.http.Cookie && !beresp.public) {
                # Risky to cache unless stated explicitly
                call vcl_beresp_hitmiss;
        }
}
```

Built-in vcl_beresp_status

```
sub vcl_beresp_status {
        if (!beresp.must_understand) {
                return;
        }
        if (beresp.status != 200 &&
            beresp.status != 203 &&
            beresp.status != 204 &&
            beresp.status != 300 &&
            beresp.status != 301 &&
            beresp.status != 304 &&
            beresp.status != 404 &&
            beresp.status != 410 &&
            beresp.status != 414) {
                call vcl_beresp_hitmiss;
        }
}
```

Built-in vcl_beresp_private

```
sub vcl_beresp_private {
        if (beresp.is_private) {
                call vcl_beresp_hitmiss;
        }
}
```

Built-in VCL miscellany

- After fiddling with `beresp` headers
  - `reset beresp;`
  - As if we just entered `vcl_backend_response` in this state
- Remove subroutines (or leave them empty)
  - `vcl_beresp_stale` (change of paradigm to validation by default)
  - `vcl_beresp_control` (what about `Surrogate-Control: no-store`?)
- Rename subroutines (or alias them)
  - `vcl_beresp_hitmiss` to `vcl_beresp_uncacheable`
  - `vcl_builtin_*` to something more specific
    - `vcl_builtin_backend_fetch` to `vcl_bereq_body`
    - `vcl_builtin_synth` to `vcl_synth_body`
    - etc

{thank you}